Prise en compte de l'énergie dans la phase d'exploitation des bases de données volumineuses

Thesis · May 2017

DOI: 10.13140/RG.2.2.36242.04807

1 author:

Amine Roukh
Université de Mostaganem
16 PUBLICATIONS **226** CITATIONS

SEE PROFILE

Université Abdelhamid Ibn Badis Mostaganem

Faculté des Sciences Exactes et Informatique

Département d'informatique

Thèse de Doctorat

Présentée par :

Amine ROUKH

Prise en compte de l'énergie dans la phase d'exploitation des bases de données volumineuses

---

Spécialité : Informatique

Option : Apprentissage Automatique et Web Intelligence

Soutenue le 14/05/2017 devant un jury composé de :

Président du jury	**Dr. Amir ABDESSAMAD**	(Université de Mostaganem)
Examinateurs	**Dr. Karim SEHABA**	(Université de Mostaganem)
	Pr. Djamal BENSLIMANE	(IUT, Université de Lyon 1, France)
Directeur de thèse	**Dr. Omar BELHAMITI**	(Université de Mostaganem)
Co-directeur de thèse	**Pr. Ladjel BELLATRECHE**	(ISAE/ENSMA, Poitiers, France)
Invité	**Pr. Nadjia BENBLIDIA**	(Université de Blida)

Résumé

À l'ère du *Big Data*, la gestion de la consommation d'énergie par les serveurs et les centres de données est devenue un défi majeur pour les entreprises, les institutions et les pays. Parmi les applications déployées sur les centres de données, on distingue les systèmes de gestion de base de données (SGBD), qui sont l'un des principaux consommateurs d'énergie lors de l'exécution des requêtes complexes impliquant une masse de données gigantesque. Par ailleurs, le traitement de ce type de données requiert des infrastructures logicielles et matérielles coûteuses, et qui consomment beaucoup d'énergie. Les pratiques actuelles d'utilisation et d'exploitation des bases de données extrêmement larges, montrent que le coût énergétique de traitement de requête est totalement négligé par les utilisateurs et également par les concepteurs. Sachant que le facteur le plus important pour l'utilisateur est la minimisation du temps de réponse des requêtes. Dans cette thèse nous proposons une formalisation multi-objectifs des problèmes d'exploitation des bases de données, en tenant compte de deux besoins non-fonctionnels : la performance et la consommation d'énergie lors de l'exécution d'une charge de requêtes. Cette formalisation nous a permis de tirer parti de certaines techniques avancées proposées dans l'état de l'art pour la résolution des problèmes d'optimisation multi-objectifs. Pour ce faire, nous développons en premier lieu des modèles de coût pour estimer le coût énergétique des requêtes exécutées d'une manière isolée ou parallèle. Ces modèles de coût sont ensuite intégrés dans l'un des modules les plus importants dans un SGBD, qui est le module de traitement de requêtes. La nouvelle tâche de ce module est la sélection des plans d'exécution des requêtes avec le compromis souhaité par les utilisateurs entre le temps et l'énergie des requêtes. De plus, nous proposons une initiative qui intègre la dimension énergétique dans la phase de conception physique des bases de données, afin de sélectionner des structures d'optimisation en prenant en compte les aspects énergétiques. Nous étudions le cas des vues matérialisées, l'une des structures d'optimisation redondantes très répondues. Dans chaque contribution de cette thèse, des expérimentations intensives on été menées en utilisant un dispositif réel pour les mesures d'énergie et les données des benchmarks TPC-H, TPC-DS et SBB avec plusieurs configurations matérielles et logicielles.

Mots-clés : Efficacité énergétique, modèles de coût, traitement de requêtes, conception physique, gestion d'énergie, optimisation multi-objectifs.

مُلَخَّصٌ

في عصرِ البيَاناتِ الضخمةِ، أصبحت إدارَةُ استهلاك الطَّاقة بالنِّسبَة للخَوَادِمِ ومراكز البيانات تحدِّيًا كبيرا للشركات والمؤسسات والدُّوَل. من بين التطبيقات المنتشرة بكثرة على مراكز البيانات، نَجِدُ أنَّ أنظمة إدارة قواعد البيانات واحدة من أكبر مستهلكي الطَّاقة الكهرَبائيَّة، وذلك أثناء تنفيذ الاسْتِغْلاَمَات المعقَّدة التي تنْطوي على حجمٍ كبير جدًّا من البيانات. وعِلاوةً على ذلك، عِلاجُ هذا النوع من قواعد البيانات يتطلَّب تكنولوجيا وبُنيَة تحتية باهظةَ الثَّمن ومستهلكةً كبيرةً للطَّاقة الكهربائيَّة. إلى زمن غيْرِ بعيد، كانت تَكْلُفَةُ الطَّاقة أثناء معالجة الاستعلامات الخاصَّة باستخدام وتشغيل قواعد البيانات كبيرةِ الحجم مُهْمَلَةً تماما، سواءا من قِبَلِ المستخدمين أو من قبل المُصمِّمين. حيث أنَّ العامل الأكثَرَ أهمِّية للمستخدِم هو تقْليلُ زمَن مُعالجَة الاستعلامات وتَلَقِّي النَّتائج بسُرعة. في هذه الأُطروحة نقترحُ صِياغةً مُتعدِدة الأهداف لمشاكل استغلال قواعد البيانات، وهذا عن طريق الأخْذِ بعيْن الاعتبار كلاًّ من الاحتياجات غيْرِ الوظِيفِيَّة: تحسينُ الأداءِ وتقليلُ استهلاك الطَّاقة عند تشغيل مجموعةٍ من الاستعلامات. هذه الصِّياغةُ سَمَحَت بالاستفادة من التِّقنيَّات المُتقدِّمة المُقترَحة في حالة التِّقنيِّة الصِناعيَّة السابقة من أجل حلِّ مُشكل الأَمْثَلَةِ مُتعدِّدَةِ الأهداف. لهذا، في أوَّلِ الأمر قُمنا بتطوير نماذجِ التَّكلُفَة لتقدير تكلُفة الطَّاقة اللاَّزمة لتشغيل الاستعلامات، بطريقة معزولةٍ أو متوازيةٍ. بعد ذلك، قمنا بدَمج هذه النَّماذج في واحِدَة من أهمِّ الوحدات في نُظم إدارة قواعد البيانات، ألا وهي وِحدة معالجة الاستعلام. الهدف الجديد لهذه الوِحدة هو اختيارُ خُطَطِ تنفيذ الاستعلام مع الأخذ بعيْن الاعتبار لمجموعةٍ من قِيَمِ التَّسْوِية والمُقايضة بيْن وقت التشغيل والطَّاقة المُستهلكة، هذه القِيمُ المستخدمون هُمُ المسؤولون عن إدخالها. أيضا، اقترحنا صيغَةً لدمجِ البُعدِ الطَّاقويِّ في مرحلة التصميم الماديِّ لقواعد البيانات، وذلك عن طريق اختيار هيَاكِلِ الأَمْثَلَةِ مع الأخذ بعين الاعتبار جانِبَ استهلاك الطَّاقة. نتيجة لذلك، قمنا بدراسة تقنيِّة حِفظ نتائج الاستعلامات، والتي تُعدُّ واحدةً من أهمِّ هياكِل الأمثَلة المُستعملة بكثرة. في كلِّ مساهمة من مساهمات هذه الأُطروحةِ، قمنا بإجراء تجاربَ واسعةِ النِّطاق باستخدام جهاز فعليٍّ لقياس استهلاك القُدرة الكهربائيَّة للخادِم، واعتمادًا على البيانات والاستعلامات الخاصَّة بمقاييسِ كلٍّ من TPC-DS، TPC-H وSBB مع الحِرص على تنْويع المكوِّنات المادِّية والبرمجيَّة في كلِّ تَجربة.

الكلمات المفتاحيَّة: كفاءةُ استخدام الطَّاقة، نماذجُ التَّكلفة، معالَجَةُ الاستعلام، التَّصميم الماديُّ، إدارة الطَّاقة، أمثَلةٌ متعدِّدة الأهداف.

Abstract

In the Big Data Era, the management of energy consumption by servers and data centers has become a challenging issue for companies, institutions, and countries. In data-centric applications, Database Management Systems are one of the major energy consumers when executing complex queries involving very large databases. Moreover, the processing of this type of data requires costly and energy-intensive computing and hardware infrastructures. Current practices in the use and exploitation of very large databases indicate that the energy cost of query is totally neglected by users and also by designers. Knowing that the most important factor for the user is minimizing the response time of queries. In this thesis we propose a multi-objective formalization of the databases exploitation techniques, taking into account two non-functional requirements : performance and energy consumption during the execution of a queries workload. This formalization allow us to take advantage of the advanced techniques proposed in the state-of-the-art for solving multi-objective optimization problems. For this purpose, we first develop cost models that estimate the energy consumption of queries executed in an isolated or parallel manner. These cost models are then integrated into one of the most important modules in a DBMS, which is the query processing module. The new objective of this module is the selection of execution plans of queries with the trade-off desired by the users between the time and the energy of the queries. Furthermore, we propose an initiative that integrates the energy dimension in the physical design phase of databases, in order to select optimization structures taking into account the energy aspects. We study the case of materialized views, one of the redundant optimization structures that is heavy used by database administrator. In each contribution of our thesis, intensive experiments are conducted using a real device for energy measurements and data of the TPC-H, TPC-DS and SBB benchmarks with various hardware and software configurations.

Keywords : Energy efficiency, cost models, query processing, physical design, energy management, multi-objective optimization.

Remerciements

Je mesure la chance qui m'a été donnée d'être encadré par des personnes aussi impliquées, ouvertes et compétentes, qui ont cru en mes capacités pour mener à bien ce travail, et sans qui ce travail n'aurait jamais vu le jour. Qu'ils trouvent ici le témoignage de toute ma gratitude :

Ladjel BELLATRECHE, qui sait faire voler en éclat chaque idée reçue, qui s'est dévoué généreusement à ma thèse et qui m'a donné goût à la recherche. Il m'a fait énormément apprendre tant sur le plan scientifique qu'humain.

Omar BELHAMITI, pour son encadrement minutieux et ses bonnes qualités pédagogiques et scientifiques. Ses conseils avisés sont précieux et sa gentillesse m'est inégalé.

Un grand merci à Karim SEHABA, Djamal BENSLIMANE et Nadjia BENBLIDIA de m'avoir fait l'honneur d'être examinateurs de cette thèse, ainsi qu'à Amir ABDESSAMAD pour avoir accepté d'être membre du jury en tant que président. Je suis très honoré de l'intérêt qu'ils ont porté à mes travaux.

Mes remerciements vont également :

À Fouad HENNI qui a été toujours disponible pour m'apporter son aide. Ses conseils m'ont été très utiles, sa rigueur au travail est un exemple pour moi. À Houari BENMEKKI qui m'a aidé à effectuer une partie de ma thèse à Poitiers et qui n'a jamais cessé de mettre à ma disposition les moyens qui m'ont permis d'aboutir cette thèse. À Emmanuel GROLLEAU, directeur du laboratoire LIAS/ENSMA pour son accueil et ses qualités humaines hors du commun.

À tous les membres d'Université de Mostaganem et du laboratoire LIAS, en particulier : Mohamed, Saliha, Asma, Djamila, Amel, Walid, Amin, Jalil, Maghnia, Liela, Houria et Sidahemd, ainsi que Ahcène, Selma, Sabrina, Abdelkader, Aymen, Lahcène avec lesquels j'ai partagé des moments inoubliables.

À mes amis et collègues sans exception, pour leur présence, leur respect et leur gentillesse.

Je ne saurais finir sans exprimer mes remerciements aux personnes qui me sont les plus proches : Mon feu Père, à qui j'adresse un retentissant hommage, pour qui entre autres, j'ai spécialement porté cette tenue aujourd'hui, et à ma Mère, pour avoir pu cimenter ma vie dans un douillet cocon familial, fait d'amour, de bravoure et de zèle au travail ; Mes deux frères Abdelatif et Sofiane, qui étaient toujours présents pour m'aider et me réconforter.

Et enfin, merci à tous ceux qui ont participé de près ou de loin à l'aboutissement de ce travail.

En Hommage à mon Père, que ton âme repose en paix !
À tous ceux qui me sont chers :
Ma mère,
Mes frères.

Table des matières

Liste des figures

Liste des tableaux

Première partie

Introduction Générale

Chapitre

1 Introduction Générale

« Et au-dessus de tout homme détenant la science, il y a un savant plus docte que lui. »
— Joseph, v. 76

Sommaire

1.1 Contexte et problématique

Depuis quelques années, la communauté internationale regroupant les états, les gouvernements, les associations, et les usagers s'intéresse de près aux changements climatiques en proposant des initiatives pour limiter le réchauffement climatique. En 2015, la France a organisé à Paris la Conférence mondiale des parties à la Convention-cadre de l'ONU portant sur cette problématique, appelée aussi « COP 21 », où 195 pays étaient représentés. L'un des objectifs primordiaux de cette convention est de sensibiliser les *pays pollueurs* et proposer des initiatives accompagnées par des solutions durables pour réduire le réchauffement climatique lié aux gaz à effet de serre (émissions de carbone ou CO_2). Rester immobiles de la part de la communauté internationale et les usagers entrainera les conséquences néfastes sur la planète comme : **(a)** la multiplication des vagues de chaleur, **(b)** des sécheresses et des inondations, **(c)** la fonte accélérée des glaciers.

Depuis quelques années, un nombre important d'initiatives ont été menées couvrant plusieurs secteurs comme le *bâtiment* (*green house*) [200], *l'automobile* (l'émergence des voitures électriques) [257], *l'électronique* [19], *les réseaux sans fil* [210], etc. Les expériences énergétiques menées dans ces secteurs doivent être *généralisées* et *capitalisées* afin de les *reproduire* et les *réutiliser* dans d'autres secteurs dans lesquels l'énergie électrique n'est pas complètement intégrée. Pour illustrer ces avancées, nous prenons l'exemple du secteur du bâtiment qui a connu d'énormes progrès sur la prise en compte de l'énergie. Il est intéressant de noter qu'il est parmi les plus grand consommateurs d'énergie. En France par exemple, il représente 43% des consommations énergétiques, soit 1,1 tonne équivalent pétrole par an et par habitant [1]. D'après les dernières statistiques, le poste « logement » représente à lui seul 30% du budget des ménages. Devant cette situation, plusieurs initiatives ont été prises pour réduire cette consommation que nous proposons de classer en deux catégories principales selon la nature du bâtiment : **(i)** les initiatives pour les bâtiments existants et **(ii)** les initiatives pour les bâtiments neufs. Pour les bâtiments existants, les pouvoirs publics ont beaucoup communiqué afin de sensibiliser et inciter les usagers à réduire leur consommation énergétique afin qu'ils aient un comportement éco-citoyens. Des gestes simples pourraient contribuer à cette réduction. Nous pouvons citer par exemple : **(a)** s'arranger pour que les appareils électriques fonctionnent à pleine charge, **(b)** penser aux heures creuses pour la mise en route des appareils, **(c)** acheter des appareils ménagers en tenant compte leur niveau de consommation électrique, **(d)** limiter le plus possible le nombre d'appareils en veille, **(e)** utiliser des ampoules basse énergie. Pour une réduction maximale d'énergie, ces gestes doivent concerner tout composant électrique du bâtiment susceptible de contribuer à la réduction de la consommation énergétique.

En ce qui concerne les initiatives pour les bâtiments neufs, l'intégration de l'énergie est prise en compte dès la conception de ce dernier. Plusieurs réglementations ont été mises en place concernant l'utilisation des matériaux et équipements plus performants et à l'utilisation des énergies. Afin d'assurer un maximum de réduction énergétique, les deux types d'initiatives doivent être combinées. Sur le plan humain, tous les acteurs de la chaîne de construction de bâtiments : les *concepteurs*, les *producteurs* de matériaux de construction et les *usagers* doivent travailler ensemble pour assurer cet objectif.

Revenons à notre domaine d'étude qui est l'informatique. Ce secteur avec ses composantes (matérielles et logicielles) et ses usagers participe intensivement au phénomène de réchauffement climatique. L'une des causes principales de cette situation est la demande continuelle de calcul intensif du *déluge de*

1. http://www.logement.gouv.fr/la-politique-energetique-dans-les-batiments-2445

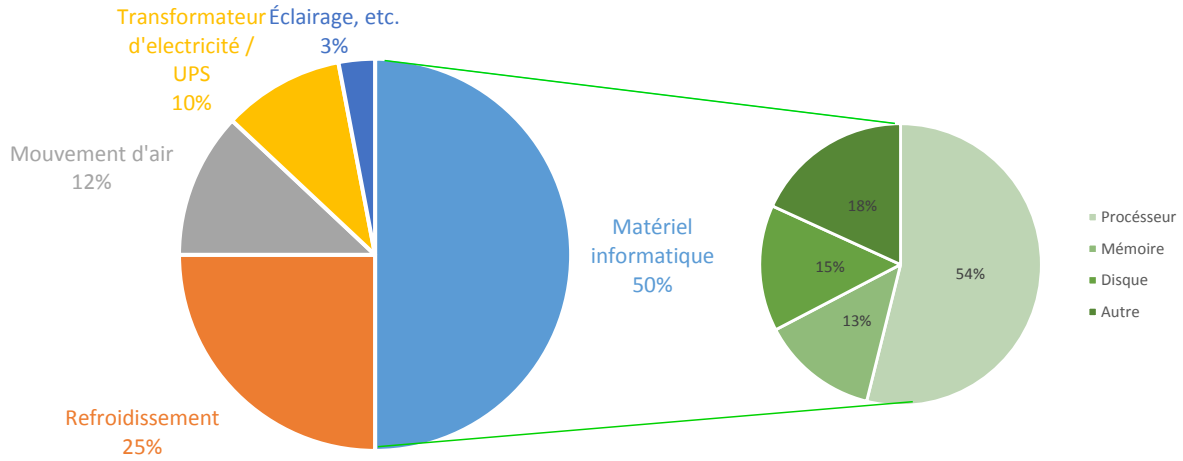


FIGURE 1.1 – La consommation d'énergie dans un centre de données.

données générées par les contenus numériques, les applications autour des *données massives* (« Big Data »), l'*E-commerce*, et le *trafic Internet*. En conséquence, les centres de données deviennent l'épine dorsale pour toute entreprise, organisme et fournisseur de stockage de données comme *Amazon*, *Facebook*, *Google*, etc. Si nous nous référons au Conseil de Défense des Ressources Naturelles, il a indiqué que les centres de données ont consommé 91 milliards de kilowatt-heures en 2013 [77]. Ceci représente environ 3% de la consommation d'électricité mondiale, avec un coût annuel de 9 milliards de dollars [149]. Citons l'exemple du centre de données construit par *Interxion*, l'installation consomme 64 mega-watts, soit l'équivalent d'une ville de 50 000 habitants [2]. Avec la chaleur qu'ils dégagent, ils génèrent 200 millions de tonnes de gaz à effet de serre [77] ; Malheureusement, pour les défenseurs de l'environnement, cette situation ne fait que s'empirer. Les experts du secteur, tels que le SMARTer 2020, rapportent que la consommation des centres de données mondiaux augmenteront de 7% chaque année jusqu'en 2020 [251], en raison du développement d'Internet et de l'augmentation des besoins de stockage des entreprises.

La consommation énergétique électrique de ces centres est répartie de la manière suivante : **(i)** le matériel informatique déployé sur ces centres tels que serveurs de traitement, **(ii)** le stockage et la communication, **(iii)** des équipements de refroidissement d'infrastructure (pour un kilowatt dépensé par un serveur, un autre kilowatt serait nécessaire pour le refroidir [168, 265]) et **(iv)** leur fonctionnement sans arrêt. Figure 1.1 illustre la distribution de consommation d'énergie des équipements informatiques dans un centre de données. Nous remarquons que le processeur, la mémoire et le disque consomment la moitié d'énergie totale.

Face à cette situation, les industriels et les chercheurs de la communauté scientifique ne sont pas restés indifférents, ils ont pris des initiatives accompagnées de solutions réelles couvrant plusieurs actions, à savoir : **(a)** la conception de matériel avec une consommation énergétique réduite, **(b)** la restructuration des logiciels pour minimiser la consommation, **(c)** l'exploitation de divers états d'alimentation disponibles dans les nouveaux matériaux, et **(d)** une bonne conception, utilisation et contrôle de l'infrastructure des centres de données [141]. Ces actions ressemblent fortement à celles utilisées dans le secteur du bâtiment. Plus précisément :

2. http://www.interxion.com/fr/Implantations/france/paris/par7/

- En ce qui concerne la conception de matériel avec une consommation énergétique réduite, les producteurs ont mis en place un label, appelé *l'efficacité énergétique* (communément appelée PUE - power usage effectiveness) mesurant la quantité d'énergie totale consommée par un centre de données. Le consortium International Green Grid [3], crée en 2007 et qui regroupe 501 membres et plus de 200 sociétés mondiales, vise au « verdissement » des centres de données. L'ETSI (European Telecommunications Standards Institute) – institut de normalisation européen des télécoms – propose en juin 2014, une nouvelle mesure de l'efficacité énergétique comparable à celle que l'on retrouve déjà dans l'électroménager [4]. Celle-ci comporte 5 niveaux de performance, du A « vert » au I « noir », en passant par le G « rouge » ;
- En général, l'efficacité énergétique n'est pas synonyme d'efficacité de calcul [141]. En particulier, des techniques comme le regroupement des calculs et des transferts de données améliorent l'efficacité énergétique des logiciels. Cela est du fait qu'elles allongent les périodes d'inactivité et permettent aux dispositifs d'entrées/sorties dans des états de faible puissance. En outre, certaines opérations consomment moins d'énergie que d'autres pour un travail effectif équivalent. Il est intéressant de noter qu'une progression remarquable est constatée dans le développement de logiciels embarqués économiques. Cependant, il manque encore un travail considérable en terme méthodologique, conceptuel, et validation pour le développement de logiciels économiques de tout bord [141].
- De nos jours, presque tous les principaux composants d'un serveur offrent des modes de contrôle d'énergie. Ils se matérialisent sous forme d'états de puissance : une collection de modes opérationnels qui compromettent la consommation d'énergie pour la performance. Un état de puissance pour un composant est dit *actif* si celui-ci reste opérationnel dans cet état, sinon l'état est dit *inactif*. L'ACPI (*Advanced Configuration and Power Interface*) fournit une nomenclature standard pour ces états et définit également les interfaces logicielles pour les gérer. L'ACPI est implémentée dans tous les principaux systèmes d'exploitation (SE) sous forme de la gestion de l'alimentation dirigée par le SE [5].
- Plusieurs technologies sont actuellement envisagées pour rendre l'infrastructure d'un centre de données plus efficace. Par exemple, la plupart des serveurs fonctionnent à une utilisation relativement faible la plupart du temps. Cependant, une double alimentation redondante (PSU) est souvent utilisée pour la fiabilité, ce qui se traduit par une charge et une efficacité relativement faible. Pour les infrastructures de refroidissement ; dans une démonstration de concept d'un centre de données exploité par Intel, le refroidissement ambiant avec un échange d'air jusqu'à 32° a été utilisé. Le résultat obtenu est une économie de 67% de puissance avec un refroidissement ambiant 91% du temps [15]. En descendant la hiérarchie, des solutions de refroidissement plus intelligentes ont émergé au niveau du châssis, des enceintes et du serveur sous la forme de ventilateurs à vitesse variable modulés par des mesures de température [141].

La discussion ci-dessous que nous avons initiée concerne le domaine informatique d'une manière générale. Nous nous focalisons sur un de ses postes importants à savoir les systèmes de gestion de données (SGBD), ou les systèmes de stockage de donnée hébergés par les centres de données. Ceci représente le contexte de cette thèse. Les SGBD sont les plus gros consommateurs d'énergie électrique.

3. http://www.thegreengrid.org/
4. http://www.etsi.org/news-events/news/798-2014-06-press-etsi-releases-the-first-global-kpi-on-energy-efficiency-in-ict
5. http://www.uefi.org/sites/default/files/resources/ACPI_6_1.pdf

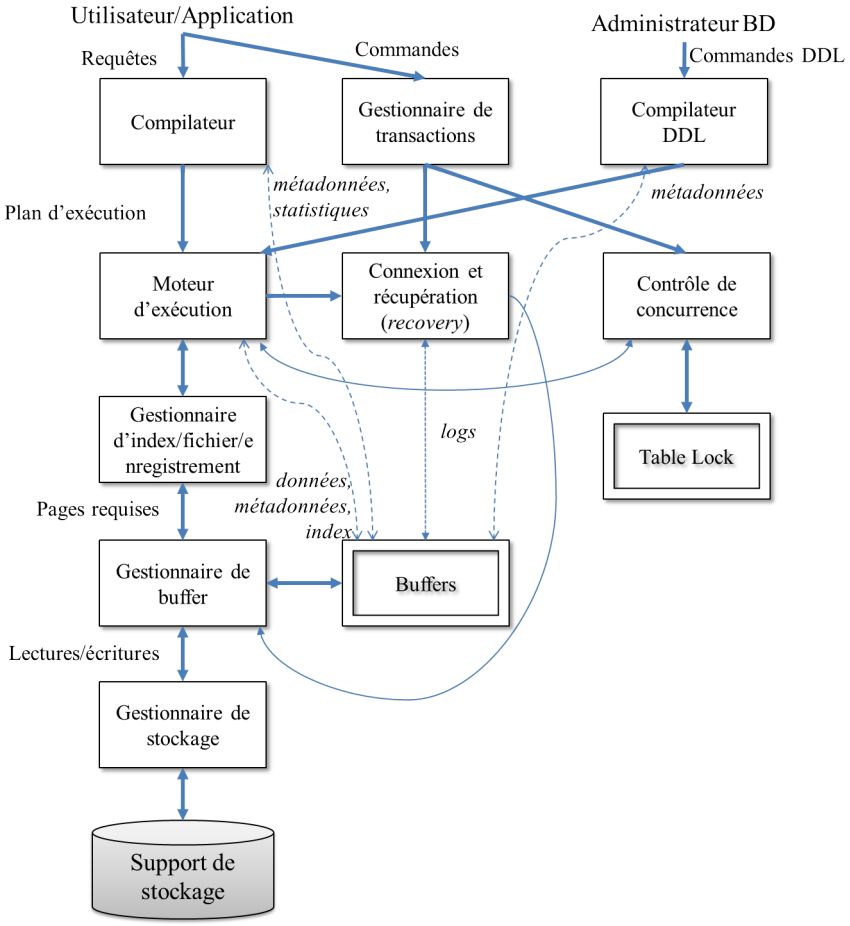


FIGURE 1.2 – Les composantes principales d'un SGBD

Cela est dû au fait qu'ils effectuent des opérations (jointure, union, agrégation, etc.), impliquant des tables ou des structures de données volumineuses qui nécessitent toutes les ressources d'un centre de données (CPU, mémoire, et réseau).

Devant cette situation critique, deux articles de vision ont été publiés par des chercheurs de la *communauté de bases de données* provenant à la fois du milieu académique (Stavros Harizopoulos et al. [116] de MIT) et industriel (Goetz Graefe [104] de HP). Ils ont mis l'accent sur l'intégration de l'énergie dans la conception et l'exploitation des bases de données. Les recommandations de ces auteurs coïncident avec ce que la communauté scientifique a identifié, à savoir l'utilisation des matériels et logiciels économiques [202]. Les SGBD peuvent bénéficier des matériels économiques existants (comme les supports de stockage amovible [228], les cartes graphiques [126], etc.).

Rappelons que la consommation énergétique d'un SGBD dépend de la base de données qu'il héberge. Plus précisément, elle dépend de son schéma (le nombre de tables et les attributs), sa population (en termes d'instances) et son exploitation via des requêtes. La plus grosse consommation d'un SGBD est due au calcul effectué par ses composantes principales : l'optimiseur de requêtes, le gestionnaire de buffer, le contrôleur de concurrence, le gestionnaire des méthodes d'accès, le gestionnaire de stockage, etc. comme

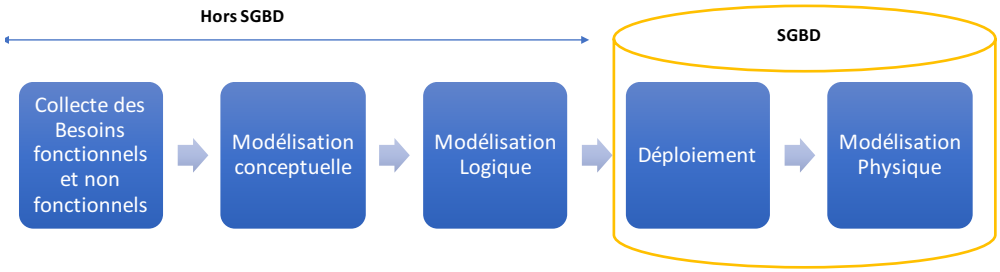


FIGURE 1.3 – L'énergie dans le cycle de vie de conception de base de données

le montre la Figure 1.2. Les travaux existants sur l'incorporation de la dimension « énergie » se focalise sur la partie traitement de requêtes. Ils supposent que la base de données est déjà conçue et déployée sur un SGBD. L'optimisation de requêtes dans les bases de données est une tâche difficile, et souvent modélisée comme un problème d'optimisation à base de contraintes. Ces dernières représentent plusieurs besoins non-fonctionnels comme le temps de réponse, le coût de stockage, le coût de maintenance. Les approches proposées pour résoudre cette problématique font souvent appel au développement de modèles de coût mathématiques estimant le ou les besoins non-fonctionnels utilisés. Les premiers travaux sur l'incorporation de l'énergie dans le monde des bases de données (côté logiciel) étaient le développement de modèles de coût mathématiques estimant la consommation énergétique lors de l'exécution de requêtes. Ces modèles prennent en compte des paramètres couvrant plusieurs entités logicielles et matérielles de la base de données cible tels que son schéma (par ex. la taille des tables, la longueur de tuples, etc.), la charge de requêtes (par ex. le type de requêtes, les facteurs de sélectivité des jointures et des prédicats de sélections, tailles des résultats intermédiaires, etc.), le matériel (par ex. la taille de tampon, la taille de page du disque, etc.) et la plate-forme de déploiement (par ex. RAM, la taille du disque, le nombre de nœuds, etc.).

D'autres efforts ont été effectués dans le développement des bancs d'essai permettant aux chercheurs de tester l'efficacité de leurs modèles économiques dédiés à la gestion de la base. Nous pouvons citer l'exemple du banc d'essai TPC-Energy [1] lancé en 2007.

L'ensemble de ces travaux suppose que la base de données est déjà déployée et s'intéresse particulièrement à la partie optimisation de requêtes effectuée par le SGBD hébergeant cette base. Or, d'autres phases de cycle de vie de la base de données peuvent être sensibles à l'énergie comme la phase physique – souvent associée à l'image de la qualité et de performance de toute base de données – dans la quelle des structures d'optimisation comme les index, les vues matérialisées, le partitionnement doivent être sélectionnées. Ces structures sont souvent utilisées par les optimiseurs de requêtes pour finir des plans efficaces (Figure 1.3).

Dans cette thèse, nous tenterons à répondre à ces limites en proposant des modèles mathématiques plus robustes pour estimer l'énergie et proposer une démarche de conception physique dirigée par la consommation énergétique.

Dans les sections suivantes, nous présentons les objectifs de notre thèse et l'ensemble de contributions réalisées.

1.2 Objectifs et contributions

Notre premier objectif est de sensibiliser les usagers et les concepteurs de bases de données de la dimension énergétique, en suivant l'expérience du domaine du bâtiment, en proposant des solutions pour des bases de données déjà existantes. Cela doit passer par la revisite des travaux sur l'optimisation de requêtes, considérée comme un poste gourmand en énergie. L'intégration de l'énergie dans les optimiseurs de requêtes peut se faire en suivant plusieurs scénario : (i) les optimiseurs de requêtes dirigées par seulement l'énergie, (ii) les optimiseurs de requêtes dirigées par plusieurs besoins non-fonctionnels y compris l'énergie. Dans la continuité de la reproduction des efforts du secteur du bâtiment, nous avons déployé nos solutions énergétiques sur un SGBD open source (PostgreSQL) afin de valider nos résultats et surtout pour sensibiliser et motiver les usagers et les concepteurs pour intégrer cette dimension dans leurs solutions. Le second objectif est de pousser plus loin notre réflexion sur la conception verte des bases de données, et l'appliquer sur une autre phase de cycle de vie qui est la phase physique.

Afin de répondre à nos objectifs, nous nous sommes fixées un ensemble d'actions qui sont :

1. *Un audit sur les composants d'un SGBD.* Pour économiser l'énergie, il faut d'abord identifier les postes sensibles. Cela est établi par un audit de l'ensemble de composants d'un SGBD (Figure 1.2). Dans un serveur informatique, il y a généralement cinq consommateurs d'énergie, à savoir, le processeur, les disques, la mémoire, les dispositifs d'entrées/sorties (E/S) et la carte mère. Il faut étudier le comportement du système lors de l'exécution d'une requête par le SGBD et son impact sur l'énergie. Atteindre une efficacité énergétique nécessite des améliorations dans le profil de la consommation d'énergie de chaque composant du système.
2. *L'adoption dynamique de composants matériels.* L'adoption de composants matériels dynamiquement pourrait aider à améliorer l'efficacité énergétique. Par exemple, l'exécution du processeur à une fréquence inférieure permet de réduire la tension, résultant d'une économie d'énergie. Cette technique, qui met en œuvre un compromis entre la performance et l'énergie, est connue comme l'ajustement dynamique de la tension et de la fréquence (DVFS), la tension et la fréquence sont ajustées à la volée. Une technique similaire existe pour les disques et mémoires vivres [241]. En outre, en employant des lecteurs de disques Solid-State Drive (SSD), des cartes graphiques puissantes pour le traitement, une grande configuration de mémoire, processeurs et mémoires de faible consommation ont pourrait diminuer la consommation d'énergie totale du système. Intégrer ces techniques directement dans l'SGBD afin de contrôler son bon fonctionnement est nécessaire, car le SGBD a la connaissance sur la donnée.
3. *Le développement des modèles de coût mathématiques.* Toute prise de décision d'un SGBD est souvent orchestrée par un modèle de coût mathématique estimant un besoin non-fonctionnel.

Dans le contexte de l'energie, l'optimiseur de requêtes doit être associé à des modèles de coût robustes qui doivent prendre en compte tous des paramètres pertinents des composantes sensibles à l'énergie électrique.

4. *Vers une sélection des structures d'optimisation vertes.* Les structures d'optimisation (les vues matérialisées, les index, etc.) qui ont émergé avec l'arrivée des entrepôts de données sont souvent sélectionnées avec des algorithmes dirigés par des modèles de coût mathématiques, estimant la performance de requêtes ou le coût de maintenance. Cette sélection doit être revisitée pour intégrer le coût énergétique.

Dans cette thèse nous considérons les actions (1), (3) et (4). L'action (2) n'est pas considéré dans notre étude pour plusieurs raisons :

- Les techniques basées sur le matériel requièrent de nouveaux matériels, car d'une part les nouvelles technologies comme le DVFS ne sont pas disponibles sur tout les processeurs et d'autre part les nouveaux disques SSD sont très cher et ont un problème de durée de vie.
- Il existe diverses gammes pour chaque composant matériel, donc développer des méthodes sur une seule gamme ou marque limite la portabilité de ces méthodes.
- Les techniques logicielles sont préférées dans le contexte des bases de données car le SGBD a une connaissance détaillée sur les charges de requêtes, les plans d'exécution, les informations sur les données, etc.

Pour répondre aux limitations précédentes, nos principales contributions, décrites dans cette thèse, sont les suivantes :

La proposition d'un modèle de coût énergétique pour des requêtes isolées. Pour exécuter une requête SQL, l'optimiseur doit sélectionner son meilleur plan d'exécution qui peut être représenté par un arbre d'opérations algébriques correspondant à cette requête. Une fois identifié, l'optimiseur exécute ce plan étape par étape [6]. Le mode d'exécution des requêtes influence la puissance électrique consommée et l'énergie totale. Cette observation est complètement ignorée par les travaux de l'état de l'art lors de la définition des modèles de coût. Ce dernier doit prendre en considération la relation entre les paramètres du modèle (linéaire, logarithmique, exponentielle. etc.) pour avoir des résultats fiables. Les travaux existants supposent que cette relation est linéaire. Dans cette thèse, nous avons montré que la relation est non-linéaire.

Modèle de coût énergétique pour des requêtes concurrentes. Dans un environnement réel des centres de données, le SGBD exécute un ensemble de requêtes simultanément de façon concurrente. Cette hypothèse n'est pas prise en compte par les travaux existants lors de la définition des modèles de coût. En conséquence, nous avons étendu notre modèle de coût pour prendre en charge ce scénario. Cela est utile pour plusieurs tâches telles que le contrôle d'admission, l'ordonnancement des requêtes et le contrôle d'exécution des requêtes avec l'objectif d'améliorer l'efficacité énergétique.

Traitement de requêtes éco-énergétique. Avec la présence de modèle de coût, nous proposons une méthodologie qui intègre l'énergie dans le processus de traitement des requêtes pour produire des plans d'exécutions réduisant leur consommation. Cette démarche a été implémentée sous le SGBD PostgreSQL.

6. Ce mode d'exécution est appelé le mode itérateur.

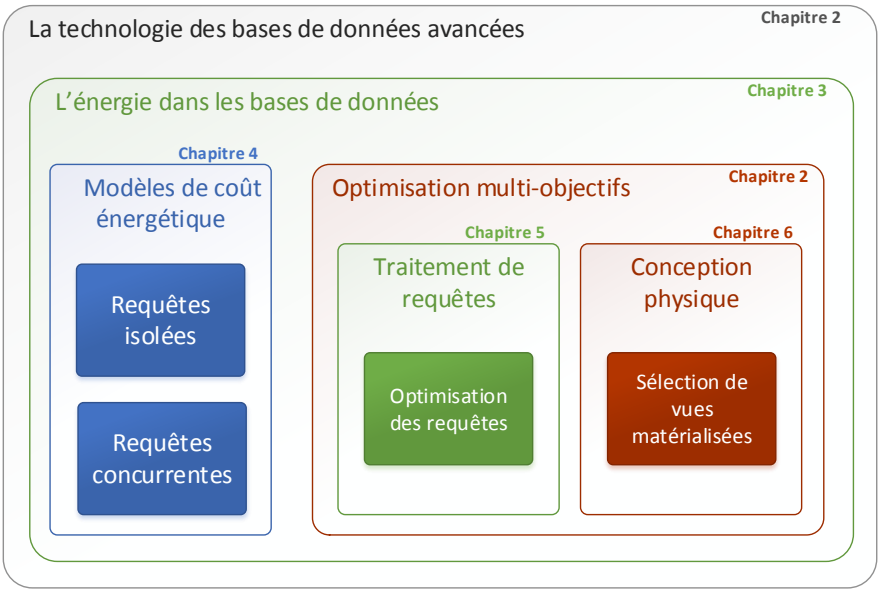


FIGURE 1.4 – La répartition des chapitres de la thèse.

Conception physique éco-énergétique. Nous avons proposé une approche de sélection de structures d'optimisation en prenant en compte plusieurs besoins non-fonctionnels : la performance de requêtes, l'espace de stockage et la consommation énergétique. Nous avons choisi le cas des vues matérialisées ; considérées comme des structures redondantes du fait qu'elles exigent une espace de stockage et un coût de maintenance. Nous avons alors formalisé le problème de sélection des vues matérialisées comme un problème multi-objectif et nous avons proposé des algorithmes avancés pour sa résolution.

1.3 Organisation de la thèse

D'un point de vue organisationnel, le reste de la thèse s'articule autour de six chapitres.

Dans le Chapitre 2, nous présentons un état de l'art sur le cycle de vie de conception des bases de données avancées, avec une focalisation particulière sur la phase physique. Le chapitre décrit aussi le module de traitement de requêtes dans un SGBD relationnel et propose la conception d'un optimiseur de requêtes éco-énergétique. Le chapitre cite également les techniques d'optimisations multi-objectifs et les solutions proposées dans le contexte des bases de données pour sa résolution.

Dans le Chapitre 3, nous présentons un état de l'art sur les différentes techniques d'amélioration de l'efficacité énergétique dans les systèmes informatiques en général, et dans les bases de données en particulier, les différents concepts requis pour ce travail ainsi que les notions de bases sur l'énergie et les modèles de coût proposés sont présentés.

Dans le Chapitre 4, nous étudions le comportement de l'énergie des bases de données sur le matériel informatique. Il s'agit d'identifier les paramètres et ses caractéristiques liées à la consommation énergétique des requêtes. A la base de cette étude empirique, nous proposons un modèle de coût pour

estimer la consommation énergétique d'une requête SQL isolée. Ensuite, le cas des requêtes exécutées simultanément de façon concurrente est étudié. Nous étendons le modèle coût pour une seule requête vers une charge de requêtes, et nous adaptons les paramètres et les algorithmes pour avoir une meilleure estimation. Nous validons notre modèle sur une variété de configurations logicielles et matérielles.

Dans le Chapitre 5, nous étudions le cas de traitement des requêtes dans un SGBD relationnel, afin de savoir l'impact de chaque phase de ce traitement sur la consommation d'énergie du système. Suivant cette étude, nous proposons une méthodologie qui intègre l'énergie dans le processus de traitement des requêtes pour produire des plans d'exécutions réduisant la dissipation d'énergie.

Le Chapitre 6 aborde le problème d'intégration de l'énergie dans la phase de conception physique des bases de données. L'analyse est effectuée en prenant le cas des vues matérialisées. Une formalisation multi-objectif est proposée, et un algorithme évolutionnaire est utilisé pour résoudre le problème de sélection des vues matérialisées.

En fin, nous concluons la thèse par un bilan général sur nos contributions, et l'évocation des perspectives qui s'offrent à l'issue de notre travail dans le Chapitre 7.

Ce mémoire comprend également trois annexes. L'Annexe A décrit le processus d'estimation des coûts dans la phase d'optimisation des requêtes dans un SGBD. Elle donne les fonctions d'estimation de cardinalité des résultats intermédiaires, et détaille les formules d'estimation du coût d'entrées/sorties des opérateurs physiques. L'Annexe B étudie les bornes de confiance dans les données et les résultats de modèle de coût énergétique afin de confirmer la qualité de la modélisation. L'Annexe C liste les requêtes utilisées pour construire le modèle de coût énergétique lors de la phase d'apprentissage.

Les principales contributions de notre thèse sont réparties dans les Chapitres Chapitre 4, Chapitre 5 et Chapitre 6. La Figure 1.4 schématise les grands axes de ce travail et leur répartition dans chaque chapitre. Chacun des ces chapitres a donné lieu à un ensemble de communications et de publications dans des conférences internationales :

Publications internationales :

[223] **Amine ROUKH**, Ladjel BELLATRECHE, Selma BOUARAR, Ahcène BOUKORCA : Eco-Physic : Eco-Physical Design Initiative for Very Large Databases. Information Systems. Elsevier, 2017. [DOI].

Communications internationales :

[220] **Amine ROUKH**, Ladjel BELLATRECHE : Eco-Processing of OLAP Complex Queries. In proceedings of 17th International Conference on Big Data Analytics and Knowledge Discovery (DAWAK). LNCS, Springer, Valencia, Spain, September 1-4, 2015. p. 229-242. [DOI].

[219] **Amine ROUKH** : Estimating Power Consumption of Batch Query Workloads. In proceedings of 5th International Conference Model and Data Engineering. Springer International Publishing (MEDI), Rhodes, Greece, September 26-28, 2015. p. 198-212. [DOI].

[222] **Amine ROUKH**, Ladjel BELLATRECHE, Ahcène BOUKORCA, Selma BOUARAR : Eco-DMW : Eco-Design Methodology for Data Warehouses. In proceedings of 15th International Workshop on Data Warehousing and OLAP (DOLAP). ACM, Melbourne, Australia, October 23 - 23, 2015. p. 1-10. [DOI].

[221] **Amine ROUKH**, Ladjel BELLATRECHE, Carlos ORDONEZ : EnerQuery : Energy-Aware Query Processing. In proceedings of 16th International on Conference on Information and Knowledge Management (CIKM). ACM, Indianapolis, Indiana, USA — October 24 - 28, 2016. p. 2465-2468. [DOI].

[224] **Amine ROUKH**, Ladjel BELLATRECHE, Nikos TZIRITAS, Carlos ORDONEZ : Energy-Aware Query Processing on Parallel Database Cluster Nodes. In proceedings of 16th International Conference on Algorithms and Architectures for Parallel Processing (ICA3PP). Springer International Publishing, Granada, Spain, December 14-16, 2016. p. 260-269. [DOI].

[194] Abdelkader OUARED, Yassine OUHAMMOU, **Amine ROUKH** : A Meta-advisor Repository for Database Physical Design. In proceedings of 6th International Conference on Model and Data Engineering (MEDI) Springer International Publishing, Almería, Spain, September 21-23, 2016. p. 72-87. [DOI].

[37] Selma BOUARAR, Ladjel BELLATRECHE, **Amine ROUKH** : Eco-Data Warehouse Design Through Logical Variability. In proceedings of 43rd International Conference on Current Trends in Theory and Practice of Informatics (SOFSEM). Springer International Publishing, Limerick, Ireland, January 16-20, 2017. p. 436-449. [DOI].

Deuxième partie

État de l’art

Chapitre

2

La technologie des bases de données

« The beginning of knowledge is the discovery of something we do not understand. »

— Frank Herbert

Sommaire

2.1 Introduction

Aujourd'hui, la disponibilité des systèmes de gestion de base de données fiables permet aux organisations de toutes tailles de gérer leurs données efficacement, de les stocker, et de déployer des applications exploitant ces données. Ces trois opérations fondamentales sont précieuses pour assurer la sécurité, l'intégrité des données et l'uniformisation des procédures administratives. Les bases de données sont actuellement au cœur des systèmes d'information des entreprises, plus particulièrement dans les tâches de gestion. Au cours des quarante dernières années, des concepts, des méthodes, et des algorithmes ont été développés pour gérer les données sur des mémoires secondaires, ils constituent aujourd'hui l'essentiel de la discipline « Bases de Données » (BD). Cette discipline est utilisée dans de nombreuses applications. De plus, Il existe un grand nombre de Systèmes de Gestion de Bases de Données (SGBD), qui permettent de gérer efficacement des bases de données volumineuses. Dans ce contexte, une théorie fondamentale sur les techniques de modélisation des données, et les algorithmes de traitement a vu le jour. Les bases de données constituent donc une discipline s'appuyant sur une théorie solide et offrant de nombreuses solutions pratiques.

Dans ce chapitre, nous présentons un état de l'art portant sur la technologie des bases de données, son historique, son cycle de vie de conception et d'exploitation. Nous nous focalisons particulièrement sur la phase de conception physique et le traitement des requêtes. Nous allons détailler les différentes techniques utilisées et travaux proposés pour chacune d'entre elles. En second lieu, nous abordons les problèmes d'optimisation multi-objectifs en citant les méthodes de résolution proposées dans le cas général et le cas des bases de données.

2.2 Technologie des bases de données

Une base de données est un ensemble structuré de données enregistrées sur des supports accessibles par ordinateur, représentant des informations du monde réel et pouvant être interrogées et mises à jour par une communauté d'utilisateurs.

La gestion et l'accès à une base de données sont assurés par SGBD. Un Système de Gestion de Bases de Données est un logiciel de haut niveau permettant aux utilisateurs de structurer, d'insérer, de modifier, de rechercher de manière efficace des données spécifiques, au sein d'une grande quantité d'informations, stockées sur mémoires secondaires partagée de manière transparente par plusieurs utilisateurs.

Suite à la progression de la technologie dans les domaines des processeurs, de la mémoire, du stockage et des réseaux informatiques, les tailles, les capacités et les performances des bases de données et SGBD ont augmenté en ordre de grandeur. Le développement de la technologie de base de données peut être divisé en trois périodes basées sur le modèle ou la structure de données : **navigationnelle**, **relationnelle** et **post-relationnelle** [246].

Lorsque les ordinateurs augmentaient en vitesse et en capacité, un certain nombre de systèmes de base de données à usage général ont vu le jour. Au milieu des années 1960, une partie de ces systèmes étaient utilisés commercialement. En 1971, le groupe de travail sur les bases de données au sein de la société CODASYL a livré son standard, qui est généralement connu sous le nom d'approche *CODASYL* ou modèle réseau [17]. Le produit commercial IDMS développé par Charles Bachman est basé sur cette

approche [274]. L'approche CODASYL s'appuie sur des modèles de données organisés autour de types d'articles constituant les nœuds d'un graphe, reliés par des types de pointeurs composant les arcs du graphe. IBM avait également son propre SGBD en 1966, connu sous le nom de système de gestion de l'information (IMS) [129]. IMS était un développement de logiciels écrits pour le programme Apollo sur le système/360. IMS était généralement semblable en concept à CODASYL, mais a utilisé un modèle *hiérarchie* stricte pour son modèle de navigation de données au lieu du modèle de réseau de CODASYL [17].

Le modèle *relationnel*, proposé pour la première fois en 1970 par Edgar F. Codd [71], s'est écarté de cette tradition en insistant sur le fait que les applications devraient chercher les données par contenu plutôt que par des liens. Le modèle relationnel utilise des ensembles de tables de style grand livre, utilisés chacun pour un type différent d'entité. Il s'agit d'un modèle mathématique défini en termes de logique de prédicat et de théorie des ensembles. IBM System R [49] était un projet novateur, il s'agissait de la première implémentation de SQL, qui est devenu depuis le langage de requête de données relationnelles standard. Il a également été le premier système à démontrer qu'un système de gestion de base de données relationnelle pourrait fournir de bonnes performances de traitement des transactions. Ceci a poussé IBM à développer une véritable version de production de System R, connue sous le nom de SQL DS, et, plus tard, Database 2 (DB2). Un autre modèle de données, le modèle *entité-association* proposé par Peter Chen est apparu en 1976 [62]. Ce dernier a gagné en popularité pour la conception des bases de données grâce à sa description des données plus familière que celle du modèle relationnel.

C'est seulement au milieu des années 1980 que le matériel informatique est devenu suffisamment puissant pour permettre un large déploiement de systèmes relationnels. Cependant, depuis le début des années 90, les systèmes relationnels demeurent dominant dans toutes les applications de traitement de données à grande échelle, tel que : IBM DB2, Oracle, MySQL, PostgreSQL et Microsoft SQL Server. Ils supportent en général une architecture répartie, avec des stations clients transmettant leurs requêtes à des serveurs puissants qui gèrent les bases de données. Le langage de base de données dominant le modèle relationnel est le SQL. Ce dernier a largement influencé les langages des autres modèles de données.

Les années 1990, parallèlement à une augmentation d'utilisation de la programmation *orientée objet*, ont connu une croissance dans la façon dont les données dans diverses bases de données ont été traitées. Les programmeurs et les concepteurs ont commencé à traiter les données dans leurs bases de données comme des objets. C'est-à-dire que si les données d'une personne figuraient dans une base de données, les attributs de cette personne, comme son adresse, son numéro de téléphone et son âge, sont désormais considérés comme appartenant à cette personne au lieu d'être des données étrangères. Cela permet aux relations entre les données d'inclure des objets et leurs attributs et non des champs individuels [173]. Des systèmes commerciaux sont apparus, tel que Illustra, Objectivity/DB, Omniscience et UniSQL.

Les produits offrant un modèle de données plus général que le modèle relationnel sont classés comme post-relationnels, d'autres termes alternatifs incluent « base de données hybride », « base de données améliorée ». Le modèle de données de ces produits incorpore des relations mais n'est pas limité par la contrainte de E.F. Codd, qui exige que : « *toutes les informations dans la base de données doivent être exprimées explicitement en termes de valeurs dans les relations et d'aucune autre manière* » [78]. Un exemple est les bases de données *XML*, qui sont un type de base de données structurée orientée document permettant d'interroger les attributs de document XML [54]. Les bases de données XML sont

principalement utilisées dans la gestion des bases de données d'entreprises, où XML est utilisé comme norme d'interopérabilité de données machine à machine. Les systèmes de gestion de base de données XML incluent les logiciels commerciaux MarkLogic et Oracle Berkeley DB XML, ainsi que les logiciels libres comme Clusterpoint Distributed XML/JSON et BaseX.

Certaines des extensions au modèle relationnel intègrent des concepts issus de technologies antérieures au modèle relationnel. Par exemple, ils permettent la représentation d'un graphe dirigé avec des arbres sur les nœuds, comme GraphDB et Neo4j. Certains produits post-relationnels étendent les systèmes relationnels en systèmes *dimensionnels* utilisés pour représenter les données dans les entrepôts de données de façon à pouvoir facilement résumer les données en utilisant le traitement analytique en ligne ou les requêtes OLAP. Dans le modèle dimensionnel, un schéma de base de données se compose d'une seule grande table de faits qui est décrite en utilisant des tables de dimensions et des mesures. Des exemples de produits dans cette tendance est IBM InfoSphere Warehouse, Sybase IQ et Teradata Enterprise Data Warehouse.

Une autre tendance, dès les années 2000, dans la génération de post-relationnels est les bases de données *NoSQL*. Les BDs NoSQL sont souvent très rapides et ne nécessitent pas de schémas de tables fixes. Ils évitent également les opérations de jointure en stockant des données dénormalisées et ils sont conçues pour être redimensionnées horizontalement [249]. Déclenché par les besoins des entreprises comme Facebook, Google et Amazon devant le Web 2.0. La raison principale de l'émergence et de l'adoption des bases de données NoSQL est le développement des clusters de serveurs et la nécessité de posséder un paradigme de bases de données adapté à ce modèle d'infrastructure matérielle. Les bases de données NoSQL sont de plus en plus utilisées dans des applications « Big Data » et web temps réel. Les structures de données utilisées par les bases de données NoSQL incluent le modèle clé-valeur, la colonne, le graphe et le document. Les systèmes NoSQL les plus populaires incluent MongoDB, Couchbase, Riak, Memcached, Redis, CouchDB, Hazelcast, Apache Cassandra et HBase [7], qui sont tous des logiciels libres.

NewSQL est une classe de bases de données relationnelles modernes qui vise à fournir la même performance évolutive des systèmes NoSQL pour les charges de traitement transactionnel en ligne (lecture-écriture) tout en utilisant SQL et en maintenant les garanties les propriétés ACID (atomicité, cohérence, isolation et durabilité) d'un système de base de données traditionnel [248]. Ces bases de données incluent ScaleBase, Clustrix, EnterpriseDB, MemSQL, NuoDB et VoltDB. Une illustration de l'évolution de la technique des bases de données est donnée dans la Figure 2.1.

Afin d'intégrer l'énergie dans les BDs, nous proposons de les étudier suivant deux dimensions : (1) le cycle de vie de la conception et (2) le traitement des requêtes dans un SGBD relationnel. Nous détaillons chacune de ces dimensions dans les prochaines sections.

2.2.1 Conception et cycle de vie

Une des tâches essentielles des développeurs de bases de données est la conception de leurs schémas. La conception de base de données est le processus de production d'un modèle de données détaillé de la base de données. Ce modèle de données contient tous les choix de conception logique et physique nécessaires et les paramètres de stockage physique nécessaires pour générer une conception dans un

7. http://db-engines.com/en/ranking

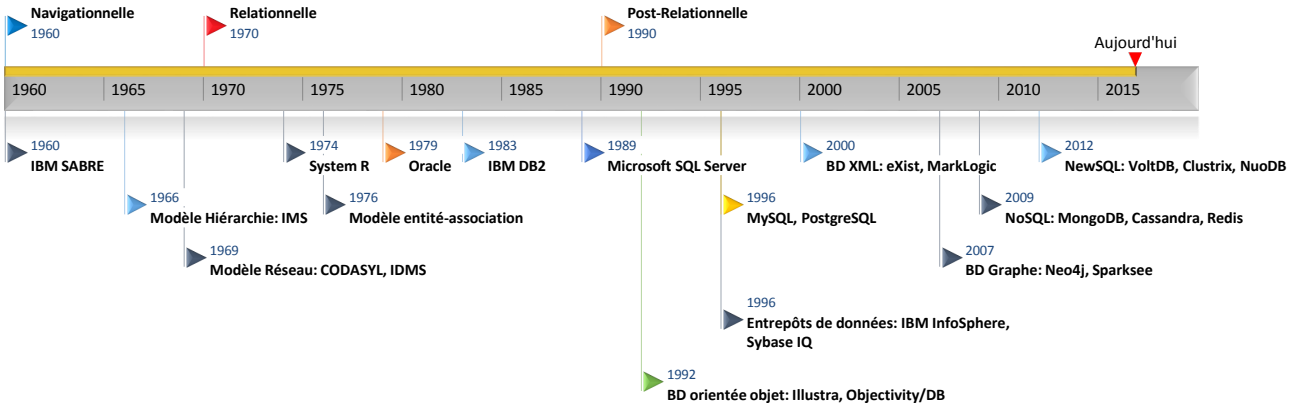


FIGURE 2.1 – Une brève évolution de la technologie des bases de données.

langage de définition de données, qui peut ensuite être utilisé pour créer une base de données. La représentation doit être juste pour éviter les erreurs sémantiques, notamment dans les réponses aux requêtes. Elle doit aussi être complète pour permettre le développement des programmes d'application souhaités. Elle doit enfin être évolutive afin de supporter la prise en compte rapide de nouvelles demandes [98].

Une base de données bien conçue facilite la gestion des données et génère des informations précises et précieuses. Une base de données mal conçue est susceptible de devenir un terrain fertile pour les erreurs difficiles à tracer qui peuvent conduire à une mauvaise prise de décision et à l'échec d'une organisation. Même un bon SGBD fonctionnera mal avec une base de données mal conçue. La conception de base de données est tout simplement trop importante pour être laissée à la chance. L'objectif principal est de créer des modèles conceptuels, logiques et physiques complets, normalisés, non redondants et entièrement intégrés.

La création et l'évolution des bases de données suivent un schéma itératif appelé cycle de vie, un processus continu de création, de maintenance, d'amélioration et de remplacement des BDs. Traditionnellement, la démarche de conception et de cycle de vie d'une base de données s'effectue par des abstractions successives et itératives, en descendant depuis les problèmes de l'utilisateur vers la base de données finale [76]. Nous proposons de distinguer cinq étapes :

1. **Analyse des besoins**. Consiste à définir et examiner les problèmes, les contraintes et les objectifs de l'entreprise. Une première évaluation des besoins et exigences, logiciels et matériels, relatives au flux d'information doit être effectuée lors de cette étape. La phase d'analyse du cycle de vie est, en fait, une vérification approfondie des besoins des utilisateurs. Le résultat est une spécification textuelle de ces besoins.
2. **Modélisation conceptuelle**. L'objectif de cette étape est de concevoir une base de données indépendante du logiciel et des détails physiques de la base de données. La sortie de ce processus est un modèle de données conceptuelles qui décrit les principales entités de données, les attributs, les relations et les contraintes d'un domaine de problème donné. Cette conception est descriptive et narrative en forme. C'est-à-dire qu'elle est généralement composée d'une représentation graphique ainsi que de descriptions textuelles des principaux éléments de données, relations et

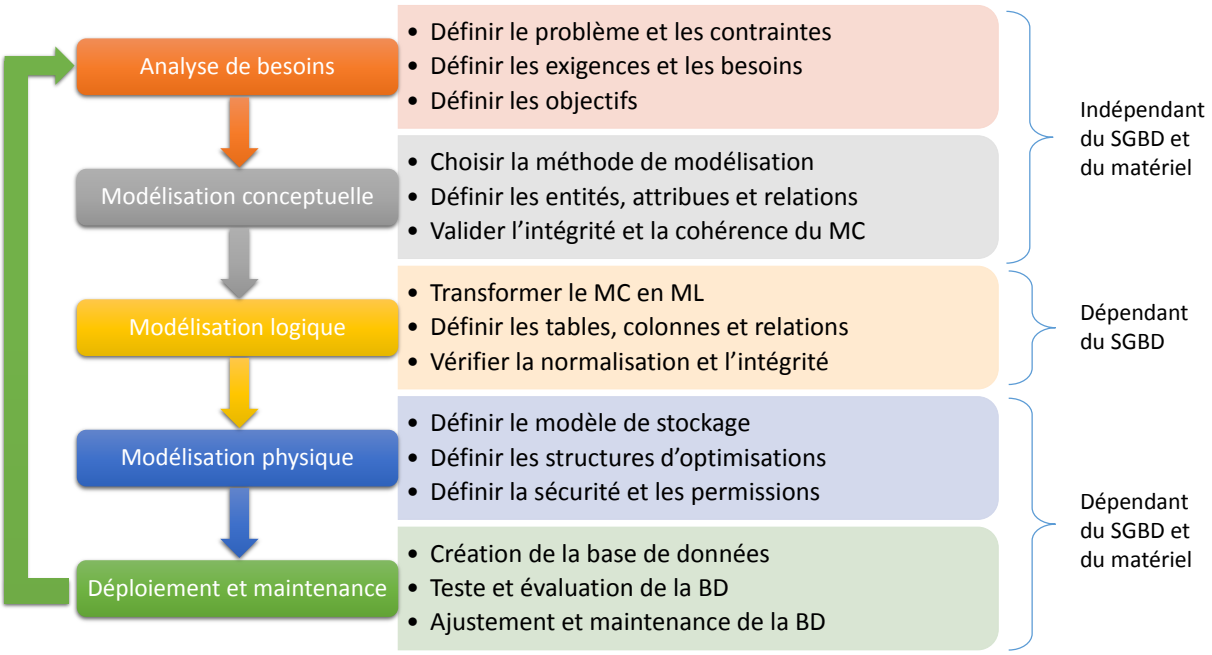


FIGURE 2.2 – Le cycle de vie des bases de données.

contraintes. L'objectif est d'intégrer toutes les parties dans un schéma conceptuel global complet, non redondant et cohérent. Ce schéma global est développé en utilisant des techniques telles que la modélisation entité-association ou Unified Modeling Language (UML).

3. **Modélisation logique**. L'objectif de la modélisation logique est de concevoir une base de données à l'échelle de l'entreprise basée sur un modèle de données spécifique, mais indépendante des détails de niveau physique. La conception logique nécessite que tous les objets du modèle conceptuel soient mappés aux structures spécifiques utilisées par le modèle de base de données sélectionné. Par exemple, la conception logique d'un SGBD relationnel comprend les spécifications pour les relations (tables). Avec un SGBD objet, il s'agit de générer des classes et des associations. La conception logique ne doit contenir que des tables correctement normalisées et nécessite également la définition des domaines d'attributs et des contraintes appropriées.
4. **Modélisation physique**. La modélisation ou conception physique est le processus de détermination de l'organisation du stockage de données et des caractéristiques d'accès aux données afin d'en assurer l'intégrité, la sécurité et les performances du système. Il s'agit de la dernière étape du processus de conception de base de données. Les caractéristiques de stockage sont déterminées en fonction des types de périphériques pris en charge par le matériel, du type de méthodes d'accès aux données prises en charge par le système et le SGBD. Dans cette étape, il faut choisir les bonnes structures d'optimisations physiques : vues matérialisées ou partitionnement de tables, index, etc. La conception physique pourrait devenir un travail très technique qui affecte non seulement l'accessibilité des données dans le(s) périphérique(s) de stockage, mais aussi la performance du système.
5. **Déploiement et maintenance**. Le résultat des phases de conception de base de données est une série d'instructions détaillant la création de tables, d'attributs, d'index, de contraintes de

sécurité, de règles de stockage, de performance, d'une architecture centralisée ou distribuée, etc. Dans cette phase, toutes ces spécifications de conception doivent être mises en œuvre. L'administrateur teste, évalue et ajuste la base de données pour s'assurer qu'elle fonctionne comme prévu. L'administrateur doit aussi effectuer des activités de maintenance dans la base de données. Certaines des activités de maintenance périodique comprennent la sauvegarde, la restauration, la modification des attributs et tables, génération de statistiques, audits de sécurité, etc.

Le schéma illustré par la Figure 2.2, représente ce cycle de vie avec les tâches principales de chaque étape. Dans les sections suivantes nous détaillons ces différentes étapes en mettant l'accent sur la phase de conception physique.

2.2.1.1 Analyse des besoins

Dans le domaine de l'ingénierie logicielle, l'ingénierie des besoins est la branche du génie logiciel qui se consacre aux objectifs, aux fonctions et aux contraintes réels des systèmes logiciels. Elle s'intéresse également à la relation entre ces facteurs et les spécifications précises du comportement des logiciels, ainsi qu'à leur évolution dans le temps et dans les familles de logiciels [293]. Autrement dit, les besoins et les exigences des utilisateurs sont traduits en un ensemble de besoins qui définissent ce que le système doit faire et comment il doit le faire. Le concepteur de base de données doit s'assurer que les besoins sont compréhensibles, sans ambiguïté, complets et concis. Les systèmes matériels et logiciels existants sont également étudiés pendant la phase d'analyse. Le résultat de l'analyse devrait être une meilleure compréhension des domaines fonctionnels du système et des problèmes actuels ou potentiels. Les utilisateurs finaux et le concepteur de systèmes doivent travailler ensemble pour identifier les processus et pour découvrir les zones à problèmes potentielles. Cette coopération est essentielle pour définir les objectifs de performance appropriés permettant de juger le nouveau système [76]. Une bonne analyse des besoins est fondamentale pour une conception réussie de la base de données. Les besoins se répartissent généralement en deux types : *besoins fonctionnels* ($\mathcal{BF}$) et *besoins non-fonctionnels* ($\mathcal{BNF}$) [293, 70, 12], tel que représenté dans la Figure 2.3.

2.2.1.1.1 Besoin fonctionnel

Le $\mathcal{BF}$ est défini comme étant tout besoin qui spécifie *ce que* le système doit faire. En d'autres termes, un $\mathcal{BF}$ décrira un comportement particulier du fonctionnement du système lorsque certaines conditions sont valides, par exemple : « envoyer un courrier électronique lorsqu'un nouveau client s'inscrit ». Des exemples des BFs dans une base de données incluent : gestion de transactions, authentification, niveaux d'autorisation, exigences de certification, données historiques, exigences légales ou réglementaires, etc.

2.2.1.1.2 Besoin non-fonctionnel

Le $\mathcal{BNF}$ est défini comme étant tout besoin qui spécifie *comment* le système effectue une certaine fonction. En d'autres termes, un $\mathcal{BNF}$ décrira comment un système doit se comporter et quelles limites il y a sur sa fonctionnalité. Les BNFs spécifient généralement les attributs ou caractéristiques de qualité

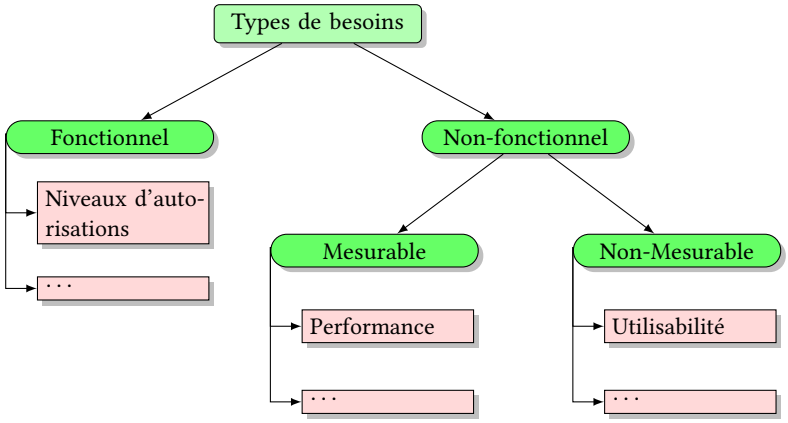


FIGURE 2.3 – Classification des types de besoins.

du système, par exemple : « les données modifiées dans une base de données doivent être mises à jour pour tous les utilisateurs qui y accèdent en moins de 2 secondes ». Par exemple, citons : performance, temps de réponse, consommation d'énergie, évolutivité, disponibilité, fiabilité, la sécurité, utilisabilité, etc. Les BNFs sont généralement plus difficiles à analyser. En particulier, les BNFs ont tendance à être des propriétés d'un système dans son ensemble et ne peuvent donc pas être vérifiés pour des composants individuels [191].

En outre, les BNFs peuvent être classés en deux catégories : $\mathcal{BNF}$ mesurable (BNFM) et $\mathcal{BNF}$ non-mesurable (BNFNM) [238].

2.2.1.1.2 BNFM

Les BNFM sont des besoins qui peuvent être mesurés sur une échelle métrique définie par l'utilisateur. La métrique est définie par l'utilisateur car pour certains besoins, tel que la performance, il existe plusieurs métriques pour la quantifier. Les mesures peuvent être réalisées automatiquement par un ordinateur comme le temps de réponse, ou manuellement par un dispositif de mesure comme la consommation d'énergie.

2.2.1.1.2 BNFNM

Cette classe contient des besoins qui ne peuvent être décrits que qualitativement à l'aide d'une échelle ordinaire ou nominale, c'est-à-dire qu'il n'existe aucune métrique à partir de laquelle nous pouvons extraire des mesures quantifiables. Par exemple, considérer l'utilisabilité. Il faut réaliser de grandes études aux niveau des utilisateurs du système pour quantifier l'influence de différentes caractéristiques sur l'utilisabilité du système. L'évaluation d'une telle étude se fait d'habitude manuellement.

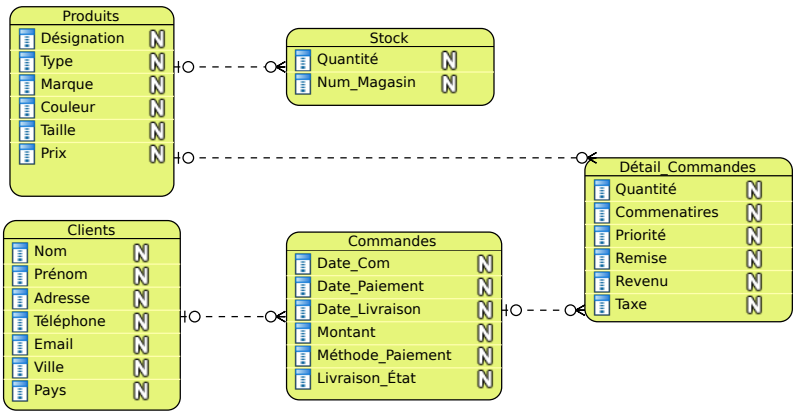


FIGURE 2.4 – Exemple d'un schéma de modélisation conceptuelle.

2.2.1.2 Modélisation conceptuelle

La modélisation conceptuelle consiste à créer un schéma conceptuel pour la base de données, à partir des besoins déjà collectés et analysés, en utilisant un modèle de données conceptuelles de haut niveau, tel que le modèle entité-association ou digramme de classe UML. Ce schéma représente les objets du monde réel de la manière la plus réaliste possible. Le modèle conceptuel doit comprendre une compréhension claire de l'entreprise et de ses $\mathcal{BF}$ et $\mathcal{BNF}$. À ce niveau d'abstraction, le type de matériel et/ou de SGBD à utiliser pourrait ne pas avoir été identifié. Par conséquent, la conception doit être indépendante du logiciel et du matériel afin que le système puisse être installé dans n'importe quelle plate-forme matérielle et logicielle choisie plus tard [76].

Exemple 1. *Une société spécialisée dans la vente souhaite dynamiser sa politique de vente en construisant un site web pour la vente de ses produits. Après l'étude des besoins de l'entreprise, le concepteur de BD a créé un schéma conceptuel avec le modèle entité-association. Voici la description de chaque entité :*

- *Clients : Cette table conserve les informations sur les clients.*
- *Commandes : Contient la liste des commandes des clients.*
- *Détail commandes : Contient les informations détaillées sur les commandes.*
- *Produits : Garde la liste des produits avec leur détaille.*
- *Stock : Sauvegarde la quantité de chaque produit en stock.*

Un exemple de schéma conceptuel défini en termes d'entités et d'associations entre ces entités est représenté dans la Figure 2.4.

2.2.1.3 Modélisation logique

La modélisation logique commence par le choix du SGBD suivant le modèle de données choisi dans l'étape précédente. Ensuite, le schéma conceptuel est transformé du modèle de données de haut niveau en modèle de données de mise en œuvre avec le SGBD [88]. La création schéma logique peut se faire d'une manière automatisée. Il existe des règles sur la façon dont le modèle entité-association ou le diagramme de classe UML est transféré à des relations du schéma logique. Dans cette étape, les clés

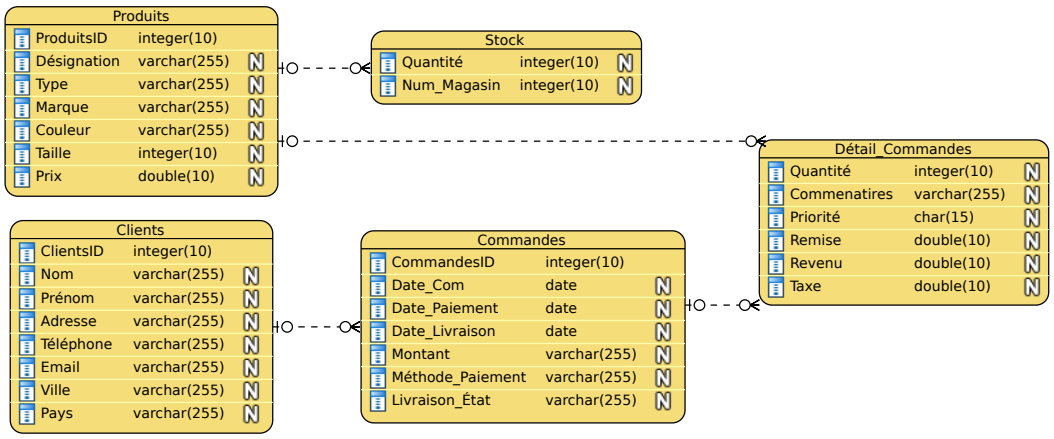


FIGURE 2.5 – Exemple d'un schéma de modélisation logique.

primaires et les clés étrangères sont définies. La normalisation est la dernière partie de la conception logique. L'objectif de la normalisation est d'éliminer la redondance et les anomalies potentielles lors de la mise à jour. La redondance signifie que les mêmes données sont enregistrées plus d'une fois dans la base de données. L'anomalie de mise à jour est une conséquence de la redondance. Si une donnée est enregistrée en plusieurs endroits, les mêmes données doivent être mises à jour en plusieurs endroits. La normalisation est une technique par laquelle on peut modifier le schéma relationnel pour réduire la redondance [88].

Exemple 2. *La Figure 2.5 représente le schéma logique de notre exemple de gestion de vente. En peut remarquer que la définition des types de données des attributs et les clés primaires ont été établies sur les tables « Stock », « Commandes » et « Clients ».*

2.2.1.4 Modélisation physique

L'étape suivante est la conception physique, au cours de laquelle les structures de stockage internes, les organisations de fichiers, les indexes, les chemins d'accès, les contraintes d'intégrité, les droits d'accès des utilisateurs et les paramètres de conception physique des fichiers de base de données sont spécifiés [88].

Dans cette étape, des structures d'optimisation ($\mathcal{SO}$) telles que les vues matérialisées, les indexes, le partitionnement de données, sont sélectionnées pour optimiser un ou plusieurs $\mathcal{BNF}$ tels que la performance des requêtes et l'énergie du système. La conception physique se concentre sur les méthodes de stockage et d'accès aux tables sur le disque qui permettent à la base de données de fonctionner avec une efficacité élevée. L'objectif est de maximiser les performances de la base de données sur l'ensemble du système. Les ressources physiques qui impliquent les performances lors de l'exécution des requêtes comprennent le CPU, les E/S (par exemple, les disques) et les connexions réseaux [169].

Exemple 3. *La Figure 2.6 illustre le schéma physique de l'exemple précédent. Les informations ajoutées sont les clés étrangères sur les tables « Stock », « Commandes » et « Détail_Commandes », et les indexes définis sur les tables « Clients » et « Produits » pour accélérer l'accès aux données de ces tables.*

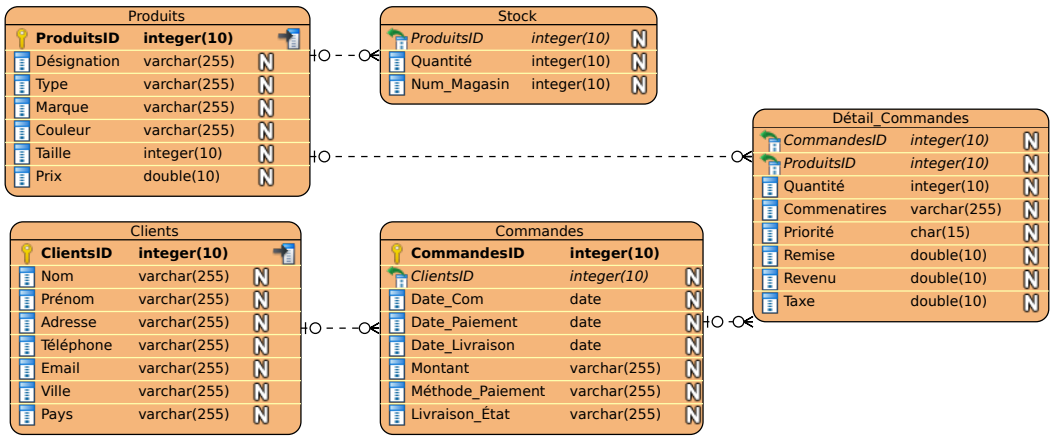


FIGURE 2.6 – Exemple d'un schéma de modélisation physique.

Les optimisations physiques ont été amplifiées par l'explosion du volume des données et la nécessité d'optimiser ses modèles d'accès, ce qui est assuré par les $\mathcal{SO}$ [59]. En exploitant les $\mathcal{SO}$, on peut réduire le temps de traitement des opérations dans certains cas de plusieurs ordres de grandeur. Une tâche critique pour l'administrateur de la base de données dans cette phase est le bon choix des $\mathcal{SO}$ pour satisfaire les $\mathcal{BNF}$ des utilisateurs. Cette tâche peut être divisée en quatre sous-tâches [154] :

1. Le choix des structures d'optimisation : redondante ou non-redondante ;
2. Le choix du mode de sélection des structures d'optimisation : isolée ou multiple ;
3. Le choix et le développement des algorithmes de résolution ;
4. La validation et le déploiement des solutions d'optimisation.

Dans ce suit, nous considérons le cas des vues matérialisées ($\mathcal{VM}$) qui est une $\mathcal{SO}$ redondante très utilisé dans les applications décisionnelles et les requêtes de type OLAP.

2.2.1.4.1 Cas d'étude : Vues matérialisées

Définition 1

Une vue est une relation logique qui utilise la définition de requête pour extraire les données à partir des tables correspondantes.

Définition 2

Une vue matérialisée est une table sur disque qui contient les résultats d'une requête. Elle peut s'agir d'une copie locale de données située à distance, ou un sous-ensemble des lignes et/ou colonnes d'un résultat de requête ou de jointure avec ou sans une fonction d'agrégation.

Les $\mathcal{VM}$ sont de petites tailles par rapport à la table réelle et sont constituées de données importantes. Par conséquent, une exécution plus rapide des requêtes est possible. Lorsque les requêtes sont générées

dans le système, les $\mathcal{VM}$ sont accédées en premier. Si les résultats ne sont pas trouvés dans les $\mathcal{VM}$, les tables réelles du système sont recherchées.

Exemple 4. *Pour créer une vue matérialisée qui comptabilise la somme des ventes de chaque client par jour de notre base de données de vente, la requête SQL s'écrit comme suit :*

```
CREATE MATERIALIZED VIEW resume_ventes AS
SELECT Nom, Date_Com, SUM(Montant)
FROM Clients Commandes
WHERE Clients.ClientsID = Commandes.ClientsID
GROUP BY Nom, Date_Com
ORDER BY Nom, Date_Com;
```

Si la relation Commandes contient des dizaines de millions de tuples, la différence entre le calcul de l'agrégation à partir des données de base et à l'aide de la vue matérialisée peut être de plusieurs ordres de grandeur.

Les défis à relever pour exploiter les vues matérialisées sont : (1) identifier les bonnes vues à matérialiser, (2) exploiter les $\mathcal{VM}$ pour répondre aux requêtes, et (3) mettre à jour efficacement les $\mathcal{VM}$ lors du changement des données de base [56].

2.2.1.4.1 Problème de sélection des $\mathcal{VM}$

L'administrateur de base de données ne peut pas matérialiser toutes les vues possibles, car il est limité par certaines ressources comme, l'espace disque et le temps de calcul, ceci est connu comme le problème de sélection des vues matérialisées ($\mathcal{PSV}$). Le $\mathcal{PSV}$ consiste à trouver un ensemble approprié de vues satisfaisant un ensemble donné de $\mathcal{BNF}$ tels que la performance des requêtes, la fiabilité, l'utilisabilité, etc. Les $\mathcal{VM}$ sélectionnées doivent répondre à un ensemble de contraintes telles que le coût de stockage, les coûts de maintenance, etc. Le $\mathcal{PSV}$ est connu pour être un problème NP-complet [110] en raison du fait que l'espace de solution s'accroît de façon exponentielle lorsque la taille du problème augmente. La formalisation générale de $\mathcal{PSV}$ est défini comme suit : Étant donné :

- un schéma d'une base de données : DB ;
- une charge de requêtes : $\mathcal{W} = \{Q_1, Q_2, \cdots, Q_n\}$;
- un ensemble de contraintes : $\mathcal{C} = \{C_1, C_2, \cdots, C_p\}$;
- un ensemble de besoins non-fonctionnelles : $\mathcal{BNF} = \{nfr_1, \cdots, nfr_k\}$.

Le $\mathcal{PSV}$ consiste à sélectionner un ensemble de vues matérialisées : $MV = \{V_1, V_2, \cdots, V_m\}$ satisfaisant des $\mathcal{BNF}$ et en respectant l'ensemble des contraintes $\mathcal{C}$.

Plusieurs instanciations de cette formalisation existent. Nous proposons de les représenter par un *cube* ayant trois dimensions principales : (i) le $\mathcal{BNF}$, (ii) les contraintes $\mathcal{C}$, et (iii) les algorithmes utilisés pour résoudre le $\mathcal{PSV}$. Ces dimensions permettent de classer les solutions existant dans l'état de l'art. La Figure 2.7 illustre notre cube.

Les $\mathcal{BNF}$. En ce qui concerne les $\mathcal{BNF}$ que les $\mathcal{VM}$ sélectionnés doivent satisfaire, elles couvrent plusieurs objectifs, tels que : la minimisation du temps de réponse des requêtes [218], la minimisation du temps de réponse et du coût de maintenance [109]. Ces objectifs peuvent être combinés pour donner

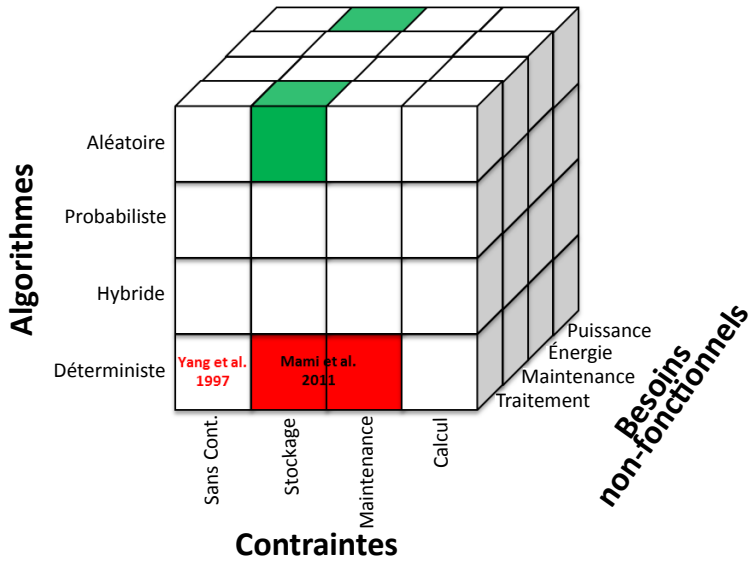


FIGURE 2.7 – Méthodes de résolutions du $\mathcal{PSV}$.

lieu à une formalisation *multi-objectif* du $\mathcal{PSV}$ comme dans [160] où le temps de réponse et le coût de maintenance ont été considérés.

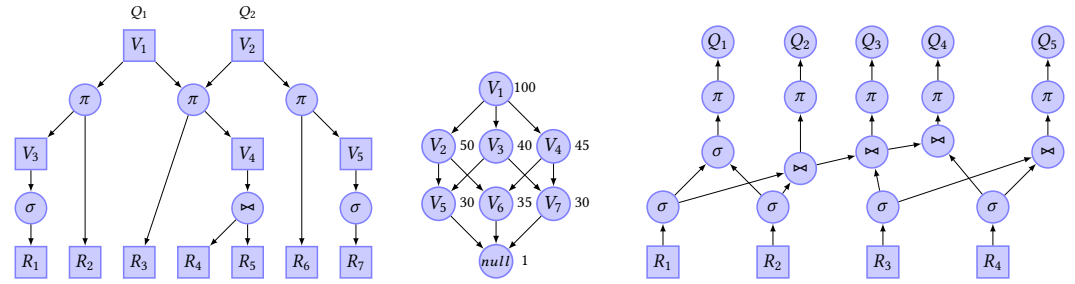
Les contraintes. Les premières études sur le $\mathcal{PSV}$ supposent l'absence de contraintes [181]. Par la suite et en raison de la grande taille de stockage des $\mathcal{VM}$ sélectionnées, l'espace de stockage devient la principale contrainte que le processus de sélection doit intégrer [109]. Ensuite, le coût de maintenance devient un paramètre important car les $\mathcal{VM}$ sélectionnées doivent être mises à jour une fois les tables de base modifiées [110]. Certaines études ont considéré les deux contraintes [139]. Récemment, des études traitent le cas du déploiement des $\mathcal{VM}$ dans un environnement distribué avec le coût de communication réseaux comme contrainte [61, 175], ou encore un environnement cloud sous la contrainte de coût monétaire [143].

Algorithmes de résolution. La résolution de $\mathcal{PSV}$ est généralement réalisée soit par des algorithmes simples ou avancés qui couvrent des algorithmes *déterministes* [110, 109, 40], algorithmes *aléatoires* [139, 160], algorithmes *hybrides* [295], et la *programmation par contraintes* [176, 175] (pour plus de détails, voir l'étude de [174, 68]).

Type de sélection. Il existe deux approches pour la sélection des vues : *statique* et *dynamique*. Une approche de sélection de vue statique suppose que la charge de requêtes est fixe, et choisit ensuite l'ensemble de vues à matérialiser [286, 40]. Alors que, dans une approche de sélection de vue dynamique, la sélection est appliquée lorsqu'une requête arrive. Par conséquent, la charge de requêtes est construite progressivement et change avec le temps [143, 151].

Structure de données. Habituellement, la première étape dans la résolution de $\mathcal{PSV}$ est l'identification des vues candidates qui sont prometteuses pour la matérialisation. De nombreuses techniques de structures de données ont été proposées pour obtenir l'ensemble des vues candidates [174] :

— ***Graphe de vue ET/OU*** : est un graphe orienté acyclique (DAG), composé de deux types de nœuds : opération et équivalence. Chaque nœud d'opération correspond à une opération algébrique dans le plan d'une requête (sélection, jointure, projection, etc.). Chaque nœud d'équivalence



(a) Exemple d'un graphe de vue ET/OU.

(b) Exemple d'un graphe de données cube treillis.

(c) Exemple d'un graphe de traitement de plan multi-vue.

FIGURE 2.8 – Des exemples de structures de données graphe orienté acyclique.

correspond à un ensemble d'expressions logiques équivalentes. De nombreux algorithmes ont été proposés dans la littérature exploitant cette structure [110, 109, 176]. La Figure 2.8a donne un exemple de graphe de vue ET/OU.

- ***Traitement de plan multi-vue (MVPP)*** : Le MVPP défini par Yang *et al* [286] est un DAG dans lequel les nœuds racine sont les requêtes, les nœuds feuilles sont les relations de base et tous les autres nœuds intermédiaires sont des vues de sélection, de projection, de jointure ou d'agrégation qui contribuent à la construction d'une requête donnée [40, 295]. Un exemple d'un MVPP est représenté sur la Figure 2.8c.
- ***Données cube treillis*** : est un DAG proposé dans le contexte des entrepôts de données multi-dimensionnel, dont les nœuds représentent les requêtes ou vues qui sont caractérisées par une opération de regroupement, et les arêtes définissent la relation entre les vues [139, 115, 290]. La Figure 2.8b donne un exemple du graphe de treillis.
- ***Transformation de requête*** : dans cette technique, le $\mathcal{PSV}$ prend comme entrer les définitions de requête. Le $\mathcal{PSV}$ est modélisé comme un problème de recherche d'espaces d'états en utilisant un ensemble de règles de transformation. Ces règles détectent et exploitent des sous-expressions communes entre les requêtes de la charge et garantissent que toutes les requêtes peuvent être répondu en utilisant exclusivement les $\mathcal{VM}$ [170, 254].
- ***Analyse syntaxique*** : cette technique analyse la charge de requêtes et sélectionne un sous-ensemble de relations à partir duquel on peut matérialiser une ou plusieurs vues, à condition de réduire le coût de la charge de requêtes de manière significative [9, 61].

2.2.1.4.1 Problème d'exploitation des $\mathcal{VM}$

Le problème d'exploitation des $\mathcal{VM}$ est de trouver des méthodes efficaces pour répondre à une requête en utilisant un ensemble de $\mathcal{VM}$ précédemment définies sur la base de données, plutôt que d'accéder aux relations de base de données, afin d'améliorer le temps de traitement des requêtes [114]. Cela se fait à l'aide de la *réécriture de requête* : l'objectif est de trouver une réécriture équivalente de la requête originale en utilisant une ou plusieurs vues. Ce problème est prouvé NP-complet [164].

Les approches proposées dans la littérature pour résoudre ce problème se basent sur l'adaptation du modèle de coût de l'optimiseur de requêtes afin de générer le meilleur plan équivalant. Les travaux peuvent être classés en algorithmes de style *System-R* et en algorithmes de *transformation*. Les travaux

initiaux ont incorporé des vues dans l'énumération de jointure d'un optimiseur de style System-R [60, 164]. Tandis que les travaux ultérieurs qui traitent un ensemble d'opérateurs SQL plus étendu ont utilisé des règles de transformation [101, 292].

2.2.1.4.1 Problème de maintenance des $\mathcal{VM}$

Lorsqu'une relation de base (table) est modifiée, les $\mathcal{VM}$ construites sur celle-ci doivent être mises à jour afin de produire des résultats de requêtes à jour. Le processus de mise à jour d'une vue matérialisée est connu sous le nom de *maintenance de vue*. Les approches de maintenance des vues peuvent être classées en trois catégories : (1) la maintenance de vue basique par le *recalcule* des vues depuis le début [107], (2) la maintenance de vue *incrémentale*, où seules les modifications des vues matérialisées sont calculées [182], et (3) *l'auto-maintenance* qui est un processus de rafraîchissement de vues progressif sans examiner aucune des relations de base [108].

Sur la base des travaux présentés dans cette section, nous proposons une classification qui incluent de nouvelles dimensions telles que les techniques de réécriture de requêtes, la stratégie de maintenance des $\mathcal{VM}$. La Figure 2.9 illustre cette classification.

2.2.1.5 Déploiement et maintenance

Une fois la conception logique et physique terminée, la base de données peut être créée et déployée grâce à la mise en œuvre des schémas précédents. Cette phase consiste à choisir la plate-forme adéquate dans laquelle la base de données cible sera déployée. Plusieurs plates-formes peuvent stocker la base de données : machines centralisées, décentralisées, parallèles, cloud, etc. Le choix de la plate-forme de déploiement dépend du budget de l'entreprise et des $\mathcal{BNF}$ fixés [76].

Lorsque la base de données est mise en marche, la surveillance indique si les $\mathcal{BF}$ et $\mathcal{BNF}$ sont respectées. Si elles ne sont pas satisfaites, des modifications ou *tuning* doivent être apportées pour améliorer les performances. D'autres modifications peuvent être nécessaires lorsque les besoins changent ou la quantité de données évolue [169]. Ainsi, le cycle de vie se poursuit avec la surveillance, le tuning et les modifications.

2.2.1.6 Bilan et discussion

Dans les sections précédentes, nous avons présenté et étudié le cycle de vie de la conception des bases de données. Cette étude nous permet de comprendre les différentes caractéristiques de chaque phase et les interdépendances qui peuvent exister entre elles. L'étude contribue également à identifier les phases potentielles pour intégrer la dimension de l'énergie. Nous pouvons constaté que l'énergie est influencée par toutes les étapes du cycle : (1) l'énergie est considérée comme un nouveau $\mathcal{BNF}$ dans une entreprise qui veut créer/modifier sa base de données à fin de minimiser la consommation d'énergie du système, (2) le choix du modèle conceptuel (entité-association ou UML par exemple) influence aussi sur les étapes suivantes du cycle et modifie par conséquent le comportement de l'énergie du système, (3) le choix du type du SGBD lors de la modélisation logique a un impact majeur sur la consommation de l'énergie car il est responsable de la gestion des données et sa structure interne diffère d'un SGBD à un

FIGURE 2.9 – Classification des travaux pour la technique de $\mathcal{VM}$.

autre (par exemple un SGBD orienté colonne améliore la performance et peut influencer sur l'énergie), (4) la conception physique influe largement sur l'énergie, l'organisation physique des données, les méthode d'accès, le choix des $\mathcal{SO}$ ont tous un effet sur l'usage de l'énergie, (4) finalement, le choix de l'environnement de déploiement de la base de données (centralisé, distribué, etc.) rend l'amélioration de l'efficacité énergétique un processus non trivial.

Notez que le choix approprié de la variante d'une technique de conception donnée dans chaque phase est crucial, et peut avoir un effet sur les autres phases et sur l'énergie globale du système. En résumé, nous concluons que chaque phase du cycle de vie de la conception des bases de données devrait être intégrée à la conception de systèmes de bases de données éco-énergétiques.

2.2.2 Traitement de requêtes dans un SGBD relationnel

Après la conception et la création de la base de données, les utilisateurs peuvent exploiter les données. Le traitement de requêtes est le groupe de composants d'un SGBD qui transforme les requêtes des utilisateurs et les commandes de modification des données en une séquence d'opérations de base de données et exécute ces opérations [97]. Notre but est d'examiner le processus de traitement de requêtes dans un SGBD pour des opportunités de minimisation d'énergie.

Dans ce chapitre, nous discuterons des techniques utilisées en interne par un SGBD pour traiter des requêtes de haut niveau. Une requête exprimée dans un langage de requête de haut niveau comme SQL doit d'abord être analysée syntaxiquement. *L'analyseur* identifie les mots clés de la requête (les mots clés SQL, les noms d'attributs et les noms de relations) qui apparaissent dans le texte de la requête, et vérifie la syntaxe de requête pour déterminer si elle est formulée selon les règles de syntaxe (règles de grammaire) du langage de requête. La requête doit également être *validée* en vérifiant que tous les noms d'attributs et de relations sont des noms valides et sémantiquement significatifs dans le schéma de la base de données. En utilisant des règles de *transformation*, une représentation interne de la requête est ensuite créée, habituellement sous la forme d'une structure de données arborescente appelée *arbre algébrique*. Le SGBD doit alors concevoir une stratégie *d'exécution* ou un *plan de requête* pour extraire les résultats de la requête à partir des fichiers de base de données. Une requête a de nombreuses stratégies d'exécution possibles, et le processus de choisir un approprié pour traiter une requête est connu sous le nom *d'optimisation de requête* [88]. La Figure 2.10 donne un aperçu du processus de traitement des requêtes par un SGBD.

2.2.2.1 Analyse

Le travail de l'analyseur est de prendre du texte écrit dans une langue comme SQL et de le convertir en un arbre d'analyse, qui est un arbre dont les nœuds correspondent soit à : (1) *atomes*, qui sont des éléments lexicaux tels que des mots clés (par exemple, `SELECT`), des noms d'attributs ou de relations, des constantes, des parenthèses, des opérateurs tels que + ou <, et d'autres éléments de schéma, ou (2) des *catégories syntaxiques*, qui sont des noms pour des sous-parties de requête qui ont un rôle similaire. Ils sont représentés par un nom descriptif entre les symboles < et > [97]. Par exemple, `<Requête>` est utilisé pour représenter une requête, et `<Condition>` représentera toute expression qui est une condition. Nous présenterons ces propos par un exemple dans la section suivante.

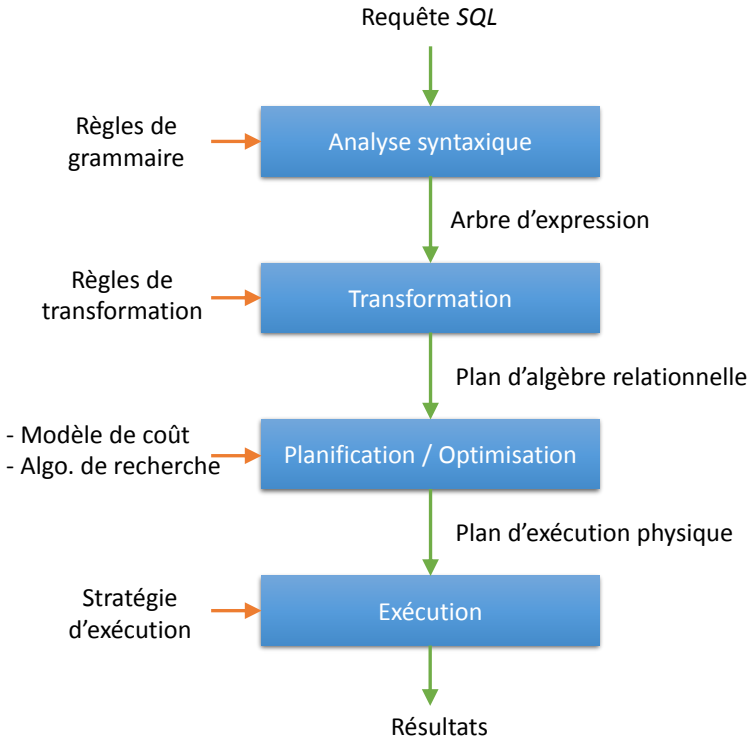


FIGURE 2.10 – Architecture globale pour le traitement de requêtes dans un SGBD.

2.2.2.1.1 Une grammaire pour un sous-ensemble simple de SQL

Nous allons illustrer le processus d'analyse en donnant quelques règles qui décrivent un petit sous-ensemble de requêtes SQL.

2.2.2.1.1 Requête

La catégorie syntaxique <Requête> est destinée à représenter certaines requêtes SQL. Nous lui donnons la règle :

```
<Requête> ::= SELECT <Sélection> FROM <Depuis> WHERE <Condition>
```

Le symbole `::=` signifie « peut-être exprimer comme ». Les catégories syntaxiques `<Sélection>`, `<Depuis>` et `<Condition>` représentent des listes pouvant suivre `SELECT`, `FROM` et `WHERE`, respectivement. Nous décrirons les formes de ces listes dans les prochaines sections.

2.2.2.1.1 Sélection

```
<Sélection> ::= <Attribut>, <Sélection>
<Sélection> ::= <Attribut>
```

Ces deux règles disent qu'une liste de sélection peut être une liste d'attributs séparés par des virgules : soit un attribut unique, soit un attribut, une virgule et toute liste d'un ou plusieurs attributs.

2.2.2.1.1 Depuis

```
<Depuis> ::= <Relation>, <Depuis>
<Depuis> ::= <Relation>
```

Ici, une liste de type `Depuis` est définie comme étant une liste de relations séparées par des virgules.

2.2.2.1.1 Condition

Les règles que nous utiliserons sont :

```
<Condition> ::= <Condition> AND <Condition>
<Condition> ::= <Attribut> IN ( <Requête> )
<Condition> ::= <Attribut> = <Attribut>
<Condition> ::= <Attribut> LIKE <Motif>
```

2.2.2.1.1 Catégories syntaxiques

Les catégories syntaxiques `<Attribut>`, `<Relation>` et `<Motif>` sont spéciales, puisqu'elles ne sont pas définies par des règles grammaticales, mais par des règles sur les atomes pour lesquels elles peuvent être utilisées. Par exemple, dans un arbre d'analyse, l'un des fils de `<Attribut>` peut être n'importe quelle chaîne de caractères qui identifie un attribut du schéma de base de données. De même, `<Relation>` peut être remplacé par n'importe quelle chaîne de caractères qui représente une relation dans le schéma, et `<Motif>` peut être remplacé par n'importe quelle chaîne entre guillemets qui est un modèle SQL légal.

Exemple 5. *Notre étude de l'analyse et de la réécriture de requêtes sera basée sur notre exemple de gestion de vente. Nous voulons répondre à la requête « trouver les noms des clients qui ont payé leurs commandes en espèces ». Nous identifions les clients qui ont payé des commandes en espèces en demandant si leur méthode de paiement est égale à « espèces », en utilisant l'opérateur* `=`.

La requête SQL utilisé pour réponde à cette question est :

```sql
SELECT Nom
FROM Clients, Commandes
WHERE Clients.ClientsID = Commandes.ClientsID AND
      Méthode_Paiement = 'espèces';
```

L'arbre d'analyse pour notre requête, selon la grammaire que nous avons proposée, est illustré dans la Figure 2.11. A la racine se trouve la catégorie syntaxique `<Requête>`. En parcourant l'arbre, nous voyons que cette requête a une forme de `select-from-where`, la liste de `<Sélection>` se compose uniquement de l'attribut `Nom` et la liste de `<Depuis>` contient plus d'une relation et deux conditions connectées par `AND`.

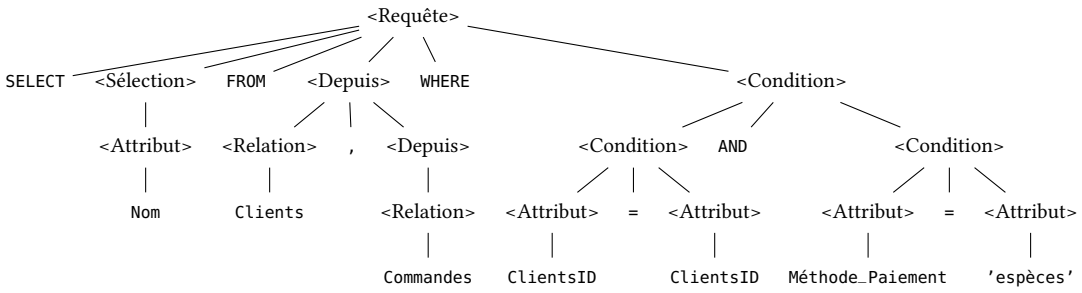


FIGURE 2.11 – Exemple d'un arbre d'analyse.

L'analyseur vérifie si une relation dans la requête est en réalité une vue virtuelle, chaque utilisation de cette relation dans la requête doit être remplacée par un arbre d'analyse qui décrit la vue [97]. L'analyseur est aussi responsable de la vérification sémantique. Même si la requête est valable syntaxiquement, elle peut en fait violer une ou plusieurs règles sémantiques sur l'utilisation des noms. Par exemple, l'analyseur doit :

- *Vérifiez les relations* : Chaque relation mentionnée dans une clause `FROM` doit être une relation ou une vue dans le schéma de la base de données courante.
- *Vérifiez et résoudre les attributs* : Chaque attribut mentionné dans la clause `SELECT` ou `WHERE` doit être un attribut d'une certaine relation dans la requête. Par exemple, l'attribut `Nom` est un attribut de `Clients`, donc l'analyseur valide cette utilisation. Il vérifie également l'ambiguïté, signalant une erreur si un attribut est dans la portée de deux ou plusieurs relations.
- *Vérifiez les types* : Tous les attributs doivent être d'un type approprié à leurs utilisations. Par exemple, `Méthode_Paiement` est utilisé dans une comparaison `=`, ce qui nécessite que cet attribut soit une chaîne ou un type qui peut être contraint à une chaîne. De même, les opérateurs sont vérifiés pour voir qu'ils s'appliquent aux valeurs des types appropriés et compatibles.

2.2.2.2 Transformation

Dans cette étape, l'arbre d'analyse résultant de l'étape précédente est transformé en une expression de l'algèbre relationnelle étendue. L'objectif de la transformation des requêtes est : (1) la construction d'un arbre standardisé pour l'optimisation des requêtes (*standardisation*), (2) l'élimination de la redondance (*simplification*), et (3) la construction d'une expression améliorée (*amélioration*) [135]. Nous verrons dans cette section comment appliquer des heuristiques qui améliorent l'expression algébrique de la requête, en utilisant certaines des nombreuses règles algébriques issues de l'algèbre relationnelle [88]. À titre préliminaire, cette section répertorie les règles algébriques qui transforment un arbre d'expression en un arbre d'expression équivalent qui peut avoir un plan de requête physique plus efficace. Le résultat de l'application de ces transformations algébriques est le plan de requête logique qui est la sortie de cette phase.

2.2.2.2.1 Règles algébriques pour améliorer les plans de requête

Une règle est dite *commutative* sur un opérateur si le résultat reste invariable si l'on intervertit les arguments de l'opérateur. Par exemple, + et × sont des opérateurs commutatifs d'arithmétique. Plus précisément, $x+y = y+x$ et $x \times y = y \times x$ pour tout nombre x et y. D'autre part, − n'est pas un opérateur arithmétique commutatif : $x - y \neq y - x$.

Une règle est dite *associative* sur un opérateur si nous pouvons regrouper deux utilisations de l'opérateur soit de la gauche ou de la droite. Par exemple, + et × sont des opérateurs arithmétiques associatifs, ce qui signifie que $(x + y) + z = x + (y + z)$ et $(x \times y) \times z = x \times (y \times z)$. D'autre part, − n'est pas associatif : $(x - y) - z \neq x - (y - z)$.

Nous allons présenter quelques règles de transformation de certain opérateurs SQL [94, 88, 97] :

1. **Décomposition de** σ. Une condition de sélection conjonctive peut être décomposée en une cascade (c'est-à-dire une séquence) d'opérations σ individuelles :
   $\sigma_{c_1 \text{ AND } c_2 \text{ AND } \cdots \text{ AND } c_n}(R) \equiv \sigma_{c_1}(\sigma_{c_2}(\cdots(\sigma_{c_n}(R))\cdots))$
2. **Commutativité de** σ. L'opération de sélection σ est commutative :
   $\sigma_{c_1}(\sigma_{c_2}(R)) \equiv \sigma_{c_2}(\sigma_{c_1}(R))$
3. **Décomposition de** π. Dans une décomposition (séquence) d'opérations de projection π, tous sauf le dernier peuvent être ignorés :
   $\pi_{Liste_1}(\pi_{Liste_2}(\cdots(\pi_{Liste_n}(R))\cdots)) \equiv \pi_{Liste_1}(R)$
4. **Commuter** σ **avec** π. Si la condition de sélection c ne concerne que les attributs $A_1, \cdots, A_n$ dans la liste de projection, les deux opérations peuvent être commutées :
   $\pi_{A_1,A_2,\cdots,A_n}(\sigma_c(R)) \equiv \sigma_c(\pi_{A_1,A_2,\cdots,A_n}(R))$
5. **Commutativité de** $\bowtie$ **(et** $\times$**).** L'opération de jointure est commutative, de même que l'opération de produit cartésien $\times$:
   $R \bowtie_c S \equiv S \bowtie_c R$
   $R \times S \equiv S \times R$
6. **Commuter** σ **avec** $\bowtie$ **(ou** $\times$**).** Si tous les attributs de la condition de sélection c ne comportent que les attributs d'une des relations jointes, par exemple R, les deux opérations peuvent être commutées comme suit :
   $\sigma_c(R \bowtie S) \equiv (\sigma_c(R)) \bowtie S$
7. **Commuter** π **avec** $\bowtie$ **(ou** $\times$**).** Supposons que la liste de projection est $L = A_1, \cdots, A_n, B_1, \cdots, B_m$, où $A_1, \cdots, A_n$ sont des attributs de R et $B_1, \cdots, B_m$ sont des attributs de S. Si la condition de jointure c implique seulement les attributs en L, les deux opérations peuvent être commutées comme suit :
   $\pi_L(R \bowtie_c S) \equiv (\pi_{A_1,\cdots,A_n}(R)) \bowtie_c (\pi_{B_1,\cdots,B_m}(S))$
8. **Commutativité des opérations ensemblistes**. Les opérations ensemblistes $\cup$ et $\cap$ sont commutatives, mais − ne l'est pas.
9. **Associativité de** $\bowtie$**,** $\times$**,** $\cup$ **et** $\cap$. Ces quatre opérations sont individuellement associatives ; C'est-à-dire si les occurrences de θ représentent la même opération qui est l'une de ces quatre opérations, on a :

$(R\ \theta\ S)\ \theta\ T \equiv R\ \theta\ (S\ \theta\ T)$

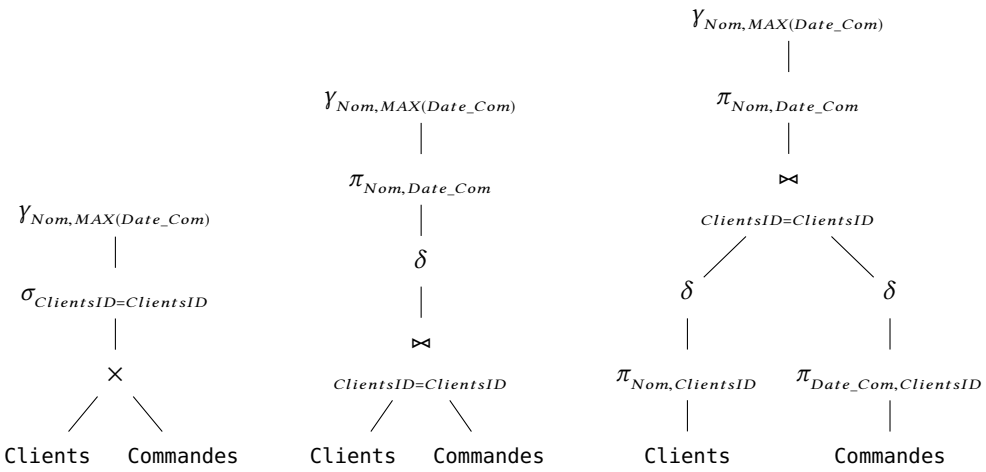
10. **Commuter σ avec des opérations ensemblistes**. L'opération σ commute avec $\cup$, $\cap$ et $-$. Si θ représente l'une de ces trois opérations dans une expression, on a :
    $\sigma_c(R\ \theta\ S) \equiv (\sigma_c(R))\ \theta\ (\sigma_c(S))$
11. **L'opération π commute avec $\cup$.**
    $\pi_L(R \cup S) \equiv (\pi_L(R)) \cup (\pi_L(S))$
12. **Conversion d'une séquence (σ, $\times$) en $\bowtie$**. Si la condition c d'une σ qui suit un $\times$ correspond à une condition de jointure, convertissez la séquence (σ, $\times$) en une $\bowtie$ comme suit :
    $(\sigma_c(R \times S)) \equiv (R \bowtie_c S)$
13. **Descendre σ en bas en conjonction avec la différence ensembliste**.
    $\sigma_c(R - S) = \sigma_c(R) - \sigma_c(S)$
14. **Descendre σ à un seul argument dans $\cap$**. Si dans la condition σ_c tous les attributs sont de la relation R, alors :
    $\sigma_c(R \cap S) = \sigma_c(R) \cap \sigma_c(S)$
15. **Élimination des doubles**. L'opérateur δ élimine les doublons d'un ensemble. Si θ représente l'une de ces trois opérations : $\bowtie$, $\bowtie_c$ ou $\times$, dans une expression, alors :
    $\delta(R\ \theta\ S) \equiv \delta(R)\ \theta\ \delta(S)$
    $\delta(\sigma_c(R)) \equiv \sigma_c(\delta(R))$.
16. **Règles de groupement et d'agrégation**. L'opérateur γ a plusieurs règles, par exemple, il absorbe δ. Nous pouvons projeter des attributs inutiles avant d'appliquer l'opération δ. Nous pouvons également supprimer les doublons dans le cas d'agrégation de type `MIN` et `MAX` :
    $\delta(\gamma_L(R)) \equiv \gamma_L(R)$
    $\gamma_L(R) \equiv \gamma_L(\pi_M(R))$; Si M est une liste contenant tous les attributs mentionnés dans L
    $\gamma_L(R) \equiv \gamma_L(\delta(R))$; L est `MIN` et/ou `MAX`
17. **Quelques transformations triviales**.
    Si S est vide, alors $R \cup S = R$
    Si la condition c dans σ_c est vraie pour tout R, alors $\sigma_c(R) = R$

Exemple 6. *Sur notre exemple de gestion de vente, supposons que nous voulons savoir pour chaque client la date de commande la plus récente. Nous pouvons exprimer cette requête comme :*

```sql
SELECT Nom, MAX(Date_Com)
FROM Clients, Commandes
WHERE Clients.ClientsID = Commandes.ClientsID
GROUP BY Nom;
```

Un plan de requête logique initial construit directement à partir de la requête est représenté dans la Figure 2.12a. La liste `FROM` est exprimée par un produit cartésien et la clause `WHERE` par une sélection. Le groupement et l'agrégation sont exprimés par l'opérateur γ. Nous pouvons appliquer les transformations suivantes :

1. Combiner la sélection et le produit dans une jointure (règle 12).



(a) Plan de requête logique initial.

(b) Plan de requête après la première transformation.

(c) Plan de requête après la deuxième transformation.

FIGURE 2.12 – Étape de transformation du plan de requête logique pour l'Exemple 6.

2. Générer un δ en dessous du γ (règle 16).
3. Générer un π entre le γ et le δ généré pour projeter sur `Nom` et `Date_Com` (règle 16).

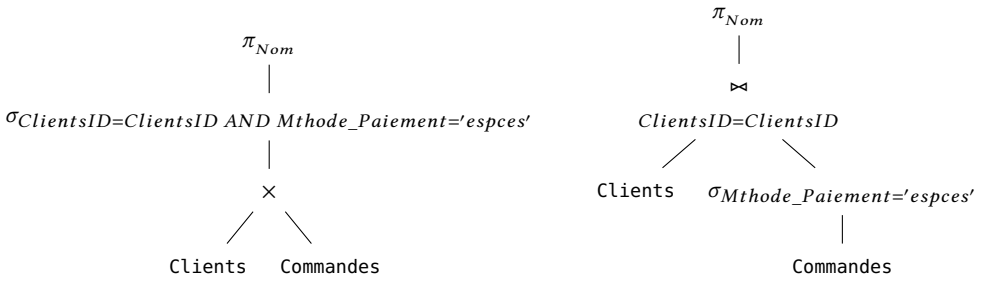
Le plan résultant est représenté à la Figure 2.12b. Nous pouvons maintenant pousser le δ sous la $\bowtie$ et introduire π (règle 15). Ce nouveau plan de requête est représenté à la Figure 2.12c.

2.2.2.2.2 Amélioration du plan de requête logique

Il existe un certain nombre de règles algébriques qui améliorent les plans de requêtes logiques. Les règles suivantes sont les plus couramment utilisées par les optimiseurs [97, 88] :

- Les sélections peuvent être poussées vers le bas dans l'arborescence d'expression dans la mesure du possible. Si une condition de sélection est composée de `AND` de plusieurs conditions, nous pouvons diviser la condition et pousser chaque partie vers le bas de l'arbre séparément. Cette stratégie est probablement la technique d'amélioration la plus efficace.
- De même, les projections peuvent être poussées vers le bas de l'arbre, ou de nouvelles projections peuvent être ajoutées.
- Les opérations d'éliminations des doubles peuvent parfois être supprimées ou déplacées vers une position plus pratique dans l'arborescence.
- Certaines sélections peuvent être combinées avec un produit ci-dessous pour transformer la paire d'opérations en une jointure, ce qui est généralement beaucoup plus efficace à évaluer que les deux opérations séparément.

Exemple 7. *Considérons l'arbre d'analyse de la Figure 2.11. L'expression d'algèbre relationnelle résultante après la transformation de cet arbre est illustrée dans la Figure 2.13a. Maintenant, nous voulons appliquer les nouvelles règles de transformation. Tout d'abord, nous pouvons diviser les deux parties de la sélection en* `ClientsID = ClientsID` *et* `Méthode_Paiement = 'espèces'`*. Ce dernier peut être poussé vers le bas de l'arbre, puisque le seul attribut impliqué,* `Méthode_Paiement`*, est de la relation* `Commandes`*. Ensuite, la sélec-*



(a) Plan de requête logique initial.

(b) Plan de requête après la transformation.

FIGURE 2.13 – Transformation du plan de requête logique pour l'Exemple 5.

tion et le produit peuvent être combiné dans une seule jointure. L'effet de ces transformations est démontré sur la Figure 2.13b.

2.2.2.3 Génération des plans et optimisation

Après avoir analysé et transformé une requête en un plan de requête logique, la prochaine étape consiste à transformer le plan logique en un plan physique. Cela se fait normalement en considérant plusieurs plans physiques qui sont dérivés du plan logique, et en évaluant ou en estimant le coût de chacun. Après cette évaluation, souvent appelée génération de plans basée sur le *modèle coût* [229], le plan de requête physique avec le coût le moins estimé est choisi. Ce plan est passé au moteur d'exécution de requêtes [97]. La Figure 2.14 donne un aperçu du processus d'optimisation des requêtes.

Le premier problème est de savoir comment estimer les coûts des plans avec précision. Ces estimations sont basées sur (1) des statistiques sur données de la base, et (2) des statistiques de système. Compte tenu des valeurs de ces paramètres, nous pouvons faire un certain nombre d'estimations raisonnables de la taille des relations qui peuvent être utilisés pour prédire le coût d'un plan physique complet.

2.2.2.3.1 Statistiques sur les données

Étant donné un arbre d'opérateur d'une requête, il est d'une importance fondamentale d'être en mesure d'évaluer avec *précision* et *efficacité* son coût. L'estimation des coûts doit être précise parce que l'optimisation est seulement aussi bonne que ses estimations de coûts. L'estimation des coûts doit être efficace car elle est dans la boucle interne de l'optimisation des requêtes et est invoquée à plusieurs reprises [55]. Un SGBD moderne permet généralement à l'utilisateur ou à l'administrateur de demander explicitement la collecte de statistiques telles que $T(R)$, le nombre de tuples dans une relation R, et $V(R, a)$, le nombre de valeurs différentes dans la colonne de relation R pour l'attribut a. Ces statistiques sont ensuite utilisées dans l'optimisation des requêtes, inchangées jusqu'à la prochaine commande de collecte de statistiques [97]. Généralement, les statistiques sont sauvegardées dans la base de données comme relations appelées *catalogue* ou *dictionnaire de données* [88].

En balayant entièrement la relation R, il est facile de compter le nombre de tuples $T(R)$ et aussi de découvrir le nombre de valeurs $V(R, a)$ différentes pour chaque attribut a. De plus, un SGBD peut

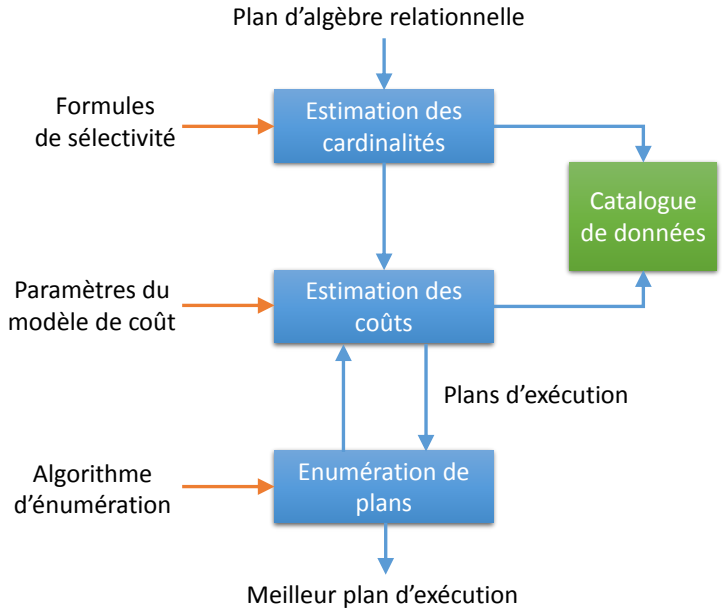


FIGURE 2.14 – Une architecture globale pour l'étape d'optimisation de requêtes.

calculer un *histogramme* de distribution des valeurs d'un attribut donné. Si $V(R, A)$ n'est pas trop grand, l'histogramme peut être constitué du nombre (ou fraction) des tuples ayant chacune des valeurs de l'attribut a. Les histogrammes les plus courants sont [205] :

- **Largeur égale.** Divise l'intervalle de valeurs en intervalles de taille égale. On choisit une largeur w, avec une constante v_0. On fournit des comptages du nombre de tuples avec des valeurs v dans les intervalles $v_0 \leq v < v_0 + w$, $v_0 + w \leq v < v_0 + 2w$, etc. La valeur v_0 peut être la valeur la plus petite ou une borne inférieure sur les valeurs vues jusqu'ici.
- **Hauteur égale.** Ajuste les limites des intervalles de sorte que chaque intervalle ait le même nombre de valeurs. Ce sont les « centile [8] » courants. On choisit une fraction p, et on énumère la valeur la plus petite, la valeur qui est la fraction p du plus bas, la fraction $2p$ du plus bas, et ainsi de suite, jusqu'à la valeur la plus grande.
- **Valeurs les plus fréquentes.** Nous pouvons énumérer les valeurs les plus communes et leur nombre d'occurrences.

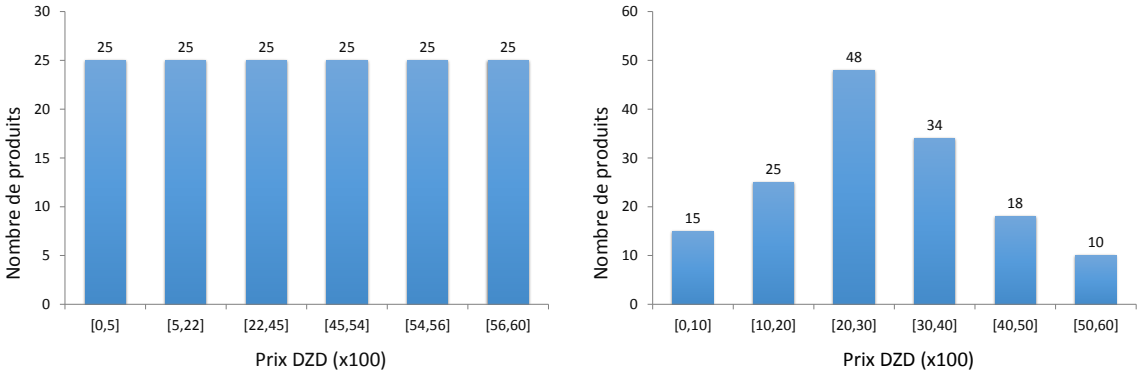
Les SGBD modernes comme IBM DB2, Informix, Microsoft SQL Server, Oracle et Sybase ASE utilisent la technique d'histogramme pour générer les statistiques sur les données [209].

2.2.2.3.2 Statistiques de système

Le coût de l'évaluation des requêtes peut être mesuré en fonction d'un certain nombre de ressources système différentes. Les ressources ou paramètres les plus importants sont les suivants :

- **Coût d'accès au stockage secondaire.** C'est le coût du transfert (lecture et écriture) des blocs de données entre le stockage de disque secondaire et les tampons de mémoire principale. Cela

8. Un centile est chacune des 99 valeurs qui divisent les données triées en 100 parts égales, de sorte que chaque partie représente 1/100 de l'échantillon de population.



(a) Histogramme hauteur égale.

(b) Histogramme largeur égale.

FIGURE 2.15 – Exemples des histogrammes de distribution des valeurs d'attribut *Prix* de la table *Produit*.

est également connu sous le nom de coût d'E/S (entrée/sortie) ou $C_{\mathcal{E}/\mathcal{S}}$. Le coût de la recherche d'enregistrements dans un fichier disque dépend du type de structures d'accès sur ce fichier et la nature d'allocation des blocs de fichiers.

- **Coût de stockage sur disque.** C'est le coût de stockage sur disque de tous les fichiers intermédiaires générés par une stratégie d'exécution de la requête.
- **Coût de calcul.** C'est le coût d'exécution des opérations sur les enregistrements en mémoire tampon pendant l'exécution de la requête. Ces opérations comprennent la recherche, le tri, la fusion d'enregistrements pour une opération de jointure ou de tri et l'exécution de calculs sur les valeurs. Cela est également connu comme le coût de CPU (unité centrale de traitement) ou C_{CPU}.
- **Coût d'utilisation de la mémoire.** C'est le coût correspondant au nombre de tampons mémoire nécessaires pendant l'exécution de la requête.
- **Coût de la communication.** Il s'agit du coût d'expédition de la requête et de ses résultats à partir du site de base de données vers le site ou le terminal d'où provient la requête. Dans les bases de données distribuées, il inclut également le coût du transfert des tables et des résultats entre différents ordinateurs au cours de l'évaluation des requêtes. Le coût de communication est abrégé C_{COM}.

2.2.2.3.2 Obtention des paramètres et leurs relations

Les paramètres du modèle de coût, tels que le coût d'E/S et de CPU, doivent être calculés. Trois méthodes existent dans la littérature pour obtenir ces paramètres : *analytique*, *empirique* et *dynamique* [195].

1. **Méthode analytique.** C'est l'approche classique employée par les premiers travaux sur les modèles de coût. Elle utilise des connaissances *détaillées* sur le SGBD, l'implémentation des opérateurs algébriques, ainsi que les caractéristiques externes du matériel, pour déterminer analytiquement les informations sur les coûts [229, 105, 112]. Cette approche est adaptée par le SGBD open-source PostgreSQL [278].
2. **Méthode empirique.** Cette approche traite le SGBD comme une *boîte noire*. Elle consiste à

identifier d'abord un ensemble de caractéristiques de requête et de données qui déterminent potentiellement les coûts des opérateurs. Puis, exécuter ces requêtes d'apprentissages sur le système. Ensuite, appliquer des modèles *statistiques* ou *d'apprentissage automatique* aux données collectées, et à partir de ceux-ci déterminer les paramètres finaux du modèle de coût et la relation entre eux [279, 96, 166]. Cette approche est également utilisée par les SGBD commerciaux comme Oracle [48].

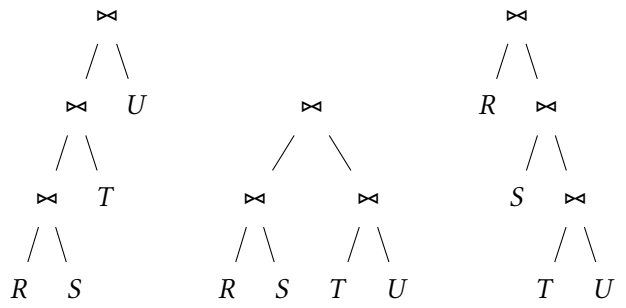
3. **Méthode dynamique.** Les méthodes précédentes supposent que l'environnement d'exécution est toujours stable. Cependant, dans la plupart des cas, les facteurs d'environnement d'exécution changent fréquemment, tels que la charge et la vitesse du CPU, le débit d'E/S, la disponibilité de mémoire, le schéma et les données de la BD, etc. Cette troisième approche se base sur la méthode empirique, et de plus, elle surveille le comportement d'exécution du SGBD et collecte et ajuste dynamiquement les informations sur les coûts pour faire face au changement d'environnement [86, 235, 146].

Compte tenu d'un plan d'exécution, nous pouvons maintenant estimer son coût en utilisant l'estimation de la cardinalité des résultats intermédiaires faite par les techniques de Section A.1 de l'Annexe A, couplées aux estimations de coûts pour divers algorithmes et méthodes d'accès décrits au Section A.2 de l'Annexe A.

2.2.2.3.3 Approches pour l'énumération des plans physiques

Maintenant, considérons l'utilisation des estimations de coûts dans la conversion d'un plan de requête logique en un plan de requête physique. L'approche de référence, appelée *exhaustive*, consiste à considérer toutes les combinaisons de choix possible. Chaque plan physique est affecté à un coût estimé, et celui avec le plus petit coût est sélectionné [88]. Cependant, il existe d'autres approches pour sélectionner un plan physique. En terme d'exploration de l'espace des plans physiques possibles, il existe deux approches principales : (1) *descendante* et (2) *ascendante*. Dans la première approche, l'estimation se fait à partir de la racine de l'arbre du plan de requête logique vers le bas en choisissant le meilleur choix à chaque étape. Cette approche est adoptée par Volcano, Cascades, Tandem et Microsoft SQL Server [171]. Tandis que dans la deuxième approche, l'estimation se fait à partir des feuilles vers la racine. Cette approche est utilisée dans System R, DB2, Oracle et Informix [171]. Nous décrirons ensuite les différentes techniques proposées dans la littérature pour l'énumération des plans physiques [97, 88].

- **Sélection heuristique.** Une option est d'utiliser la même approche pour sélectionner un plan physique qui est généralement utilisé pour sélectionner un plan logique : faire une séquence de choix basée sur l'heuristique (Section 2.2.2.2). Il existe d'autres heuristiques qui peuvent être appliquées. Par exemple, si le plan logique appelle une sélection $\sigma_{A=c}(R)$, et la relation stockée R a un index sur l'attribut A, effectuer un balayage d'index pour obtenir seulement les tuples de R avec une valeur de A égale à c.
- **Séparation et évaluation (*Branch-and-Bound*).** Cette approche, souvent utilisée en pratique, commence par utiliser des heuristiques pour trouver un bon plan physique pour l'ensemble du plan de requête logique. Soit C le coût de ce plan. Alors, lorsque nous considérons d'autres plans pour les sous-requêtes, nous pouvons éliminer tout plan qui a un coût supérieur à C, puisque ce plan ne pourrait pas participer à un plan complet meilleur que celui que nous avons déjà. De



(a) Arbre de jointure linéaire gauche.

(b) Arbre de jointure ramifié.

(c) Arbre de jointure linéaire droit.

FIGURE 2.16 – Exemple des méthodes pour la jointure de quatre relations.

même, si nous trouvons un plan qui a un coût moins que C, nous remplaçons C par le coût de ce meilleur plan dans l'exploration ultérieure de l'espace des plans de requêtes physiques.

— **Escalade (*Hill climbing*).** Cette approche, dans laquelle nous cherchons une « vallée » dans l'espace des plans physiques et de leurs coûts, commence avec un plan physique heuristiquement choisi. Nous pouvons alors apporter de petits changements au plan, par exemple en remplaçant une méthode pour exécuter un opérateur par une autre, ou en réordonnant des jointures en utilisant les lois associatives et/ou commutatives, pour trouver des plans « proches » qui ont un coût inférieur. Lorsque nous trouvons un plan tel qu'aucune modification ne permet pas de donner un plan de coût inférieur, nous choisissons ce plan comme un plan physique.

— **Programmation dynamique.** Dans cette variante de la stratégie ascendante, nous gardons pour chaque sous-expression seulement le plan de moindre coût. Au fur et à mesure que nous traversons l'arbre, nous considérons les implémentations possibles de chaque nœud, en supposant que le meilleur plan pour chaque sous-expression est également utilisé.

— **System-R.** Cette approche améliore l'approche de programmation dynamique en gardant pour chaque sous-expression non seulement le plan de moindre coût, mais certains autres plans qui ont un coût plus élevé, alors qu'ils produisent un résultat trié dans un ordre qui peut être utile plus haut dans l'expression de l'arbre. Car les attributs déjà triés dans une opération tel que la sélection, le groupement ou la jointure peuvent être utilisés dans un algorithme de tri d'une manière efficace [229].

2.2.2.3.3 Problème d'ordre de jointure

Les règles de transformation algébriques pour l'opération de jointure peuvent produire de nombreuses expressions de jointure équivalentes. En conséquence, le nombre d'arborescences de requêtes alternatives croît très rapidement avec l'augmentation du nombre de jointures dans une requête. En général, pour un bloc de requête comportant n relations, il existe $n!$ ordres de jointure [88]. Un arbre d'une requête ayant plus de deux jointures peut être exprimé en trois structures :

1. **Arbre de jointure linéaire gauche.** Est un arbre binaire où l'enfant gauche de chaque nœud de feuille est une relation de base (Figure 2.16a).

2. **Arbre de jointure linéaire droit.** Est un arbre binaire dans lequel l'enfant droit de chaque nœud non-feuille est toujours une relation de base. Un exemple est illustré dans la Figure 2.16c.

3. **Arbre de jointure ramifié.** Est un arbre binaire où l'enfant gauche ou droit d'un nœud interne peut être un nœud de jointure (Figure 2.16b).

La plupart des optimiseurs de requêtes considèrent les arbres de jointure linéaire gauche comme arbre de jointure préféré, puis choisi un parmi les $n!$ ordres de jointure possibles. Le problème d'ordre de jointure peut être résolu par trois classes d'algorithme : déterministe, aléatoire ou génétique (pour plus de détails, voir l'étude de [268]).

— **Algorithmes déterministes.** Également connu sous le nom d'algorithme de programmation dynamique de recherche exhaustive. Les algorithmes de cette classe effectuent une certaine recherche déterministe de l'espace de solution, soit par traverse complète, soit par l'application de certaines heuristiques d'élagage de l'espace [229, 245].
— **Algorithmes aléatoires.** Ces algorithmes effectuent une marche aléatoire dans l'espace de la solution, se déplaçant d'un point à un autre. Un déplacement est possible si la solution du premier point peut être transformée en solution du second point en lui appliquant une seule règle de transformation. L'exécution de l'algorithme se termine soit quand aucun déplacement valide ne peut être effectué ou quand un temps d'exécution prédéfini s'est écoulé. La meilleure solution trouvée pendant la marche aléatoire est le résultat de l'optimisation [133, 132].
— **Algorithmes génétiques.** Les algorithmes génétiques imitent l'évolution biologique pour rechercher la solution optimale. L'idée principale est, à partir d'un ensemble initial (population) de solutions, génère des descendants par des opérations de croisement aléatoire et mutation. Les meilleurs individus de la population survivent à chaque génération et forment la nouvelle population. L'algorithme s'arrête soit après un certain nombre de générations, soit lorsque la population devient homogène au-dessus d'un certain seuil suivant de la fonction de coût [27, 247].

2.2.2.4 Exécution

La dernière étape du processus est l'exécution de la requête. Dans cette étape, toutes les opérations d'E/S et de CPU indiquées dans le plan physique sont exécutées. Un opérateur physique d'une requête peut être exécuté en mode *matérialisé* ou *pipeline* et/ou *parallèle*.

2.2.2.4.1 Matérialisée

Avec une exécution matérialisée, le résultat d'une opération est stocké comme une relation temporaire. Chaque résultat est physiquement matérialisé sur le dispositif de stockage et s'ajoute au coût global du traitement des requêtes. Par exemple, le résultat de l'opération $\bowtie$ peut être calculée et stocké comme une relation temporaire. Ce résultat est considéré comme entrée par l'algorithme qui calcule l'opération π pour produire le résultat final de la requête. La génération et le stockage de gros fichiers temporaires sur le disque prend beaucoup de temps et peuvent être inutile dans de nombreux cas, car ces fichiers seront immédiatement utilisés comme entrée pour la prochaine opération [88].

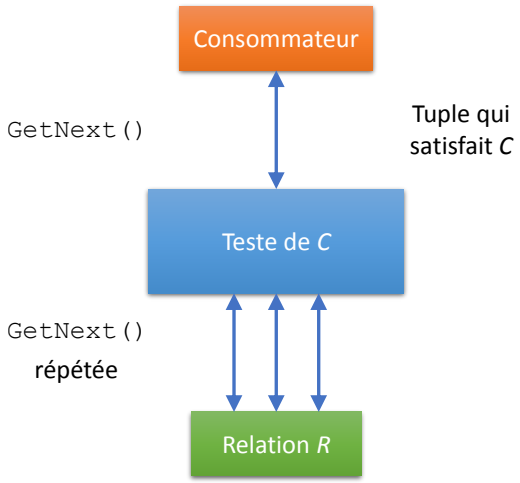


FIGURE 2.17 – Exécution d'une sélection en mode pipeline à l'aide d'itérateurs.

2.2.2.4.2 Pipeline

D'autre part, avec l'exécution en pipeline, lorsque les tuples résultants d'une opération sont produits, ils sont envoyés directement à l'opération suivante dans la séquence de requête. Par exemple, les tuples sélectionnés d'une relation sont produits par l'opération σ, et placés dans un tampon mémoire. L'opération $\bowtie$ consomme alors les tuples de la mémoire tampon, et les tuples qui résultent sont pipelinés à l'opération π. L'avantage du pipelining est les économies de coût d'écriture des résultats intermédiaires sur le disque et la relecture pour l'opération suivante. Il permet également de produire des résultats rapidement avec certaines opérations algébriques.

Différents algorithmes pour des opérations algébriques impliquent la lecture d'un ou plusieurs fichiers comme entrée, le traité, et ensuite la génération d'un fichier de résultat comme sortie. Si l'opération est mise en œuvre de telle sorte qu'elle produit qu'un tuple à la fois, elle peut être considérée comme un *itérateur*. Par exemple, une implémentation pour la jointure à boucle imbriquée génère un tuple à la fois comme sortie après chaque jointure. L'interface itérateur comprend généralement les méthodes suivantes [57] :

1. **`Open()`** : Cette méthode initialise l'opérateur en allouant des tampons pour son entrée et sa sortie et en initialisant toutes les structures de données nécessaires à l'opérateur. Il est également utilisé pour transmettre des arguments tels que les conditions de sélection nécessaires pour effectuer l'opération. Il appelle à son tour `Open()` pour obtenir les arguments dont il a besoin.
2. **`GetNext()`** : Cette méthode appelle le `GetNext()` sur chacun de ses arguments d'entrée et appelle le code spécifique à l'opération exécutée sur les entrées. Le prochain tuple de sortie généré est renvoyé, et l'état de l'itérateur est mis à jour suivant la quantité d'entrée traitée. Lorsque aucun tuple ne peut être renvoyé, il place une valeur spéciale dans le tampon de sortie.
3. **`Close()`** : Cette méthode met fin à l'itération après la génération de tous les tuples possible ou si le nombre requis/demandé de tuples a été retourné. Elle appelle également `Close()` sur les arguments de l'itérateur.

Cependant, certains opérateurs physiques ne supportent pas le concept d'interface itérateur et ne peuvent donc pas prendre en charge le pipeline. Comme par exemple l'opérateur de tri et de groupement, car ils ne produisent aucune sortie tant qu'ils n'ont pas consommé leurs entrées complètement [172].

Exemple 8. *Le consommateur du résultat du pipeline appelle* `GetNext()` *chaque fois qu'un tuple est nécessaire. Dans le cas d'une projection, il suffit d'appeler* `GetNext()` *une fois sur la source des tuples, ensuite projeter ce tuple de manière appropriée et retourner le résultat au consommateur. Pour une sélection* σ_c*, il peut être nécessaire d'appeler* `GetNext()` *plusieurs fois à la source, jusqu'à ce qu'un tuple qui satisfait la condition C est trouvé. La Figure 2.17 illustre ce processus.*

2.2.2.4.3 Parallèle

Les opérations de base de données, qui prennent souvent beaucoup de temps et impliquent beaucoup de données, peuvent généralement bénéficier d'un traitement parallèle. Trois approches principales ont été proposées pour les bases de données parallèles. Ils correspondent à trois configurations matérielles différentes de processeurs et de périphériques de stockage secondaires (disques) pour supporter le parallélisme. Dans une *architecture à mémoire partagée*, plusieurs processeurs sont connectés à un réseau d'interconnexion et peuvent accéder à une région de mémoire principale commune. Le deuxième type d'architecture est connu sous le nom *d'architecture à disque partagé*. Dans cette architecture, chaque processeur a sa propre mémoire, qui n'est pas accessible à partir d'autres processeurs. Cependant, chaque machine a accès à tous les disques à travers le réseau d'interconnexion. Le troisième type est *l'architecture sans partage*. Dans cette architecture, chaque processeur accède à sa propre mémoire principale et à son stockage disque [253].

L'architecture sans partage offre la possibilité d'atteindre le parallélisme dans le traitement des requêtes à quatre niveaux, y compris (1) le parallélisme inter-requêtes, (2) le parallélisme intra-requêtes, (3) le parallélisme inter-opérations et (4) le parallélisme intra-opérations [195].

1. **Parallélisme inter-requêtes.** Le parallélisme inter-requêtes est le « parallélisme entre les requêtes », c'est-à-dire que des requêtes ou des transactions différentes sont exécutées en parallèle les unes avec les autres d'une façon concurrente. L'utilisation principale du parallélisme entre les requêtes consiste à mettre à l'échelle les systèmes de traitement des transactions pour supporter un grand nombre de transactions par seconde.
2. **Parallélisme intra-requêtes.** Une requête est divisée en plusieurs opérations. Le parallélisme intra-requête est une exécution d'une seule requête en parallèle sur plusieurs processeurs et disques. Dans ce cas, les opérations multiples dans une requête sont exécutées en parallèle. Par conséquent, le parallélisme intra-requête est le « parallélisme au sein d'une requête ».
3. **Parallélisme inter-opérations.** Puisque les opérations de base de données fonctionnent sur des tables contenant de grands ensembles de données, nous pouvons paralléliser les opérations en les exécutant en parallèle sur différents sous-ensembles de la table. Par conséquent, le parallélisme inter-opérations est souvent appelé parallélisme partitionné, c'est-à-dire parallélisme dû à la partition des données. Comme le nombre d'enregistrements dans une table peut être important, le degré de parallélisme est potentiellement énorme. Par conséquent, le parallélisme inter-opérations est naturel dans les systèmes de base de données.

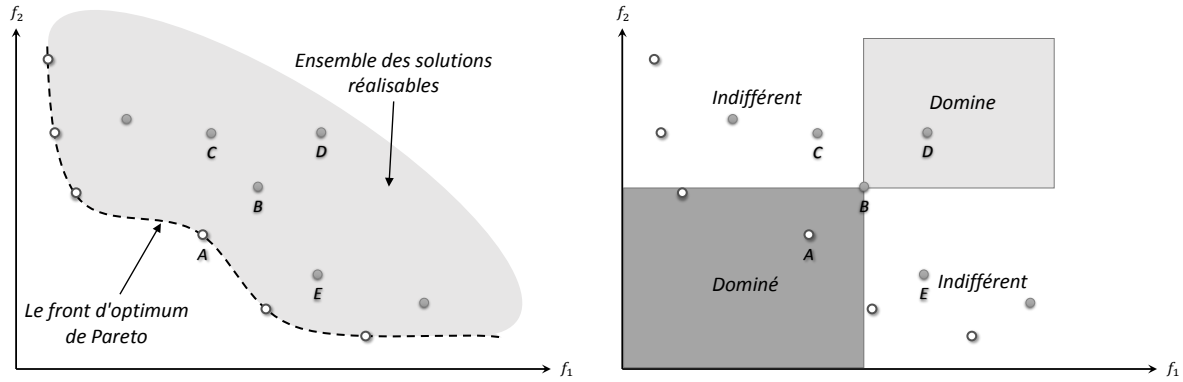
4. **Parallélisme intra-opérations.** Le parallélisme intra-opérations est là où le parallélisme est créé en exécutant simultanément différentes opérations au sein d'une même requête. Par exemple, la séquence d'opérations $R \bowtie S \bowtie T \bowtie U$. Dans ce cas, on peut traiter le $R \bowtie S$ en parallèle avec le $T \bowtie U$.

2.2.2.5 Bilan et discussion

Comme nous avons vu au cours de cette section, le principal objectif lors du traitement de requête par un SGBD est la minimisation du temps de réponse des requêtes, à l'aide de techniques sophistiquées, qui comprennent des algorithmes pour sélectionner et exécuter un plan optimal pour une requête. Cependant, nous proposons une nouvelle façon de penser sur le traitement et l'optimisation des requêtes. L'idée est de reformuler le problème d'optimisation des requêtes en : *trouvant et exécutant le plan le plus éco-énergétique qui répond à un certain objectif de temps de réponse.* Cela peut être achevé (1) en étendant l'optimiseur de requêtes existant avec un modèle de coût de consommation d'énergie, en plus du modèle de temps de réponse traditionnel, et (2) avec un modèle d'évaluation des coûts des plans d'exécution. Le modèle d'évaluation des coûts est utilisé pour sélectionner les plans d'exécution avec la capacité d'ajuster dynamiquement sa préférence entre l'énergie et la performance dans la sélection du plan de requête.

2.3 Compromis entre la performance et l'énergie : Optimisation multi-objectifs

La proposition des techniques, comme les vues matérialisées ou le traitement de requêtes, pour minimiser l'énergie dans les base de données est une seule partie du problème, car ces techniques ne garantissent pas un temps de réponse acceptable des traitements du SGBD. La deuxième partie du problème consiste à développer des techniques qui offrent le meilleur compromis entre la minimisation du temps de réponse aux requêtes et la minimisation de la consommation d'énergie du système. Ceci est considéré comme un problème *multi-objectif* parce que le problème consiste à optimiser plus d'un problème simultanément. Alors que dans l'optimisation mono-objectif la solution optimale est généralement clairement définie, cela n'est pas valable pour les problèmes d'optimisation multi-objectifs. Au lieu d'un seul optimum, il existe plutôt un ensemble de compromis alternatives, généralement connues sous le nom de solutions *Pareto-optimales*. Ces solutions sont optimales dans le sens qu'aucune autre solution dans l'espace de recherche est meilleure lorsque tous les objectifs sont pris en compte. Dans cette section, les principes de l'optimisation multi-objectifs sont décrits et les concepts de base sont formellement définis. Ceci est suivi d'une discussion sur les approches classiques et les algorithmes évolutionnaires utilisés pour approcher l'ensemble des solutions Pareto-optimales. Ensuite, nous exposons l'état de l'art sur l'optimisation multi-objectifs dans le contexte des bases de données, et plus particulièrement, le problème de sélection de vues matérialisées et le traitement de requêtes.



(a) Exemple de front d'optimum de Pareto.

(b) Exemple des relations entre les solutions.

FIGURE 2.18 – Exemples de la notion de dominance de Pareto dans l'espace objectif.

2.3.1 Problème d'optimisation multi-objectifs

Généralement, un problème d'optimisation multi-objectifs ($\mathcal{PMO}$) comprend un ensemble de n paramètres (variables de décision), un ensemble de k fonctions objectifs et un ensemble de m contraintes. L'objectif d'optimisation est de minimiser ou maximiser les k fonctions objectifs en respectant les contraintes m. Plus formellement, le $\mathcal{PMO}$ est défini comme suit [299] :

$$\begin{aligned} &\text{minimiser} && y = f(x) = (f_1(x), f_2(x), \cdots, f_k(x)) \\ &\text{avec} && e(x) = (e_1(x), e_2(x), \cdots, e_m(x)) \leq 0 \\ &\text{et} && x = (x_1, x_2, \cdots, x_n) \in X \\ & && y = (y_1, y_2, \cdots, y_k) \in Y \end{aligned} \tag{2.1}$$

Où x est le vecteur de décision, y est le vecteur objectif, X est désigné comme l'espace de décision, et Y est appelé l'espace objectif. Les contraintes $e(x) \leqslant 0$ déterminent l'ensemble des *solutions réalisables*. L'ensemble des solutions réalisable X_f est défini comme l'ensemble des vecteurs de décision x qui satisfont les contraintes $e(x)$:

$$X_f = \{x \in X | e(x) \leqslant 0\} \tag{2.2}$$

L'image de X_f, c'est-à-dire la région réalisable dans l'espace objectif, est désignée par :

$$Y_f = f(X_f) = \bigcup_{x \in X_f} \{f(x)\} \tag{2.3}$$

Dans le reste de la section, nous supposons un problème de minimisation. Pour les problèmes de maximisation ou de maximisation/minimisation mixtes, les définitions présentées ici sont similaires.

Dans l'optimisation mono-objectif ($\mathcal{OMO}$), l'ensemble réalisable est totalement ordonné selon la fonction objectif f : pour deux solutions $a, b \in X_f$ soit $f(a) \leqslant f(b)$ ou $f(b) \leqslant f(a)$. Le but est de trouver la ou les solutions qui donnent la valeur minimale de f [81]. Cependant, lorsque plusieurs objectifs sont impliqués, la situation change : X_f en général n'est pas totalement ordonné, mais partiellement ordonné. Ceci est illustré à la Figure 2.18a. La solution représentée par le point B est meilleure que la

solution représentée par le point D. Elle serait préférable si c'est un problème $\mathcal{OMO}$. En effet, dans le cas entre les points C et D : malgré que les f_2 sont égaux, C obtient de meilleures valeurs dans f_1 que D. Pour exprimer mathématiquement cette situation, les relations $=$, $\leqslant$ et $<$ sont étendues aux vecteurs objectifs par analogie avec le cas $\mathcal{OMO}$. Pour deux vecteurs objectifs quelconques u et v :

$$\begin{aligned} u = v \quad &\Leftrightarrow \quad \forall i \in \{1, \cdots, k\} : u_i = v_i \\ u \leqslant v \quad &\Leftrightarrow \quad \forall i \in \{1, \cdots, k\} : u_i \leqslant v_i \\ u < v \quad &\Leftrightarrow \quad u \leqslant v \wedge u \neq v \end{aligned} \tag{2.4}$$

En utilisant cette notion, on considère que $A < B$, $B < D$, et par conséquent, $A < D$. Cependant, en comparant B et E dans les deux fonctions objectives, aucune des solutions ne peut être considérée comme meilleure, puisque $B \nleqslant E$ et $E \nleqslant B$. Par conséquent, deux vecteurs de décision a, b peuvent avoir trois possibilités avec $\mathcal{PMO}$ en ce qui concerne la relation $\leqslant$ (contrairement à deux possibilités avec $\mathcal{OMO}$) : $f(a) \leqslant f(b)$, $f(b) \leqslant f(a)$ ou $f(a) \nleqslant f(b) \wedge f(b) \nleqslant f(a)$. Les notions suivantes sont utilisées pour classer les différentes situations.

Définition 3

(Dominance de Pareto) Pour deux vecteurs de décision quelconques a et b,

$$\begin{aligned} a \prec b \text{ (a domine b)} \quad &\Leftrightarrow \quad f(a) < f(b) \\ a \preccurlyeq b \text{ (a domine faiblement b)} \quad &\Leftrightarrow \quad f(a) \leqslant f(b) \\ a \sim b \text{ (a est indifférent à b)} \quad &\Leftrightarrow \quad f(a) \nleqslant f(b) \wedge f(b) \nleqslant f(a) \end{aligned} \tag{2.5}$$

Dans la Figure 2.18b, le rectangle gris clair encapsule la région dans l'espace objectif qui est *dominée* par le vecteur de décision représenté par B. Le rectangle gris foncé contient les vecteurs objectifs dont les vecteurs de décision correspondant *dominent* la solution associée à B. Toutes les solutions pour lesquelles le vecteur objectif résultant n'est dans aucun rectangle sont *indifférentes* à la solution représentée par B.

Sur la base du concept de dominance de Pareto, le *critère d'optimalité* pour les $\mathcal{PMO}$ peut être introduit. Toujours en référence à la Figure 2.18b, A est unique parmi B, C, D et E : son vecteur de décision a correspondant n'est pas dominé par aucun autre vecteur de décision. Cela signifie que a est optimal dans le sens où il ne peut pas être amélioré dans une objective sans causer une dégradation dans au moins une autre objective. Ces solutions sont appelées *optimum de Pareto* [42].

Définition 4

(Optimalité de Pareto) On dit qu'un vecteur de décision $x \in X_f$ est non-dominé pour un ensemble $A \subseteq X_f$ si et seulement si :

$$\nexists\, a \in A : a \prec x \tag{2.6}$$

En outre, x est dit optimum de Pareto si et seulement si x est non-dominé pour X_f.

Dans la Figure 2.18b, les points blancs représentent des solutions optimales de Pareto. Ils sont indifférents les uns aux autres. Cela rend la principale différence à la $\mathcal{OMO}$ clair : il n'existe pas de solution optimale unique, mais plutôt un ensemble de compromis optimaux. Aucune solution d'entre elles ne peut être identifiée comme meilleure que les autres à moins que l'information de préférence soit incluse (par

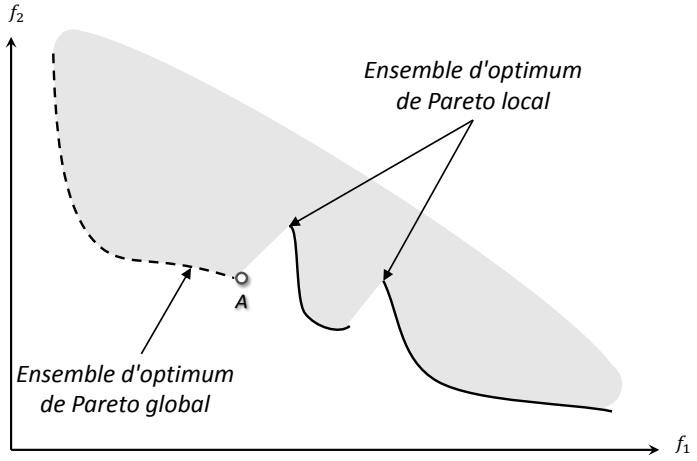


FIGURE 2.19 – Exemple d'ensembles de solutions localement optimales et de solutions globalement optimales dans l'espace objectif.

exemple, un classement des objectifs).

L'ensemble de toutes les solutions optimales de Pareto est appelé *ensemble d'optimum de Pareto*; Les vecteurs objectifs correspondants forment *le front ou la surface d'optimum de Pareto* [81].

Définition 5

(Ensembles et fronts non dominés) Soit $A \subseteq X_f$. La fonction $p(A)$ donne l'ensemble des vecteurs de décision non dominés en A :

$$P(A) = \{a \in A | a \text{ est non-dominé sur } A\} \tag{2.7}$$

$p(A)$ est l'ensemble non-dominé sur A, l'ensemble correspondant de vecteurs objectifs $f(p(A))$ est le front non-dominé de A. En outre, $X_p = p(X_f)$ est nommé l'ensemble d'optimum de Pareto et $Y_p = f(X_p)$ est désigné comme le front d'optimum de Pareto.

L'ensemble d'optimum de Pareto comprend les solutions globalement optimales. Cependant, il peut y avoir des optima locaux qui constituent un ensemble non-dominé dans un certain voisinage. Ceci correspond aux concepts des ensembles d'optimum de Pareto globaux et locaux [81] :

Définition 6

Considérons un ensemble de vecteurs de décision $A \subseteq X_f$. L'ensemble A est nommé comme un ensemble d'optimum de Pareto local si et seulement si :

$$\forall a \in A : \nexists x \in X_f : x \prec a \wedge ||x - a|| < \epsilon \wedge ||f(x) - f(a)|| < \delta \tag{2.8}$$

Où $||.||$ est une métrique de distance correspondante et $\epsilon > 0$, $\delta > 0$. L'ensemble A est appelé ensemble d'optimum de Pareto global si et seulement si :

$$\forall a \in A : \nexists x \in X_f : x \prec a \tag{2.9}$$

La différence entre optimum local et global est visualisée dans la Figure 2.19. La ligne pointillée constitue un front d'optimum de Pareto global, alors que la ligne continue représente un front d'optimum de Pareto local. Enfin, notez qu'un ensemble d'optimum de Pareto global ne contient pas nécessairement toutes les solutions d'optimum de Pareto et que tout ensemble d'optimum de Pareto global est aussi un ensemble d'optimum de Pareto local.

2.3.2 Méthodes de résolution

En résolvant un $\mathcal{PMO}$, deux types de problèmes peuvent être identifiés [42] : (1) la *recherche* et (1) la *prise de décision*. Le premier aspect concerne le processus d'optimisation dans lequel l'ensemble des solutions réalisable est échantillonné pour trouver les solutions d'optimum de Pareto. Le second aspect traite le problème de sélection d'une solution de compromis appropriée à partir de l'ensemble d'optimum de Pareto. Un décideur humain est nécessaire pour faire le compromis, souvent difficile, entre les objectifs contradictoires.

Suivant la combinaison du processus d'optimisation et la prise de décision, les méthodes d'optimisation multi-objectifs peuvent être classées en trois catégories [127, 178, 72] :

- **Méthodes a priori.** Prise de décision *avant* la recherche : Les fonctions objectifs du $\mathcal{PMO}$ sont regroupées en une seule objectif qui inclut implicitement les informations de compromis fournies par le décideur.
- **Méthodes interactives.** Prise de décision *durant* la recherche : Le décideur peut donner ses préférences de compromis au cours du processus d'optimisation d'une manière interactive. Après chaque étape d'optimisation, un certain nombre de compromis alternatifs sont présentés aux décideur, sur la base desquels il spécifie une autre information de préférence, qui guide respectivement la recherche.
- **Méthodes a posteriori.** Prise de décision *après* la recherche : L'optimisation est effectuée sans aucune préférence donnée. Le résultat du processus de recherche est un ensemble de solutions candidates (d'optimum de Pareto) dont le choix final de la meilleure solution est fait par le décideur.

Les méthodes de différentes classes ont leurs avantages et inconvénients et, pour cette raison, différentes approches sont nécessaires pour résoudre un $\mathcal{PMO}$. Signalons que la classification donnée ici n'est pas complète ou absolue. Le chevauchement et les combinaisons d'approches sont possibles et certaines méthodes peuvent appartenir à plusieurs classes en fonction de différentes interprétations [42]. Cependant, nous allons utiliser la classification proposée par *Deb* [81], où les méthodes sont classées en deux catégories : (1) méthodes *classiques* et (2) méthodes *évolutionnaires*. Dans ce qui suite, nous présentons chacune des catégories.

2.3.2.1 Méthodes classiques

Les méthodes classiques pour résoudre le $\mathcal{PMO}$ regroupent les objectifs en une seule fonction objectif paramétrée. Il existe une panoplie de méthodes appartenant à cette classe, nous citons parmi elles : la méthode de pondération, la méthode ϵ-contrainte, la méthode programmation par but, etc [72, 42, 81].

2.3.2.1.1 Méthode de pondération

Dans la méthode de pondération ou somme pondérée, le $\mathcal{PMO}$ original est convertie en un problème $\mathcal{OMO}$ en formant une combinaison linéaire des objectifs [99, 291] :

$$\begin{aligned} &\text{minimiser} \quad y = f(x) = \sum_{i=1}^{k} \omega_i \cdot f_i(x) \\ &\text{avec} \quad x \in X_f \end{aligned} \tag{2.10}$$

Les ω_i sont appelés poids et sont normalisés tels que $\sum \omega_i = 1$. Résoudre le problème d'optimisation ci-dessus pour un certain nombre de combinaisons de poids différentes donne un ensemble de solutions. A condition que tous les poids soient positifs, cette méthode produira uniquement des solutions d'optimum de Pareto. La méthode de pondération peut être utilisée comme une méthode a posteriori de sorte que différents poids sont utilisés pour générer différentes solutions d'optimum de Pareto, et ensuite le décideur est invité à sélectionner la plus satisfaisante d'entre elles. Alternativement, le décideur peut être invité à spécifier les poids, dans ce cas la méthode est utilisée comme une méthode a priori.

Exemple 9. *La Figure 2.20 illustre le fonctionnement de la méthode de pondération. Nous considérons le problème à deux objectifs. Avec deux objectifs, il existe deux poids ω_1 et ω_2, mais un seul est indépendant. Connaissant un, l'autre peut être calculé par une simple soustraction. Connaissant les poids, nous pouvons également calculer la fonction composite $f(x)$. Ses surfaces d'hyperplan (une droite) peuvent alors être visualisées dans l'espace objectif, comme le montrent les lignes « a », « b », « c » et « d » dans la Figure 2.20. Puisque notre problème nécessite une minimisation de $f(x)$, la tâche consiste à trouver l'hyperplan avec la valeur $f(x)$ minimale. Cela se produit avec l'hyperplan qui est tangent à l'espace de recherche et se trouve également dans le coin inférieur gauche de cet espace. Sur la figure, cette ligne est marquée comme « d ». Le point tangent A est la solution minimale de $f(x)$ et est par conséquent la solution d'optimum de Pareto.*

L'avantage de cette méthode est sa facilité d'implémentation et le fait qu'elle puisse être utilisée avec les méthodes et techniques définis pour $\mathcal{OMO}$. Cependant, comme mentionné précédemment, il est important dans $\mathcal{PMO}$ que les solutions d'optimum de Pareto soient générées et que toute solution puisse être trouvée. À cet égard, la méthode de pondération présente un sérieux défaut. La solution d'optimum de Pareto peut être trouvée en modifiant les poids seulement si le problème est convexe. Ainsi, il peut arriver que certaines solutions d'optimum de Pareto pour des problèmes non convexes ne puissent être trouvées, peu importe la façon dont les poids sont sélectionnés [42]. C'est le cas du point A dans la Figure 2.18a.

2.3.2.1.2 Méthode ϵ-contrainte

Afin d'atténuer les difficultés rencontrées par la méthode de somme pondérée dans la résolution de problèmes ayant des espaces objectifs non convexes, la méthode de ϵ-contrainte est utilisée. L'idée de cette méthode est la transformation des $k-1$ fonctions objectives en contraintes. La fonction objective

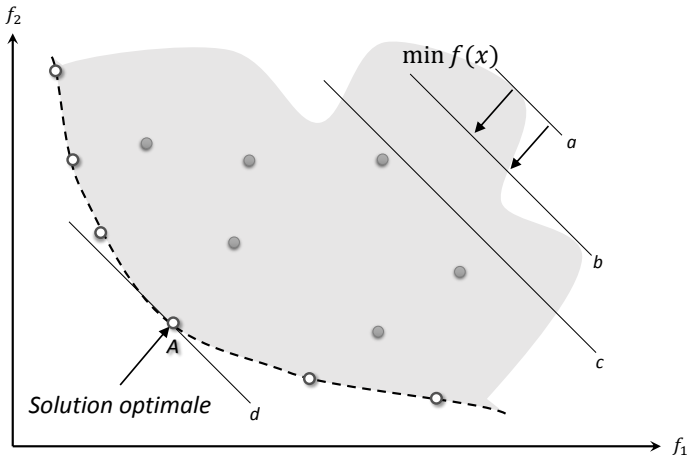


FIGURE 2.20 – Exemple de la méthode de pondération.

restante, qui peut être choisi arbitrairement, est la fonction objective du OMO résultant [113, 270] :

$$\begin{array}{ll} \text{minimiser} & y = f(x) = f_h(x) \\ \text{avec} & e_i(x) = f_i(x) \leqslant \epsilon_i \quad (1 \leqslant i \leqslant k, i \neq h) \\ \text{et} & x \in X_f \end{array} \tag{2.11}$$

Les limites supérieures, ϵ_i, sont les paramètres à varier afin de trouver des solutions d'optimum de Pareto. En pratique, il peut être difficile de spécifier les limites supérieures pour avoir des solutions aux problème OMO, c'est-à-dire que la région réalisable ne soit pas vide. Cette difficulté est soulignée lorsque le nombre de fonctions objectives augmente [42]. Cependant, il est possible d'utiliser la méthode de manière a priori et demander au décideur de spécifier la fonction à optimiser et les limites supérieures.

2.3.2.1.3 Méthode programmation par but

L'idée principale de la programmation par but est de trouver des solutions qui atteignent une cible prédéfinie pour une ou plusieurs fonctions objectives [51]. Dans cette méthode, le décideur est invité à spécifier les niveaux cibles $z_i (i = 1, \cdots, k)$ pour les fonctions objectives. Ensuite, les écarts par rapport à ces niveaux cibles sont minimisés. Une fonction objective conjointement avec un niveau de cible est appelée un *but*. Pour les problèmes de minimisation, les buts sont de la forme $f_i(x) \leqslant z_i$ et les niveaux de cibles sont supposés être sélectionnés. Après la formations des buts, les *écarts* $\delta_i = max[0, f_i(x) - z_i]$ des valeurs de la fonction objective sont minimisés.

La méthode comporte plusieurs variantes. Dans l'approche de *programmation par but pondérés* [52], la somme pondérée des écarts est minimisée. Cela signifie qu'en plus des niveaux de cibles, le décideur

doit spécifier des poids positifs. Puis nous résolvons le problème :

$$\begin{aligned} &\text{minimiser} && \sum_{i=1}^{k} \omega_i \delta_i \\ &\text{avec} && f_i(x) - \delta_i \leqslant z_i \text{ et } \delta_i \geqslant 0 \\ &\text{et} && x \in X_f \end{aligned} \tag{2.12}$$

D'autre part, dans l'approche de *programmation par but lexicographiques*, le décideur doit spécifier un ordre lexicographique pour les buts en plus des niveaux de cible. Après l'ordonnancement lexicographique, le problème des déviations en tant que fonctions objectives est résolu lexicographiquement [72]. Mentionnons également une approche de *programmation par but min-max* où le maximum d'écarts est minimisé [81].

2.3.2.2 Méthodes évolutionnaires

Les algorithmes évolutionnaires (AE) sont des méthodes de recherche inspirées de la nature pour résoudre des problèmes complexes avec un grand espace de recherche. Certaines caractéristiques de cette technique, tel que la possibilité de trouver des solutions optimales multiples dans une seule simulation, les rendent populaires et motivent les chercheurs à les utiliser dans divers domaines d'applications comme alternative aux méthodes classiques décrites précédemment.

Il existe plusieurs familles historiques d'algorithme évolutionnaires qui se sont développées de façon indépendante : la programmation évolutionnaire, les algorithmes génétiques, les stratégies d'évolutions et la programmation génétique. Aujourd'hui, ces terminologies sont considérées comme sous-domaines des AE [42].

Cette section vise à fournir un aperçu d'un certain nombre d'AE bien connus, conçus pour traiter des problèmes d'optimisation multi-objectifs, par la présentation du vocabulaire et le principe général de leurs fonctionnements.

2.3.2.2.1 Principe de fonctionnement

En général, un AE se caractérise par trois faits :

1. un ensemble de solutions candidates est maintenu, ce qui
2. subit un processus de sélection, et qui
3. est manipulé par les opérateurs génétiques, généralement le croisement et la mutation.

Par analogie avec l'évolution naturelle, les solutions candidates sont appelées *individus* et l'ensemble des solutions candidates s'appelle la *population*. Chaque individu représente une solution possible, et il est codé sur la base d'une structure appropriée. Sans perte de généralité, on suppose que cette structure est un vecteur, par exemple un vecteur de bits ou un vecteur de valeur réelle. L'ensemble de tous les vecteurs possibles constitue *l'espace des individus* I. Dans cette terminologie, la population est un ensemble de vecteurs $i \in I$.

Dans le processus de *sélection*, qui peut être stochastique ou complètement déterministe, les individus de faible qualité sont retirés de la population, tandis que les individus de haute qualité sont reproduits. L'objectif est de concentrer la recherche sur des portions particulières de l'espace de recherche et d'augmenter la qualité moyenne au sein de la population. La qualité d'un individu par rapport à la tâche d'optimisation est représentée par une valeur scalaire, qui est le *fitness*.

Le *croisement* et la *mutation* visent à générer de nouvelles solutions dans l'espace de recherche par la variation de celles existantes. L'opérateur de croisement prend un certain nombre de parents et crée un certain nombre d'enfants en recombinant les parents. Pour simuler la nature stochastique de l'évolution, une probabilité de croisement est associée à cet opérateur. En revanche, l'opérateur de mutation modifie les individus en changeant de petites parties dans les vecteurs associés selon un taux de mutation donné. L'opérateur de *l'élitisme* assure que la nouvelle population garde que les meilleures solutions de la population actuelle.

Sur la base des concepts ci-dessus, l'évolution naturelle est simulée par un processus itératif de calcul. Premièrement, une population initiale est créée au hasard ou suivant un schéma. Ensuite, une boucle (*génération*) comprenant les étapes : évaluation (affectation de fitness), sélection, croisement et/ou mutation est exécutée un certain nombre de fois fixé. À la fin, le meilleur individu(s) dans la population finale, ou trouvé pendant tout le processus d'évolution, est le résultat de l'AE.

Algorithme 2.1 Algorithme général d'AE

Entrée : T ▷ nombre maximal de générations
Sortie : A ▷ ensemble non-dominé

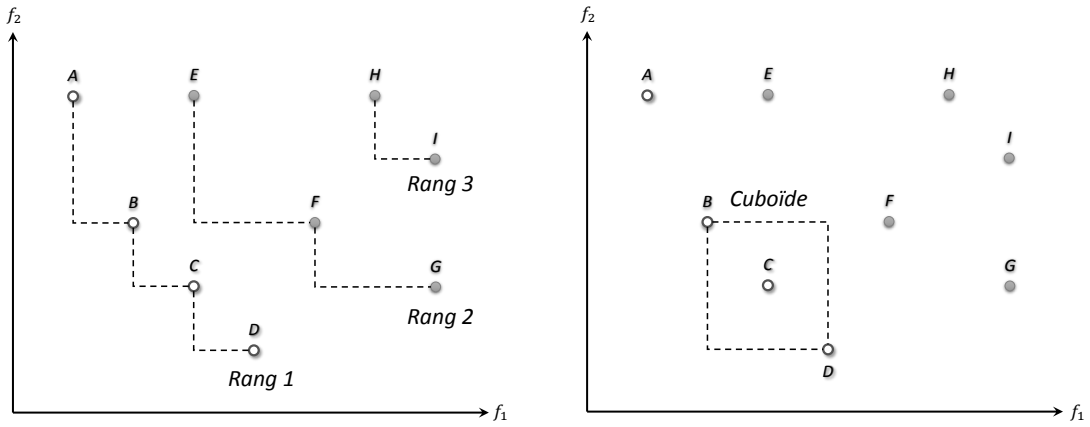
```
1 : t = 0;
2 : Initialisation(P_t);
3 : répéter
4 :     Evaluation(P_t);
5 :     P'_t = Selection(P_t);
6 :     P''_t = Croisement(P'_t);
7 :     P_{t+1} = Mutation(P''_t);
8 :     t = t + 1;
9 : jusqu'à Termination(P_t, P_{t+1});
10 : retourner A;
```

Dans l'Algorithme 2.1, la procédure de base d'un AE est donnée. La population P à une certaine génération t est représentée par le symbole P_t.

Les AE multi-objectifs (AEMO) peuvent être classés généralement en trois approches basées sur : *la dominance*, *la décomposition* et *les indicateurs* [271]. Dans les sections suivantes, nous les détaillons et nous présentons un algorithme représentatif de chaque approche.

2.3.2.2.2 Approches basées sur la dominance

Les méthodes de cette catégorie se base sur le concept de la dominance de Pareto déjà introduit. Ils sont les approches les plus populaires dans la littérature, et ils existent depuis les deux dernières décennies. Les AEMO de cette catégorie peuvent être classés en deux groupes principaux : des algorithmes *non-élitistes* et des algorithmes *élitistes*. Comme décrit précédemment, l'élitisme est un mécanisme pour préserver les



(a) La méthode de *ranking*.

(b) La méthode de la distance de *crowding*.

FIGURE 2.21 – Exemples des méthodes de NSGA-II.

bons individus de la génération actuelle en les sauvant et en les transmettant à la génération suivante, ce qui accélère la convergence de la population vers le front de Pareto. Le premier groupe inclue : *Non-Dominated Sorting Genetic Algorithm* (NSGA) [244], *Niched-Pareto Genetic Algorithm* (NPGA) [121], *Multi-Objective Genetic Algorithm* (MOGA) [93], etc. Le deuxième groupe comprend : *Elitist Non-Dominated Sorting Genetic Algorithm* (NSGA-II) [82], *Strength Pareto Evolutionary Algorithm* (SPEA-II) [302] et *Pareto Envelope-based Selection Algorithm* (PESA-II) [75], et plusieurs autres algorithmes.

2.3.2.2.2 NSGA-II

Un exemple classique de cette catégorie est NSGA-II par Deb *et al* [82]. Cet algorithme génétique utilise une méthode de *ranking* qui attribue aux solutions non-dominées de la population courante le meilleur score. Par la suite, les solutions qui ne sont dominées que par les solutions potentiellement efficaces, reçoivent le second meilleur score et ainsi de suite. De cette manière, la population est organisée en couches où chaque couche contient des solutions non comparables entre elles. Le ranking d'une population est illustré dans la Figure 2.21a. Dans cet exemple, les solutions A, B, C, D reçoivent le rang 1 car elles ne sont dominées par aucune solution. Les solutions E, F et G ont le rang 2 car elles ne sont dominées que par des solutions de rang 1. Enfin, H et I sont affectées au rang 3. NSGA-II utilise aussi une seconde mesure, la distance de *crowding*, visant à améliorer la diversité de la population du premier rang de l'espace des objectifs. La distance de crowding est une mesure permettant de déterminer le surpeuplement des individus du même front. Il s'agit d'une estimation de la densité de solutions qui entourent un individu. Comme le montre la Figure 2.21b, la distance de crowding d_i pour l'individu C est définie comme un demi-périmètre de cuboïde formé par les individus voisins les plus proches de C dans l'espace objectif (les points B et D).

2.3.2.2.3 Approches basées sur la décomposition

Les méthodes basées sur la décomposition, connues aussi sous le nom d'approches d'agrégation pondérée, où un $\mathcal{PMO}$ est décomposé en un certain nombre de problèmes d'optimisation objectifs simples

en utilisant un certain nombre de combinaisons de poids. Ces poids sont générés soit de façon aléatoire [134], soit dynamiquement et continuellement modifiées [136], soit prédéfinies et uniformément réparties [185]. Parmi les algorithmes les plus courant : *Multiple Single Objective Pareto Sampling* (MSOPS) [125], *Multiobjective Evolutionary Algorithm based on Decomposition* (MOEA/D) [296] et *Repeated Single Objective* (RSO) [124].

2.3.2.2.3 MOEA/D

Cette variante récente d'algorithmes de décomposition a reçu une attention croissante en raison de sa simplicité de calcul et sa performance dans la recherche des solutions. Le principe de cet algorithme est le suivant : Premièrement, il faut définir un P vecteurs de poids uniformément distribués dans l'espace objectif. Ces vecteurs définissent des directions pour chercher les solutions. En effet, chaque vecteur de poids est utilisé pour définir un sous-problème mono-objectif en utilisant des fonctions scalaires comme la méthode de pondération ou la méthode de Tchebycheff. L'algorithme maintient également une population de solutions de taille P. Une solution est alors attachée à chaque fonction scalaire.

Le but de MOEA/D est de trouver les meilleures solutions par rapport à chacun des vecteurs de direction à l'aide de *la notion de voisinage*. Étant donné un vecteur de direction λ_i et le sous-problème scalaire sous-jacent pour tout $i \in \{1, \cdots, P\}$. L'algorithme définit l'ensemble $B(i)$ des voisins de i comme étant les sous-problèmes correspondant aux vecteurs de direction les plus proches de λ. L'algorithme parcourt les sous-problèmes de façon itérative, et pour chacun, deux solutions parmi les voisins sont sélectionnées. Ces solutions permettent alors de produire une nouvelle solution, notée y', en utilisant des opérateurs de variations (mutation, croisement). La nouvelle solution y' est comparée aux solutions des voisins. Si y' permet d'améliorer la solution d'un voisin alors elle remplace cette solution et ainsi de suite pour tous les voisins. Ce mécanisme est répété pour toutes les directions jusqu'à ce qu'une condition d'arrêt soit satisfaite. A la fin, l'ensemble des solutions calculées par rapport à toutes ces directions est alors retourné en sortie.

2.3.2.2.4 Approches basées sur les indicateurs

Une idée différente est d'attribuer un fitness aux individus basé sur un indicateur de performance pour évaluer la qualité, la convergence et la diversité des individus de la population. L'indicateur de *l'hypervolume* [303], qui mesure le volume de la portion faiblement dominée par un ensemble de point dans l'espace objectif, a été largement adopté par les différents AEMO basés sur les indicateurs de performance. Cela est dû à sa précision et la diversité des ensembles de solutions non-dominées produits [271]. Parmi les algorithmes proposés dans cette catégorie : *Indicator-based Evolutionary Algorithm* (IBEA) [301] qui utilise une métrique de performance binaire comme indicateur, *S-metric Selection-EMOA* (SMS-EMOA) [29] qui utilise l'indicateur d'hypervolume et plus récemment le *Hypervolume Estimation Algorithm for Multiobjective Optimization* (HypE) [18] qui adopte la simulation Monte Carlo pour approximer la valeur exacte de l'hypervolume. Le principal défi de cette classe d'AEMO est la complexité élevée pour calculer l'indicateur de performance, en particulier lorsque le nombre d'objectifs est élevé [30].

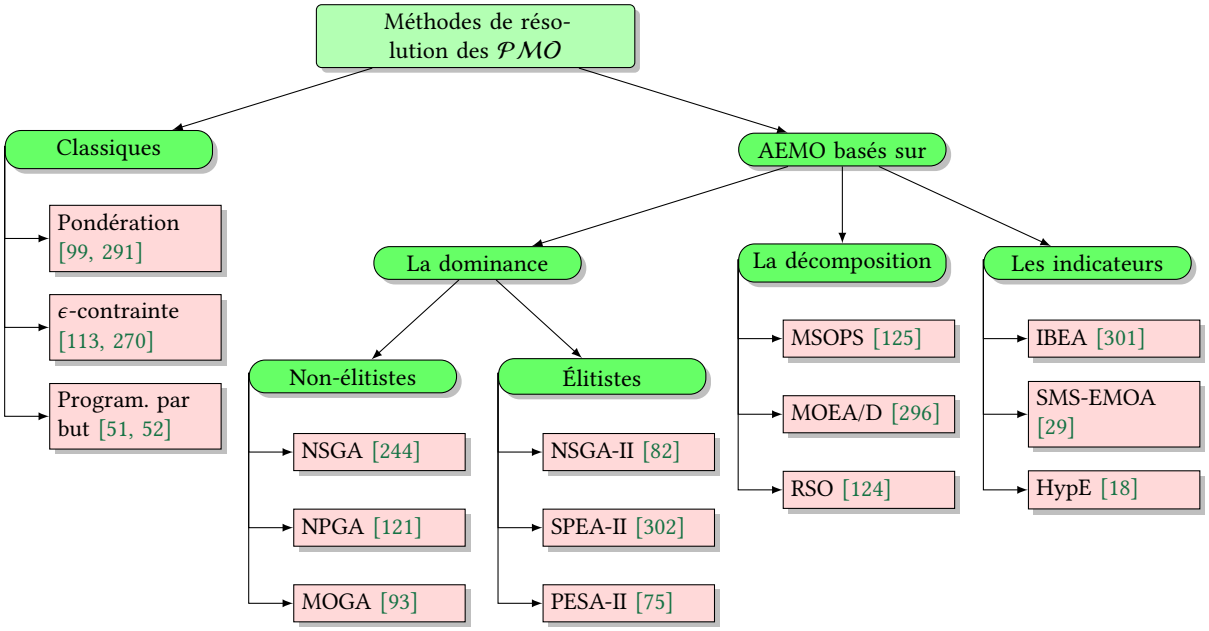


FIGURE 2.22 – Classification des méthodes de résolutions d'un $\mathcal{PMO}$.

2.3.2.2.4 IBEA

L'idée principale de IBEA [301] est de formaliser les préférences entre individus en utilisant un indicateur de performance binaire de qualité I arbitrairement choisi. Ensuite, chaque individu x^1 est attribué une valeur de fitness $F(x^1)$ correspondant à la mesure de la « perte en qualité » si cet individu avait été retiré de la population P.

$$F'(x^1) = \sum_{x^2 \in P \setminus \{x^1\}} I(\{x^1\}, \{x^2\}) \tag{2.13}$$

Les auteurs proposent une autre formule modifiée qui amplifie l'influence des solutions non-dominées sur les solutions dominées. En d'autre terme, donner le plus grand score possible à une solution selon sa contribution par rapport aux autres solutions de la population.

$$F(x^1) = \sum_{x^2 \in P \setminus \{x^1\}} -e^{-I(\{x^1\}, \{x^2\})/k} \tag{2.14}$$

Où $k > 0$ est un facteur d'échelle qui dépend de l'indicateur I utilisé (par exemple, l'hypervolume ou epsilon) et du problème à optimiser.

La Figure 2.22 montre les différentes méthodes de résolution d'un $\mathcal{PMO}$ discutées dans cette section.

2.3.3 Optimisation multi-objectifs dans les bases de données

Nous passons maintenant sur l'application des techniques d'optimisation multi-objectifs décrites dans la section précédente dans le contexte des bases de données. En étudiant les travaux de littérature, nous

trouvons que les techniques d'optimisation multi-objectifs ont été largement employées pour résoudre des problèmes diverses dans les base de données sur plusieurs niveaux : le Cloud Computing (ex : minimisation des coûts monétaire) [147, 85, 118], les bases de données temps réel (ex : ordonnancement de requêtes) [255, 297], la conception physique des bases de données (ex : le problème de sélection d'index [44, 45], la fragmentation [23, 24, 21], les vues matérialisées [160, 252, 102]), le traitement de requête [198, 261, 36], etc.

Cependant, nous allons focaliser notre étude sur les vues matérialisées et la phase de traitement des requêtes par un SGBD.

2.3.3.1 $\mathcal{PSV}$ multi-objectifs

Le $\mathcal{PSV}$ discuté dans le Paragraphe 2.2.1.4.1 (ou une autre structure d'optimisation en général) peut être reformulé pour sélectionner un ensemble de vues avec plusieurs objectifs à optimiser en même temps. En effet, dans les travaux qui se basent sur l'optimisation mono-objectif, seul le coût de traitement de requête est considéré comme fonction objectif. Cependant, le coût de stockage ou de maintenance ont été considérés comme des contraintes à satisfaire.

En analysant les travaux de littérature, la première formalisation de $\mathcal{PSV}$ multi-objectif a été proposée dans [160], dans lequel le temps de réponse de requête et le temps de maintenance doivent être minimisés simultanément sous la contrainte d'espace de stockage. La structure de représentation de données employée est la technique de cube treillis. Dans ce but, deux AEMO non-élitistes, MOGA et NPGA ont été adoptés pour résoudre le problème de sélection de vue. Pour faire face aux contraintes, l'auteur a proposé deux méthodes. La première méthode intègre la contrainte dans l'objectif et définit la notation de dominance de telle sorte qu'un individu infaisable est toujours dominé par un individu réalisable. La seconde permet à un individu infaisable d'être crée et d'utiliser une fonction de réparation pour le convertir en un faisable. Les résultats d'expérimentations sur des ensembles de données réelles et synthétiques montrent que les AEMO sont très compétitifs par rapport aux algorithmes de glouton classique.

Les auteurs de [252] ont proposé un algorithme génétique basé sur le poids (WBGA) pour résoudre le problème de sélection de vue. L'objectif est de minimiser la somme pondérée du temps de réponse de requêtes et du temps de maintenance en fonction de la contrainte d'espace de stockage. Le cube treillis été utilisé comme structure de données. Le WBGA multiplie chaque objectif i à un facteur de poids correspondant w_i pour calculer une valeur de fitness. Contrairement à l'approche de la somme pondérée classique avec des poids fixes, la WBGA utilise le mécanisme d'optimisation génétique pour rechercher des solutions avec des poids différents en parallèle. Par conséquent, la population converge vers un ensemble de solutions optimales avec différents vecteurs de poids en un seul passage.

Dans [102], les auteurs ont mis en œuvre un algorithme à évolution différentielle (ED) multi-objectifs pour résoudre le $\mathcal{PSV}$ dans un entrepôt de données. Le temps de réponse et le coût de maintenance sont considérés comme fonctions objectifs, tandis que l'espace de stockage comme contrainte. Les solutions candidates sont représentées par la structure de MVPP. L'algorithme ED est une méthode méta-heuristiques stochastiques d'optimisation des espaces continus. La technique s'avère appropriée pour sélectionner des solutions représentatives à partir d'un grand nombre de solutions non-dominantes du $\mathcal{PSV}$.

2.3.3.2 Traitement de requêtes multi-objectifs

Il existe souvent d'autres indicateurs de coûts en plus du temps d'exécution qui sont pertinents pour comparer les plans de requêtes, tel que le coût d'argent dans un scénario de Cloud Computing par exemple. Les travaux qui implémentent les techniques d'optimisation multi-objectifs dans le contexte du traitement de requêtes peuvent être divisés en deux catégories. La première catégorie considère le niveau d'optimisation de requêtes, afin de produire des plans d'exécutions respectant les objectifs prédéfinis. D'autre part, la deuxième catégorie considère le niveaux données ou requêtage, en cherchant dans les données et produisant des résultats suivant les objectifs. Nous citons les travaux de chaque catégorie dans les sections suivantes.

2.3.3.2.1 Optimisation de requêtes

L'optimisation de requêtes multi-objectifs modélise le coût d'un plan de requête comme un vecteur de coût où chaque composante vectorielle représente le coût selon une métrique de coût différente. L'optimisation classique des requêtes peut être considérée comme un cas particulier d'optimisation de requête multi-objectif où la dimension de l'espace de coût est égale à un (c'est-à-dire le nombre de composantes de vecteur de coût).

Différentes métriques de coût peuvent entrer en conflit. Par conséquent, l'objectif de l'optimisation est de trouver un plan de requête qui réalise le meilleur compromis entre les différentes métriques de coûts. Le meilleur compromis dépend des préférences de l'utilisateur fournies en entrée à l'optimiseur (par exemple, un utilisateur peut définir des pondérations entre différentes métriques de coûts pour exprimer une importance relative ou définir des bornes supérieurs/inférieurs de coût).

Le travail de [198] présente un algorithme d'approximation multi-objectifs pour exécuter des opérateurs d'une requête vers des sites dans le cas d'une base de données distribuée. Il suppose que chaque site soumet une « offre » pour la requête, en spécifiant un délai pour la délivrance du résultat et un coût monétaire associé. L'optimiseur de requêtes compare alors ces offres à un compris entre le coût/délai fourni par l'utilisateur (une fonction $u(d)$, qui spécifie pour chaque valeur d du délai, le montant d'argent que l'utilisateur est prêt à payer afin de recevoir les résultats de la requête dans le temps d) et tente de déterminer la combinaison des exécutions de la sous-requête qui maximise les préférences de l'utilisateur. Les auteurs proposent un algorithme glouton et une preuve mathématique pour résoudre ce problème.

Les auteurs de [89] ont proposé un framework de traitement des requêtes pour l'intégration des données. Ils considèrent la couverture, la densité et la latence des sources de donnée comme des fonctions objectives, et ils ont développé un modèle de coût pour estimer chacun d'entre eux. Dans la phase d'optimisation, ils ont utilisé une méthode de pondération pour combiner les fonctions objectives en une seule fonction agrégée, en utilisant des paramètres de poids permettant à l'utilisateur de changer l'importance relative associée aux différents objectifs. Les expérimentations montrent que le modèle de coût est capable d'estimer efficacement chaque type d'objectif, et la procédure d'optimisation est capable de faire des compromis flexibles entre les objectifs lors du traitement des requêtes.

Les travaux récents de Trummer et Koch traitent le problème d'optimisation de requêtes ayant plusieurs objectifs (*many-objective*). Grâce à l'avancement de la technologie de matériel informatique, les auteurs ont proposé de revisiter les suppositions faites par les chercheurs lors du développement

des méthode d'optimisation de requêtes les premiers jours, afin d'exploiter les capacités logicielles et matérielles actuelles. Ils ont présenté le problème d'optimisation de requête multi-objectif où les plans de requêtes sont comparés selon plusieurs métriques de coût, comme par exemple le temps d'exécution, les frais monétaires, les mesures de consommation de ressources système (le nombre de noyaux utilisés, la quantité d'espace mémoire consommée). Les techniques utilisées pour résoudre ce problème inclues : des schémas d'approximation qui permettent de relaxer progressivement l'optimalité d'un plan pour accélérer le processus d'optimisation [260], un algorithme incrémental qui divise l'optimisation en plusieurs petites étapes progressives, permettant aux utilisateurs de guider l'algorithme après chaque étape, ce qui rend de l'optimisation des requêtes un processus interactif [259], une optimisation de requêtes paramétrée, avec l'introduction d'une étape de pré-traitement si les requêtes correspondent à des modèles de requêtes connus à l'avance [261], une approche de décomposition qui permet de paralléliser l'optimisation de requêtes classique sur des clusters de grande taille avec des centaines de nœuds [263], l'utilisation d'un algorithme aléatoire (un Hill Climbing modifié) pour l'optimisation de requêtes multi-objectif qui est capable de traiter des requêtes avec des centaines de jointures [258], la transformation du problème d'optimisation de requêtes en un programme linéaire en nombres entiers mixtes qui permet d'appliquer des solveurs de la programmation en nombres entiers pour traiter des espaces de recherche plus grands [264], et enfin, ils ont présenté des résultats d'expérimentations pour résoudre le problème d'optimisation de requêtes sur l'ordinateur quantique D-Wave 2X avec plus de 1000 qubits (100 millions de fois plus rapide qu'un ordinateur classique) [262].

2.3.3.2.2 L'opérateur Skyline

L'opérateur Skyline est utilisé dans une requête et effectue un filtrage des résultats à partir d'une base de données afin qu'il ne conserve que les objets qui ne sont pas dominés par les autres.

Cet opérateur est une extension de SQL proposée par Börzsönyi *et al.* [36]. Un exemple classique d'application de l'opérateur Skyline implique de choisir un hôtel pour des vacances. L'utilisateur veut que l'hôtel soit à la fois bon marché et proche de la plage. Cependant, les hôtels qui sont près de la plage peuvent également être coûteux. Dans ce cas, l'opérateur Skyline ne présenterait que les hôtels qui ne sont pas pires que tout autre hôtel à la fois dans le prix et la distance. Dans le cas d'un ensemble de données composé d'objets multidimensionnels, un objet domine un autre objet s'il est aussi bon dans toutes les dimensions et mieux dans au moins une dimension [256].

Exemple 10. *Börzsönyi et al. ont proposé la syntaxe suivante pour l'opérateur Skyline [36] :*

```
SELECT ... FROM ... WHERE ...
GROUP BY ... HAVING ...
SKYLINE OF [DISTINCT] d1 [MIN | MAX | DIFF],
                ..., dm [MIN | MAX | DIFF]
ORDER BY ...
```

Où $d_1, \cdots d_m$ désignent les dimensions de Skyline et `MIN`, `MAX` *et* `DIFF` *spécifient si la valeur de cette dimension doit être minimisée, maximisée ou simplement la différence.*

L'opérateur Skyline peut être implémenté directement dans SQL en utilisant des instructions SQL courantes, mais cela s'est avéré être très lent [36]. D'autres algorithmes qui utilisent la méthode diviser

pour régner, les indexes [36], MapReduce [184] et calcul généraliste sur les cartes graphiques [34] ont été proposés. Les requêtes Skyline sur les flux de données ont été étudiées dans le contexte du traitement parallèle des requêtes sur des systèmes multi-cœurs, en raison de leur large utilisation dans les problèmes de prise de décision en temps réel et les analyses de flux de données [80]. Une étude récente et détaillée sur les différentes techniques d'implémentation du l'opérateur Skyline peut être trouvée dans [256].

2.3.4 Bilan et discussion

Nous avons présenté dans cette section le problème de d'optimisation multi-objectifs, sa formulation et ses méthodes de résolution. Nous avons détaillé en particulier la méthode de la somme pondérée et les algorithmes NSGA-II, MOEA/D et IBEA car ils sont les plus populaires. De plus, nous avons cité les principaux travaux qui appliquent ce concept dans les bases de données, en particulier sur le problème de sélection de vues et le traitement de requêtes.

Dans notre étude, l'intégration du nouveau besoin non-fonctionnel qui est l'énergie dans la base de données, est considéré comme une fonction objective à optimiser. De ce fait, nous proposons de reformuler le problème de sélection des structures d'optimisations et le traitement de requête en un problème multi-objectifs, où il faut optimiser deux objectifs : le temps d'exécution de la charge de requêtes et la consommation d'énergie, avec la prise en compte des contraintes telles que le coût de maintenance et l'espace de stockage. Pour résoudre ce nouveau problème, il faut (1) proposer un modèle de coût qui permet de calculer la fonction objectif d'énergie, (2) adopter une méthode de prise de décision pour avoir un compromis entre la performance et l'énergie (une méthode a priori, interactive ou a posteriori), et (3) choisir la bonne configuration d'algorithme et des paramètres pour résoudre le $\mathcal{PMO}$.

2.4 Conclusion

Afin d'intégrer l'énergie dans une base de données, il faut, en premier lieu, l'étudier pour identifier ses composantes qui peuvent avoir un impact sur la consommation énergétique du système. Dans ce chapitre nous avons décrit l'ensemble des phases traditionnelles de cycle de vie de conception d'une base de données : l'analyse de besoins, la modélisation conceptuelle, logique, physique et le déploiement. Nous avons identifié l'opportunité d'intégrer l'énergie dans la phase de conception physique, plus précisément, dans le choix d'une structure d'optimisation. Nous nous sommes focalisés sur la technique redondante qui est les vues matérialisées, et nous avons proposé de reformuler le problème de sélection de vues en un problème multi-objectifs ou il faut minimiser le temps de réponse et la consommation d'énergie en respectant certaines contraintes. Nous avons également cité les travaux d'état de l'art qui traitement le $\mathcal{PSV}$ en version mono et multi-objectifs.

En seconde lieu, nous avons étudier la phase de traitement de requêtes dans un SGBD relationnel, qui inclue : la phase d'analyse, la transformation, la génération de plans et optimisation et enfin l'exécution. Cette étude nous a permis d'identifier les paramètres clés qu'il faut prendre en considération lors de conception d'un optimiseur de requêtes éco-énergétique. Ces paramètres incluent un modèle de coût pour estimer la consommation d'énergie d'une requête SQL et une technique d'intégration de ce modèle de coût dans la phase de génération de plans. En parallèle à la description des composantes d'un moteur de traitement de requête, nous avons passé en revue les principaux techniques et travaux proposés dans

l'état de l'art.

En troisième lieu, nous avons focalisé notre attention sur les problèmes d'optimisation multi-objectifs. Car notre nouveau problème d'incorporation de l'énergie nécessite une reformulation multi-objectifs, afin de proposer aux utilisateurs et administrateurs de base de données un ensemble de solutions avec des compromis variés. Nous avons présenté la formulation des $\mathcal{PMO}$, ses principales méthodes de résolutions : classiques et évolutionnaires, et la démarche à suivre pour intégrer ces méthodes par rapport à l'utilisateur (méthode a priori, interactive ou a posteriori).

Dans le prochain chapitre, nous allons présenter le concept d'énergie, ses propriétés, et un état de l'art sur les travaux de minimisation d'énergie dans les systèmes informatiques.

Chapitre

3

L'énergie dans les systèmes informatiques

« With out passion you don't have energy, with out energy you have nothing. »
— Donald Trump

Sommaire

3.1 Introduction

La demande croissante de traitement de l'information a entraîné la demande de systèmes de gestion de données plus économiques, plus rapides et plus vastes. En le même temps, une part importante et croissante du coût total de propriété de ces systèmes sont l'énergie et le refroidissement. Ces dernières années, l'augmentation des coûts de l'énergie est devenue l'une des questions cruciales dans les centres de données. La nouvelle approche qui s'oriente vers les technologies de préservation de l'énergie a vu l'apparition et l'émergence des concepts tels que les systèmes éco-énergétiques. Par conséquent, l'objectif des concepteurs du système informatique est allé vers la puissance électrique et l'efficacité énergétique.

Toutefois, pour résoudre à la fois le problème énergétique des centres de données tout en continuant à alimenter la demande des ressources en matière de gestion des données, nous devrons passer de l'optimisation des systèmes de gestion des données aux performances pures à l'optimisation de l'efficacité énergétique. Les bases de données constituent une grande partie des ressources dans un centre de données, et consomment ainsi une large partie d'énergie. Traditionnellement, les recherches antérieures se sont uniquement concentrées sur l'amélioration des caractéristiques de performance des bases de données au cours des phases de conception et d'exploitation. Le *modèle standard* a pour but de maximiser les performances du SGBD ou, en d'autres termes, de minimiser les temps de réponse lors de l'exécution des requêtes. Cependant, les BDs n'ont pas la capacité de gérer la consommation d'énergie lors de son fonctionnement. Contrairement à l'approche traditionnelle, le *modèle éco-énergétique*, introduit dans ce chapitre, utilise l'*efficacité énergétique* comme étant un aspect supplémentaire qui devrait être considéré dans le processus interne du SGBD.

Pour identifier les défis ouverts dans ce domaine et faciliter des futurs progrès, il est essentiel de synthétiser et classer les recherches sur la conception éco-énergétique réalisées à ce jour. Ce chapitre traite le problème de la consommation de l'énergie et présente ainsi une taxonomie des travaux sur la conception économe en énergie des systèmes informatiques couvrant les niveaux : matériels, systèmes d'exploitations, applications et bases de données.

3.2 L'énergie dans la technologie de l'information

Dans cette section, nous mettons en avant le problème de l'efficacité énergétique et nous décrivons certaines façons possibles permettant de l'aborder.

3.2.1 Le concept de l'énergie

Pour faciliter la compréhension du reste de notre thèse, certains concepts et définitions préliminaires sont donnés. Le mot « énergie » vient du grec **energeia** signifiant « ***force en action*** » [41], cette terminologie vient d'une approche physique du problème, décrivant l'énergie comme [217] :

Définition 7

> *L'énergie est une mesure de la capacité d'un système à modifier un état, à produire un travail entraînant un mouvement, un rayonnement électromagnétique ou de la chaleur.*

L'énergie peut exister sous diverses formes telles que l'énergie mécanique, l'énergie thermique, l'énergie de la lumière, l'énergie sonore, l'énergie magnétique, l'énergie électrique, l'énergie chimique et l'énergie nucléaire. Nous focalisons sur l'énergie électrique. Le transfert d'énergie en une seconde est défini comme la puissance électrique. Plus précisément :

Définition 8

La puissance est la quantité d'énergie d'un système par unité de temps, ou le rythme de faire un travail.

L'énergie est généralement mesurée en *Joules* tandis que la puissance est mesurée en *Watts*. Formellement, l'énergie et la puissance peuvent être définies comme suit :

$$P = \frac{W}{t} \tag{3.1}$$

$$E = P \times t \tag{3.2}$$

où P, t, W et E représentent respectivement, une puissance, une période de temps, le travail total effectué dans ce laps de temps, et de l'énergie. Ces concepts de puissance, travail et énergie sont utilisés différemment dans divers contextes. Dans le contexte de la technologie d'information, le travail implique des activités associées à l'exécution de programmes (par exemple : addition, soustraction ou opération dans la mémoire), la puissance est le taux pour lequel l'ordinateur consomme de l'énergie électrique (ou la dissipe sous forme de chaleur) et l'énergie est l'énergie électrique totale que consomme l'ordinateur (ou se dissipe en chaleur) au fil du temps [269].

Cependant, l'énergie peut être réduite soit par la minimisation du laps du temps ou de la puissance consommée. En général, la consommation de la puissance électrique d'un système donné peut-être divisée en deux parties :

1. *Puissance de base* : La consommation de la puissance lorsque le système est inactif. Cela inclut la consommation des ventilateurs, processeur, mémoire, périphériques d'E/S et les autres composants de la carte mère dans un état d'inactivité.
2. *Puissance active* : La consommation de la puissance lors de l'exécution d'une charge de travail. Les composants qui influencent sur la puissance active sont déterminés par le type de charge de travail qui s'exécute sur la machine, et la façon avec laquelle elle utilise le processeur, la mémoire et les dispositifs d'E/S.

Deux concepts de puissance électrique doivent être pris en considération lors de l'évaluation de l'utilisation d'énergie dans un système : *la puissance moyenne* représentant la puissance moyenne consommée au cours d'exécution de la charge de requête, et *le pic* représentant la puissance maximale (*peak power*). Dans cette thèse, nous traitons la puissance moyenne.

Lorsqu'on évoque des économies de l'énergie, on utilise les terme « *efficacité* » ou « *efficience* ». L'efficacité énergétique (ou efficience énergétique) est définie comme :

Définition 9

La consommation d'énergie d'un système par rapport au service rendu.

En d'autres termes, l'efficacité énergétique (EE) est équivalente au rapport de la performance, mesurée

comme le taux de travail effectué (TE), à la puissance utilisée [116, 265] :

$$EE = \frac{TE}{E} = \frac{TE}{P \times t} = \frac{t}{P} \tag{3.3}$$

Pour une quantité de travail fixe, maximiser l'efficacité énergétique est la même que minimiser l'énergie. Ainsi, contrairement à l'optimisation des performances, nous pouvons améliorer l'efficacité énergétique en réduisant la puissance, le temps ou les deux à la fois.

3.2.2 Méthodes d'évaluation de l'EE

La comparaison entre les différentes méthodes éco-énergétique et l'évaluation des méthodes développées quel que soit leur correspondance ou pas à la réalité, constitue une importante question. Les utilisateurs intéressés par l'énergie, les fabricants d'ordinateurs ainsi que les chercheurs, doivent être en mesure d'évaluer et de comparer l'EE des systèmes informatiques afin de prendre des décisions d'achat ou d'identifier des technologies prometteuses. Des règles bien définies sont primordiales pour fournir des comparaisons normalisées et justes de l'EE des ordinateurs [213]. Des efforts considérables ont été fournis dans le domaine d'académie et d'industrie concernant les mesures d'évaluation et les techniques de modélisation de l'énergie. Elles peuvent être regroupés en deux catégories : modèles de coût et benchmarks [272].

3.2.2.1 Modèles et métriques de coût

La consommation d'énergie réelle d'un système spécifique dépend de nombreux facteurs, tels que la charge de travail, l'équilibre du système et les paramètres environnementaux. La mesure de la consommation d'énergie nécessite des modèles énergétiques et thermiques précis pour les composants individuels (processeur, mémoire, disque, carte graphique, réseau, etc.), les systèmes, les centres de calcul et les applications. Deux types d'approches de modélisation existent dans la littérature, y compris les approches basées sur l'analyses détaillées (analytiques) et les approches boîtes noires (ou expérimentales) [212].

3.2.2.1.1 Modèles analytiques

Cette approche est généralement utilisée par les concepteurs des systèmes qui l'ont déjà écrit et possèdent les détails sur le fonctionnant interne, et souhaitent comprendre les caractéristiques de leur consommation d'énergie. Ces modèles varient dans le détail et la précision à partir des modèles avec des connaissances de circuit détaillées aux autres qui utilisent une combinaison d'informations architecturales et de paramètres de circuit, tels que les compteurs de performance de processeur (des registres matériels qui comptent différents types d'événements sur le processeur et la mémoire). Ils peuvent-être très précis sans la nécessité d'utiliser des équipements de mesurage coûteux, mais ils ne parviennent généralement pas à offrir une portabilité et une généralité car ils dépendent des systèmes spécifiques. De plus, ils sont très difficiles à être implémenter. Des exemples des systèmes de cette approche incluent : Wattch [43], Tempo [232], SoftWatt [111] et les travaux basés sur les compteurs de performance : [74, 137, 167].

3.2.2.1.2 Modèles expérimentaux

La deuxième approche de la modélisation d'énergie, et celle la plus utilisée dans la littérature, est de construire un modèle d'énergie basé uniquement sur l'ajustement d'un modèle mathématique qui prend en entrer des mesures collectées en temps réel sur les données et des mesures sur l'énergie consommée. La procédure générale consiste à calibrer le modèle en exécutant une suite des charges de travail synthétiques conçus pour générer une plage de valeurs pour chaque métrique ; par exemple, si le nombre d'instructions à virgule flottante est l'une des métriques à collecter, les charges de travail devraient mettre l'unité à virgule flottante sous tension à différents niveaux d'intensités.

Une approche générale pour gérer la consommation d'énergie des systèmes informatiques consiste en quatre étapes principales (voir la Figure 3.1) : extraction des paramètres, construction des modèles, validation des modèles et application du modèle à une tâche telle que la prédiction [79].

1. **Extraction des paramètres.** Afin de réduire la consommation d'énergie d'un système, nous devons d'abord mesurer la consommation d'énergie de ses composants et identifier où la majeure partie de l'énergie est dépensée. C'est la tâche de la phase d'extraction des paramètres.
2. **Construction des modèles.** Deuxièmement, les paramètres d'entrée sélectionnées sont utilisés pour construire un modèle de consommation d'énergie en utilisant des techniques d'analyse telles que la régression et l'apprentissage automatique. L'un des principaux problèmes auxquels nous sommes confrontés dans cette étape est que certains paramètres importants du système, à l'image de la consommation d'énergie d'un composant particulier ne peuvent pas être mesurés directement. Les méthodes d'analyse classiques ne pourront pas produire des résultats précis dans de telles situations, tandis que les techniques d'apprentissage automatique peuvent mieux fonctionner. Le résultat de cette étape est un modèle d'énergie.
3. **Validation du modèle.** Ensuite, le modèle doit-être validé pour son adéquation aux fins prévues.
4. **Utilisation du modèle.** Enfin, le modèle construit peut être utilisé pour prédire la consommation d'énergie des composants ou du système. De telles prédictions peuvent alors être utilisées pour (1) améliorer l'EE, par exemple en incorporant le modèle dans des techniques comme l'ordonnancement des tâches, l'optimisation des algorithmes existants, la commutation des composants à des états de faible puissance, etc, (2) la conception des composants et des systèmes éco-énergétique et (3) la prévision des tendances en matière d'EE des systèmes.

La modélisation de l'énergie est un domaine de recherche actif. Il existe une panoplie des travaux dans la littérature, des revues comme [25, 211, 100, 193, 79] donne un aperçu global sur ces travaux sur plusieurs niveaux (architecture, système, composants, circuit, système d'exploitation, application, etc.) et avec une classification détaillée. Cependant, un modèle de coût énergétique de base peut être formuler comme suit :

$$E = \alpha\ I_{cpu} + \beta\ I_{mem} + \gamma\ I_{es} + \delta\ I_{com} + \epsilon \tag{3.4}$$

Où I_{cpu}, I_{mem}, I_{es} et I_{com} représentent le nombre d'instructions du processeur, l'utilisation de la mémoire, le débit d'E/S de disque et le nombre de transfert réseau respectivement. ϵ est une valeur d'erreur ou perturbation, qui représente les erreurs dans les mesures. Les poids α, β, γ et δ sont obtenus par une phase d'extraction de paramètres et l'étude du comportement des composants matériels sur la consommation d'énergie. Souvent, cette phase s'appuie sur les techniques d'apprentissage automatique [180].

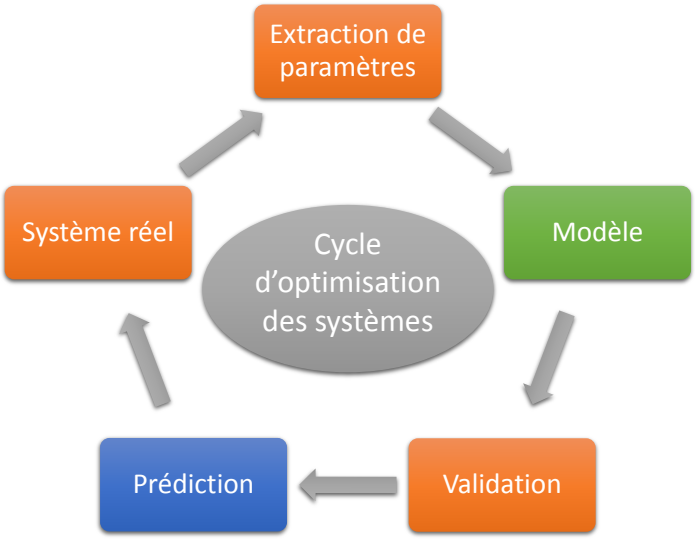


FIGURE 3.1 – Le cycle d'optimisation éco-énergétique des systèmes informatiques.

3.2.2.2 Benchmarking

Dans la deuxième catégorie des méthodes d'évaluation de l'énergie on trouve les benchmarks. Un indice de référence, ou benchmark, est un instrument d'analyse, de sensibilisation et de motivation des chercheurs. Il s'agit de définir des références pour se comparer aux meilleurs techniques de la branche d'EE, dans le but d'exploiter les potentiels d'amélioration en matière de gestion énergétique et de réaliser des économies [214].

Les chercheurs, les agences gouvernementales et les consortiums standard de l'industrie pour les mesures de la performance, y compris *Transaction Processing Performance Council* (TPC), *Standard Performance Evaluation Corporation* (SPEC) et *Storage Performance Council* (SPC) ont développé des benchmarks pour mesurer la consommation d'énergie des systèmes informatiques. Poess *et al.* ont donné un aperçu très complet des benchmarks énergétiques actuellement disponibles. Il ont analysé leurs points communs et leurs différences selon diverses dimensions, y compris les composantes matérielles, la charge de requêtes et le type d'application, ainsi que les métriques et les exigences de précision et de calibration [204, 202].

3.2.2.2.1 TPC

Le TPC a développé la norme *TPC-Energy* [289, 1] conçue pour augmenter les benchmarks de TPC existants avec des mesures d'énergie, afin que les utilisateurs finaux puissent comprendre les coûts d'énergie associés à un résultat d'un benchmark spécifique. Le nouvel benchmark répond à la demande émergente des mesures liées à l'énergie. Alors que TPC-C, TPC-E et TPC-CH ne fournissent que des mesures de performances, TPC-Energy introduit des mesures pour la consommation d'énergie pendant le traitement des requêtes. TPC-Energy régule en outre comment les mesures de puissance doivent être effectuées. Par exemple, quels sont les dispositifs de mesure à utiliser et quelle précision de mesure doit être maintenue. Les mesures définies par TPC-Energy sont la *consommation d'énergie* par rapport au

travail effectué exprimée en *Joule* par *transaction* [227].

3.2.2.2.2 SPEC

SPEC a introduit le benchmark *SPECpower_ssj2008* [2] pour mesurer les performances et la consommation d'énergie d'un système exécutant des charges de travail basées sur Java. Le benchmark exerce les CPU, les caches, la hiérarchie de mémoire et l'évolutivité des processeurs de mémoire partagée sur plusieurs niveaux de charge. Le benchmark fonctionne sur une grande variété des systèmes d'exploitation et d'architectures matérielles avec la prise en charge des multi-nœuds. Contrairement à TPC-Energy, le benchmark SPEC mesure la consommation électrique de 0% de charge à 100% de charge et agrège les mesures par la moyenne géométrique pour former un seul résultat. En outre, les nouvelles versions du benchmark SPEC comme *SPECweb_2009* et *SPECvirt_sc2010* intègrent les méthodes de mesure de puissance de SPECpower.

3.2.2.2.3 SPC

De plus, le SPC, dont les benchmarks sont focalisés sur l'évaluation des composants de stockage, a défini des extensions liées à l'énergie pour leurs benchmarks [3]. Ces extensions ne suivent pas la puissance maximale consommée à la charge, mais la mesurent à 80% (notée intense) et 50% (notée modérée) des performances maximales ainsi qu'en mode veille. En outre, ils introduisent la consommation d'énergie moyenne pondérée basée sur trois modes d'utilisation différents (faible, moyen et élevé).

3.2.2.2.4 Green500

Le projet *Green500* [4] vise à la sensibilisation sur l'impact environnemental et à la durabilité à long terme des supercalculateurs haut de gamme en fournissant un classement des supercalculateurs les plus éco-énergétiques au monde. La liste de Green500 utilise « Performance par Watt » comme métrique pour classer l'EE des supercalculateurs. La performance est définie comme la performance maximale obtenue, GFLOPS (Giga opération en virgule flottante par seconde), par le benchmark Linpack [9] sur l'ensemble du système. La puissance est définie comme la consommation d'énergie moyenne du système pendant l'exécution de Linpack.

3.2.2.2.5 JouleSort

À part les benchmarks proposés par les organismes spécialisés, la communauté des bases de données elle-même a proposé un benchmark en matière d'énergie pour évaluer l'EE des systèmes informatiques. Le benchmark *JouleSort* [213] est un benchmark de tri, dont l'idée est de mesurer l'énergie consommée pour trier une taille de données. Il mesure l'EE des systèmes à leur maximum d'utilisation (100% de la charge). Il s'agit d'une extension du *Sort Benchmark* [10], qui est utilisé pour mesurer la performance et le coût-performance des systèmes informatiques. JouleSort mesure le coût d'une certaine quantité de

9. https://www.top500.org/project/linpack/
10. http://sortbenchmark.org/

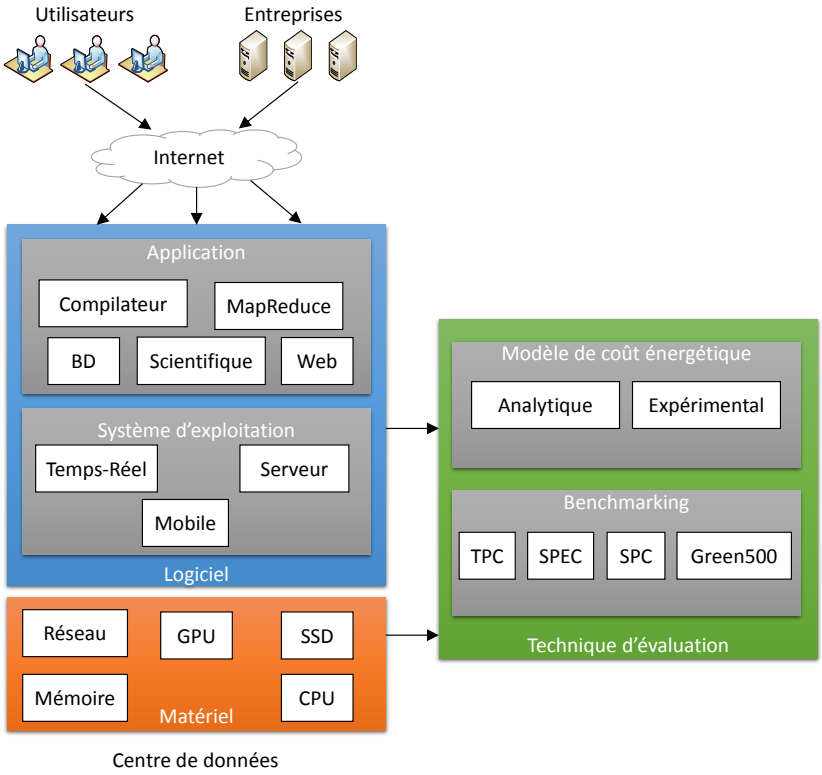


FIGURE 3.2 – La consommation d'énergie dans les systèmes informatiques sur les trois niveaux.

travail, avec une certaine mesure de consommation de puissance, par exemple : la puissance moyenne, la puissance de pic et l'énergie totale.

3.2.3 Approches d'EE dans les systèmes informatiques

Aujourd'hui, les enjeux de l'informatique éco-énergétique sont de plus en plus étudiés à tous les niveaux dans les infrastructures informatiques. Compte tenu d'une brève description des méthodes d'évaluation d'énergie dans les systèmes, nous présentons dans cette section un état de l'art des travaux d'EE des systèmes informatiques sur trois niveaux : (1) matériel, (2) système d'exploitation et (3) application. La Figure 3.2 illustre l'interdépendance des différents niveaux qui ont un impact sur la consommation d'énergie des systèmes informatiques.

3.2.3.1 Approches d'EE au niveau matériel

Un ordinateur de calcul comprend plusieurs composants, chacun pouvant être optimisé pour économiser l'énergie. Nous allons évoquer brièvement les techniques importantes proposées pour chaque composant.

Afin de réduire la consommation d'énergie du CPU, les auteurs de [275] ont proposé un travail original. L'idée de base est le contrôle dynamique de la vitesse d'horloge pour réduire la consommation d'énergie pour un travail particulier. L'avantage provient de la réduction dans la tension électrique qui

suit le ralentissement de l'horloge. Comme un inconvénient pour appliquer cette technique, le temps d'exécution augmente. Ce travail a largement inspiré le développement de la technique d'ajustement dynamique de la tension (DVS), et également la communauté des chercheurs. Encouragés par ces avantages, les chercheurs dans [287] ont construit un modèle théorique sur la base de travail précédent. L'objectif est de trouver le moyen le plus économe en énergie pour ordonnancer les tâches tout en respectant les délais d'exécution. L'ordonnancement heuristique en ligne des tâches apériodiques tout en conservant la faisabilité des ensembles de tâches périodiques est présenté dans [119]. Un algorithme éco-énergétique pour l'ordonnancement sans préemption est proposée dans [120]. Une autre méthode tente de ralentir le CPU à chaque fois qu'il y a une seule tâche éligible pour l'exécution, a été développée dans [236]. Une approche plus agressive est présentée dans [16], où deux algorithmes hors et en ligne sont considérés en respectant les délais, tout en réduisant la vitesse du cycle CPU autant que possible. Une comparaison systématique des différents algorithmes d'ordonnancement sur le rapport temps/performance est démontrée dans [103]. Dans [201], les informations de la date limite sont adoptées dans le système d'exploitation en temps réel avec la technique DVS. La norme ACPI (*Advanced Configuration and Power Interface*), disponible sur la plupart des systèmes d'exploitation, fournit une interface standard pour gérer les états de puissance du processeur [28]. Pour résumer, la plupart de ces algorithmes d'ordonnancement proposés tentent de tirer parti de l'EE et des contraintes temporelles d'un système en temps réel, à fin d'adapter le meilleur compromis raisonnable entre la tension électrique et les performances.

Par rapport à l'intérêt de la recherche dans l'utilisation du processeur, il y avait d'autres voix disant que le DVS peut-être également appliquée sur d'autres composantes, telle que la gestion de la mémoire. Les auteures de [90] illustrent par simulation que ni la gestion d'alimentation de mémoire ni encore la technique DVS sur le processeur, peuvent économiser l'énergie de façon spectaculaire. Mais en combinant les deux techniques ensemble, 89% d'économie d'énergie, par rapport au cas de base, peut être achevé. Les techniques d'optimisation et les algorithmes proposés [162, 199, 277] se sont concentrés sur les opportunités de basculer la mémoire entière ou une partie d'elle en mode faible consommation, soit pendant ou au moment de l'exécution des processus. D'autres technologies émergentes telles que la mémoire à changement de phase (PCM) [300], transfert de spin [273] et memristor [190] ont également été proposées comme des alternatives de la mémoire principale économe en énergie.

Les unités de traitement graphique (GPU) sont également devenues une partie essentielle des centres de données hébergeant des applications scientifiques à cause de leur efficacité dans les fonctions à virgule flottante. De plus, les GPUs sont très efficaces sur le plan énergétique par rapport aux processeurs. Les deux principales machines supercalculateur à haut rendement énergétique sont basées sur GPU [4]. Les GPUs fournissent près de 85 fois moins de consommation d'énergie pour un calcul en virgule flottante que les processeurs x86 [142].

Il existe deux techniques de base pour atteindre l'EE dans les équipements de stockage : (1) rendre le matériel de stockage économes en énergie, et (2) réduire la redondance des données [288]. Le principal objectif du matériel de stockage éco-énergétique est constitué par les disques SSDs (Solid-State Drive). Les disques SSD utilisent une mémoire Flash, c'est-à-dire une mémoire non volatile ayant des caractéristiques similaires à la mémoire morte effaçable électriquement et programmable (EEPROM). Les SSDs ont des propriétés proportionnelles à l'énergie, du fait que l'énergie utilisée est proportionnelle aux opérations d'E/S par seconde [237]. D'autre part, les disques durs (HDDs) consomment 85% d'énergie lorsqu'ils sont

inactifs [187]. Les SSDs deviennent l'avenir du stockage primaire économe en énergie dans les systèmes.

Les périphériques réseau constituent une autre ressource énergétique dans les centres de données [193]. La consommation d'énergie dans les périphériques réseau peut être gérée par des techniques de taux de liaison adaptatif (ALR) [33, 32]. Les techniques d'ALR atteignent l'EE par : (1) des débits de données réduits, ou (2) des transitions d'état de puissance inférieure/inactive qui sont gérées en fonction des charges de données [31]. Les protocoles de routage efficaces jouent un rôle important dans l'utilisation de la capacité réseau. Shang *et al.* [233] ont avancé l'idée du routage énergétique où peu de périphériques réseau fournissent le routage de base et le débit en fonction d'un seuil de performance tandis que le reste des périphériques sont hors tension.

3.2.3.2 Approches d'EE au niveau système d'exploitation

La communauté de la recherche a rapidement réalisé l'impact significatif de la couche système sur la consommation d'énergie. Le travail de [26] classe les composants du système consommant la majeure partie de l'énergie en trois catégories : unités de calcul, unités de communication et unités de stockage. Les auteurs présentent également un état de l'art des techniques éco-énergétique autour de trois phases principales d'une conception de système : la conceptualisation et la modélisation, la conception et la mise en œuvre, et la gestion d'exécution. Les recherches montrent ainsi que les systèmes d'exploitation (SE) ont une consommation d'énergie hétérogène et peuvent-être optimisés pour consommer moins d'énergie. La consommation varie même entre les versions du même SE. Par exemple, la consommation d'un ordinateur sous différentes versions de Windows et des noyaux Linux, suivent des méthodologies de mesures similaires, présente des variations non négligeables [193]. Les auteurs de [197] ont développé un gestionnaire d'alimentation en temps réel dans le noyau pour le système d'exploitation Linux appelé le gouverneur à la demande (LGD). Le gestionnaire surveille en permanence l'utilisation du processeur et définit une fréquence d'horloge qui correspond aux exigences de performances actuelles, ce qui minimise la consommation d'énergie. Le travail de [294] a proposé et développé *ECOsystem*, un framework pour la gestion de l'énergie en tant que ressource de première classe dans un SE destinée aux dispositifs alimentés à l'aide de batterie tels que les mobiles. ECOsystem fournit une interface pour définir une durée de vie de la batterie cible et des priorités d'application utilisées pour déterminer la quantité d'énergie qui sera allouée aux applications à chaque période. Des techniques similaires ont été proposées pour les mobiles comme : *Nemesis OS* [189], *GRACE* [225, 267] et *Coda/Odyssey* [92], et pour les serveurs : *Linux/RK* [208], *PowerNap* [179] et *Barely Alive* [13].

3.2.3.3 Approches d'EE au niveau application

L'EE au niveau application est devenu un domaine de recherche actif pour les raisons suivantes : (1) les techniques d'optimisation de bas niveau (matériel et SE) dépend fortement de l'estimation précise des profils de puissance des applications. Cependant, beaucoup d'informations et de statistiques nécessaires pour ces estimations ne sont disponibles que dans le niveau application. (2) De nombreuses applications peuvent être exécutées de différentes manières pour accomplir la même tâche de calcul. Les applications éco-énergétique, qui adaptent leurs comportements en fonction des états liés à l'alimentation des systèmes sous-jacents, peuvent offrir des possibilités supplémentaires d'économie d'énergie. De nombreux projets de recherche dans ce domaine sont consacrés aux services Web [35, 234, 53, 64],

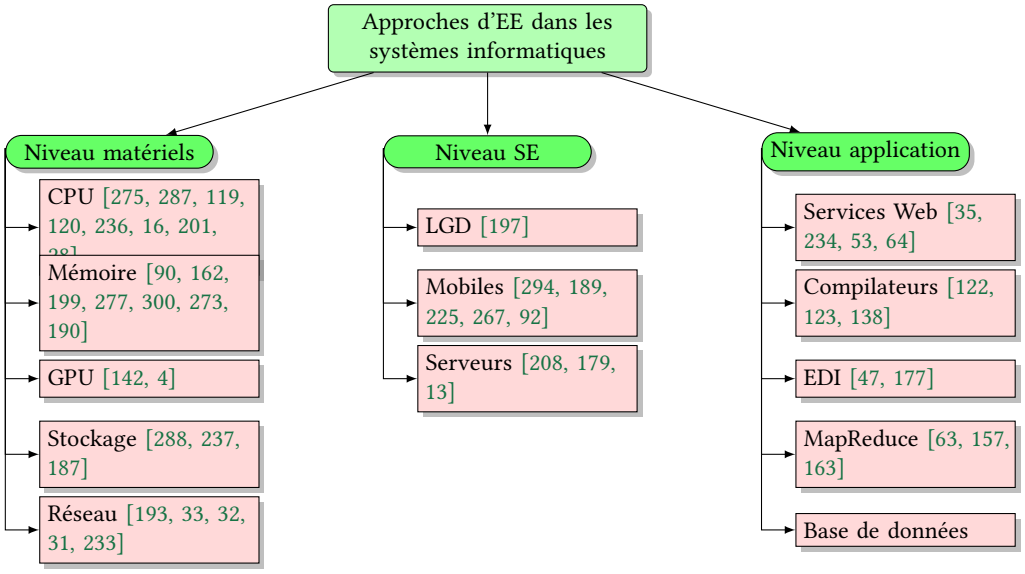


FIGURE 3.3 – Classification des approches d'EE dans les systèmes informatiques.

les compilateurs [122, 123, 138] et les environnements de développement (EDI) [47]. Dans [280], un framework de programmation à usage général a été développé pour permettre aux applications d'être exécutées avec des plans différents en fonction de leurs coûts d'énergie. *SEEDS* est un framework pour aider les ingénieurs des logiciels à développer des applications éco-énergétiques de haut niveau [177]. SEEDS fournit une analyse automatisée, une prise de décision et une mise en œuvre visant à optimiser la consommation d'énergie d'une application donnée. Une application récente des techniques éco-énergétique inclut *MapReduce* [63, 157, 163].

Et encore plus récemment, la consommation d'énergie dans les bases de données commence à attirer l'attention de la communauté des chercheurs. L'objectif principal de ces efforts est de concevoir des SGBDs avec la consommation optimisée d'énergie. Notre étude s'inscrit dans cette catégorie, et nous allons détailler dans une section séparée les différents travaux de littérature sur l'EE dans les bases de données. La Figure 3.3 représente une classification des travaux d'EE dans les systèmes informatiques sur les trois niveaux discutés.

3.3 Approches d'EE dans les BD

La gestion de l'énergie des systèmes de bases de données est devenue un sujet de recherche important au cours des dernières années. Il y a eu une pléthore d'études et d'initiatives de la communauté des chercheurs. Lors de l'étude de la littérature des travaux, on distingue principalement deux approches principales : *Approches Orientées Matériels* ($\mathcal{AOM}$) et *Approches Orientées Logicielles* ($\mathcal{AOL}$). Dans les sections suivantes, nous détaillons les travaux de chaque approche.

3.3.1 $\mathcal{AOM}$

Les approches matérielles utilisent des solutions « naïves » comme la désactivation de composants électroniques pendant les périodes d'inactivité, et des solutions avancées telles que la modification dynamique de la performance des composants matériels pour une meilleure EE, qui peut être illustrer par la technique DVFS offerte par les CPU d'aujourd'hui [25]. Par ailleurs, les techniques appliquées au niveau du matériel peuvent être divisées en deux catégories : (i) dispositif de traitement et (ii) la gestion du stockage.

3.3.1.1 Dispositif de traitement

Les techniques appartenant à cette catégorie utilisent de nouvelles approches pour minimiser la consommation d'énergie des appareils informatiques. Lang *et al.* ont proposé le mécanisme PVC (Processor Voltage/Frequency Control) pour équilibrer la consommation d'énergie et la performance [156]. Il vise à exécuter des instructions à une tension et une fréquence de processeur basse en tirant parti de la capacité des processeurs modernes. Une technique similaire est utilisée dans [266], les auteurs rapportent 51,3% des économies d'énergie. La technique proposée ajuste dynamiquement le niveau DVFS du processeur en fonction des performances du SGBD ou le plan de requête choisi pour l'exécution. Xu *et al.* [285] aussi utilisent la technique DVFS avec un mécanisme de contrôle de rétroaction pour les charges de requêtes d'une base de données. Afin d'économiser l'énergie, ils gèrent le niveau DVFS en fonction du débit de travail (*throughput*) et des caractéristiques de la charge de requêtes (par exemple, en basant sur le nombre d'E/S ou de CPU). Dans une autre étude, une co-ingénierie entre Oracle et Intel tente de réduire les coûts d'exploitation et de répondre efficacement aux objectifs d'informatique durable (*green computing*) a donné lieu à une nouvelle version de *Oracle Exadata Database Machine* [130]. La solution proposée met dynamiquement le processeur Intel Xeon et la mémoire dans l'état de puissance disponible le plus bas convenable pour répondre aux exigences de la charge de requêtes. Cependant, les techniques mentionnées ci-dessus souffrent des frais additionnels pour la transition vers des états de faible puissance, qui conduisent généralement à une consommation d'énergie supplémentaire et des délais d'exécution causés par la réinitialisation des composants [25].

Pour accélérer les applications de données intensives, les Field-Programmable Gate Array (FPGA) sont utilisés à la place des processeurs classiques, principalement parce qu'ils permettent la construction de matériel personnalisé à des coûts de développement relativement faibles. De plus, il a déjà été montré dans [276] que les solutions à base de FPGA ont le potentiel d'améliorer la performance et en même temps de réduire la consommation d'énergie. Les auteurs rapportent que l'utilisation de ces propositions pour certaines requêtes, permet au système de consommer que 216 *Joules*, alors que les techniques traditionnelles consomment 3888 *Joules*.

Les unités de traitement graphique (GPU) ont été considérées comme une excellente alternative aux processeurs pour leur haute performance, leur haut débit de traitement ainsi que leur économie d'énergie. Une étude a rapporté que l'utilisation d'un GPU est plus économe en énergie lorsque l'amélioration de la performance est supérieure à une certaine borne, par rapport à une solution CPU classique [216]. Le CPU-GPU co-traitement est une autre tendance intéressante dans la technologie de base de données, qui consiste à exploiter les avancées matérielles ainsi que la conception d'algorithmes optimisés pour améliorer les performances de traitement des requêtes et l'EE. Les résultats de comparaison entre une

architecture CPU-GPU discrète et une architecture CPU-GPU couplée montrent que la consommation moyenne d'énergie de l'architecture discrète est entre 36% et 44% supérieurs à celle de l'architecture couplée [126].

D'autre part, des études récentes telles que [84] affirment que l'utilisation de disques Smart Solid State (SSSDs) pour traiter les données peuvent-être plus avantageux à la fois en terme de performance et de consommation d'énergie. Les SSSDs se sont des dispositifs de stockage flash qui intègrent une mémoire et un processeur à l'intérieur du dispositif de SSD, et pourrait être utilisé pour exécuter des programmes généraux définis par l'utilisateur. Les auteurs de [84] améliorent l'EE du traitement des requêtes en réduisant son temps d'exécution et/ou en exécutant le traitement sur le processeur de faible puissance qui vient avec les SSSDs. Les premiers résultats de l'exécution d'un sous-ensemble de requêtes SQL (simples requêtes de sélection et d'agrégation) sur un SSSD fournissent jusqu'à 3x de réduction de la consommation d'énergie. Les SSSD ont beaucoup de potentiel, mais les capacités de calcul des appareils embarqués dans les SSSD actuels sont trop limitées pour gérer le volume de données énormes.

3.3.1.2 Gestion du stockage

Les dispositifs de traitement ont été considérés comme les principaux contributeurs à la consommation d'énergie dans un serveur. Cela dit, la consommation d'énergie des dispositifs de stockage secondaire et de la mémoire ont également attiré une forte attention des chercheurs en raison de leurs capacités de croissance significative au cours des dernières années. Les approches de gestion de stockage tentent de trouver un bon niveau de consolidation de la charge, l'amélioration des techniques de mise en cache et de préchargement et l'optimisation des modes d'alimentation dans les matrices de disques afin de minimiser la dissipation d'énergie [266]. Dans [116], les auteurs fournissent des expériences pour montrer que les choix effectués par les systèmes de base de données peuvent améliorer l'EE. Plus précisément, repartitionner la base de données sur 66 disques propose une augmentation de 14% de l'efficacité contre la dégradation de 45% de la performance. Les auteurs de [266] ont proposé un modèle d'optimisation intégrée pour la migration des données et l'ajustement de l'état du disque, dans lequel ils placent dynamiquement les enregistrements dans un fragment selon la fréquence des enregistrements de données d'accès. Les résultats présentés montrent que près de 50% de l'énergie peut être économisé en utilisant les techniques proposées. Dans la même direction, le travail de [22] propose un modèle de gestion de puissance dynamique qui prend des décisions en temps réel sur le passage des disques à des modes de faible puissance. Le modèle est intégré dans le SGBD et utilisé pour obtenir le compromis optimal entre la consommation d'énergie et le temps de réponse des requêtes. Le rapport montre 60% des économies d'énergie.

L'utilisation de disques SSD comme un périphérique de stockage des bases de données pour améliorer l'EE est une autre direction qui a été étudiée par des travaux récents. Dans [228], les auteurs ont étudié le comportement de la performance et la consommation d'énergie des SSD pour des schémas d'accès typiques pour les applications de base de données à forte intensité d'E/S. Ils montrent, en outre, que la technologie SSD offre un nouveau niveau de performance et d'EE, et suggèrent l'exploitation de leurs nouvelles caractéristiques par les serveurs de base de données. De même, le travail de [65] analyse et évalue les performances de traitement des données et l'EE des SSD et système de stockage. Comme son prédécesseur, les auteurs montrent l'amélioration de la performance et l'EE.

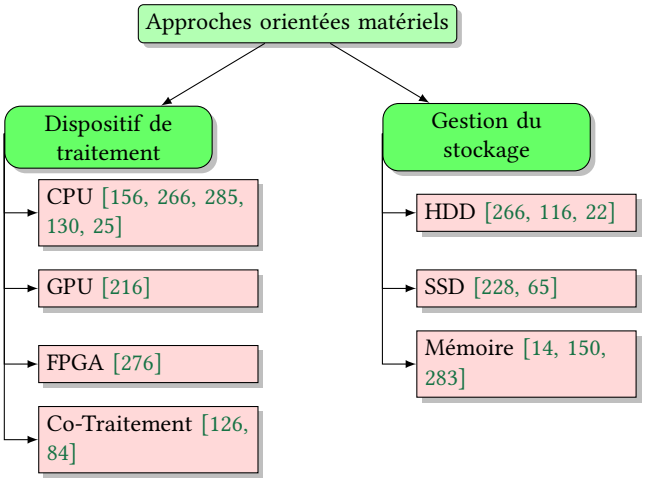


FIGURE 3.4 – Classification des approches d'EE orientées matériels.

La mémoire est un autre consommateur d'énergie important dans les systèmes informatiques, en particulier avec l'utilisation croissante des bases de données orientée colonnes où le stockage des données se fait directement dans la mémoire. De nombreux travaux de recherche récents commencent à étudier l'impact du dispositif de mémoire sur la consommation d'énergie. L'enquête faite dans [14] souligne que l'utilisation croissante des bases de données en mémoire va bientôt émerger la mémoire en tant que consommateur de puissance électrique dominante au lieu de CPU. Comme pour les composants précédents, la mémoire principale a également des différents états de puissance et de la fréquence à modifier en fonction de son utilisation. Sur cette base, les auteurs ont utilisé la fréquence mise à l'échelle et des modes de mise hors tension des DRAM pour améliorer la consommation d'énergie sans sacrifier les performances. Dans [150], les auteurs rapportent un résultat préliminaire sur une technique d'allocation de mémoire basée sur les rangs pour économiser l'énergie des systèmes de base de données. La technique permet au SGBD de déplacer les rangs inutiles de la mémoire aux états de faible puissance et réduire ainsi la consommation d'énergie globale. L'évaluation indique que la méthode proposée atteint 40% de diminution de la puissance lorsque la taille de la base de données est plus petite que la quantité de la mémoire déjà provisionnée sur le serveur. Le travail de [117] propose une architecture de mémoire hybride comprenant à la fois Dynamic Random Access Memory (DRAM) et de la mémoire non volatile (NVM) pour réduire la consommation d'énergie de base de données, grâce à une politique de gestion des données au niveau de l'application qui décide de placer des données sur la DRAM ou la NVM. Ils analysent d'abord l'exécution d'applications et de collectent des statistiques d'accès mémoire pour des objets individuels. Ceux-ci sont évalués en utilisant un modèle d'analyse de la performance et de l'énergie pour identifier la technologie de mémoire la plus adaptée pour stocker ces données. Les résultats présentés indiquent que l'utilisation de la mémoire hybride peut réduire l'énergie du système de mémoire d'environ 80%.

3.3.2 Bilan et discussion

Dans les sections précédentes, nous avons classé les techniques appliquées au niveau du matériel en deux catégories, dispositif de traitement et la gestion du stockage. Les études précitées dans pour chaque catégorie ont en effet apporté une contribution importante à la conservation de l'énergie (voir Figure 3.4). Cependant, les $\mathcal{AOM}$ ne sont pas simple à appliquer et ont plusieurs inconvénients en particulier dans le domaine des bases de données. Tout d'abord, ils ont besoin d'utiliser du nouveau matériel, puisque les nouvelles technologies tels que DVFS ne sont pas disponibles sur tous les processeurs ou mémoires actuels. Deuxièmement, les SSD émergents sont très chers et ont un problème de durée de vie. Dernier point, mais non des moindres, il existe différentes marques/versions pour chaque composant matériel, ce qui rend ces méthodes peu pratiques en raison de leurs problèmes de portabilité. Par conséquent, les $\mathcal{AOL}$ sont beaucoup préférés dans le cadre des bases de données, car ils sont indépendants du matériel sous-jacent. En outre, le SGBD a une connaissance détaillée des éléments internes du système, tels que la charge de requêtes, les plans d'exécution, les statistiques de base de données, etc. [150]. Dans [116], Harizopoulos *et al.* de laboratoire de HP confirment que « les approches matérielles ne sont qu'une partie de la solution, et les approches logicielles seront la clé dans l'optimisation de l'efficacité énergétique ».

3.3.3 $\mathcal{AOL}$

Les approches matérielles ne sont qu'une partie de la solution pour réduire les valeurs de puissance ; les approches logicielles sont l'autre élément clé dans les mécanismes d'optimisation de l'EE. A ce niveau, les papiers de vision sur l'énergie [104, 116] recommandent de revoir les techniques d'optimisation des requêtes, les algorithmes d'ordonnancement, la conception physique et les techniques de mise à jour de base de données en tenant compte de l'énergie. La plupart des importantes études concernent deux aspects principaux : (i) la définition des modèles de coûts pour prédire l'énergie et (ii) la proposition des techniques basées sur les modèles de coûts pour réduire l'énergie. Cette situation est tout à fait semblable à celle vécue dans les approches de conception traditionnelles axées sur l'amélioration de la performance, où la plupart des optimiseurs de requêtes des SGBD open source [11] et commerciale [12] utilisent des approches fondées sur les modèles de coûts. Dans ces approches, un optimiseur basé sur les coûts estime le coût de chaque plan d'exécution possible en appliquant des formules heuristiques utilisant un ensemble de statistiques relatives à la base de données (taille des tables, des indexes, longueur de tuples, les facteurs de sélectivité de jointure et prédicats de sélection, la taille des résultats intermédiaires, etc.) et au matériel (taille du tampon mémoire, la taille de page, etc.).

3.3.3.1 Définition des modèles de coûts

Dans [282, 283], les auteurs proposent une extension du modèle de coût de PostgreSQL pour prédire la consommation d'énergie d'une seule requête exécutée d'une façon isolée (pas de parallélisme inter-requêtes). Un profil de puissance statique pour chaque opération de base de données basique est défini dans la phase du traitement des requêtes. Le coût de l'énergie d'un plan peut être calculé à partir des opérations de base de données SQL, comme le coût de la puissance du processeur pour accéder à un tuple,

11. http://www.postgresql.org/docs/current/static/planner-optimizer.html
12. http://docs.oracle.com/cd/B10500_01/server.920/a96533/optimops.htm#51003

coût du disque pour la lecture/écriture d'une page et ainsi de suite, et par l'intermédiaire de différentes méthodes d'accès et d'opérations de jointure, en utilisant une technique de régression linéaire simple. La formule générale pour calculer l'énergie un opérateur x est :

$$\hat{E}_x = W_{cpu} \times N_{tuples}^m \times N_{tuples} + \bar{W}_{E/S} \times N_{pages} \tag{3.5}$$

Où W_{cpu} et $\bar{W}_{E/S}$ sont le coefficient de consommation d'énergie d'un tuple traité par le CPU (N_{tuples}), et le coefficient statique d'une page traitée en E/S (N_{pages}), et m est un coefficient de modèle pour la consommation d'énergie du CPU ($m = 0,5$ après la régression). Les auteurs adaptent leur modèle statique pour des charges de requêtes dynamiques à l'aide d'un mécanisme de contrôle de rétroaction pour mettre à jour périodiquement les paramètres du modèle à travers des mesures d'énergie en temps réel.

Dans [152], une recherche profonde dans la modélisation de la puissance de pic des opérations de requête est donnée. Un modèle de plans d'exécution sur la base de pipeline a été développé pour identifier les sources de consommation d'énergie de pic pour une requête isolée et de recommander des plans avec une faible puissance de pic. Pour chacun de ces pipelines, une fonction mathématique est appliquée, qui prend en entrée les taux et les tailles des données circulant à travers les opérations de pipelines. En tant que sortie, une estimation de la consommation d'énergie de pic est donné. Les auteurs ont utilisé la technique de régression multiple affine par morceaux pour construire leurs modèles de coût. Pour un nœud de base B (par exemple : un accès séquentiel), le débit est calculé comme :

$$Taux_B = \frac{Entre_B}{TempsDisque_B} \tag{3.6}$$

Où $Entre_B$ indique la taille des données extraites du disque par B et $TempsDisque$ est le temps de transfert du disque.

$$\begin{aligned} &\text{soit} \quad Base_N = SousArbre_N^{PL} \cap Base^{PL} \\ &\text{alors} \quad Taux_N = \frac{\sum_{i \in Source_N^{PL}} Sortie_i}{max_{x \in Base_N} TempsDisque_x} \end{aligned} \tag{3.7}$$

N est un nœud générique en aval dans le pipeline, $SousArbre_N^{PL}$ est le sous-arbre du pipeline PL avec la rince N, $Base^{PL}$ est l'ensemble des nœuds de bases dans le pipeline PL, $Source_N^{PL}$ est l'ensemble des nœuds dans le pipeline PL qui fournissent directement des entrées au nœud N, et $Sortie_i$ désigne la taille de la donnée en sortie par le nœud i. Les tailles des données sont estimées à partir de catalogue de la base de données.

Dans le même sens, le travail de [155] propose un framework pour un traitement éco-énergétique des requêtes dans les bases de données. Il étend les plans de requêtes produites par l'optimiseur de requête traditionnelle avec une prédiction de la consommation d'énergie pour certains opérateurs de bases de données spécifiques comme la sélection, projection et la jointure en utilisant la technique de régression linéaire. L'Équation (3.8) montre le modèle le plus précis proposé pour estimer la puissance moyenne du système pendant l'exécution d'une requête isolée. Le modèle est défini par trois types de variables :

1. Temps T

2. Paramètres de requête Q : nombre d'instructions de processeur (I_Q), lecture de page disque (R_Q), écriture de page disque (W_Q) et nombre d'accès au mémoire (M_Q)

3. Paramètres du système : CPU (C_{cpu}), lecture disque (C_R), écriture disque (C_W), accès mémoire (C_{mm}), reste du système (C_{autres}). Ces paramètres quantifient la puissance des composants matériels et sont obtenus par une régression.

$$P_{moy} = C_{cpu} \times \frac{I_Q}{T} + C_R \times \frac{R_Q}{T} + C_W \times \frac{W_Q}{T} + C_{mm} \times \frac{M_Q}{T} + C_{autres} \tag{3.8}$$

L'intuition derrière ce modèle est de calculer la sommation de la puissance moyenne du processeur ($C_{cpu} \times \frac{I_Q}{T}$), du lecture disque ($C_R \times \frac{R_Q}{T}$), d'écriture disque ($C_W \times \frac{W_Q}{T}$), du mémoire ($C_{mm} \times \frac{M_Q}{T}$) et du reste du système (C_{autres}) pendant l'exécution de la requête Q.

L'étude de [215] tente de modéliser l'énergie et la puissance de pic des requêtes de sélection simples sur les relations individuelles en utilisant une régression linéaire. Le modèle développé pour la consommation d'énergie d'une requête Q avec une relation R est le suivant :

$$E_Q = \beta_0 + \beta_1\ S + \beta_2\ SF_p(R)\ S + \beta_3\ |R|\ SF_p(R) + \beta_4\ |R| < R > SF_p(R) \tag{3.9}$$

Où β_i sont des coefficients de régression et $|R|, < R >, SF_p(R)$ et S sont respectivement : la cardinalité de la relation R, le nombre de colonnes de R, la sélectivité du prédicat p dans R et le nombre de serveurs utilisés. Les requêtes sont supposées exécuter d'une façon isolée.

Alors que les travaux cités considèrent seulement une architecture de base de données centralisée, l'étude de Lang *et al.* [158] a proposé un modèle de coût pour une architecture de base de données en grappe de serveurs. Le modèle prend en compte les configurations de serveur (par exemple, le CPU, les disques et la bande passante du réseau) et les paramètres de traitement de données (par exemple, la sélectivité des prédicats, la taille de la table de hachage), et prédit les performances de traitement des données et l'EE pour un opérateur de jointure par hachage. L'erreur maximale rapportée est inférieure à 10%.

3.3.3.2 Techniques basées sur des modèles de coûts

Les modèles de coûts proposés pour prédire l'énergie sont ensuite utilisés dans diverses techniques pour optimiser la consommation d'énergie de base de données. Les auteurs de [156] ont proposé un mécanisme pour améliorer l'EE des requêtes en introduisant des retards explicites, en utilisant l'agrégation des requêtes pour tirer parti des composants de requêtes communs dans la charge. Le retard explicite permet de grouper les requêtes dans une file d'attente en mémoire tampon, ce qui conduit alors à une évaluation plus efficace de l'ensemble du lot par des techniques d'optimisation multi-requêtes. Pour la comparaison de l'énergie et du temps, la métrique utilisée est le Produit Énergétique/Délai (EDP) [159], qui est le produit de l'énergie et du délai :

$$EDP = \text{énergie}(Joules) \times \text{temps}(sec) \tag{3.10}$$

Après la substitution, la formule finale de comparaison est :

$$EDP = V^2/F \tag{3.11}$$

Où V est le voltage et F est la fréquence du processeur. Les résultats d'expérimentations pour les requêtes de sélection simples montrent que cette technique peut être utilisé pour trouver un compromis entre la performance et l'énergie, ce qui réduit la consommation d'énergie de 54% contre 43% d'augmentation dans le temps de réponse moyen.

Le travail de [207] indique que l'ordonnancement éco-énergétique de plusieurs requêtes peut améliorer l'EE de certains opérateurs SQL basique qui peuvent être exécutées en parallèles, tels que les agrégations et les algorithmes d'accès aux données. La métrique utilisée est pour choisir le meilleur compromis entre le temps et l'énergie est EDP (Équation (3.10)). Les résultats expérimentaux montrent que l'application des méthodes proposées, l'EE est augmenté jusqu'à 4x. Dans le même sens, les auteurs de [140] proposent un système de gestion de la mémoire cache éco-énergétique pour les bases de données en temps réel sur la mémoire flash. La technique introduite crée une partition logique du pool global de la mémoire tampon dans des pools de mémoire tampon de lecture et d'écriture, et le contrôle de rétroaction dynamique ajuste la taille des pools pour satisfaire les objectifs de performance et de consommation d'énergie.

Le travail de [155] a montré que le traitement d'une requête aussi vite que possible ne s'avère pas toujours être la façon la plus économe en énergie dans un SGBD. Sur la base de leurs modèles de coût proposé pour prédire l'énergie, ils choisissent des plans de requêtes qui réduisent la consommation d'énergie et répondent aux objectifs de performance traditionnels. Ce choix est fait manuellement. Les résultats montrent que les économies d'énergie peuvent atteindre jusqu'à 23% pour une augmentation de 5% du temps de réponse en ce qui concerne les requêtes équi-jointure.

Dans une tendance similaire, les auteurs de [283] intègrent leurs modèles de coût dans les paramètres du PostgreSQL pour choisir les plans de requêtes ayant des valeurs de faible puissance au cours de la phase d'optimisation. La méthode d'évaluation des plans pour les deux nouveaux objectifs, l'énergie E et le temps T, est une variation de la méthode EDP. Plus formellement, le coût C d'un plan est :

$$C = PT^n = ET^{n-1} \tag{3.12}$$

Où n est un coefficient qui reflète l'importance relative de P et T et $E = PT$ est le coût énergétique total d'un plan. Ils considèrent trois scénarios, lorsque $n = \infty$, seul le coût du temps est considéré ; pour $n = 0$, l'optimisation est pour la consommation d'énergie ; et lorsque $n = 1$, le temps et la puissance sont prises en compte. Les auteurs fournissent une analyse approfondie du profil de puissance pour les requêtes typiques de benchmarks TPC à fin d'identifier les caractéristiques de la consommation de ressources pendant le traitement des requêtes. Les résultats expérimentaux montrent que les économies d'énergie de l'ordre de 11% à 22% peuvent être obtenus en équipant le SGBD avec un optimiseur de requête qui sélectionne les plans de requêtes basées sur les besoins de temps de traitement et de la puissance estimés.

Dans [152], en utilisant leurs modèles de coût déjà mis en place, les auteurs ont montré des possibilités dans la création de plans de requêtes avec des compromis intéressants entre la puissance pic et le temps d'exécution. La création des plans est réalisée manuellement dans leur travail. Les résultats préliminaires indiquent que pour certaine requête dans les benchmarks TPC, la puissance de pic peut être réduite de

35 *Watts*, avec une pénalité de 80% en performance. En outre, les auteurs discutent des extensions de leur framework pour exploiter des techniques telles que les charges multi-requêtes.

Le travail de [265] mène une étude en profondeur dans l'analyse de l'EE d'un serveur de base de données sous différents paramètres de configuration, en faisant valoir que les avantages obtenus dans les systèmes de base de données sont de petite taille. Cependant, dans leur environnement, ils donnent de l'importance à la puissance du processeur plutôt que la puissance active globale du système. Néanmoins, dans la pratique, il existe des requêtes qui sont intensive en E/S. Considérant que le CPU à la place de l'ensemble des composants peut réduire considérablement la marge d'amélioration d'EE [272]. Dans une autre ligne de recherche, les auteurs de [158, 226] ont abordé le cas des bases de données parallèles et distribués sur un cluster de nœuds pour économiser l'énergie et concevoir des SGBDs avec une énergie proportionnelle [13]. À cette fin, les auteurs ont exposé les décisions de conception qui doivent-être pris en considération, tels que la taille de cluster, l'allocation de stockage et le traitement de données dynamique sur les nœuds, et l'augmentation/réduction des nœuds du cluster en fonction des changements de la charge de travail. Des résultats expérimentaux sont présentés pour motiver leurs propositions, dans [158], la métrique de comparaison entre l'énergie et le temps utilisé est EDP.

3.3.3.3 Base de données en tant que référentiel pour le stockage et la gestion des données de l'énergie

Une autre approche vers les centres de données intelligents et verts est la gestion des données énergétiques. Les données temporelles de la demande d'énergie provenant de multiples sources hétérogènes (par exemple, les systèmes de contrôle et d'acquisition de données de surveillance, les compteurs intelligents, les systèmes de production d'énergie renouvelable, etc.) sont appelées *séries chronologiques* [95]. Par conséquent, la prévision, l'agrégation, la planification et la désagrégation des données énergétiques en séries chronologiques est essentielle pour optimiser le flux d'énergie dans un réseau intelligent et produit généralement une valeur commerciale ajoutée à l'entreprise [240]. Par exemple, un rapport de recherche montre qu'une augmentation de 1% de la précision de prévision pourrait donner une conservation des coûts d'exploitation estimés d'environ 14,14 millions de dollars [240].

Des projets comme MIRABEL vise à développer une approche sur un niveau conceptuel et infrastructure qui permet aux entreprises de distribution d'énergie d'équilibrer les sources d'énergie renouvelables disponibles et la demande actuelle [239]. Le défi est que la production d'énergie dépend de facteurs externes (vitesse et direction du vent, la quantité de lumière du soleil, etc.). Par conséquent, la puissance disponible ne peut qu'être prédit, mais non pas planifier, ce qui le rend assez difficile pour les distributeurs d'énergie à inclure efficacement les sources d'énergie renouvelables dans leur planification quotidienne. Le projet prévoit que l'utilisation de ses techniques de prévision, se traduira par des réductions de demande de pointe d'environ 8-9% pour la totalité de la grille [239]. Dans le même projet, les technologies de base de données sont utilisées pour le stockage et l'interrogation des valeurs anciennes et (la prévision) futures des séries chronologiques [145, 91]. L'utilisation des technologies de base de données permet des optimisations qui améliorent l'efficacité, la cohérence et la transparence du processus global de prévision. L'idée principale est de représenter de façon compacte des séries

13. La proportionnalité est la possibilité de restreindre la consommation d'énergie du matériel et du logiciel à la charge de travail réelle.

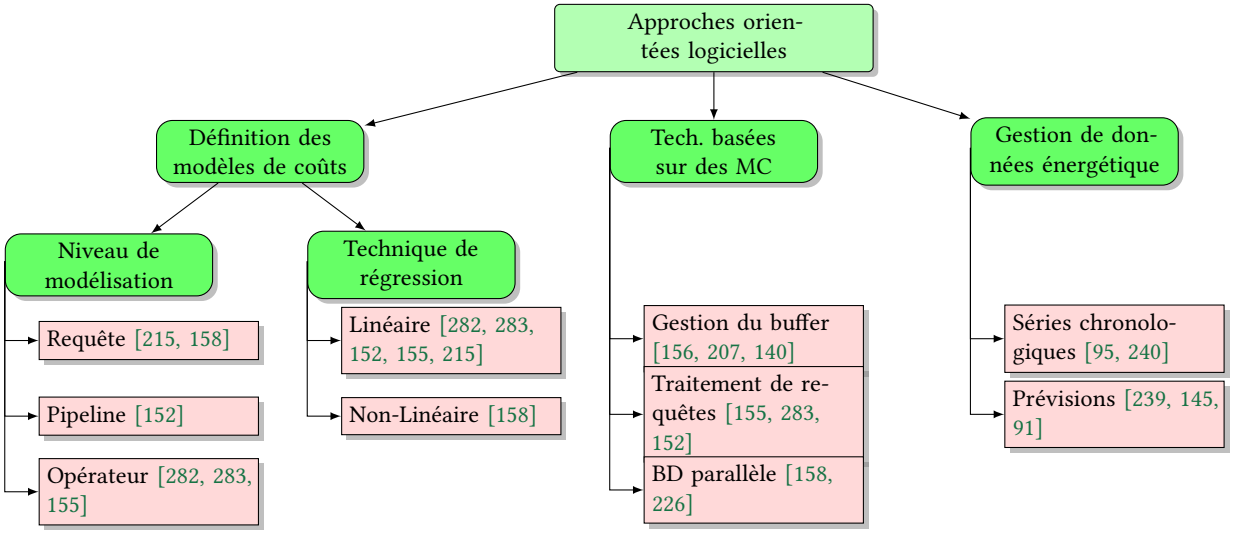


FIGURE 3.5 – Classification des approches d'EE orientées logicielles.

chronologiques en tant que modèles et les intégrer dans un SGBD comme un nouveau modèle de données. De plus, ils proposent des requêtes de prévisions pour l'accès aux données de séries chronologiques, ce qui améliore considérablement les performances.

3.3.4 Bilan et discussion

Dans cette section, nous avons présenté les travaux d'EE dans les base de données. Ces travaux peuvent être divisés en deux catégories : approches matérielles set approches logicielles (voir Figure 3.5). Nous avons décri les problèmes majeurs de la première catégorie dans la Section 3.3.2. Pour la deuxième catégorie, les travaux se focalise soit sur la création des modèles de coût, soit sur la proposition des techniques basées sur les modèles de coûts pour minimiser l'énergie. Nous avons constaté que les modèles de coût ont plusieurs problèmes :

- Le niveau de modélisation : Comme nous l'avons mentionné, chaque travail choisi son propre niveau de modélisation (requête, pipeline ou opérateur), nous proposons de revoir la décision sur ce niveau pour concevoir un modèle coût ayant les meilleurs résultats d'estimation d'énergie.
- La portabilité de modèle : Les paramètres de modèles dans les travaux précédents sont basés sur le type et l'implémentation des opérateurs (par exemple, l'accès *séquentiel* ou jointure par *hachage*). Ceci limite la portabilité du modèle sur un nouveau SGBD ou un nouvel opérateur. Nous proposons de choisir des paramètres qui ne dépendent pas du SGBD.
- La technique de modélisation : La plupart des travaux utilisent une régression linéaire simple pour la partie modélisation. Nous proposons de revoir ce choix à fin d'adapter la meilleure technique qui décrit le comportement de l'énergie pendant l'exécution d'une requête.
- Requêtes isolées : La façon d'exécution des requêtes supposée dans les travaux précédents est *isolée*, une seule requête en encours d'exécution dans un SGBD à un moment donné. Nous proposons de prendre en considération le cas le plus général où un ensemble de requêtes en train

de s'exécuter d'une façon *concurrente*.

Pour les méthodes de minimisation d'énergie, la plupart des travaux se basent sur le traitement de requêtes. Cette technique est très importante dans un SGBD et peut avoir des effets considérables sur l'énergie. Par contre, il existe plusieurs d'autres composantes dans la base de données qui peuvent contribuer pour l'amélioration de l'EE. Nous proposons de les investiguer et exploiter. D'autres part, la méthode de comparaison et le choix des compromis entre le temps et l'énergie utilisée dans les travaux précédents se font manuellement par de le DBA ou à travers la méthode EDP. C'est deux méthodes sont très simples et ne permettent pas d'exploiter l'espace de recherche des solutions (par exemple, l'ensemble des plans d'exécutions d'une requête) pour offrir une variété de choix aux DBA. Cependant, nous proposons d'utilisé les AEMO décrits dans le chapitre précédent pour faire ce choix.

3.4 Vers des bases de données vert

Ces dernières années, l'EE est devenue l'une des exigences de conception les plus importantes pour les systèmes informatiques modernes, allant des serveurs uniques aux centres de données et aux Cloud, car ces équipements continuent de consommer une énorme quantité d'énergie électrique. Outre les coûts d'exploitation élevés supportés par les ressources informatiques, des émissions importantes de CO_2 sont générées dans l'environnement. Par exemple, les infrastructures informatiques représentent actuellement environ 2% de l'empreinte carbone totale [25]. Sans avoir développer des techniques et des algorithmes économisant l'énergie des ressources informatiques, la contribution des systèmes informatiques à la consommation mondiale d'énergie et aux émissions de CO_2 devrait augmenter rapidement. Cela est évidemment inacceptable à l'ère du changement climatique et du réchauffement planétaire. Pour faciliter l'amélioration des situations dans ce domaine, il est essentiel d'examiner et d'étudier l'ensemble des connaissances existantes. Ce chapitre a présenté une taxonomie et une étude sur les différentes techniques d'amélioration de l'EE dans les systèmes informatiques. Les approches récentes ont été discutées et classées au niveau matériel, système d'exploitation et application. Nous avons soutenu que ces approches n'étaient qu'une partie de la solution pour contrôler et réduire l'augmentation des coûts énergétiques dans les grands centres de données. Les bases de données joueront finalement un rôle important dans l'optimisation de l'EE.

Les bases de données représentent aujourd'hui l'un des principaux défis en matière d'EE. La réduction d'énergie dans les BD est d'une grande importance économique, car ils sont utilisés par la plupart des environnements informatiques des entreprises. Dans un centre de données typique, la majorité des ressources informatiques sont dédiées aux serveurs de bases de données, ce qui en fait le plus grand consommateur d'énergie parmi toutes les applications logicielles déployées. Nous avons détaillé dans ce chapitre les techniques proposé pour l'EE des BDs et nous avons identifié les lacunes de chacun et proposé des méthodes pour les améliorer.

Sur la base de la discussion dans ce chapitre et le chapitre précédent, nous concluions que l'énergie est influencée par toutes les phases de conception et d'exploitation des bases de données. Dans ce contexte, nous avons identifié trois dimensions principales qui doivent être prises en compte lors de l'intégration de l'énergie dans les bases de données :

— Un modèle de coût précis, potable et robuste est une dimension essentielle pour bien estimer la consommation d'énergie dans une BD.

- Les structures d'optimisation offertes par le SGBD qui doivent être exploitées par DBA pour assurer la performance de la charge de requêtes. Ces structures incluent l'optimisation de requêtes, la sélection de vue, la fragmentation, etc. L'objectif est de choisir les bons algorithmes et paramètres pour améliorer l'EE.
- La plate-forme de déploiement utilisée, y compris le type de SGBD et le matériel. La possibilité de changer les états d'activité des composantes matériels directement à travers les SGBD peut donner aux DBA et aux utilisateurs une grande opportunité pour minimiser la consommation énergétique des systèmes.

Dans la suite de notre thèse, nous mettons en avant nos trois propositions citées ci-dessus : la modélisation de l'énergie dans les bases de données, l'intégration de ce modèle dans la phase de traitement de requêtes et la sélection des techniques d'optimisation dans la phase de conception physique. Dans chaque proposition, nous modélisons ces problématiques comme étant des problèmes multi-objectifs à résoudre.

Troisième partie

Contributions

Chapitre

4 Modèle de coût énergétique pour des requêtes isolées et concurrentes

« Passion is energy. Feel the power that comes from focusing on what excites you. »
— Oprah Winfrey

Sommaire

4.1 Introduction

La consommation d'énergie dans les BD est devenue un vrai problème pour la communauté d'informatique verte ainsi que pour les scientifiques et les administrateurs de BD. L'énergie doit être optimisée à plusieurs niveaux : matériel, système d'exploitation et SGBD. Toutefois, l'estimation du coût énergétique dans les opérations de BD a une importance majeure dans la conception de BD minimisant l'énergie. En effet, c'est la première étape à réaliser dans la plupart des techniques d'optimisation logicielles. Les solutions existantes (voir Chapitre 3) proposent des approches limitées pour la mesure de l'énergie d'une manière efficace et précise. Celles-ci s'avèrent simples, ne considérant pas les requêtes complexes, ou ne sont pas portables dans différentes configurations n'ayant pas de modèle mathématiques robuste.

Nous présentons dans ce chapitre le modèle de coût que nous avons proposé pour estimer l'énergie consommée par une requête donnée. Nous commençons par présenter les motivations d'utilisation d'un modèle de coût énergétique dans notre approche. Nous présentons ensuite les hypothèses que nous avons considérées pour établir ce modèle de coût. Nous donnons la démarche de construction de notre modèle, à savoir l'identification des paramètres, le niveau de modélisation et la technique de régression. Nous proposons deux types de modèles de coûts : (1) un modèle de coût pour les requêtes exécutées d'une manière isolée et (2) un pour le cas des requêtes concurrentes ou parallélisées. Enfin, nous évaluons notre modèle contre des benchmarks réels avec différentes configurations physiques pour montrer l'utilité et l'efficacité de notre approche.

4.2 Pourquoi un modèle de coût ?

Les modèles de consommation d'énergie sont très importants pour de nombreux systèmes où l'on exige une efficacité énergétique [214]. La conception du matériel informatique utilisé par les centres de données nécessite un modèle de coût, car il est impossible de construire des systèmes physiques afin d'évaluer tous les effets des choix de conception sur la consommation d'énergie. Dans l'utilisation quotidienne des systèmes informatiques, les utilisateurs et les opérateurs de centres de données doivent savoir et comprendre comment leurs modes d'utilisation affectent la consommation d'énergie de la machine afin de maximiser son efficacité énergétique, que ce soit dans un seul système ou sur un ensemble de systèmes. L'utilisation d'un matériel de mesure n'est pas une solution, car il ne peut pas prédire l'énergie, et il est généralement coûteux et inflexible face au changement des systèmes. Finalement, les modèles de coût permettent de développer des techniques d'optimisation de la consommation d'énergie.

Une modélisation d'énergie idéale adaptée à une utilisation dans les BD devrait répondre à plusieurs critères de conception. La modélisation basée seulement sur le matériel ne peut pas répondre à ces critères, car elle ne peut pas prédire la consommation d'énergie des futures requêtes, et elle ne décrit pas le lien entre l'utilisation des ressources et la consommation d'énergie, qui est nécessaire pour l'optimisation d'efficacité énergétique des BD. Les principaux critères sont :

- **La précision.** Le modèle doit être suffisamment précis lors des estimations d'énergie. Une erreur importante dans les résultats risque de générer de mauvaises décisions lors de la phase d'utilisation du modèle.
- **La portabilité et la robustesse.** Le modèle doit maintenir une grande précision indépendam-

ment des variations des systèmes matériels ou logiciels, et les caractéristiques de la charge de requête. La génération du modèle pour un nouveau système doit être facile et ne requiert pas une re-conception.
- **La rapidité.** Le modèle doit générer des prédictions assez rapidement pour être exploitées dans les optimisations en temps réel.
- **La simplicité.** Le modèle doit être simple en terme de nombre de paramètres d'entrée et en complexité algorithmique. Il ne doit pas impliquer des paramètres autres que ceux utilisés par les SGBD traditionnels.
- **La faible surcharge.** Le processus de modélisation ne doit ni interférer avec les activités normales du SGBD, ni impliquer des modifications importantes sur le matériel. Il ne devrait pas générer une surcharge du système.

4.2.1 Approches de modélisation

Dans la littérature, il existe deux approches pour modéliser l'énergie d'un système : (1) une approche *analytique* (boite blanche) ou (2) une approche *expérimentale* (boite noire) [212]. Chaque approche a ses avantages et ses inconvénients. L'approche analytique modélise un système dont on peut prévoir le fonctionnement interne car on connaît les caractéristiques de fonctionnement de l'ensemble des éléments qui le composent. La connaissance détaillée des éléments du système rend le modèle plus précis. Par contre, cette approche est complexe à réaliser, puisqu'elle implique un grand nombre de paramètres et requiert un temps de calcul important et une grande capacité de calcul et de maintenance face à l'évolution du système.

Une approche expérimentale fournit quant à elle la représentation d'un système sans considérer ni son fonctionnement interne, ni son architecture. Elle se base seulement sur les données des expérimentations et les techniques d'identification. La mise en œuvre de ce types d'approches est simple et rapide puisqu'elle ne requiert pas une connaissance approfondie du système. En revanche, la non-connaissance des composantes du système et leur comportement influence sur la qualité des résultats du modèle final. Le modèle peut avoir besoin d'une nouvelle adaptation si le système change.

Nous proposons de combiner les deux approches afin de profiter des avantages de chacune, en traitant le système comme une *boite grise*. Cette approche prend en compte une partie du fonctionnement interne du système d'un côté et se base sur des données expérimentales de l'autre côté. Dans la pratique, nous allons lancer des expérimentations pour mesurer l'énergie du système et étudier sa variation suivant les caractéristiques de la charge de requêtes, afin de trouver les paramètres et les fonctions mathématiques clés qui décrivent la consommation d'énergie du système. La Figure 4.1 résume les trois approches de modélisation citées ci-dessus.

4.2.2 Niveau de modélisation

Lorsqu'une requête SQL est soumise à un SGBD, elle sera traduite en un arbre d'opérateurs algébriques puis en plan d'exécution. L'optimiseur de requêtes applique des fonctions analytiques pour trouver le meilleur plan d'exécution. Ensuite, la requête sera exécutée suivant un mode itératif (pipeline), où un ensemble d'opérateurs algébriques s'exécutent en même temps lorsqu'il y a des données à traiter.

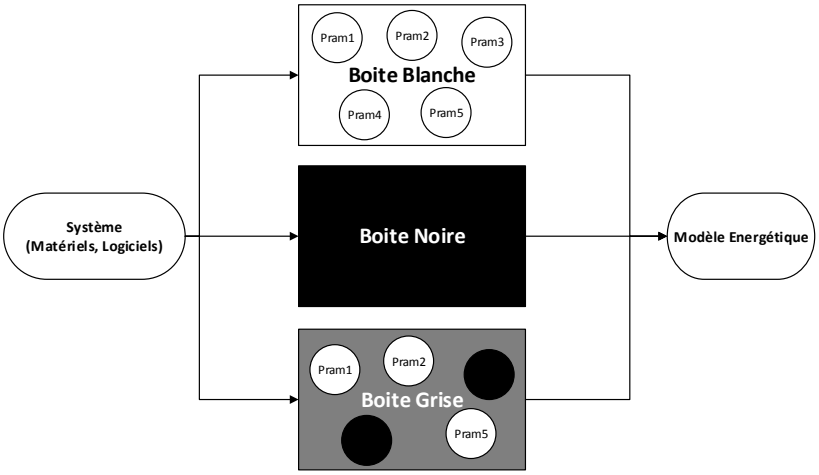


FIGURE 4.1 – Les approches de modélisation d'énergie.

Intuitivement, la modélisation d'énergie d'une requête, peut se faire à trois niveaux :

1. **Niveau requête.** Considérer les caractéristiques de la requête elle-même, telles que le nombre d'entrées/sorties requises pour exécuter la requête. Ce niveau est facile à mettre en œuvre mais ne garantis pas la précision désirée.
2. **Niveau pipeline.** Considérer les caractéristiques de l'ensemble d'opérateurs qui s'exécutent simultanément. Le défi dans ce niveau est la détermination de ces opérateurs.
3. **Niveau opérateur.** Considérer les caractéristiques de l'opérateur individuellement. La modélisation à ce niveau exige des détails sur les opérateurs et leurs implémentations, ce qui rend le processus complexe.

Le bon choix du niveau de modélisation est crucial pour la qualité du modèle final. Ce choix peut se faire à l'aide d'une étude expérimentale comme décrit dans la section précédente.

4.2.3 Mode d'exécution des requêtes

La plupart des SGBDs modernes sont des systèmes multi-utilisateurs où plusieurs utilisateurs peuvent se connecter à la BD et exécuter des requêtes de façon concurrente. Il incombe au SGBD de gérer la concurrence d'accès aux données, et de garantir la fiabilité des résultats des requêtes. Comme nous allons montrer, la consommation énergétique pour une seule requête n'est pas forcément la même pour un ensemble de requêtes qui s'exécutent en même temps en parallèle. Par conséquent, il existe deux modes d'exécution de requêtes à considérer dans la phase de conception d'un modèle de coût énergétique : (1) le mode isolé et (2) le mode concurrent.

1. **Mode isolé.** Une seule requête est en train d'être exécutée de façon isolée par le SGBD dans une instance donnée. Ce mode est la base de toute conception d'un modèle de coût énergétique due à sa complexité réduite.
2. **Mode concurrent.** Plusieurs requêtes sont en train d'être exécutées de façon concurrentes par le SGBD dans une instance donnée. Ce mode est plus générique que le premier et plus utile pour

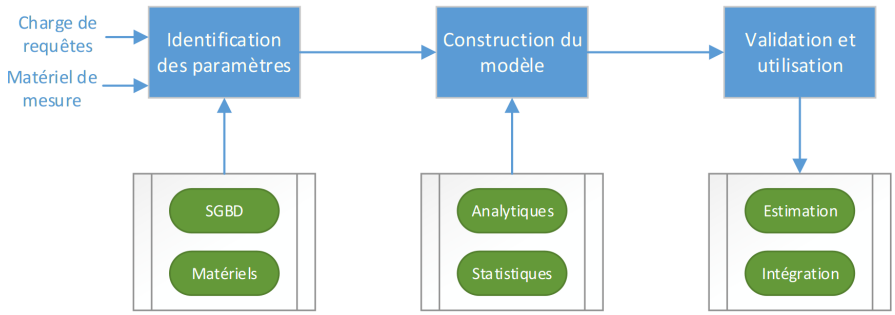


FIGURE 4.2 – Les étapes de développement de notre modèle de coût énergétique.

de nombreuses tâches de gestion de BD, à l'exemple du contrôle d'admission, l'ordonnancement des requêtes et de contrôle d'exécution. Cependant, la conception de ce mode est plus complexe.

4.3 Démarche pour la construction d'un modèle de coût énergétique

Dans cette section, nous présentons notre méthodologie pour créer un modèle de coût qui estime la consommation d'énergie d'une seule requête SQL. Nous adoptons une approche logicielle basée sur une étude empirique pour trouver les paramètres clés ayant un impact sur l'énergie, et pour identifier le niveau adéquat pour modéliser l'énergie des requêtes. L'étude empirique permet aussi de sélectionner l'algorithme mathématique qui modélise mieux le comportement de l'énergie des requêtes SQL. Les principales étapes pour développer notre modèle de coût sont (cf. Figure 4.2) :

- **Identification des paramètres.** Afin de réduire la consommation d'énergie d'un système, nous devons d'abord mesurer la consommation d'énergie de ses composantes et identifier là où la plupart de l'énergie est dépensée. Ceci est la tâche de la phase d'extraction des paramètres du modèle. Elle prend comme entrée le matériel utilisé, le SGBD et la charge de requêtes.
- **Construction du modèle.** Les paramètres d'entrée sélectionnés précédemment sont utilisés pour construire le modèle de consommation d'énergie en utilisant des techniques *analytiques* ou *statistiques* telles que la régression, l'apprentissage automatique, etc. L'un des principales difficultés de cette étape est que certains paramètres importants du système tels que la consommation d'énergie d'un composant particulier, ne peuvent pas être mesurés directement. Les méthodes analytiques classiques peuvent ne pas produire des résultats précis dans de telles situations, les techniques d'apprentissage automatique peuvent donner de meilleurs résultats.
- **Validation et utilisation.** Le modèle doit être validé selon différents scénarios pour son adaptation à ses fins prévues. Il peut être utilisé a postériori comme une base pour prédire la consommation d'énergie des requêtes. Ces estimations peuvent ensuite être utilisées pour améliorer l'efficacité énergétique des SGBD, par exemple en incorporant le modèle dans des techniques d'optimisation telles que la conception physique, l'ordonnancement des requêtes, l'ajustement dynamique de la fréquence de la tension (DVFS), l'amélioration des algorithmes et ainsi de suite.

4.3.1 Nos hypothèses

Dans notre étude, et à cause de la complexité du problème traité, nous limitons les possibilités liées à l'environnement. La multitude des environnements (p. ex. centralisé, distribué et parallèle), des SGBD (p. ex. relationnel, NoSQL ou NewSQL) ou encore des requêtes (p. ex. sélection et mise à jour) rend la démarche plus complexe. Dans notre travail, nous considérons les hypothèses suivantes :

1. L'environnement de travail est centralisé, la BD est déployée dans une seule station de travail. Par conséquent, le coût de communication est ignoré.
2. Nous ne considérons pas des requêtes de définition de données (p. ex. `CREATE`, `DROP`, `ALTER`) ou encore de requêtes de manipulation de données (p. ex. `INSERT`, `DELETE`, `UPDATE`). Les requêtes considérées sont de type transactionnel et analytique, ayant la forme suivante :

```
SELECT [DISTINCT ou ALL] colonnes
FROM tables
[WHERE predicats]
[GROUP BY expression]
[HAVING condition]
[ORDER BY colonnes]
```

3. Les requêtes sont exécutées de façon isolée et concurrente (c-à-d parallélisme inter-requêtes).
4. Le cache est vidé entre l'exécution des requêtes.

4.3.2 Identification des paramètres

La première étape de notre approche est l'identification des paramètres clés de notre modèle qui influencent sur la consommation d'énergie des requêtes. L'identification est réalisée suivant une approche d'apprentissage automatique. Cette stratégie expérimentale est utilisée lorsque les paramètres du $\mathcal{BNF}$ sont inconnus au préalable. Dans cette stratégie, nous testons l'exécution de plusieurs requêtes simples et complexes, isolées et concurrentes, dans différentes BD et de tailles différentes, puis nous observons les paramètres liés à la stratégie d'exécution des requêtes, les caractéristiques de la BD et le matériel utilisé.

4.3.2.1 Effet de stratégie d'exécution des requêtes

Pour étudier l'effet de la stratégie d'exécution des requêtes sur la consommation d'énergie, nous commençons par montrer un exemple de la requête *Q9* du benchmark TPC-H [5]. La Figure 4.3 montre la syntaxe de cette requête qui détermine le bénéfice de la vente d'une ligne de pièces, groupé par la nation et l'année des fournisseurs.

Les ensembles de données et la configuration du système que nous avons utilisés dans nos expérimentations peuvent être trouvés dans la Section 4.4. La Figure 4.4a montre la consommation d'énergie active pendant l'exécution de la requête qui a débuté à la $10^{\text{ème}}$ seconde et a fini à la 143ème seconde. Nous pouvons y distinguer trois segments différents qui s'avèrent correspondre à la segmentation en *pipelines*. Ces derniers permettent l'exécution simultanée d'une séquence contiguë d'opérateurs SQL souvent implémentés selon un modèle itératif [73].

```sql
select nation, o_year, sum(amount) as sum_profit
from
  (
    select
      n_name as nation, extract(year from o_orderdate) as o_year,
      l_extendedprice * (1 - l_discount) - ps_supplycost * l_quantity as amount
    from
      part, supplier, lineitem,
      partsupp, orders, nation
    where
      s_suppkey = l_suppkey and ps_suppkey = l_suppkey
      and ps_partkey = l_partkey and p_partkey = l_partkey
      and o_orderkey = l_orderkey and s_nationkey = n_nationkey
      and p_name like '%sandy%'
  ) as profit
group by nation, o_year
order by nation, o_year desc;
```

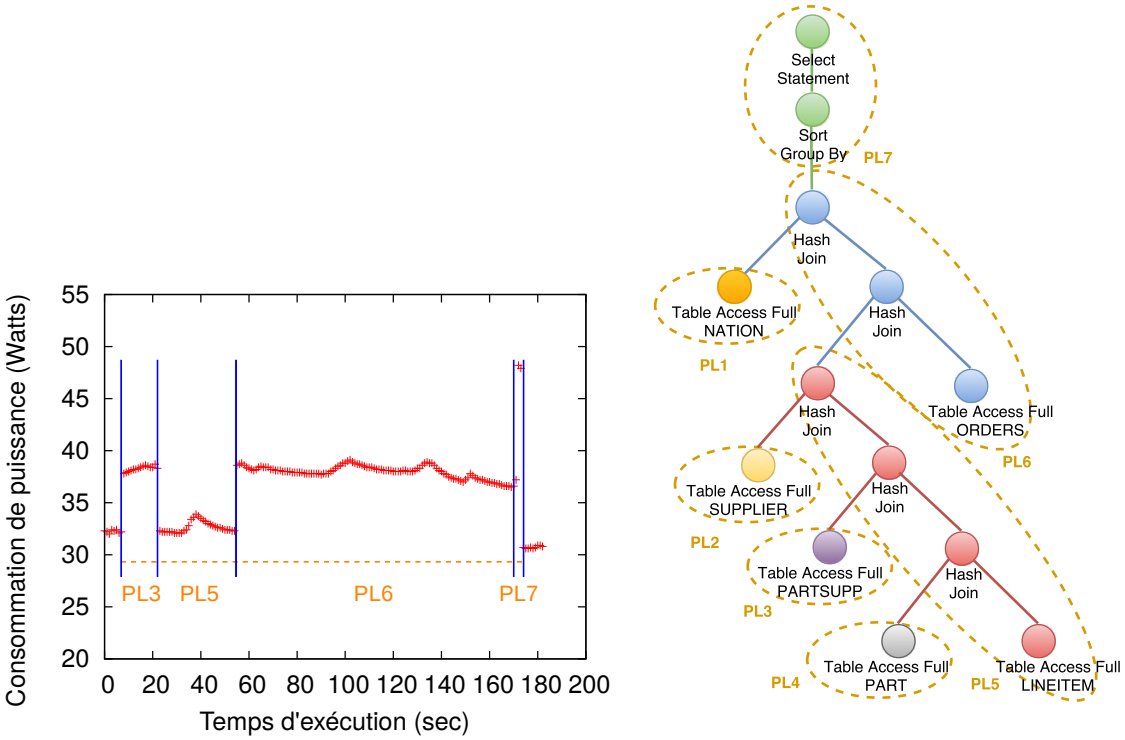
FIGURE 4.3 – La requête *Q9* issue du benchmark TPC-H.

La segmentation en pipelines du plan d'exécution de la requête *Q9* est montrée dans la Figure 4.4b. Il y a 7 pipelines dont l'ordre d'exécution est déterminé par leurs opérateurs *bloquant* (p. ex. *PL*6 ne peut pas commencer avant que *PL*5 soit terminé). Bien que la requête a sept pipelines différents, seulement quatre sont importants dans notre discussion (les autres terminent très rapidement). La détermination des segments énergie-pipelines est effectuée à l'aide du *module de surveillance SQL temps réel* intégré dans le SGBD [231], qui nous donne en temps réel les statistiques à chaque étape du plan d'exécution avec le temps écoulé. Sur la base de ces informations, nous avons segmenté l'intervalle de l'énergie en pipelines correspondants.

Si l'on compare les intervalles de changement de la consommation d'énergie avec le changement des pipelines dans la Figure 4.4, nous pouvons voir que *lorsqu'une requête passe d'un pipeline à l'autre, sa consommation d'énergie également change.* Au cours de l'exécution d'un pipeline, la consommation d'énergie a généralement tendance à être stable. L'hypothèse faite par Xu *et al.* stipulant que lors de l'exécution d'une seule requête dans un système, le coût de l'énergie est stable (steady state) [284] s'avère donc être fausse. En effet, de nombreuses requêtes ont différents segments de consommation d'énergie au cours de leur exécution. Ceci est dû au fait que l'estimation de Xu *et al.* a été réalisée au niveau de la requête. Pour avoir des modèles de coûts d'énergie plus précis, l'estimation doit être effectuée au niveau plus fin qui constitue les pipelines.

4.3.2.2 Effet de la taille de BD

Habituellement, nous admettons le fait suivant : *« grande taille, plus d'énergie »*. Pour vérifier ce fait, nous menons des expérimentations, en termes de consommation de puissance active, en tenant compte des requêtes du benchmark TPC-DS [6] avec deux ensembles de données. Le premier ensemble est de taille 10 Go et le second de 100 Go. Nous voyons clairement dans le Tableau 4.1, que les requêtes interrogeant une petite taille de données peuvent *parfois* consommer plus de puissance que des requêtes interrogeant une grande taille de données. Nous avons créé un autre jeu de données du benchmark



(a) La segmentation de la consommation d'énergie suivant les pipelines

(b) La segmentation du plan d'exécution suivant les pipelines

FIGURE 4.4 – La consommation d'énergie et le plan d'exécution de la requête *Q9* issue du benchmark TPC-H avec l'annotation des pipelines correspondante.

TPC-H avec des requêtes modifiées et nous avons trouvé le même comportement (les résultats ne sont pas présentés car ils sont très semblables). Nous notons ici que le plan d'exécution généré par le SGBD pour chaque requête est le même dans les deux ensembles de données.

Nous expliquons ce constat par le fait que pour certaines requêtes, la lecture de gros fichiers de données a besoin de plus de tâches d'E/S. Ainsi, lorsque les résultats intermédiaires ne peuvent pas être stockés dans la mémoire, ils seront écrits sur le disque et lus plus tard. Cette manœuvre se traduit par plus de temps d'attente de CPU, car la requête passe son temps dans la lecture/écriture de ses données plus que dans le traitement de ses tuples. Ainsi, les requêtes dominées par des coûts d'E/S ont moins de consommation d'énergie. De l'autre côté, lorsque les données sont de petite taille, la lecture de données termine rapidement et dans son temps restant, l'exécution de la requête est dominée par le traitement du processeur, ce qui peut se traduire par une forte consommation d'énergie.

Cette observation nie la prémisse faite par Xu *et al.* stipulant que la consommation d'énergie marginale (énergie active) d'une requête est positivement liée à la taille de données interrogées [282]. En effet, nous avons trouvé beaucoup de contre-exemples où des requêtes ont une consommation différente indépendamment de la taille de données, comme indiqué précédemment dans le Tableau 4.1.

Le coût du CPU et le coût des E/S fournis par le SGBD sont candidats pour être les paramètres du modèle pour les raisons suivantes :

— ils sont déjà calculés et disponibles à partir du SGBD optimiseur,

Tableau 4.1 – Comparaison de la consommation de puissance active des requêtes du benchmark TPC-DS avec différentes tailles de BD.

Requête	Puissance d'électricité (W)	
	10 Go	100 Go
$Q4$	44,6	40,6
$Q9$	42,7	37,4
$Q11$	43,9	40,9
$Q39$	42,9	38,5
$Q48$	47,0	37,9
$Q72$	36,4	34,1
$Q88$	43,3	37,6

— ils couvrent un large éventail d'opérateurs SQL mises en œuvre par les SGBD.

Nous avons décidé d'utiliser les coûts CPU et E/S en tant que paramètres dans notre modèle, contrairement à Kunjir *et al.* [152] qui utilisent le débit et la taille de données comme paramètres pour modéliser la consommation de puissance maximale (peak power).

4.3.2.3 Effet du mode d'exécution des requêtes

Nous étudions l'effet du mode d'exécution des requêtes (isolé ou concurrent) sur la consommation d'énergie du système. Pour cela, nous utilisons les données et les charges de requêtes issues du benchmark TPC-H. Nous avons créé de nouvelles charges de requêtes pour chaque mode d'exécution à partir des 22 requêtes d'origine en se basant sur un taux de multiprogrammation donné.

Définition 10

Un taux de multiprogrammation (Multiprogramming Level (MPL)) est défini comme le nombre de requêtes exécutées en même temps à un instant donné.

La charge de requêtes contient les mêmes requêtes, mais avec différentes instances. Nous avons exécuté les charges de requêtes et enregistré leurs consommation d'énergie. Ensuite, nous avons calculé la puissance minimale, maximale et moyenne pour chaque MPL. Les résultats sont présentés dans la Figure 4.5. Nous notons que le taux de multiprogrammation x signifie que x requêtes s'exécutent simultanément.

Comme prévu, la consommation d'énergie augmente à mesure que l'on augmente le degré de concurrence, car le système devient plus chargé. En fait, le système atteint son débit maximal à *MPL = 40* et consomme sa puissance maximale (95 Watts) tel que mentionné par le fabriquant du matériel. De plus, on observe une grande différence entre le minimum, le maximum, et la moyenne de consommation d'énergie du même MPL. Ainsi, la modélisation pour des requêtes s'exécutant en mode concurrent est largement plus difficile que pour une seule requête autonome. Par conséquent, la considération du mode d'exécution dans la modélisation est cruciale pour une prédiction avec précision de la puissance d'une charge de requête.

Inspiré par les constats des sections précédentes, nous avons conçu et mis en œuvre un modèle

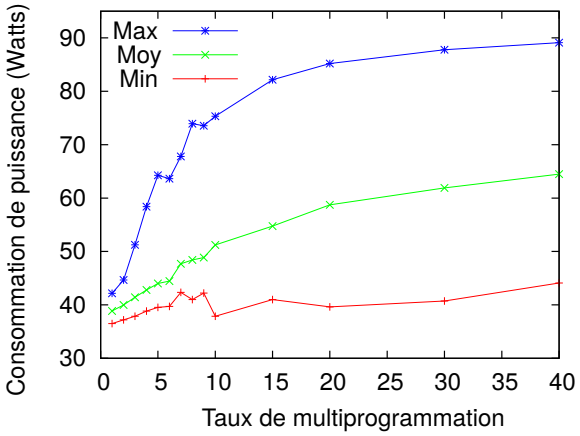


FIGURE 4.5 – Variation de la consommation d'énergie de la charge de requêtes suivant les différents taux de multiprogrammation (MPLs).

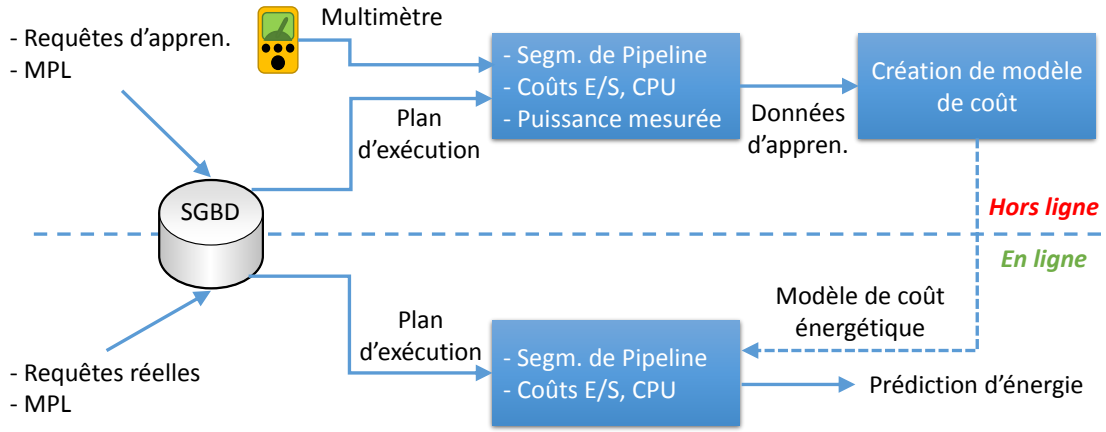


FIGURE 4.6 – Le processus de conception du modèle de coût énergétique.

d'énergie à base de coût. La Figure 4.6 montre le flux de travaux de notre processus de modélisation d'énergie. Les caractéristiques de notre modèle comprennent : (1) la segmentation d'un plan d'exécution en un ensemble de segments d'énergie/pipelines suivant les opérateurs bloquants et semi-bloquants, (2) l'utilisation du coût CPU et E/S des pipelines et de l'énergie mesurée on mode *hors ligne* pour construire le modèle et (3) l'estimation de l'énergie des futures requêtes en mode *en ligne* basée sur le modèle de coût final. Ainsi, nous bénéficions des avantages suivants :

— nous ne nous appuyons pas sur type d'opérateur SQL ou sur son implémentation physique, car la modélisation de chaque opérateur algébrique est complexe et diffère d'un SGBD à un autre.
— notre modèle peut être facilement intégré dans l'optimiseur des SGBD existants.

Dans les sections suivantes, nous détaillons chaque étape de notre processus de modélisation.

4.3.3 Segmentation en pipelines

La deuxième étape de notre approche est la segmentation en pipelines. Quand une requête est soumise à un SGBD, l'optimiseur de requêtes choisit un plan d'exécution. Un plan est un arbre composé

d'un ensemble d'opérateurs physiques, tel que la lecture, la jointure ou le tri. La Figure 4.4b présente un exemple de plan d'exécution renvoyée par l'optimiseur de requêtes. Un opérateur physique peut être *bloquant* ou *non bloquant*.

Définition 11

> *Un opérateur est bloquant s'il ne peut produire aucun tuple sans lire au moins une de ses entrées (p. ex. l'opérateur de tri).*

Basé sur la notion des opérateurs bloquant/non bloquant, nous décomposons le plan d'exécution en un ensemble de pipelines délimités par les opérateurs bloquants. Ainsi, un pipeline est défini comme suit.

Définition 12

> *Un pipeline est un ensemble d'opérateurs s'exécutant simultanément [165].*

Comme dans les travaux précédents [57, 172], les pipelines sont créés de manière inductive, à partir des opérateurs feuilles du plan à l'instar de l'accès séquentiel et l'accès par index. Chaque fois que nous rencontrons un opérateur bloquant, comme le tri et le groupage, le pipeline en cours se termine et un nouveau commence. Pour une jointure de hachage, l'opérateur de jointure est inclus dans le pipeline du nœud de calcul de jointure (*probe phase* en anglais), et le nœud de la construction de la table de hachage (*build phase*) est la racine d'un autre pipeline. Pour une opération de jointure par tri-fusion, les pipelines contenant ses nœuds d'entrée et l'opérateur de jointure sont réunis pour créer un seul pipeline. Pour une jointure par boucles imbriquées ou boucles imbriquées avec index, les nœuds externes, l'opérateur de jointure et toute sa sous-arborescence des nœuds intérieurs font partie d'un même pipeline. La définition donnée ci-dessus peut être étendue à d'autres opérateurs physiques. L'Algorithme 4.1 illustre un pseudo code de ce principe avec quelques opérateurs, une implémentation en C++ est mentionnée dans le Chapitre 5. Par conséquence, le plan d'exécution d'origine peut être considéré comme un arbre de pipelines. La Figure 4.4b montre le plan d'une requête ayant 7 pipelines :

1. Les pipelines $PL1$, $PL2$, $PL3$, $PL4$ font la lecture séquentielle des tables de données et construisent leur table de hachage en même temps. Ce sont des opérateurs bloquants.
2. Le pipeline $PL5$ fait la jointure par hachage (opérateurs non-bloquant) entre la table `LINEITEM` et le $PL1$ puis le résultat avec les autres pipelines. À la fin, le pipeline construit la table de hachage du résultat final.
3. Le pipeline $PL6$ fait la jointure par hachage entre $PL5$ et la table `ORDERS` puis le résultat avec le $PL1$.
4. Le pipeline $PL7$ calcule l'opérateur bloquant qui est le tri de $PL6$ et affiche le résultat final de la requête.

En outre, sur la base de notre analyse, nous avons constaté que le SGBD exécute les pipelines dans un ordre séquentiel, donc il n'y a pas de pipeline en concurrence. Comme nous l'avons mentionné auparavant, nous décomposons le plan en un ensemble de pipelines, et nous les ordonnons sur la base des informations fournies par le module de surveillance SQL temps réel du SGBD. Ce module affiche des statistiques en temps réel afin d'identifier les problèmes de performance d'exécution des requêtes avec longue durée. Cela permet aux administrateurs de BD d'analyser l'exécution des requêtes et de décider des stratégies d'optimisation les plus appropriées [231]. La Figure 4.7 montre une partie de ce module, les informations les plus importantes sont :

Algorithme 4.1 Algorithme de segmentation des pipelines

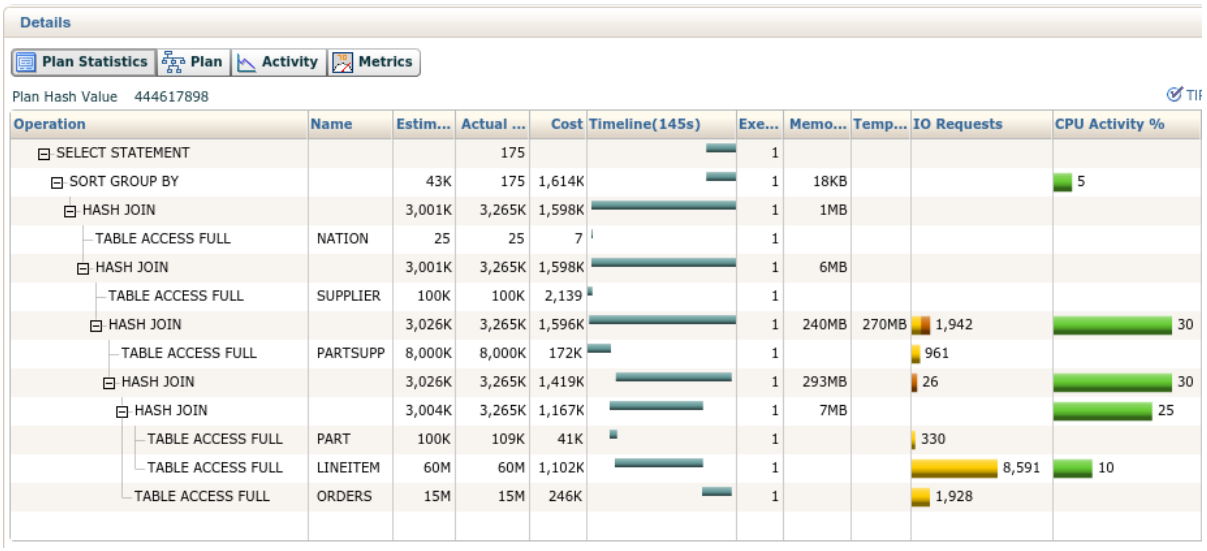
Entrée : *Plan* ▷ un plan d'exécution d'une requête SQL contenant des opérateurs *OP*
Sortie : *PL* ▷ une liste contenant la segmentation des pipelines

```
1 : PLActuel ← ∅ ;
2 : pour tout opérateur OP_i ∈ Plan faire
3 :     si estFeuille(OP_i) alors
4 :         j ← i ;
5 :         tant que j ∈ Plan et OP_j ≠ « Tri » et OP_j ≠ « Agrégation » et OP_j ≠
    « Matérialisation » · · · faire
6 :             PLActuel ← PLActuel + OP_j ;
7 :             j ← j + 1 ;
8 :         fin tant que
9 :         PL ← PLActuel + PL ;
10 :        PLActuel ← ∅ ;
11 :    fin si
12 : fin pour
13 : retourner PL ;
```

- La colonne `Operation` indique le type d'opération SQL dans le plan d'exécution, le nom associé à cette opération est affiché dans la colonne `Name`. Par exemple, le nom de la table `LINEITEM` est associé à l'opération de lecture `TABLE ACCESS FULL`.
- La colonne `Estimated Rows` affiche les cardinalités estimées pour chaque opération lors de la génération des plans d'exécutions. Idem pour `Cost` qui affiche le coût estimé des opérations en terme de CPU et d'E/S, les valeurs sont cumulées de bas vers le haut et son unité de mesure n'est pas spécifié par Oracle.
- La colonne `Actual Rows` affiche les cardinalités réelles de chaque opération après l'exécution de la requête.
- La colonne `Timeline` est la plus importante dans notre travail. Elle affiche le temps de début d'exécution de l'opérateur pour la première fois et sa durée totale, ce qui permet d'identifier et de segmenter les pipelines. La colonne `Execution` indique le nombre de fois que l'opérateur a été exécuté.
- Les colonnes `Memory (Max)` et `Temp (Max)` affichent la taille mémoire et l'espace temporaire maximaux qui ont été consommés durant l'exécution de l'opérateur.
- La colonne `IO Requests` indique le nombre de demandes de lecture et d'écriture sur un support de stockage externe. Selon la figure, des demandes de lecture ont été effectués par les opérateurs de lecture, et des demandes de lecture et écriture par l'opérateur `HASH JOIN`. Les demandes par l'opérateur de jointure sont dues à la saturation de l'espace mémoire alloué pour le SGBD.
- La colonne `CPU Activity (%)` affiche la portion d'utilisation de la ressource CPU par chaque opérateur. Dans notre exemple, le CPU a été utilisé par les opérateurs de jointure et l'opérateur de tri `SORT GROUP BY`.

Cette fonctionnalité de surveillance des requêtes est aussi disponible dans d'autres SGBD, à l'exemple de SQL Server [250], PostgreSQL [14] et MySQL [192].

14. http ://www.postgresql.org/docs/current/static/sql-explain.html



Operation	Name	Estim...	Actual ...	Cost	Timeline(145s)	Exe...	Memo...	Temp...	IO Requests	CPU Activity %
SELECT STATEMENT			175			1				
SORT GROUP BY		43K	175	1,614K		1	18KB			5
HASH JOIN		3,001K	3,265K	1,598K		1	1MB			
TABLE ACCESS FULL	NATION	25	25	7		1				
HASH JOIN		3,001K	3,265K	1,598K		1	6MB			
TABLE ACCESS FULL	SUPPLIER	100K	100K	2,139		1				
HASH JOIN		3,026K	3,265K	1,596K		1	240MB	270MB	1,942	30
TABLE ACCESS FULL	PARTSUPP	8,000K	8,000K	172K		1			961	
HASH JOIN		3,026K	3,265K	1,419K		1	293MB		26	30
HASH JOIN		3,004K	3,265K	1,167K		1	7MB			25
TABLE ACCESS FULL	PART	100K	109K	41K		1			330	
TABLE ACCESS FULL	LINEITEM	60M	60M	1,102K		1			8,591	10
TABLE ACCESS FULL	ORDERS	15M	15M	246K		1			1,928	

FIGURE 4.7 – Aperçu du module de surveillance SQL temps réel d'Oracle.

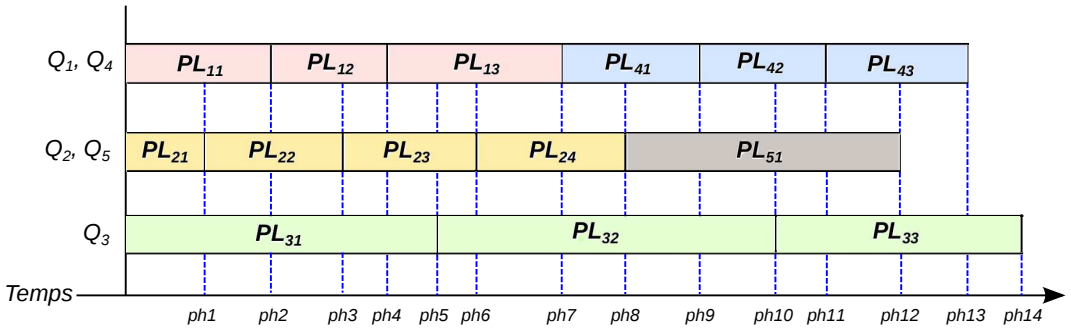


FIGURE 4.8 – Les segmentation des pipelines d'une charge de requêtes concurrentes.

4.3.3.1 Cas des requêtes concurrentes

Supposons que nous avons une charge de requêtes $q_1, q_2, \cdots, q_n$. Après avoir segmenté leurs plans d'exécution en un ensemble de pipelines, la charge de requêtes peut être considérée comme de multiples phases de mixes de pipelines. Ceci est illustré par la Figure 4.8. Dans cet exemple, nous avons une charge de 5 requêtes $q_1, \cdots, q_5$. Après la segmentation de leurs plans, q_1 est représentée comme une séquence de 3 pipelines $P_{11}P_{12}P_{13}$, q_2 est représentée comme une séquence de 4 pipelines $P_{21}P_{22}P_{23}P_{24}$, et ainsi de suite pour les autres requêtes. Nous utilisons $PLij$ pour désigner le j^{ime} pipeline de la i^{ime} requête, et nous utilisons ph_k pour désigner la phase où un certain pipeline PL_{ij} termine son exécution. Comme nous le voyons, nous pouvons identifier un nouveau mix de pipelines dès qu'un certain pipeline termine son exécution. Dans notre exemple, à la fin, nous aurons 14 mixes de pipelines, séparés par les lignes bleues en pointillés qui indiquent le temps de fin d'exécution de chaque pipeline.

L'exécution de la charge de requêtes est assimilée à l'exécution d'une suite de mixes de pipeline. L'exécution de chaque mix est appelée une phase de pipeline. Un changement de phase se produit quand un pipeline termine son exécution et un autre pipeline commence. Notre objectif est de déterminer les phases de pipeline et les transitions entre eux et d'estimer la quantité d'énergie consommée par chaque

Algorithme 4.2 Algorithme de prédiction

Entrée : $W = \{q_1, \cdots, q_n\}$ ▷ une charge de n requêtes SQL
Sortie : P_w ▷ la consommation d'énergie estimée pour la charge W

pour tout requête $i \in W$ **faire**
  $Plan_i \leftarrow GetPlan(i)$;
  $PL_i \leftarrow GetPipelines(Plan_i)$;
fin pour
$PLMixActuel \leftarrow \emptyset$;
pour tout requête $i \in W$ **faire**
  $PLMixActuel \leftarrow PLMixActuel + PL_{i1}$;
fin pour

tant que $PLMixActuel \neq \emptyset$ **faire**
  $PL_{ij} \leftarrow PlusCourtPipeline(PLMixActuel)$;
  $t_{min} \leftarrow t_{ij}$;
  $Coût \leftarrow 0$;
  pour tout $PL_{ik} \in CurrentMix$ **faire**
    $Coût \leftarrow Coût + Coût_{ik} \times \frac{t_{min}}{t_{ik}}$;
  fin pour
  $P_w \leftarrow P_w + EstimePuissance(Coût)$;
  Retirer PL_{ij} depuis $MixActuel$;
  si $aPlusDePipelines(PL_i)$ **alors**
    $PLMixActuel \leftarrow PLMixActuel + PL_{i(j+1)}$;
  fin si
fin tant que

retourner P_w ;

phase. La consommation d'énergie de la charge de requêtes est la consommation totale de toutes les phases. Cette idée est présentée dans l'Algorithme 4.2.

Dans cet algorithme, nous générons d'abord le plan d'exécution $Plan_i$ pour chaque requête Q_i en appelant l'optimiseur de requêtes, puis nous créons la segmentation de pipelines PL_i de chaque $Plan_i$. Le premier pipeline PL_{i1} dans chaque PL_i est ajouté au mix en cours d'exécution $CurrentPLMix$. Ensuite, nous continuons l'exécution par phase tant que le mix actuel ne soit pas vide. Nous déterminons le pipeline PL_{ij} avec le temps d'exécution le plus court t_{min}. Ensuite, nous calculons *le coût énergétique du travail effectué* par chaque pipeline PL_{ik} dans le mix actuel, en multipliant le coût de chaque pipeline par $\frac{t_{min}}{t_{ik}}$, où t_{ik} est le temps d'exécution du pipeline PL_{ik}. Nous appelons la procédure d'estimation d'énergie pour obtenir l'énergie du mix de pipeline actuel, et nous retirons P_{ij} du mix. Si PL_i contient plus de pipelines après la fin d'exécution du P_{ij}, nous ajoutons le suivant $P_{i(j+1)}$ dans le mix en cours et nous répétons ce processus jusqu'à ce que toutes les requêtes de la charge sont terminées. La consommation d'énergie finale de la charge de requêtes est la somme de consommation de chaque phase.

4.3.4 Modélisation des pipelines

Après l'identification des paramètres de notre modèle et le niveau de modélisation. La prochaine étape consiste à modéliser ces paramètres. Dans cette section, nous allons donner une formalisation pour notre approche. Ensuite, nous construisons notre modèle en identifiant le modèle mathématique qui décrit notre problème d'estimation d'énergie. Cette phase de construction combine les paramètres identifiés dans les sections précédentes avec une méthode d'apprentissage automatique pour trouver le meilleur modèle (boite grise).

4.3.4.1 Coût CPU et E/S des pipelines

Dans un SGBD traditionnel, le coût d'exécution de la requête est traité comme une combinaison linéaire de trois composantes : le coût du CPU, d'E/S et le coût de la communication [282]. Inspiré par cette méthodologie, nous proposons d'utiliser ces coûts pour modéliser les pipelines des requêtes. Pour une charge de requête donnée, l'optimiseur de requêtes est responsable de l'estimation des coûts CPU et d'E/S. Notre stratégie pour la modélisation des pipelines est de profiter des modèles de coûts qui sont déjà intégrés dans les SGBD pour l'optimisation des requêtes. Dans un SGBD, pour traiter les tuples, chaque opérateur appartenant à un pipeline doit effectuer des tâches de CPU et/ou d'E/S. Le coût de ces tâches représente le « *coût du pipeline* », qui est la puissance d'électricité active censée être consommée afin de terminer les tâches. Dans cette thèse, nous nous concentrons sur une configuration de serveur centralisé et laissons l'étude des BD distribuées pour les futurs travaux. Ainsi, le coût de la communication peut être ignoré. Plus formellement, pour une charge de requête donnée W composée par des requêtes Q, notre modèle de coût énergétique pour W est défini comme suit :

$$Puissance(W) = \sum_{i=1}^{n} Puissance(Q_i) \tag{4.1}$$

Soit Q_i une requête composée de p_i pipelines $\{PL_1, PL_2, \ldots, PL_p\}$. Le coût de l'énergie $Puissance(Q_i)$ de la requête Q_i est donné par l'équation suivante :

$$Puissance(Q_i) = \frac{\sum_{j=1}^{p} Puissance(PL_j) * Time(PL_j)}{Time(Q_i)} \tag{4.2}$$

$Time(PL_j)$ et $Time(Q_i)$ représentent respectivement, le temps d'exécution du pipeline j et la requête Q_i. Ces deux facteurs sont complètement ignorés par Xu *et al.* [282]. Dans notre modèle, le temps est un facteur important pour déterminer le pipeline dominant par les coûts CPU ou E/S dans une requête. Bien que le module statistique du SGBD nous fournisse ces informations, nous allons utiliser le temps d'exécution réel des requêtes afin de tester la qualité de notre modèle.

Notez que le coût d'un pipeline j doit également être décomposé car il peut impliquer plusieurs opérations algébriques. Soit n_j le nombre de ces opérations ($\{OP_1, OP_2, \ldots, OP_n\}$). Le coût énergétique $Puissance(PL_j)$ d'un pipeline j est donné par l'équation suivante :

$$Puissance(PL_j) = \beta_{cpu} \times \sum_{k=1}^{n_j} COUT\_CPU_k + \beta_{es} \times \sum_{k=1}^{n_j} COUT\_ES_k \tag{4.3}$$

Où β_{cpu} et β_{es} sont les paramètres du modèle (les unités des coûts énergétiques).

Pour une requête donnée, l'optimiseur utilise le plan d'exécution, les estimations de cardinalité et les formules pour les opérateurs afin de générer les valeurs des différents types d'opérations CPU et d'E/S. Il convertit ensuite ces valeurs en temps en utilisant des paramètres spécifiques au système, tels que la vitesse du processeur et la vitesse du transfert E/S. Par conséquent, dans notre modèle, nous prenons les estimations de CPU et d'E/S avant de les convertir en temps. Les paramètres du système sont calculés à la création de la BD, et pourraient être peu fiables. Suivant les recommandations du fournisseur de SGBD, nous calibrons ces paramètres pour assurer la précision de l'estimation des ressources et l'optimalité du plan d'exécution.

Le *COUT_CPU* est le nombre estimé de *Cycle de CPU* et de *lecture du cache mémoire* nécessaires au SGBD pour l'exécution de l'opérateur spécifié. De même, le *COUT_ES* est le nombre estimé d'*E/S* nécessaire au SGBD pour l'exécution de l'opérateur spécifié.

Dans la littérature, l'estimation de ces deux coûts peut se réaliser suivant deux catégories de techniques : (1) techniques basées sur des fonctions analytiques et (2) techniques basées sur l'optimiseur de requêtes du SGBD [39]. La première catégorie de techniques [40, 144, 38] définit une fonction analytique pour calculer le coût d'une requête à partir d'un plan d'exécution prédéfini. À chaque opérateur algébrique, une formule qui prend en entrée des paramètres et des statistiques de la BD est utilisée pour le calcul de coût. L'avantage de cette méthode réside dans la facilité et la rapidité de la mise en œuvre et du calcul, mais elle reste limitée par les hypothèses simples et parfois irréalistes. De plus, elle ne garantit pas l'optimalité du plan de la requête choisie. Par conséquent, cette méthode ne répond pas aux critères régissant la construction d'un modèle de coût énergétique de qualité, qui ont été décrits dans la Section 4.2.

La deuxième catégorie de techniques [46, 58, 55] fait appel à l'optimiseur de requête du SGBD utilisé pour estimer le coût d'une requête. L'optimiseur évalue les différents plans d'exécution de la requête

et retourne l'optimal avec son coût. Le plan et le coût sont fiables comme exigé par nos critères de modélisation. De plus, cette technique rend l'intégration de notre modèle dans un SGBD réel plus facile.

Dans le cadre de cette thèse, nous adaptons une technique appartenant à la deuxième catégorie. Comme indiqué précédemment, en utilisant les coûts CPU et E/S retournés par le SGBD, nous ne dépendons pas du type ou de l'implémentation des opérateurs SQL comme dans [282]. Par conséquent, notre modèle est portable et peut traiter des requêtes plus complexes comme celles du benchmark TPC-DS.

4.3.4.2 Estimation des paramètres du modèle

Le principal défi dans la résolution de l'Équation (4.3) est de trouver les paramètres du modèle β_{cpu} et β_{es}. Notre méthodologie utilise la technique de régression pour calculer ces paramètres de coût.

La notion de régression est fondamentale dans toutes les sciences appliquées, elle consiste à analyser une relation entre deux variables quantitatives et à l'exploiter pour estimer la valeur inconnue de l'une à l'aide de la valeur connue de l'autre. Elle est couramment utilisée dans les techniques de gestion et de commercialisation, pour expliquer un chiffre d'affaires en fonction des dépenses publicitaires, effectuer des prévisions de bénéfices, de ventes, etc. Dans les sections suivantes, nous détaillons cette technique, et nous formalisons la démarche utilisée pour calculer les paramètres de notre modèle de coût.

4.3.4.2.1 Régression linéaire

On étudie en régression deux variables quantitatives, dont l'une appelée variable *expliquée*, est considérée comme *dépendante* de l'autre, appelée variable *explicative* ou *indépendante*. On note habituellement la variable expliquée Y, et la variable explicative X. Un exemple de cette dépendance est lorsqu'on mesure la consommation d'une voiture à des vitesses choisies pour établir la relation entre la consommation (variable dépendante) et la vitesse (variable indépendante) [183, 188, 153].

On qualifie un modèle de régression de *simple* lorsque qu'il y a une seule variable explicative. À l'inverse, on le qualifie de *multiple* en présence d'au moins de deux variables explicatives. Le modèle de régression simple est une équation censée représenter cette relation entre les deux variables. Il s'écrit sous cette forme :

$$Y = f(X) + \epsilon \tag{4.4}$$

La variable Y est supposée être approximativement égale à une fonction f de X, le terme ϵ caractérisant la marge d'erreur ou d'imprécision du modèle. L'objectif est de préciser la nature de la régression (la fonction f), de mesurer le degré d'imprécision (le terme ϵ), de détecter les observations qui ne suivent pas le modèle et d'effectuer des prévisions de Y pour différentes valeurs de X. Nous supposons d'abord que la nature de la liaison entre les deux variables est linéaire. Le modèle de régression s'exprime alors de la façon suivante :

$$y = \beta x + a + \epsilon \tag{4.5}$$

Où β et a représentent les coefficients théoriques de la régression et leurs valeurs sont inconnues. L'estimation de ces coefficients peut se faire avec plusieurs méthodes. Parmi les méthodes les plus utilisées on trouve la méthode des moindres carrés.

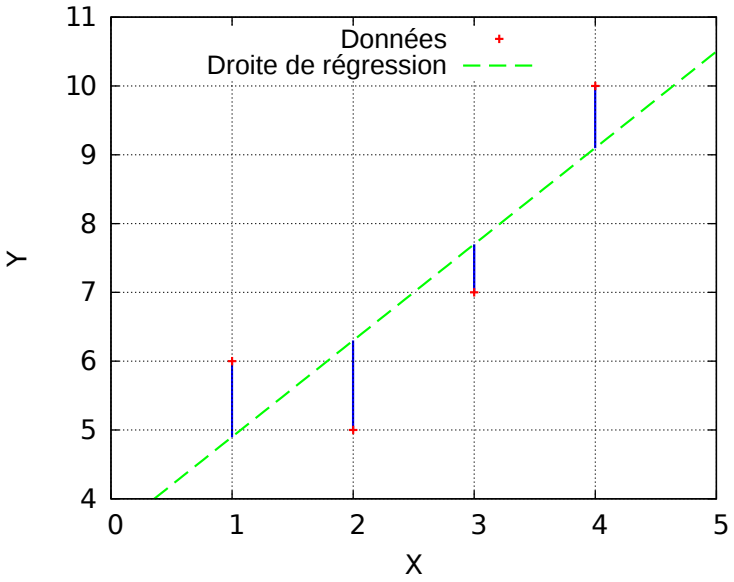


FIGURE 4.9 – La régression linéaire par la méthode des moindres carrés.

4.3.4.2.1 Critère des moindres carrés

Supposons que les données de l'étude de la régression se présentent sous la forme d'une suite de n couples $[x(i), y(i)]$, numérotés de $i = 1$ à $i = n$. Nous avons représenté sur la Figure 4.9 cet ensemble de données. L'objectif est de déterminer les coefficients de la droite $y = \beta x + a$, appelée *droite de régression*, la plus proche possible des n points.

Plus précisément, il s'agit de reconstruire le mieux possible la variable Y en fonction de la variable X et de déterminer la droite de façon à ce que les termes d'erreur de la forme $\epsilon(i) = y(i) - [\beta x(i) + a]$ soient les plus petits possible (les plus proches de 0). Ces termes $e(i)$ sont appelées aussi *résidus*. Pour mesurer la proximité de la valeur 0 à ces erreurs, on cherche les coefficients β et a tel que la somme des carrés de ces termes soit minimale. Ces sommes sont représentées sur la Figure 4.9 par les longueurs des segments de couleur bleu.

Les estimations des coefficients de régression théoriques β et a sont telles que la somme des carrés des erreurs soit la plus petite possible. Elles sont données par les formules ci-dessous [188] :

$$\begin{gathered} \beta = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n}(x_i - \bar{x})^2} \text{ et } a = \bar{y} - \beta\bar{x} \\ \text{avec : } \bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i \text{ et } \bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i \end{gathered} \tag{4.6}$$

4.3.4.2.1 Modèle énergétique en régression linéaire

Appliquant les techniques expliquées dans les sections précédentes sur notre modèle de coût, le coût de l'énergie $Puissance_{PL_j}$ du pipeline p_j, avec une relation linéaire entre les paramètres du modèle, est

calculé comme suit :

$$Puissance(PL_j) = \beta_0 + \beta_1 \times COUT\_ES + \beta_2 \times COUT\_CPU + \epsilon \tag{4.7}$$

où $COUT\_ES$, $COUT\_CPU$ désignent les coûts CPU et E/S du pipeline, respectivement. Ces coûts sont fournis par le module de statistiques SGBD, et ϵ_i est un terme d'erreur ou perturbation, qui représente les erreurs de mesure. Les paramètres β sont des coefficients de régression qui seront estimés, tout en apprenant le modèle à partir de données d'apprentissage. Ainsi, les modèles de régression linéaire sont résolus par l'estimation des paramètres du modèle β, et cela se fait par trouver la solution des moindres carrés.

Nous avons commencé simplement par une régression linéaire avec la formule ci-dessus (Équation (4.7)), comme déjà fait dans [282, 152, 155]. Malheureusement, ce modèle ne donne pas de meilleurs résultats surtout quand la taille des données change. En effet, la relation entre la taille des données et la puissance d'électricité n'est pas linéaire. En d'autres termes, le traitement de gros fichiers ne se traduit pas *toujours* par une forte consommation d'énergie. Cela dépend plutôt du type de pipeline (dominé par CPU ou E/S) dans la requête et de son temps d'exécution. Par conséquent, nous avons décidé d'utiliser le modèle de régression polynomiale.

4.3.4.2.2 Régression polynomiale

La régression polynomiale est une analyse statistique qui décrit la variation d'une variable expliquée y, en fonction d'une variable explicative x. On cherche, par régression, à lier les variables par un polynôme de degré p. Le modèle est de la forme [153] :

$$y = \beta_0 + \beta_1 x + \beta_2 x^2 + \cdots + \beta_p x^p + \epsilon \tag{4.8}$$

Pour simplifier les notations, nous introduisons une variable explicative supplémentaire x_0, qui est constante et égale à 1. On peut alors écrire :

$$y = \sum_{j=0}^{p} \beta_j x^j + \epsilon \tag{4.9}$$

Soit, pour chaque unité statistique :

$$y(i) = \sum_{j=0}^{p} \beta_j x^j(i) + \epsilon(i) \tag{4.10}$$

Le critère utilisé pour calculer les estimations β_j est le même que précédemment : on cherche les valeurs $\beta_0, \beta_1, \cdots, \beta_j, \cdots, \beta_p$ tel que l'ajustement soit le meilleur possible au sens des moindres carrés. On minimise donc la somme S :

$$S = \sum_{i=0}^{n} [y(i) - \sum_{j=0}^{p} \beta_j x^j(i)]^2 \tag{4.11}$$

La régression polynomiale est considérée comme un cas particulier de la régression linéaire multiple. Cette technique convient quand il y a une relation *non linéaire* entre les variables indépendants et la

variable dépendante [188].

4.3.4.2.2 Modèle énergétique en régression polynomiale

Nous avons utilisé la régression polynomiale dans notre modèle. Selon nos expérimentations, l'ordre *p=2* nous donne les meilleurs résultats (la somme des carrés des résidus est la plus petite). Le modèle de régression polynomiale avec le degré *p=2* est exprimé comme suit :

$$\begin{aligned} Puissance(PL_j) = \beta_0 + \beta_1 \times (COUT\_ES) + \beta_2 \times (COUT\_CPU) + \\ \beta_3 \times (COUT\_ES^2) + \beta_4 \times (COUT\_CPU^2) + \beta_5 \times (COUT\_ES \times COUT\_CPU) + \epsilon \end{aligned} \tag{4.12}$$

Les résultats de cette méthode sont beaucoup plus véridiques qu'une régression linéaire.

4.3.4.3 Apprentissage

Comme mentionné ci-dessus, les paramètres β sont estimés tout en apprenant le modèle à partir de données d'apprentissage. Ensuite, nous effectuons des séries d'observations où les requêtes sont bien choisies et les valeurs de l'énergie consommée par le système sont récupérées au moyen d'un équipement de mesure lors de l'exécution de ces requêtes. En même temps, pour chaque instance d'apprentissage, nous segmentons les opérateurs de requête en un ensemble de pipelines, et nous calculons leurs coûts.

4.3.4.3.1 Données d'apprentissage

Pour générer des données d'apprentissage, nous avons créé notre charge de requête personnalisée basé sur les données du benchmark TPC-H (au facteur d'échelle 10 et 100). La charge contenant 110 requêtes est caractérisée par (i) des requêtes ayant des opérations qui épuisent le processeur du système (requêtes gourmandes en CPU) et (ii) des requêtes avec des opérations qui épuisent les ressources de stockage (requêtes gourmandes en E/S). Pour ce faire et en s'inspirant des travaux précédents [152], nous varions d'une requête à une autre : le nombre de tables à lire, le nombre/type des prédicats de jointure, le nombre des opérations du tri/regroupement, le type des fonctions d'agrégation allant de la plus simple (`count(*)`) à des fonctions plus complexes. L'ensemble de requête interrogeant des données de 10 Go de taille est présenté dans l'Annexe C (page 207).

Nous notons que pour le coût de CPU de certaines requêtes, en particulier celles contenant des agrégations et des fonctions analytiques, n'est pas calculé par le SGBD malgré la puissance intensive consommée par ces deux types de fonctions. Pour surmonter cette contrainte, nous proposons de calculer *manuellement* les cycles de CPU requis pour exécuter ces fonctions d'agrégation, en utilisant les coûts de notre machine pour faire les instructions de base telles que *ADD, SUB, MUL, DIV* et *COMPARE* [131]. Les estimations obtenues sont ensuite multipliées par le nombre de lignes.

4.3.4.3.2 Régression avec le logiciel R

Après avoir collecté la consommation de puissance, les coûts CPU et E/S des requêtes d'apprentissage, nous appliquons l'équation de régression (Équation (4.12)) pour trouver les paramètres du modèle et

```
> data ← read.csv('energy-data.csv', comment.char = "#", sep='\t')
> result ← lm(Power ~ IO + CPU, data=data)
> summary(result)
Call :
lm(formula = Power ~ IO + CPU, data = data)

Residuals :
    Min      1Q  Median      3Q     Max
-2.7331 -0.5331 -0.0666  0.1971  6.5852

Coefficients :
              Estimate Std. Error t value Pr(>|t|)
(Intercept)  3.878e+01  1.824e-01  212.63   <2e-16 ***
IO          -2.891e-07  2.623e-08  -11.02   <2e-16 ***
CPU          3.722e-12  2.327e-13   15.99   <2e-16 ***
---
Signif. codes :  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error : 1.436 on 139 degrees of freedom
Multiple R-squared :  0.6656,    Adjusted R-squared :  0.6608
F-statistic : 138.3 on 2 and 139 DF,  p-value : < 2.2e-16
```

FIGURE 4.10 – Exemple d'une régression linéaire avec le logiciel R.

ainsi décrire le comportement des pipelines vis-à-vis de la consommation de l'énergie. Finalement, après l'extraction des paramètres, nous pouvons estimer la puissance des nouvelles requêtes sans avoir à utiliser les équipements de mesure.

Nous décrivons dans cette section la mise en œuvre de la régression à l'aide de R [15]. R est un logiciel libre et gratuit et un langage d'analyse statistique et graphique [128] brassant de nombreuses fonctions traitant la modélisation linéaire et non linéaire, la statistique multivariée, l'analyse de séries chronologiques, la classification, le clustering, etc.

Pour notre problème de régression, R dispose de la commande `lm()`, qui signifie *linear model* ou modèle linéaire, permettant d'obtenir l'équation de la droite de régression. La lecture de données, l'application de la régression et l'affichage de résultat peut être réalisé de la manière présentée dans la Figure 4.10.

Le `Intercept` correspond ici à β_0 de notre Équation (4.7) et le IO, CPU correspondent respectivement au β_1, β_2 de notre équation. Les valeurs de ces termes sont dans la colonne `Estimate`. L'équation de droite de régression est donc :

$$Puissance(PL_j) = 38,78 - 2,891 \times 10^{-7} \times COUT\_ES + 3,722 \times 10^{-12} \times COUT\_CPU \tag{4.13}$$

La commande `summary()` affiche également d'autres informations sur le modèle, tel que les valeurs de résidus, l'erreur standard, le coefficient de détermination, etc. De même pour la régression polynomiale de degré 2, on utilise la commande présentée dans la Figure 4.11 suivante.

15. http ://www.r-project.org/

```
> result ← lm(Power ~ poly(IO, CPU, degree=2, raw=TRUE), data=data)
> summary(result)

Call :
lm(formula = Power ~ poly(IO, CPU, degree = 2, raw = TRUE), data = data)

Residuals :
    Min      1Q  Median      3Q     Max
-2.3579 -0.3407  0.0850  0.3690  4.0238

Coefficients :
                                          Estimate Std. Error t value Pr(>|t|)
(Intercept)                              3.841e+01  1.946e-01 197.378  < 2e-16
poly(IO, CPU, degree = 2, raw = TRUE)1.0 -1.281e-06  1.851e-07  -6.922 1.60e-10
poly(IO, CPU, degree = 2, raw = TRUE)2.0  9.026e-14  1.532e-14   5.892 2.85e-08
poly(IO, CPU, degree = 2, raw = TRUE)0.1  2.934e-11  1.652e-12  17.763  < 2e-16
poly(IO, CPU, degree = 2, raw = TRUE)1.1 -2.236e-18  1.561e-19 -14.319  < 2e-16
poly(IO, CPU, degree = 2, raw = TRUE)0.2 -2.663e-25  1.575e-25  -1.691   0.0931
...
```

FIGURE 4.11 – Exemple d'une régression polynomiale avec le logiciel R.

On obtient les mêmes sorties comme pour la commande précédente. L'équation de la droite de régression dans ce cas est :

$$\begin{aligned} Puissance(PL_j) = 38,41 - 1,281 \times 10^{-6} COUT\_ES + 2,943 \times 10^{-11} COUT\_CPU \\ +9,026 \times 10^{-14} COUT\_ES^2 - 2,663 \times 10^{-25} COUT\_CPU^2 \\ -2,236 \times 10^{-19} COUT\_ES \times COUT\_CPU \end{aligned} \tag{4.14}$$

4.3.5 Méthode d'évaluation

Nous élaborons une phase de test où le coût d'énergie estimé *(E)* par notre modèle est comparé avec la consommation d'énergie réelle *(M)* du système, relevée à l'aide des équipements de mesure d'énergie. Pour quantifier la précision du modèle, nous avons utilisé la métrique du taux d'erreur suivante pour une seule requête :

$$Erreur = \frac{|M - E|}{M} \tag{4.15}$$

Et pour une charge de n requêtes, nous avons utilisé la métrique d'erreur moyenne et maximale suivantes :

$$\begin{aligned} Erreur_{moy} &= \frac{1}{n} \times \sum_{i=1}^{n} \frac{|M_i - E_i|}{M_i} \\ Erreur_{max} &= Max(\frac{|M - E|}{M}) \end{aligned} \tag{4.16}$$

La Figure 4.12 résume les étapes de notre approche composée de trois étapes. Nous avons d'abord identifié le niveau de modélisation le plus adéquat comme étant les pipelines et les paramètres du modèle comme étant le coût E/S et CPU. Dans l'étape de construction du modèle, la méthode choisie est la

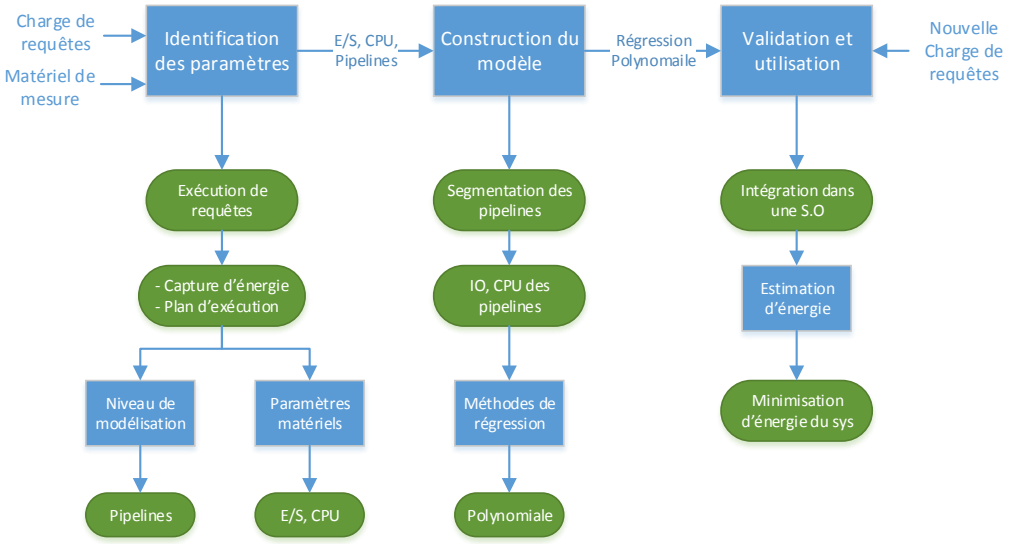


FIGURE 4.12 – Vue d'ensemble sur le modèle de coût énergétique.

régression polynomiale. Finalement, l'étape de validation permet d'estimer l'énergie d'une requête sans recourir à l'équipement de mesure et intégrer le modèle validé dans une structure d'optimisation afin de minimiser la consommation d'énergie dans les BD.

4.4 Évaluation de notre modèle de coût

Dans cette section, nous présentons des résultats expérimentaux montrant l'efficacité de notre modèle proposé. Nous décrivons d'abord l'architecture et le jeu de données d'expérimentations, puis nous évaluons la performance de notre modèle d'estimation d'énergie.

4.4.1 Architecture d'expérimentations

Pour mesurer la consommation réelle d'énergie, nous avons créé une configuration similaire aux travaux précédents [215, 283, 152] que nous utilisons même dans le reste de nos expérimentations présentées dans les prochains chapitres. Cette configuration utilise le multimètre « *Watts UP? Pro ES* » [16] qui a une résolution maximale d'une seconde. Le dispositif est directement connecté entre la prise d'alimentation et la station de travail sur laquelle la BD est déployée. Cet ordre permet de mesurer la consommation d'énergie globale de la station de travail. Les valeurs mesurées sont enregistrées et traitées dans une machine de contrôle séparée connectée avec le multimètre via un câble USB.

Nous avons utilisé une station de travail Dell PowerEdge R210 II ayant un processeur Intel Xeon E3-1230 V2 de 3.30GHz, 10 Go de mémoire DDR3 et une configuration disque dur de 2x500 Go. A noter que des techniques comme la *tension-fréquence dynamique du processeur* (DVFS) ne sont pas appliquées dans nos expérimentations. Nous répétons chaque expérience plusieurs fois pour assurer la confiance dans les valeurs observées.

16. https ://www.wattsupmeters.com/

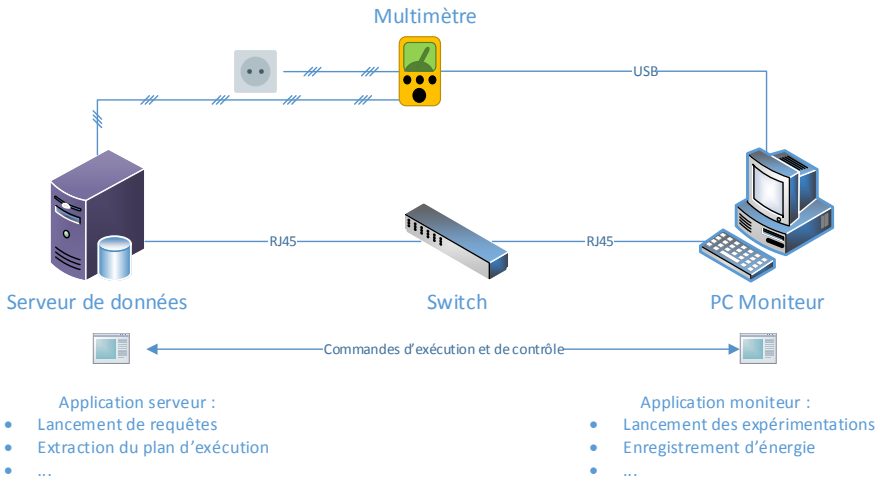


FIGURE 4.13 – Applications de gestion d'expérimentations.

Notre station de travail est installée avec la dernière version du SGBD Oracle 11gR2 sous Ubuntu Server 14.04 LTS avec noyau Linux 3.13. Afin de minimiser les influences indésirables, nous avons désactivé les tâches de fond inutiles et vidé le cache du système et le buffer d'Oracle avant chaque exécution de requête.

4.4.1.1 Exécution des requêtes et capture d'énergie

Pour réaliser nos expérimentations, nous avons développé une application et un script pour automatiser le processus des tests. La Figure 4.13 représente l'architecture logicielle des expérimentations. Une application client/serveur basée sur les sockets et implémentée en langage C est mise en œuvre. La machine de contrôle contient la partie serveur de l'application dont les tâches se résument à :

1. l'établissement d'une connexion via le réseau avec la station de travail ;
2. le lancement/l'arrêt des expérimentations ;
3. l'enregistrement des valeurs d'énergie via le multimètre en invoquant son pilote.

La station de travail contient la partie client de l'application et ses tâches consistent à :

1. l'établissement de la connexion avec l'application serveur ;
2. le vidage du cache de la BD à l'aide des commandes Oracle suivantes :

```
-- Clean the cache
ALTER SYSTEM FLUSH SHARED_POOL;
ALTER SYSTEM FLUSH BUFFER_CACHE;
```

3. le vidage du cache du système d'exploitation à l'aide du script *shell* suivant :

```
#!/bin/bash
# To free pagecache, dentries and inodes (linux kernel)
sync && echo 3 | tee /proc/sys/vm/drop_caches
```

4. la connexion avec le SGBD ;

Tableau 4.2 – Cardinalité des tables de faits du benchmark TPC-DS avec une taille de 100 Go.

Table	Cardinalité
store_sales	288 millions
store_returns	28,8 millions
catalog_sales	144 millions
catalog_returns	14,4 millions
web_sales	72 millions
web_returns	7,2 millions
inventory	400 millions

5. le lancement et la gestion d'ordonnancement de requêtes d'apprentissage et de tests ;
6. l'extraction du plan d'exécution avec tous ses informations (coût d'E/S et de CPU, temps d'exécution des opérateurs, etc.), à l'aide de la commande Oracle suivante :

```
SELECT DBMS_SQLTUNE.report_sql_monitor(
  sql_id       => '526mvccm5nfy4',
  type         => 'ACTIVE',
  report_level => 'ALL') AS report
FROM dual;
```

4.4.2 Jeu de données

Dans les expérimentations, nous utilisons les jeux de données et les requêtes issues des deux benchmarks TPC-H et TPC-DS [5, 6], interrogeant des données de 10 Go et de 100 Go de taille (facteur d'échelle). Le TPC-H est un benchmark décisionnel conçu pour le requêtage ad-hoc où les requêtes ne sont pas connues à l'avance. Le schéma modélise un modèle produits-commandes-fournisseurs contenant 2 tables de faits et 6 tables de dimensions (cf. Figure 4.14), avec une charge de 22 requêtes décisionnelles caractérisées par un large volume de données et un degré élevé de complexité. TPC-DS qui est le successeur annoncé de TPC-H, modélise un entrepôt de données. Le schéma de TPC-DS représente les fonctions décisionnelles d'un détaillant sous la forme de plusieurs schémas en flocon de neige. Il comprend 17 dimensions partagées par 7 tables de faits, ce qui le transforme en un schéma en constellation. Le Tableau 4.2 représente le nombre de lignes dans les tables de faits avec une taille de 100 Go (facteur d'échelle = 100). La charge de requêtes de TPC-DS est composée de 99 requêtes et constituée de quatre classes de requêtes : requêtes de rapports, requêtes décisionnelles ad-hoc, requêtes de traitement analytique en ligne (On-Line Analytical Processing, OLAP) et requêtes d'exploration de données [186].

L'ensemble des données et de requêtes des deux benchmarks ont été générés avec l'outil `dbgen` fourni par le TPC. Une fois les données générées, des scripts sont utilisés pour peupler la BD à l'aide du l'outil `sqlloader` fourni par Oracle. Dans le cas d'exécution des requêtes en mode isolé, chaque charge contient une seule requête (une seule requête est en exécution par le SGBD à un instant donné). Dans le cas du mode concurrent, nous avons varié le taux de multiprogrammation (MPL $\in \{2, \cdots, 10, 13, 15\}$).

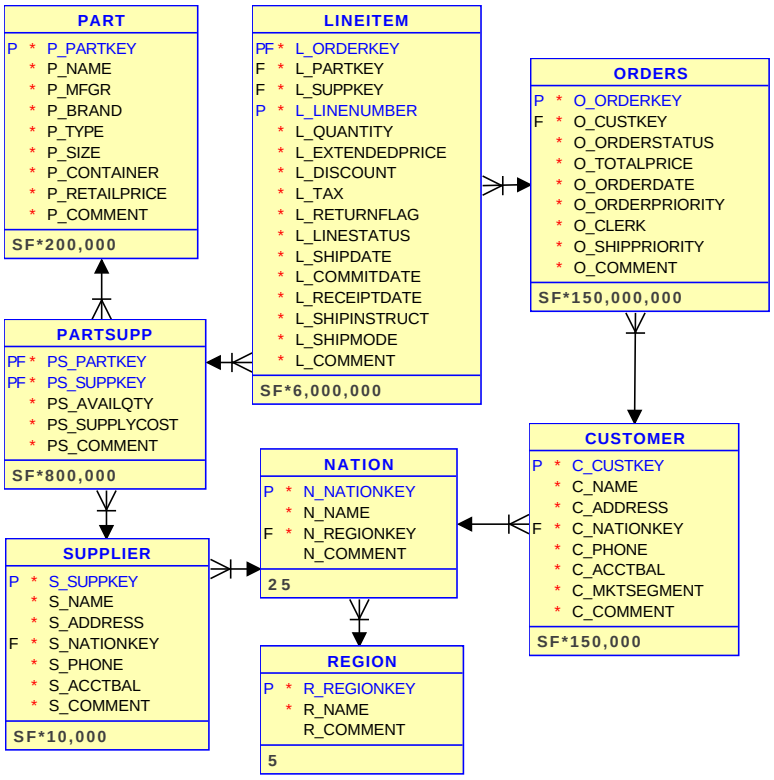
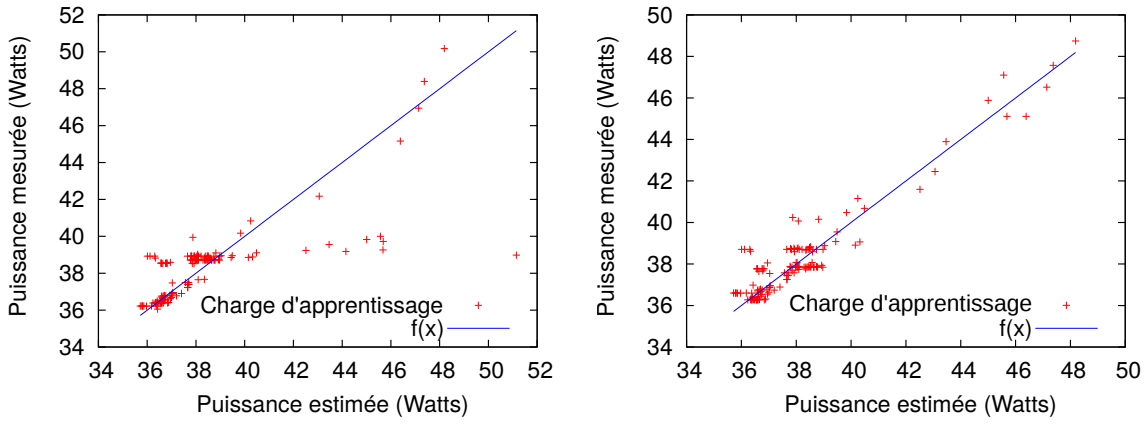


Figure 4.14 – Schéma du benchmark TPC-H.



(a) Modèle de régression linéaire

(b) Modèle de régression polynomiale

FIGURE 4.15 – La qualité des méthodes de régressions pour la phase d'apprentissage du modèle de coût énergétique.

4.4.3 Résultats pour une requête isolée

Comme décrit dans la section précédente, notre modèle de coût est composé de deux parties : les paramètres d'énergie qui peuvent être calculés par le biais d'une procédure d'apprentissage, et les paramètres de pipeline (p. ex. les coûts CPU et E/S) qui peuvent être lus à partir des statistiques de BD déjà disponibles. Dans cette section, nous présentons les résultats de ces deux étapes.

4.4.3.1 Résultat d'apprentissage

Dans la phase d'apprentissage, nous collectons à chaque exécution de requête, des statistiques relatives à l'exécution des pipelines et à la consommation d'énergie du système. Ces données collectées sont utilisées par le logiciel R pour trouver les meilleures valeurs des paramètres du modèle. Sur la base des résultats obtenus, une preuve probabiliste pour démontrer les bornes de confiance de notre modèle et des résultats est donnée dans l'Annexe B (page 203).

Les résultats de la phase d'apprentissage pour les valeurs retournées par le modèle linéaire et polynomiale (l'Équation (4.7) et l'Équation (4.12)) sont tracés dans la Figure 4.15.

Comme nous pouvons le voir sur la Figure 4.15a, il existe des différences significatives entre la consommation d'énergie estimée et réelle pour de nombreuses requêtes. Néanmoins, la consommation d'énergie estimée et réelle rapprochent de près les lignes diagonales lors de l'utilisation d'une régression polynomiale (Figure 4.15b).

4.4.3.2 Résultat avec TPC-H

Pour tester notre modèle avec un grand ensemble de données, nous exécutons tous les 22 requêtes du benchmark TPC-H avec une taille de deux facteurs d'échelle, 10 Go et 100 Go. Nous notons que la plupart des requêtes contiennent plus de 4 pipelines. Les résultats sont présentés dans le Tableau 4.3.

Tableau 4.3 – Erreurs d'estimation d'énergie dans les requêtes du benchmark TPC-H avec différentes tailles de BD.

Requête	Erreur (%)		Requête	Erreur (%)	
	10 Go	100 Go		10 Go	100 Go
$Q1$	1,2	0,9	$Q12$	1,9	0,05
$Q2$	10,1	8,9	$Q13$	12,8	6,1
$Q3$	1,0	0,1	$Q14$	0,4	0,6
$Q4$	0,5	0,3	$Q15$	2,3	1,0
$Q5$	1,2	0,6	$Q16$	0,4	1,7
$Q6$	1,3	0,1	$Q17$	0,2	1,4
$Q7$	1,1	0,7	$Q18$	1,9	3,7
$Q8$	0,5	0,1	$Q19$	0,6	1,0
$Q9$	1,9	0,8	$Q20$	1,8	2,1
$Q10$	0,6	1,2	$Q21$	0,9	0,4
$Q11$	4,8	0,3	$Q22$	0,007	2,1

Comme nous pouvons le voir dans le tableau, l'erreur moyenne est généralement très minimale (1,6% pour l'ensemble de données de 10 Go et 2,1% pour celui de 100 Go), et l'erreur maximale est habituellement au-dessous de 5%. Les plus grandes erreurs dans les estimations faites par notre modèle de coût (p. ex. *Q2* et *Q13*) sont dues à des erreurs d'estimation de cardinalité faite par l'optimiseur de requêtes.

Par exemple, le troisième pipeline de *Q2* de l'ensemble de données de 100 Go est le plus long pipeline dans la requête. Le nombre estimé de lignes est de seulement 180 692 lignes tandis que le nombre réel de lignes est de 1 362 975. Ainsi, les valeurs des coûts de CPU et de E/S sont systématiquement fausses. Puisque notre étude dans ce stade traite les SGBD comme une « boîte noire », nous ne pouvons pas vérifier le modèle basé sur des cardinalités réelles qui sont obtenues après l'exécution de la requête.

4.4.3.3 Résultat avec un schéma complexe : TPC-DS

Pour vérifier la portabilité du modèle sur un nouveau schéma de BD, nous avons pris le benchmark TPC-DS qui est plus complexe que le benchmark TPC-H en raison de son schéma diverse, sa distribution de données et sa charge de requête d'aide à la décision. Nous avons utilisé une BD d'une taille de 100 Go.

Motivé par [203], nous avons sélectionné 16 requêtes sur les 99 requêtes, en veillant à ce que les deux types soient présents (requêtes gourmandes en CPU et/ou en E/S). Le nombre moyen des pipelines est 7. La Figure 4.16 montre les résultats d'expérimentation. On constate que l'erreur des prédictions du modèle est inférieure à 10%. De même, nous observons la présence d'une seule requête avec une erreur importante, à savoir la *Q47*. En analysant la structure de cette requête, nous avons constaté que l'erreur vient du fait que le coût de la fonction analytique `rank()` n'est pas donné par le SGBD. Pour le vérifier, nous avons supprimé la partie de code SQL contenant la fonction `rank()` dans *Q47* et nous l'avons renommée et exécutée sous le nom de *Q47p*. Comme nous pouvons le voir dans la Figure 4.16, l'erreur d'estimation est descendue à 7,5%.

Cette situation (calcul du coût des agrégations et des fonctions d'analyse) doit être considérée par les

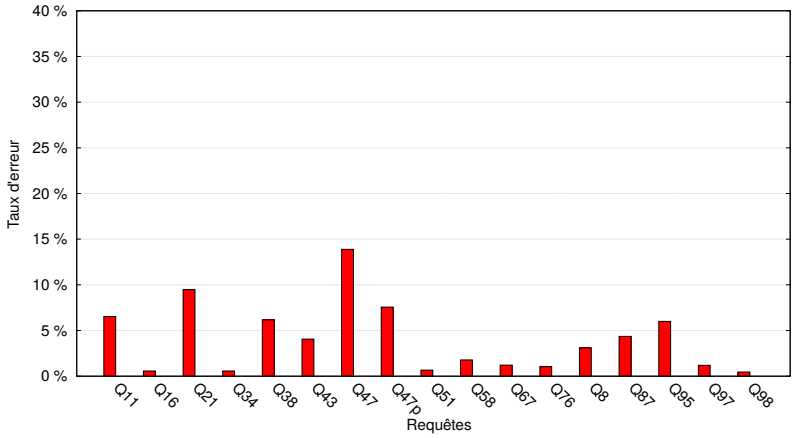


FIGURE 4.16 – Erreurs d'estimation d'énergie dans les requêtes du benchmark TPC-DS.

Tableau 4.4 – Erreurs d'estimation d'énergie dans les requêtes du benchmark TPC-H avec 2 Go de mémoire.

Requête	**Erreur 10 Go (%)**	**Requête**	**Erreur 10 Go (%)**
$Q1$	1,4	$Q12$	0,8
$Q2$	11,9	$Q13$	4,2
$Q3$	1,4	$Q14$	1,7
$Q4$	0,7	$Q15$	1,9
$Q5$	2,4	$Q16$	2,4
$Q6$	1,5	$Q17$	0,8
$Q7$	2,7	$Q18$	6,5
$Q8$	1,5	$Q19$	0,0
$Q9$	2,2	$Q20$	0,3
$Q10$	2,7	$Q21$	2,3
$Q11$	5,6	$Q22$	0,07

vendeurs de SGBD pour faciliter le développement de BD énergétique, comme précédemment souligné dans [116, 104].

4.4.3.4 Résultat avec une nouvelle configuration matérielle

Dans un autre test, nous avons réduit la mémoire de BD à 2 Go et exécuté la charge de requêtes du TPC-H. Les résultats sont présentés dans le Tableau 4.4. Dans ce scénario, lorsque les résultats intermédiaires des opérateurs (comme la phase de construction de la table de hachage ou de tri externe) ne peuvent pas tenir dans la mémoire disponible, le SGBD va les enregistrer sur le disque. Cela conduit à plus d'E/S et un changement de comportement de consommation d'énergie. L'expérience montre que l'erreur d'estimation moyenne est de 2,5%. Ainsi, notre modèle peut faire face à cette situation.

Tableau 4.5 – Erreurs d'estimation d'énergie pour les requêtes du benchmark TPC-H avec PostgreSQL.

Requête	Erreur (%)		Requête	Erreur (%)	
	10 Go	100 Go		10 Go	100 Go
$Q1$	1,03	0,2	$Q11$	4,2	-
$Q2$	-	-	$Q12$	0,9	0,02
$Q3$	1,5	1,2	$Q13$	4,7	4,4
$Q4$	0,6	0,5	$Q14$	2,8	2,4
$Q5$	1,2	3,07	$Q15$	0,4	2,7
$Q6$	4,1	2,7	$Q16$	5,4	0,03
$Q7$	0,4	1,4	$Q18$	0,4	-
$Q8$	0,07	1,09	$Q19$	1,6	0,9
$Q10$	0,6	0,3	$Q22$	1,2	0,4

4.4.3.5 Résultat avec un nouveau SGBD

Dans ces expérimentations, nous avons à la fois changé le SGBD et le matériel de déploiement pour voir la robustesse de notre modèle. Précisément, nous avons utilisé le SGBD libre PostgreSQL en version 9.4.5, et comme matériel, nous avons utilisé une station de travail Dell PowerEdge R310 ayant un processeur Xeon X3430 de 2,40 GHz et 32 Go de mémoire DDR3. Pour la BD, nous avons utilisé le même benchmark TPC-H avec un facteur d'échelle de 10 et de 100 Go. Les résultats de l'exécution de la charge de requêtes sont présentés dans le Tableau 4.5. Nous notons que certaines requêtes ont été interrompues car elles dépassent les 72 heures d'exécution dans notre environnement de test.

Les résultats dans cette nouvelle configuration logicielle et matérielle sont aussi de bonne qualité, comme nous pouvons le voir dans le tableau. L'erreur moyenne est 0,1% dans les deux ensembles de données (100 Go et 10 Go) et l'erreur maximale est au-dessous de 6%. L'expérience montre la robustesse de notre modèle de coût, indiquant qu'il est suffisamment prêt pour être intégré dans les structures d'optimisation d'énergie.

4.4.4 Résultats pour des requêtes concurrentes

Dans cette section, nous présentons les résultats d'évaluation de notre modèle de coût énergétique dans le cas des requêtes exécutées en mode concurrent.

4.4.4.1 Résultat d'apprentissage

Dans la phase d'apprentissage, nous avons créé une charge de requêtes avec 12 modèles de requête issus du benchmark TPC-H. Plus précisément, les modèles que nous avons utilisés sont les requêtes : 1, 3, 4, 5, 9, 11, 14, 16, 18, 19, 21, et 22, avec le facteur d'échelle de 10 Go. Nous nous référons à cet ensemble comme l'ensemble des modèles *connus*. Pour chaque MPL, nous avons généré des mixes de requêtes contenant le même modèle de requête, mais avec différentes instances en utilisant des outils de TPC-H. Ceci résulte en 287 mixes de pipeline composés de 132 charges de requêtes. À chaque exécution d'un mix de requêtes, nous collectons les statistiques des pipelines et nous enregistrons la consommation

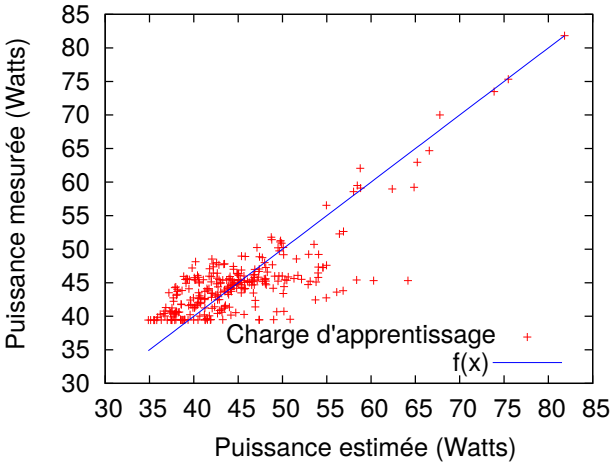


FIGURE 4.17 – La qualité de la régression polynomiale pour la phase d'apprentissage du modèle de coût énergétique en mode concurrent.

d'énergie du système. Ces données sont utilisées par le logiciel R pour trouver les meilleures valeurs de nos paramètres du modèle. Nous ne considérons pas de grandes tailles de données dans cette partie d'étude, car les requêtes prennent beaucoup de temps pour terminer leur exécution. Par exemple, un simple mix de cinq requêtes générées à partir du benchmark TPC-H avec une taille de 100 Go prend 3 heures pour terminer son exécution. Cela peut requérir des semaines d'apprentissage et de test pour notre modèle. En outre, il est probable dans ce cas qu'un mix de requêtes pourrait compléter l'ensemble de leurs exécutions dans un mode isolé plus rapide que l'exécution dans un mode concurrent.

Les résultats de la phase d'apprentissage comparés aux valeurs ajustées du modèle polynomial sont illustrés par la Figure 4.17. Comme nous pouvons voir sur la figure, la consommation d'énergie estimée et réelle rapprochent de près les lignes diagonales, indiquant ainsi la qualité de l'estimation que notre modèle peut générer. Il y a un certain écart entre l'estimation et l'énergie observée pour certains mixes de requêtes. Comme dans le cas du mode d'exécution isolé, une grande partie de ces erreurs est provoquée par l'optimiseur de requêtes, qui ne réussit pas à estimer les cardinalités de façon correcte lors de la phase de génération des plans d'exécution.

4.4.4.2 Résultat avec des requêtes connues

Pour évaluer notre modèle, nous avons d'abord créé des charges de requêtes basées sur les modèles de requête TPC-H *connus* d'une façon aléatoire. Pour chaque MPL, nous avons généré de nouveaux mixes à partir des 12 modèles, qui se traduit par 132 charges de requêtes. Cependant, la génération aléatoire de mixes (échantillonnage) peut produire des mixes contenant les mêmes requêtes. Ainsi, cette technique ne couvre pas bien l'espace de mixes de requêtes possibles. Par conséquent, nous avons utilisé la méthode d'échantillonnage par hypercube latin (Latin Hypercube Sampling (LHS)) [10]. LHS crée un hypercube avec la même dimensionnalité que le nombre de modèles de requête T. Chaque dimension est divisée en n classes égales. La classe i représente le nombre d'instances possibles du modèle de requête i. LHS sélectionne alors n mixes de l'espace de telle sorte que chaque classe de chaque modèle de requête apparaît dans exactement un mix. Intuitivement, LHS a une meilleure couverture de l'espace de mixes

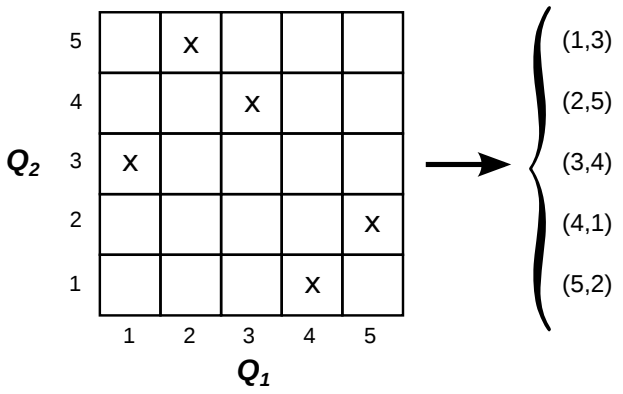


FIGURE 4.18 – Exemple d'échantillonnage par hypercube latin en 2-D.

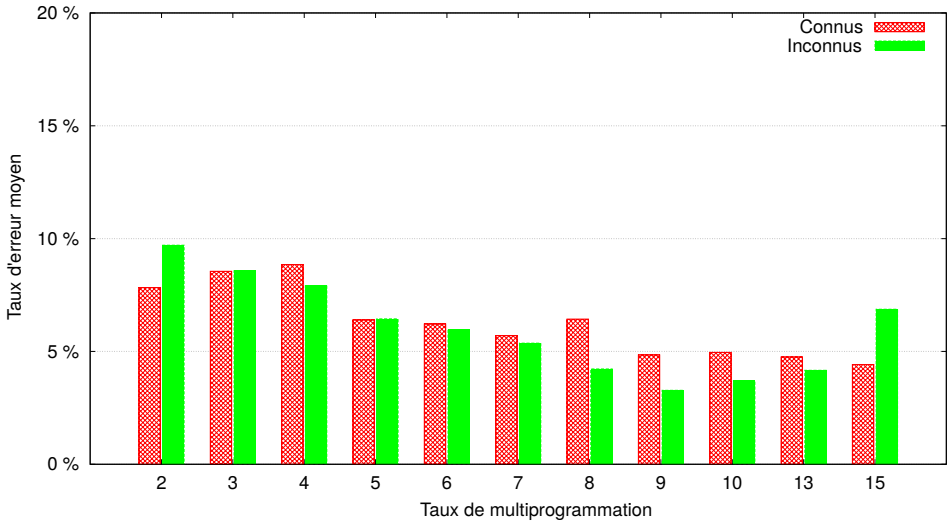


FIGURE 4.19 – Erreurs d'estimation d'énergie dans le benchmark TPC-H pour les modèles de requêtes connus et inconnus.

que la technique d'échantillonnage aléatoire.

La Figure 4.18 montre un exemple où nous avons deux modèles de requête ($T = 2$), et nous devons sélectionner $n = 5$ mixes en utilisant LHS. Les deux axes Q_1 et Q_2 dans la figure désigne le nombre d'instances de requêtes de chaque type de requête dans un mix. La méthode d'échantillonnage par hypercube latin divise chacune de ces dimensions en 5 intervalles égaux. Les symboles « x » désignent l'ensemble des mixes sélectionnés par la méthode LHS.

Nous avons testé la précision de la prédiction de notre modèle sur les mixes générés des modèles connus. La Figure 4.19 montre les résultats des expérimentations.

Comme illustré par la figure, l'erreur moyenne sur tous les est de 6,3% qui est considérée petite étant donné la variété de niveau de concurrence. En outre, l'erreur maximale est généralement inférieure à 9%.

Notons en passant que nous pouvons voir que l'erreur est légèrement plus grande lorsque le MPL est petit (MPL = 2, 3 ou 4). Nous avons constaté que lorsque le MPL est élevé, la durée d'exécution des phases de pipeline est grande et notre modèle peut capturer leur consommation d'énergie. À l'inverse,

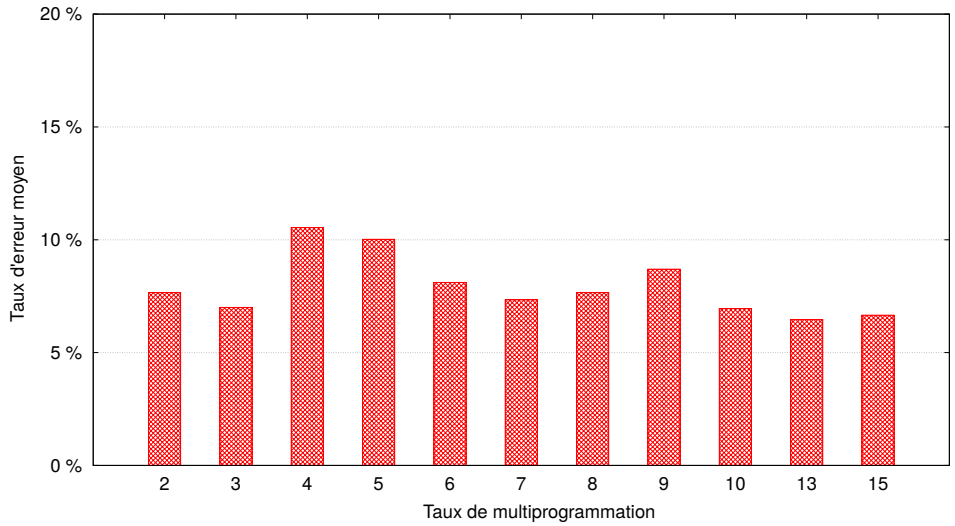


FIGURE 4.20 – Erreurs d'estimation d'énergie dans les requêtes du benchmark TPC-DS en mode concurrent.

lorsque le MPL est petit, certaines phases de pipeline finissent leur exécution rapidement et notre modèle ne peut pas estimer leur consommation d'énergie avec précision. Ce problème est également observé dans certaines autres expérimentations.

4.4.4.3 Résultat avec des requêtes inconnues

Afin de tester notre modèle de coût contre de *nouveaux* modèles de requêtes inconnues, nous avons créé des charges de requêtes basées sur le reste des requêtes du benchmark TPC-H qui ne sont pas utilisés dans la phase d'apprentissage, en utilisant la technique LHS. Plus précisément, pour chaque MPL nous avons généré 10 nouveaux mixes de requêtes, qui donnent lieu à 110 charges de requêtes. Les résultats sont présentés dans la Figure 4.19. On peut observer que les erreurs des prédictions du modèle de coût sont moins de 10% pour les modèles de requêtes *inconnus*, confirmant ainsi sa robustesse.

4.4.4.4 Résultat avec un schéma complexe

Pour vérifier la portabilité du modèle sur un nouveau schéma, nous avons choisi le schéma du benchmark TPC-DS. Nous nous sommes basés sur le même ensemble de 16 requêtes sélectionné dans le cas du mode d'exécution isolé. En utilisant la technique LHS, nous avons généré 16 mixes de requêtes pour chaque MPL, qui se traduit par 176 charges de requête.

La Figure 4.20 montre les résultats d'expérimentations. On peut observer que l'erreur des prédictions du modèle est inférieure à 10,5%, ce qui montre que la modélisation de pipeline est un indicateur robuste pour la prédiction d'énergie dans le cas des données et des requêtes inconnues.

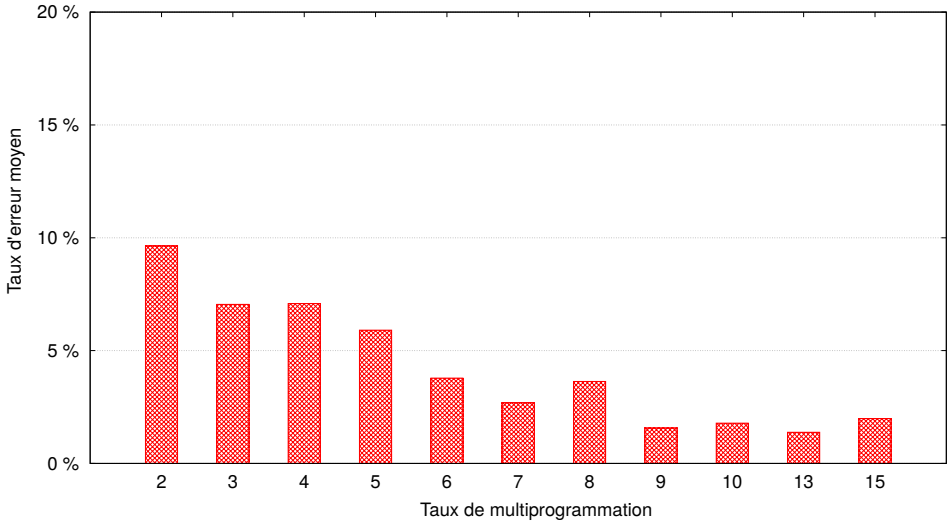


FIGURE 4.21 – Erreurs d'estimation d'énergie dans le benchmark TPC-H pour tous les modèles de requêtes.

4.4.4.5 Résultat avec des charges de requêtes aléatoires

Enfin, nous évaluons la précision de notre modèle pour prédire la consommation d'énergie pour tous les modèles de requête TPC-H de façon plus réaliste. Dans ces expérimentations et pour créer la charge de requêtes, nous passons par la liste de modèles de requêtes et nous ajoutons une instance de chaque modèle. En passant par la liste de modèles de nouveau d'une manière round-robin, nous continuons d'ajouter une instance de chaque modèle de requête à la charge jusqu'à ce que toutes les requêtes sont ajoutées. En se basant sur les 22 requêtes de TPC-H, nous avons utilisé 4 instances pour chaque modèle de requêtes, ce qui résulte en une charge de 88 requêtes. Pour chaque MPL M, nous exécutons le premier mix de requête M, chaque fois qu'une requête se termine parmi les M requêtes en cours d'exécution, une nouvelle requête à partir de la charge sera programmée à sa place sur la base de la politique d'ordonnancement First in, First out (FIFO). Cela nous donne une charge de requêtes d'une longue durée d'exécution qui prend en moyenne 2 heures pour terminer.

Les résultats sont présentés dans la Figure 4.21. Comme dans l'expérimentation précédente, les petits MPL donnent des erreurs importantes par rapport aux grands MPL. Notre suggestion pour traiter ce scénario, est de modéliser l'énergie pour chaque groupe de MPL, par exemple, la création de modèle pour MPL $M1 \in \{2, \cdots, 5\}$, $M2 \in \{6, \cdots, 10\}$, $M3 \in \{11, \cdots, 15\}$ et ainsi de suite. Pour un futur travail, nous pouvons étendre notre framework avec une stratégie de regroupement similaire. L'expérimentation montre que l'erreur d'estimation maximale est de 9,6%. Ainsi, notre modèle peut faire face à des charges de requêtes réalistes.

4.4.5 Bilan et discussion

Les résultats expérimentaux ci-dessus indiquent clairement que notre modèle offre des résultats d'estimation d'énergie comparables à ceux donnés par les travaux connexes manipulant des modèles de requêtes connus exécutées en mode isolé [282, 152]. De plus, notre framework offre une amélioration

significative par rapport aux techniques existantes, car il peut offrir une précision de prédiction similaire dans des scénarios plus complexes (à savoir des modèles de requêtes et des ensembles de données inconnus) et les requêtes exécutées en parallèle avec un taux de concurrence réaliste.

4.5 Conclusion

Dans nos jours, la consommation d'énergie des ordinateurs est devenue une préoccupation majeure pour l'industrie et la société. En conséquence, les chercheurs ont mis au point des techniques matérielles et logicielles pour assurer une efficacité énergétique.

Dans ce chapitre, nous proposons un modèle de coût basé sur les pipelines pour estimer la consommation d'énergie lors de l'exécution d'une requête SQL et un ensemble de requêtes exécutées simultanément. Nous avons aussi indiqué comment la modélisation de pipeline pourrait être un indicateur robuste pour la prédiction d'énergie. Le modèle de coût est construit sur la base des résultats empiriques obtenus à partir d'une charge de requêtes d'apprentissage qui a été soigneusement créée. Les paramètres matériels sont obtenues par un algorithme de régression polynomiale multiple. En outre, nous avons effectué des tests sur notre modèle avec une BD réelle en exécutant un ensemble de requêtes, et avons comparé leurs coûts énergétiques avec ceux prédits par le modèle. Nos résultats montrent que le modèle, qui utilise uniquement des coûts de l'optimiseur en entrées, peut prédire l'énergie avec une petite erreur.

Chapitre

5

Intégration de l'énergie dans le traitement de requêtes d'un SGBD relationnel

« Energy and persistence conquer all things. »
— Benjamin Franklin

Sommaire

5.1 Introduction

Dans les centres de données, la majorité des ressources de calculs et de stockage sont dédiées aux serveurs de bases de données, ce qui rond les SGBDs l'un des principaux consommateurs d'énergie. Cela est dû à la grande quantité de données interrogée par des requêtes complexes ; exécutées quotidiennement par un SGBD. La performance est l'objectif principal qui vient à l'esprit lorsque nous parlons des capacités des SGBDs. Ceci est particulièrement nécessaire pour les scénarios de traitement de requêtes transactionnelles en ligne (OLTP) où des centaines et des milliers de transactions comme la mise à jour ou l'insertion de nouveaux tuples dans une relation doivent être exécutées en temps réel. Au fil des années, le temps de traitement des requêtes complexes a diminué grâce à diverses améliorations apportées aux composants matériels et logiciels.

Comme déjà indiqué, les SGBDs existants se concentrent à la haute performance lors de la phase de traitement des requêtes, tout en ignorant généralement la consommation d'énergie. Cependant, des études récentes indiquent qu'une bonne performance n'est plus le seul critère de qualité pour un « bon » SGBD. Selon le rapport Claremont sur la recherche pour les bases de données, un SGBD éco-énergétique deviendra l'un des principaux domaines de recherche à l'avenir [8, 7].

Dans ce chapitre, nous proposons une méthodologie, soutenue par un outil appelé *EnerQuery*, qui crée des bases de données tout en économisant l'énergie lors de la phase de traitement des requêtes. Pour montrer son efficacité, nous mettons en œuvre cette outil dans le module de traitement de requêtes du SGBD libre PostgreSQL. Un modèle mathématique de coût est utilisé pour estimer la consommation d'énergie. Les paramètres de ce dernier sont identifiés par une méthode d'apprentissage automatique. Nous avons mené des expériences intensives en utilisant nos modèles de coûts, et un outil de mesure dédié à calculer l'énergie, tout en basant sur des données de benchmark TPC-H, afin de montrer l'efficacité de nos propositions.

5.2 Pourquoi le traitement des requêtes ?

Dans ce chapitre, nous nous concentrons sur le module de traitement de requêtes, qui fait partie des principales composantes d'un SGBD. Plusieurs travaux ont été élaborés dans le domaine de traitement des requêtes depuis le début des années 1970 pour les bases de données traditionnelles. Parmi ces travaux, divers algorithmes et systèmes ont été proposés, tels que le projet *System-R*, dont les résultats ont été largement intégrées dans de nombreux optimiseurs commerciaux (voir Chapitre 2).

Pour améliorer l'efficacité énergétique lors de l'exécution des requêtes, deux approches sont possibles : (1) nous pouvons soit réduire le temps d'exécution de la requête, (2) soit réduire l'énergie consommée. Le premier aspect a été étudié de manière exhaustive dans plusieurs travaux de littérature, ce qui a conduit au développement des SGBDs relationnels à hautes performances d'aujourd'hui. Le deuxième aspect a été ignoré dans la plupart des études antérieures, mais il est de plus en plus pris en compte dans les projets de recherche récents comme nous avons vu dans le Chapitre 3.

La phase de traitement de requêtes est une tâche primordiale, car elle est au cœur d'un SGBD. Cette phase doit être exécutée constamment dans le cas des entreprises et des sites web. L'optimiseur de requêtes peut générer un grand nombre de plans d'exécution pour la même requête, ce qui donne de

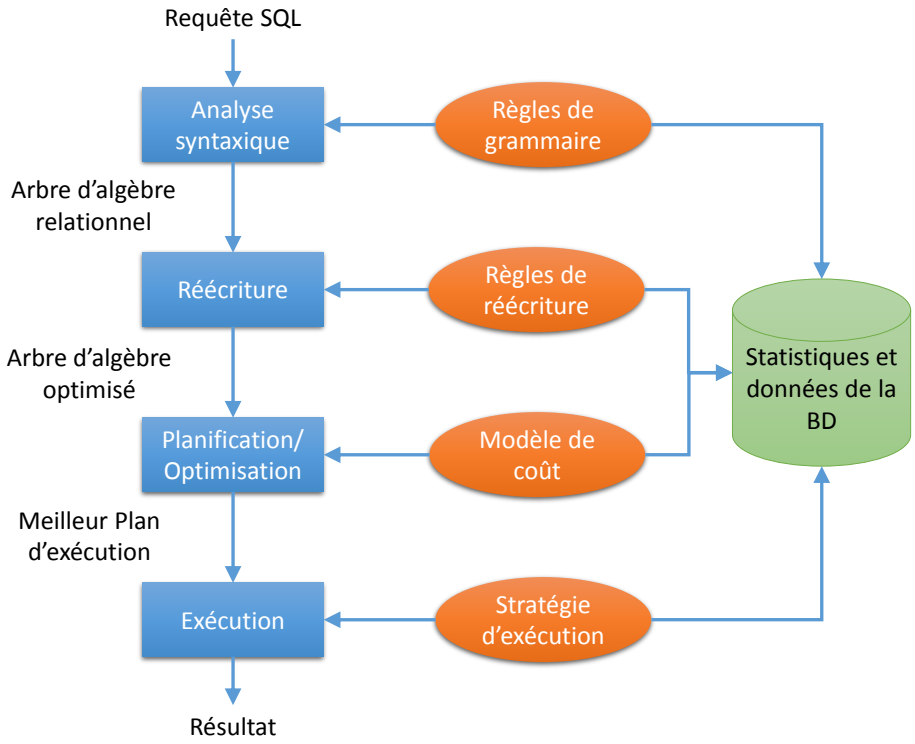


FIGURE 5.1 – Étapes de traitement de requêtes dans PostgreSQL.

diverses choix de compromis entre les coûts de performance et d'énergie. De plus, un SGBD maintient des statistiques sur les données de la base dans le but d'un traitement efficace des requêtes. Ces informations peuvent être directement utilisées pour dériver le profil d'énergie des requêtes, et aider à la prise de décision liée à la minimisation d'énergie.

5.3 Démarche d'intégration de l'énergie dans le traitement de requêtes

Afin de concevoir un traitement de requêtes éco-énergétique, nous proposons tout d'abord un audit de chaque composant, dans le but de vérifier s'il est sensible à l'énergie ou non. Après cette vérification, nous présentons en détail notre méthodologie pour construire le module de traitement de requêtes.

5.3.1 Un audit de traitement de requêtes

Rappelons que le module de traitement de requête est responsable de l'exécution des requêtes suivant un ou plusieurs besoins non fonctionnels, tels que le temps de réponse. Le processus d'exécution d'une requête donnée passe généralement par quatre étapes principales : (i) l'analyse syntaxique, (ii) la réécriture, (iii) la planification et l'optimisation et (iv) l'exécution (cf. Figure 5.1). Pour illustrer ces étapes, nous considérons comme une étude de cas le SGBD PostgreSQL [17].

PostgreSQL est un système de gestion de base de données relationnelle et objet. C'est un outil libre

17. https://www.postgresql.org/

et open-source développé à l'université de Californie à Berkeley par une équipe sous la direction de Michael Stonebraker en 1985. Le nom original était POSTGRES et fut renommé Postgres95. Ce dernier fut changé à la fin de 1996 en PostgreSQL. Il supporte diverses plates-formes matérielles et différents systèmes d'exploitation. PostgreSQL est entièrement conforme à ACID (atomicité, cohérence, isolation et durabilité), avec la prise en charge de nombreuses fonctionnalités comme les clés étrangères, jointures, vues matérialisées, triggers, récupération ponctuelle, tablespaces, réplication asynchrone, transactions imbriquées, sauvegardes en ligne, etc [18].

5.3.1.1 Analyse syntaxique

L'analyseur syntaxique doit vérifier la chaîne de caractères de la requête pour valider sa syntaxe à l'aide d'un ensemble de règles de grammaire. Si la syntaxe est correcte, un *arbre syntaxique* est construit et retourné. La fonction responsable de cette tâche dans PostgreSQL est : `raw_parser()` située dans le fichier : `src/backend/parser/parser.c`. L'analyseur défini dans `gram.y` et `scan.l` est construit en utilisant les outils Unix `bison` et `flex`. Après la fin de la tâche d'analyseur, le processus de transformation prend l'arbre syntaxique en entrée et fait l'interprétation sémantique nécessaire pour comprendre quelles tables, fonctions et opérateurs référencés par la requête. La structure de données construite pour représenter cette information est appelée *arbre de requête*. La fonction de transformation nommée `parse_analyze()` est située dans le fichier `src/backend/parser/analyze.c`. Le coût de cette phase est généralement ignoré par le SGBD car elle se termine très rapidement. Nous suivons la même logique et nous supposons que la consommation d'énergie de cette étape est négligeable.

5.3.1.2 Réécriture

La phase de réécriture (ou simplification) des requêtes traite l'arbre remis par l'étape d'analyse syntaxique et le réécrit à un autre arbre en utilisant un ensemble de règles. Ces dernières sont définies par l'utilisateur ou le système. Cette phase basée sur des règles est également utilisée dans la réécriture des requêtes dans le cas des vues matérialisées. La fonction de réécriture est nommée `QueryRewrite()` définie dans le fichier `src/backend/rewrite/rewriteHandler.c`. Quant à l'étape précédente, le coût est ignoré en raison de l'achèvement rapide de cette phase.

5.3.1.3 Planification/optimisation

La tâche du planificateur/optimiseur est de créer un plan d'exécution optimal. Une requête SQL donnée peut effectivement être exécutée de différentes manières, dont chacune va produire le même ensemble de résultats. La tâche de l'optimiseur est d'estimer le coût de l'exécution de chaque plan en utilisant une approche basée sur les coûts, et de trouver celle qui doit être exécutée rapidement.

5.3.1.3.1 Planification

Le planificateur commence par générer des plans d'exécution pour la lecture de chaque relation individuelle (table) utilisée dans la requête. Les plans possibles sont déterminés par les *indexes* disponibles

18. https://www.postgresql.org/docs/current/static/intro-whatis.html

Tableau 5.1 – Planification de la requête *Q8* de TPC-H avec différentes stratégies de recherche.

Stratégie de recherche	Temps de planification (s)	Énergie (J)
Par défaut	0,110006	5200,362
Algorithme génétique	0,977013	5387,648
Manuelle	0,092054	5160,036

sur chaque relation. Il y a toujours la possibilité d'effectuer un *accès séquentiel* sur une relation, donc un plan de accès séquentiel est toujours créé. Si la requête nécessite la jointure de deux ou plusieurs relations, tous les plans possibles pour joindre ces relations sont générés. Les stratégies de jointure disponibles dans PostgreSQL sont : *jointure par boucle imbriquée*, *jointure par fusion* et *jointure par hachage*. Lorsque la requête implique plus de deux relations, le résultat final doit être construit par un arbre de jointures, chacune avec deux entrées. Le planificateur examine les différentes séquences de jointure possibles pour trouver le moins cher en terme de coût. Si la requête utilise un nombre de relations moins d'un certain seuil défini (fixé à 12 relations dans PostgreSQL), une recherche quasi-exhaustive est effectuée pour trouver les meilleures séquences de jointure ; autrement, un algorithme heuristique à base de génétique est utilisé.

Pour étudier les effets de ces stratégies de recherche, considérons la requête *Q8* du benchmark TPC-H [5]. C'est une requête complexe qui implique la jointure de 7 tables. Nous modifions le planificateur de PostgreSQL selon trois manières : (i) la recherche d'un plan d'exécution en utilisant la stratégie du SGBD actuelle (ii) en utilisant l'algorithme génétique, et (iii) manuellement en forçant le planificateur de choisir un certain plan. Pour chaque stratégie, nous calculons son temps d'exécution et sa consommation totale d'énergie lors de l'exécution de la requête sur une base de données d'une taille de 10 Go. Les résultats sont présentés dans le Tableau 5.1.

D'après le tableau, on peut constater que la stratégie de choix d'un plan de requête manuel donne les meilleurs résultats, en terme de temps et d'énergie. Alors que la stratégie de recherche par défaut (semi-exhaustive) conduit à un temps d'exécution et de consommation d'énergie un peu plus élevée. L'algorithme génétique donne les pires résultats dans cet exemple, peut-être en raison du petit nombre de tables présent dans la requête, puisque cette stratégie est utilisée par le SGBD où il y a plus de 12 tables. Compte tenu de ce petit nombre de tables, si nous allons dans les bases de données opérationnelles réelles où il y a une centaine de tables, la stratégie de recherche utilisé par le planificateur peut conduire à une consommation d'énergie notable. Ainsi, la stratégie de recherche des plans d'exécution de requêtes manuellement par l'administrateur de base de données est recommandée dans les grandes bases de données pour gagner en efficacité énergétique.

5.3.1.3.2 Optimisation

Pour évaluer le temps de réponse pour chaque plan d'exécution, des fonctions de coût sont définis pour chaque opérateur SQL basique. La formule générale pour estimer le coût de l'opérateur *op* peut être exprimée comme :

$$Coût_{op} = \alpha \times E/S \ \oplus \ \beta \times CPU \ \oplus \ \gamma \times Com \tag{5.1}$$

Tableau 5.2 – Présentation des paramètres de modèle de coût de PostgreSQL.

T	Désignation	Définition	c
s	seq_page_cost	le coût d'E/S pour accéder séquentiellement à une page	1,0
r	random_page_cost	le coût d'E/S pour accéder aléatoirement à une page	4,0
t	cpu_tuple_cost	le coût CPU pour traiter un tuple	0,01
i	cpu_index_tuple_cost	le coût CPU pour traiter un tuple via un accès index	0,005
o	cpu_operator_cost	le coût CPU pour effectuer une opération (ex : hachage)	0,0025

Où E/S, CPU et Com sont des estimations pour le nombres de pages, nombres de tuples et de messages de communication, respectivement, nécessaires pour exécuter l'opérateur op. Ils sont généralement calculés à l'aide des statistiques de base de données et des formules de sélectivité. Les coefficients α, β et γ sont utilisés pour convertir les estimations à l'unité souhaitée (par exemple, le temps, l'énergie, etc). $\oplus$ représente la relation entre les paramètres (linéaires, non-linéaires).

L'optimiseur de PostgreSQL utilise cinq types de paramètres dans son modèle de coût, $T \in \{s, r, t, i, o\}$. Le coût d'un opérateur op dans le plan de requête est calculé comme suit :

$$Coût_{op} = \sum_{i=1}^{T} c_i \times n_i \tag{5.2}$$

Où n représente l'estimation de coût d'opérations de type T requises pour exécuter l'opérateur, et c est le coefficient de coût de performance pour le même type. Les fonctions de calcul des coûts d'un opérateur seront représentées dans la Section 5.4.1.1. Le Tableau 5.2 donne un aperçu des types d'opérateur définis dans PostgreSQL et les valeurs par défaut des coefficients de coût de performance. Le coût total estimé d'une requête est donc la somme des coûts des opérateurs individuels dans le plan d'exécution.

Les paramètres des coefficients et leur relation peuvent être obtenus en utilisant diverses techniques telles que la calibration, la régression et les statistiques. Ainsi, un modèle de coût d'énergie doit être défini à ce niveau, avec les nouveaux paramètres T les plus pertinents.

L'arbre du plan d'exécution final est constitué des opérations d'accès séquentiels ou d'accès par index des relations de base, plus des opérations de jointure par boucle imbriquée, fusion ou hachage, ainsi que des étapes auxiliaires, tels que les opérations de tri ou d'agrégation.

5.3.1.4 Exécution

L'exécuteur prend le plan créé par le planificateur/optimiseur et le traite récursivement pour extraire l'ensemble de tuples requis dans un mécanisme de *pipeline*. Chaque fois qu'un nœud du plan est appelé, il doit livrer le prochain tuple, ou signaler qu'il n'y a plus de tuples à livrer. Les requêtes complexes peuvent impliquer de nombreux niveaux de nœuds dans leurs plan d'exécution, mais l'approche générale reste la même : chaque nœud calcule et retourne sa prochaine sortie de tuple, à chaque fois qu'il est appelé. Chaque nœud est également responsable de l'application des expressions de sélection ou de projection qui ont été attribuées par le planificateur.

Pour étudier l'effet de l'étape d'exécution sur l'éco-conception d'un optimiseur de requêtes, nous

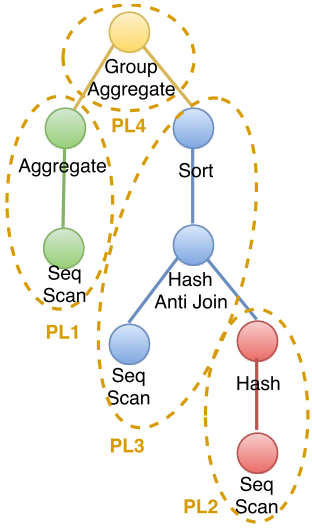


FIGURE 5.2 – Plan d'exécution du requête *Q22* issue du benchmark TPC-H avec l'annotation de pipeline correspondante.

considérons un exemple de requête *Q22* issue du benchmark TPC-H. La Figure 5.2 présente le plan d'exécution renvoyé par l'optimiseur de requête de PostgreSQL. Comme nous l'avons montré dans le Chapitre 4, la consommation d'énergie est directement influencée par le modèle d'exécution du SGBD. Par conséquent, le plan d'exécution peut être divisé en un ensemble de segments, nous nous référons à ces segments comme *pipelines*. Ces dernières correspondant à l'exécution simultanée d'une séquence contiguë d'opérateurs. La segmentation du pipeline du plan d'optimiseur pour la requête *Q22* est représentée sur la Figure 5.2. Dans cette dernière, il y a 4 pipelines, et l'ordre d'exécution de ces pipelines est géré par les opérateurs de blocage (par exemple, PL3 ne peut débuter jusqu'à ce que PL2 se termine).

Dans notre étude précédente (cf. Chapitre 4), nous avons montré que *lorsqu'une requête passe d'un pipeline à l'autre, sa consommation d'énergie change aussi*. Lors de l'exécution d'un pipeline, la consommation d'énergie a généralement tendance à être constante. Par conséquent, ce modèle d'exécution de pipeline est très important, car il a un impact direct sur la consommation d'énergie lors de l'exécution d'une requête. L'éco-conception d'un optimiseur de requête devrait prendre en considération la stratégie d'exécution. Cet aspect n'est pas pris en charge dans les travaux de Xu *et al.* [283].

5.4 Méthodologie de conception d'un module de traitement de requêtes éco-énergétique

Dans cette section, nous décrivons la méthodologie de conception et la mise en œuvre de notre proposition dans le SGBD PostgreSQL. Comme nous l'avons mentionné ci-dessus, les phases de planification/optimisation et d'exécution ont un impact sur la consommation d'énergie, et doivent être prises en compte dans la conception d'un module de traitement de requêtes éco-énergétique. Le workflow de notre méthodologie est décrit dans la Figure 5.3. Nous avons étendu le module de traitement de requêtes de PostgreSQL pour inclure un modèle de coût estimant l'énergie des opérateurs SQL. Nous avons aussi

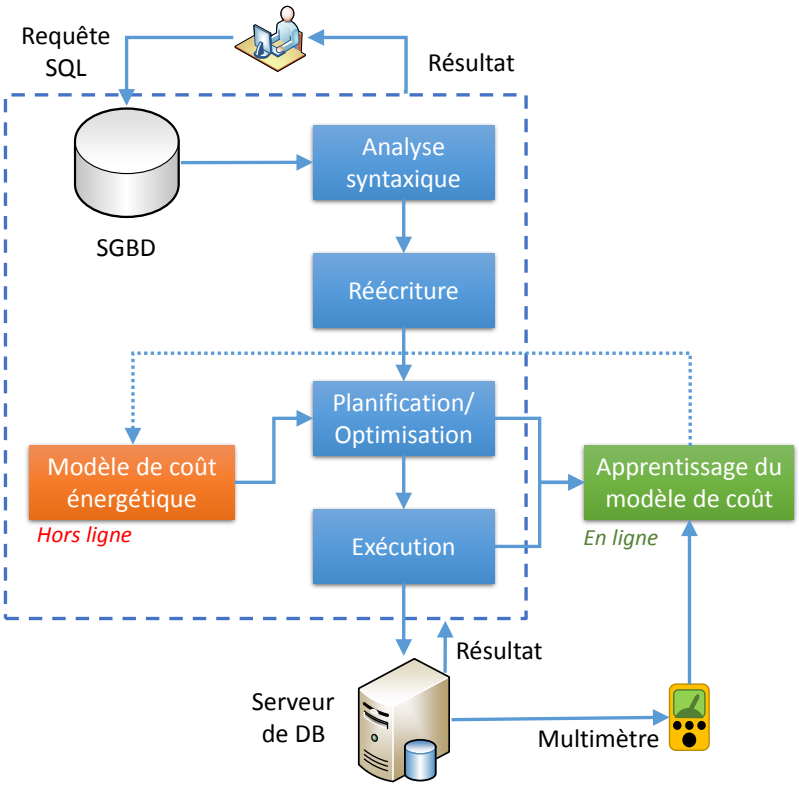


FIGURE 5.3 – La méthodologie de conception

modifié le processus de sélection des plans d'exécution *mono-objectif* en un processus *multi-objectifs* tout en incluant la dimension énergétique. Notre démarche ne génère pas de frais généraux au processus de traitement de requêtes car nous avons implémenté notre modèle énergétique à côté du modèle de performance existant, pour ne pas refaire les calculs pour chaque opérateur.

5.4.1 Modèle de coût d'énergie

Dans cette section, nous présentons notre méthodologie pour l'estimation de la consommation d'énergie des requêtes dans PostgreSQL. Nous nous sommes basé sur notre démarche introduite dans le Chapitre 4 pour construire notre modèle. Les caractéristiques de ce modèle comprennent : (i) la segmentation d'un plan d'exécution en un ensemble de pipelines, (ii) l'utilisation des paramètres de pipeline pour construire le modèle en utilisant des techniques d'apprentissage automatique (*hors ligne*), et (iii) l'estimation de l'énergie du nouveau pipeline en se basant sur le modèle de coût final (*en ligne*). Les fonctions de régression et de calibration sont les mêmes que celles utilisées dans le Chapitre 4.

5.4.1.1 Paramètres du modèle de coût

Étant donné une requête SQL, l'optimiseur de requête est responsable de l'estimation des coûts d'E/S et de CPU pour chaque opérateur algébrique de base. Notre stratégie pour la modélisation d'énergie est d'étendre le modèle de coût de ces opérateurs qui sont intégrés dans le SGBD PostgreSQL, afin d'estimer

sa consommation d'énergie. Pour chaque opérateur, les paramètres dans l'Équation (4.3) (les coûts d'E/S et de CPU) doivent être modifiés en fonction de son implémentation et comportement d'exécution. La suite de cette section sera consacrée à présenter des modèles d'énergie pour un ensemble d'opérateurs relationnels de base.

5.4.1.1.1 Accès séquentiel

L'opération d'accès séquentiel parcourt la relation complète telle qu'elle est stockée sur disque, puisque chaque page et tuple doivent être examinés. Le coût d'E/S est égal au nombre de pages dans la relation (p), et le coût CPU est égal au nombre de tuples (t).

5.4.1.1.2 Accès par index

L'opération d'accès par index réalise un parcours d'un arbre d'index, passe à travers des nœuds feuilles pour trouver toutes les entrées correspondantes, et récupère les données correspondantes de la table. Pour chaque index, le nombre de tuples est déterminé par la fraction de tuples dans une relation qui satisfait les clauses de restriction. Cette quantité fractionnaire est appelée « sélectivité » (f).

5.4.1.1.3 Accès par bitmap

Un accès par bitmap récupère tous les pointeurs de ligne dans l'index en une seule fois, les trie en mémoire en utilisant une structure bitmap, puis visite les lignes dans la table en suivant l'ordre de leur emplacement physique. Puisque l'accès par bitmap est basé sur l'accès par index, ses coûts d'E/S et de CPU sont calculés de la même façon.

5.4.1.1.4 Jointure par boucle imbriquée

La jointure par boucle imbriquée joint deux tables en récupérant le résultat d'une table (*inner*), et en recherchant chaque ligne de la première table dans la seconde (*outer*). Pour toutes les stratégies de jointure, l'estimation de la taille des deux relations *inner* et *outer* est obtenue par la multiplication de la taille originale de la relation par la sélectivité f de chaque clause de restriction dans la jointure. Le coût d'E/S dans ce type de jointure est la somme du coût de la relation *inner* et *outer*, le coût CPU est la multiplication de nombres de tuples dans chaque relation.

5.4.1.1.5 Jointure par tri-fusion

La jointure par tri-fusion combine deux listes triées comme une fermeture éclair. Les deux côtés de la jointure doivent être triés par les prédicats de jointure. Le coût d'E/S est la somme des coûts de la relation *inner* et *outer*. Le coût CPU est égal à la somme des coûts pour trier les tuples des deux relations.

5.4.1.1.6 Jointure par hachage

La jointure de hachage charge les tuples candidats d'un côté de la jointure dans une table de hachage dont chaque enregistrement est ensuite testé avec l'autre côté de la jointure. Le coût d'E/S est le coût de lecture des deux relations *outer* et *inner*, et le coût CPU est égal à la somme des coûts dans la phase de construction de la table de hachage n_hash plus la phase de sondage des tuples satisfaisants les clauses de jointures p_hash.

5.4.1.1.7 Tri

Cet opérateur trie l'ensemble de données (s) sur les colonnes mentionnées dans la requête. L'opération a besoin d'une quantité de mémoire tampon m pour matérialiser le résultat intermédiaire. Deux cas sont distingués, si l'opération de tri peut se faire dans la mémoire ($s < m$) alors PostgreSQL applique l'algorithme de tri rapide, où le coût d'E/S est nul. Sinon, le coût de sauvegarde et de la relecture de données s'ajoutent au coût d'E/S. Dans les deux cas, le coût CPU est $log(t)$ le nombre de tuples d'entrée.

5.4.1.1.8 Agrégation

Cette opération utilise une table de hachage temporaire pour grouper les tuples. Pour chaque groupe, les fonctions d'agrégat demandées sont appliquées. Une fois tous les enregistrements en entrée traitées, la table de hachage est renvoyée comme résultat. Le coût CPU est le nombre de tuples à grouper t.

5.4.1.1.9 Groupement

Dans cette opération, un ensemble pré-trié suivant la clause « group by » est agrégé. L'algorithme trie les données en entrée, en suivant la clé de groupement pour que les lignes de chaque groupe se suivent les unes les autres en une succession immédiate. Le coût CPU est le nombre de tuples à grouper t multiplié par le nombre de colonnes de groupement n_{group}.

Le Tableau 5.4 résumé les formules utilisées pour calculer les coûts d'E/S et de CPU de chaque opérateur basique avec les symboles figurants dans le Tableau 5.3.

5.4.2 Comparaison des plans de requêtes

L'optimiseur de requêtes PostgreSQL est basé sur l'énumération des plans de façon ascendante. À chaque niveau, les plans possibles sont évalués et les meilleurs sont conservés. Un plan A et meilleur qu'un plan B si le coût de A est inférieur à celui de B. En ajoutant le critère de l'énergie, nous devons ajuster les fonctions de comparaison des chemins pour refléter les compromis entre le coût de l'énergie et le temps de traitement. L'objectif de notre nouveau problème d'optimisation de requêtes multi-objectifs (PRMO) est de sélectionner des plans de requêtes qui minimisent à la fois le temps d'exécution et la consommation d'énergie. Le modèle de coût de performance traditionnel $Temps(Q)$ pour une requête Q

Tableau 5.3 – Les notations des paramètres du modèle de coût.

Paramètre	Définition
β_{cpu}	la puissance électrique d'effectuer une opération d'E/S
β_{io}	la puissance électrique d'effectuer une opération CPU
m	taille de la mémoire tampon pour l'opération de tri
$block$	taille d'une page dans un SGBD
T_i	la taille de la table i
t_i	nombre de tuples d'entrée pour l'opérateur i
p_i	nombre de pages d'entrée pour l'opérateur i
f	sélectivité d'index
s	la taille de la relation d'entrée pour l'opérateur de tri
n_{hash}	nombre de clauses de hachage en phase de construction
p_{hash}	nombre de partitions de hachage en phase de sondage
$p_{outer/inner}$	nombre de pages pour l'opérateur de jointure
$t_{outer/inner}$	nombre de tuples pour l'opérateur de jointure
n_{group}	nombre de colonnes de groupement

Tableau 5.4 – Les paramètres du modèle de coût pour les opérateurs SQL.

Paramètre	Coût d'E/S	Coût de CPU
Accès séquentiel	p_{seq}	t_{seq}
Accès par index	p_{index}	$t_{index} \cdot f$
Accès par bitmap	p_{bitmap}	$t_{bitmap} \cdot f$
Jointure par boucle imbriquée	$p_{outer} + p_{inner}$	$t_{outer} \cdot t_{inner}$
Jointure par tri-fusion	$p_{outer} + p_{inner}$	$t_{sort(outer)} + t_{sort(inner)}$
Jointure par hachage	$p_{outer} + p_{inner}$	$t_{outer} \cdot n_{hash} + t_{inner} \cdot p_{hash}$
Tri	$p_{sort}, s < m;$ $0, else$	$t_{sort} \cdot log_2(t_{sort})$
Agrégation	0	$t_{agg} \cdot$
Groupement	0	$t_{group} \cdot n_{group}$

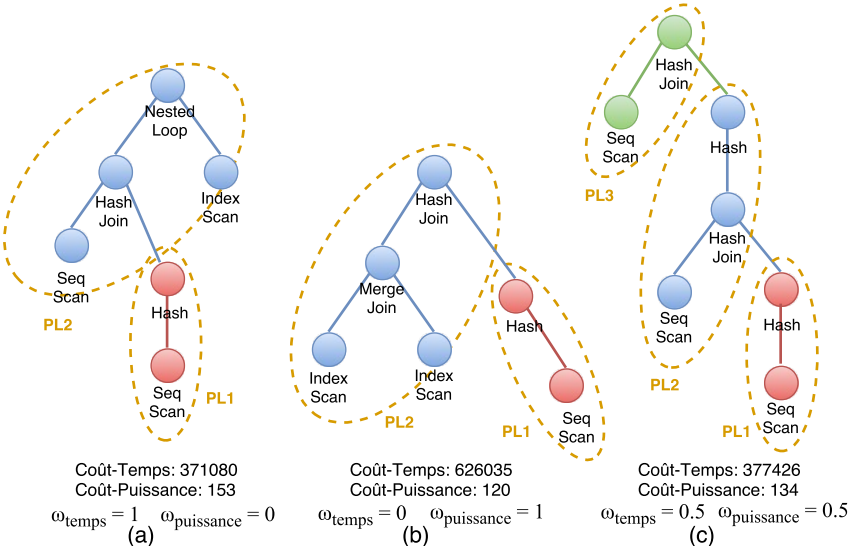


FIGURE 5.4 – Le plan optimal pour la requête *Q3* de TPC-H lors du changement des préférences d'utilisateur.

composé de n opérations algébriques $\{OP_1, OP_2, \ldots, OP_n\}$, est défini comme suit :

$$Temps(Q) = \alpha_{cpu} \times \sum_{i=1}^{n} COUT\_CPU_n + \alpha_{es} \times \sum_{i=1}^{m} COUT\_ES_i \tag{5.3}$$

Où $COUT\_ES$, $COUT\_CPU$ sont déjà décrits dans le Chapitre 4. Les paramètres α sont des coefficients spécifiés pour notre machine d'expérimentation. Ces derniers sont utilisés pour convertir les coûts de la requête en une quantité de temps. α_{cpu} est le temps *CPU* requis pour exécuter un *Cycle de CPU* et α_{es} est le temps d'*E/S* nécessaire pour que le périphérique exécute une opération d'*E/S*.

Afin de résoudre le PRMO, nous proposons d'utiliser la méthode de la somme pondérée des fonctions de coût pour donner à l'administrateur de base de données une solution avec le compromis souhaité. Dans cette méthode d'agrégation, nous calculons la somme pondérée des fonctions de coût (en utilisant l'Équation (5.3) et l'Équation (4.12) [page 108]) pour agréger les objectifs dans une seule fonction objectif équivalente pour être optimisée. Dans notre cas, cette méthode peut être définie comme suit :

$$\begin{aligned} \text{minimiser} \quad & y = \omega_1 \times Temps(Q) + \omega_2 \times Puissance(Q) \\ \text{avec} \quad & \omega_1 + \omega_2 = 1 \end{aligned} \tag{5.4}$$

Où ω_i sont les coefficients de pondération représentant respectivement l'importance relative des fonctions temps et puissance. y représente le coût du plan d'une requête Q. Nous avons implémenté ces deux coefficients comme un paramètre externe dans le SGBD, de sorte que l'administrateur ou les utilisateurs de base de données peuvent les modifier à la volée.

La Figure 5.4 montre le plan d'exécution optimal retourné par notre planificateur/optimiseur modifié pour la requête *Q3* issue du benchmark TPC-H, et la façon dont le plan change lorsque les préférences des utilisateurs varient. Dans un premier temps, nous avons utilisé une optimisation favorisant uniquement

la performance, le coût total du traitement est estimé à 371080 et la puissance totale est estimée à 153. Lorsque l'optimisation est modifiée pour favoriser seulement la puissance, le coût de traitement a augmenté à 626035, mais la puissance a diminué à 120. Dans la configuration de compromis, le coût de traitement est à 377426 et la puissance est à 134. Dans la Figure 5.4 (a), l'opérateur de boucle imbriquée attire la grande quantité de puissance dans la requête (33 watts), mais le plan est choisi par l'optimiseur, car il est très rapide en terme de temps d'exécution. Dans la Figure 5.4 (b) , nous nous rendons compte que l'opérateur de jointure par fusion est le plus lent dans la requête, son coût de traitement est à 539200, mais sa puissance est minime. Les deux opérateurs de jointure de hachage utilisés dans la Figure 5.4 (c) donnent un bon compromis, pour un 1,7% de dégradation des performances, nous obtenons 12,4% d'économie d'énergie.

5.5 L'implémentation dans un SGBD : EnerQuery

Dans cette section, nous détaillons l'implémentation de notre méthodologie appelée *EnerQuery*[19], dans le SGBD libre PostgreSQL.

5.5.1 Architecture du système

Dans cette section, nous décrivons l'architecture de *EnerQuery*. Il est composé de deux parties, une partie arrière-plan et une partie IHM (Interface Homme Machine). La partie arrière-plan inclut le SGBD avec le nouveau modèle de coût énergétique et la technique de comparaison des plans d'exécution. Elle est responsable de l'exécution de requêtes avec le compromis souhaité et de retourner les résultats aux utilisateurs finaux. Dans notre étude, nous nous sommes basés sur PostgreSQL, surtout parce qu'il est open source. Cependant, nos techniques peuvent être intégrées dans d'autres SGBD facilement. La partie interface graphique permet de manipuler *EnerQuery*, pour définir les paramètres et visualiser en temps réel leur impact sur la consommation d'énergie de la station de travail. L'architecture de notre système est décrite dans la Figure 5.5.

5.5.2 La partie arrière-plan

La partie arrière-plan concerne principalement les modifications dans le SGBD PostgreSQL. Ces modifications peuvent être classées en trois types : (1) des modifications dans l'optimiseur afin d'ajouter les fonctions de calcul des coûts énergétiques des opérations SQL ; (2) des modifications dans le planificateur pour ajouter le critère de l'énergie lors de l'évaluation des plans d'exécution ; et (3) diverses modifications tel que l'ajout des poids d'énergie et de puissance (ω_i) comme paramètres à ajuster par les administrateurs et les utilisateurs de base de données. Dans les sections suivantes, nous détaillons les principales modifications pour construire *EnerQuery*.

19. *EnerQuery* est téléchargeable à partir de : `http://www.lias-lab.fr/forge/projects/ecoprod`
Une vidéo de démonstration de ses fonctionnalités est disponible ici : `https://youtu.be/cWK5rf4MBNQ`

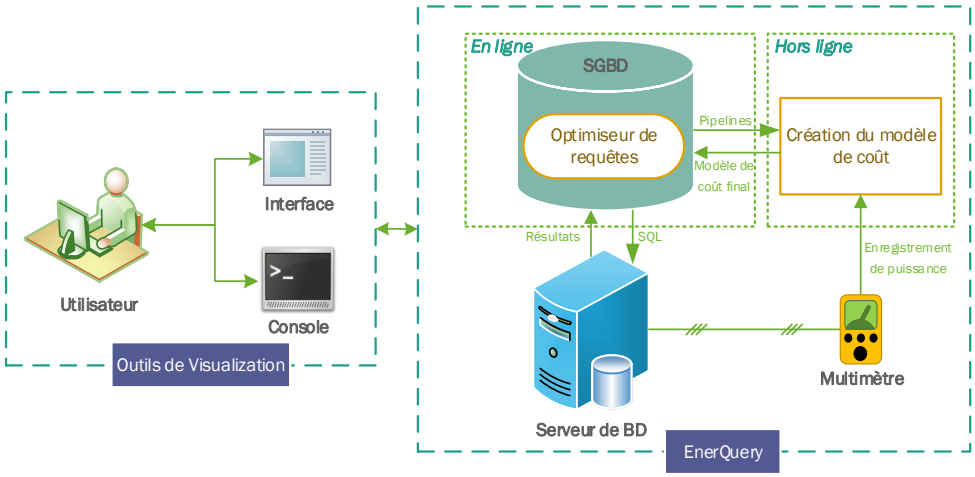


FIGURE 5.5 – Architecture du système *EnerQuery*.

5.5.2.1 Modifications dans l'optimiseur

Le fichier responsable du calcul des coûts des opérateurs algébriques est : `src/backend/optimizer/path/costsize.c`. Nous avons étendu chaque opérateur avec des instructions pour le calcul des opérations d'E/S et de CPU, et l'appel de la fonction `count_power_cost()` qui applique l'équation de régression Équation (4.12) (page 108) et retourne la puissance estimée. Un exemple d'un opérateur d'accès séquentiel modifié pour calculer la performance et la puissance est illustré dans la Figure 5.6. Un bout de code de la fonction `count_power_cost()` est représenté dans la Figure 5.7. Où les β_i sont les coefficients de régression.

5.5.2.2 Modifications dans le planificateur

Le fichier responsable de comparer et de choisir les meilleurs plans d'exécution est situé dans : `src/backend/optimizer/util/pathnode.c`. Nous avons étendu les trois fonctions : `compare_path_costs()`, `compare_fractional_path_costs()` et `compare_path_costs_fuzzily()` pour inclure la dimension d'énergie. Un exemple de code de la première fonction est représenté dans la Figure 5.8. Où `time_weight` et `power_weight` représentent les poids de chaque critère, et la méthode d'échelle logarithmique *log*() est utilisée pour rendre compte les deux unités de mesure.

5.5.2.3 Modifications diverses

Nous avons également modifié les fichiers de configuration de PostgreSQL pour ajouter des poids aux fonctions objectifs, afin de permettre aux utilisateurs de définir les valeurs de compromis entre l'énergie et la performance. Les fichiers concernés sont : `postgresql.conf.sample` et `guc.c`, situés dans : `src/backend/utils/misc/`. Les poids peuvent être affichés avec la commande de PostgreSQL `SHOW` et modifiés avec la commande `SET`, un exemple de modification est présenté dans la Figure 5.9.

Un outil puissant dans PostgreSQL pour visualiser un plan de requête est l'instruction `EXPLAIN`. Cette instruction produit une représentation textuelle du plan d'exécution optimal pour une requête

```
void
cost_seqscan(Path *path, PlannerInfo *root,
       RelOptInfo *baserel, ParamPathInfo *param_info)
{
  Cost    startup_cost = 0, run_cost = 0, io_cost = 0, cpu_cost = 0, power_cost = 0;
  double    spc_seq_page_cost;

  /* disk costs */
  run_cost += spc_seq_page_cost * baserel->pages;
  io_cost = baserel->pages;

  /* CPU costs */
  get_restriction_qual_cost(root, baserel, param_info, &qpqual_cost);

  startup_cost += qpqual_cost.startup;
  cpu_per_tuple = cpu_tuple_cost + qpqual_cost.per_tuple;
  cpu_cost = cpu_per_tuple * baserel->tuples;
  power_cost = count_power_cost(io_cost, cpu_cost);
}
```

FIGURE 5.6 – La fonction responsable de calcul des coût d'un opérateur de accès séquentiel.

```
Cost
count_power_cost(double io_cost, double cpu_cost)
{
  total_power_cost = B0
  + pow(io_cost, 1) * pow(cpu_cost, 0) * B1 + pow(io_cost, 2) * pow(cpu_cost, 0) * B2
  + pow(io_cost, 3) * pow(cpu_cost, 0) * B3 + pow(io_cost, 4) * pow(cpu_cost, 0) * B4
  + pow(io_cost, 0) * pow(cpu_cost, 1) * B5 + pow(io_cost, 1) * pow(cpu_cost, 1) * B6
  + pow(io_cost, 2) * pow(cpu_cost, 1) * B7 + pow(io_cost, 3) * pow(cpu_cost, 1) * B8
  + pow(io_cost, 0) * pow(cpu_cost, 2) * B9 + pow(io_cost, 1) * pow(cpu_cost, 2) * B10
  + pow(io_cost, 2) * pow(cpu_cost, 2) * B11 + pow(io_cost, 0) * pow(cpu_cost, 3) * B12
  + pow(io_cost, 1) * pow(cpu_cost, 3) * B13 + pow(io_cost, 0) * pow(cpu_cost, 4) * B14;

  return total_power_cost;
}
```

FIGURE 5.7 – La fonction responsable de calcul d'énergie d'un opérateur SQL.

```
int
compare_path_costs(Path *path1, Path *path2, CostSelector criterion)
{
  Cost total_cost1 = time_weight * log(path1->total_cost) + power_weight * log(path1->power_cost)
      ;
  Cost total_cost2 = time_weight * log(path2->total_cost) + power_weight * log(path2->power_cost)
      ;

  if (total_cost1 < total_cost2)
      return -1;
  if (total_cost1 > total_cost2)
      return +1;
  return 0;
}
```

FIGURE 5.8 – La fonction de comparaison des coûts de *EnerQuery*. Retourner -1, 0, ou +1 en fonction des coûts des chemins d'accès, si le chemin d'accès 1 est moins cher que le chemin d'accès 2, la valeur retournée est -1, s'ils ont le même coût, la fonction retourne 0, sinon la valeur est +1.

```
set power_weight 0.4
set time_weight 0.6
```

FIGURE 5.9 – Attribution des valeurs de poids d'énergie et de performance pour le planificateur des plans.

donnée en affichant les opérateurs de l'arbre ainsi que des informations supplémentaires telles que le coût initial et total de la performance (première et deuxième valeur de *cost*), la cardinalité (*rows*) et la taille des lignes en octets (*width*) pour chaque opérateur. Nous avons ajouté le coût de puissance électrique comme troisième valeur de *cost*, au niveau de chaque opérateur (la fonction est dans le fichier `src/backend/commands/explain.c`). La représentation textuelle du plan produit pour la requête définie précédemment est illustrée dans la Figure 5.10.

```
                                    QUERY PLAN
-----------------------------------------------------------------------------------------------
 Aggregate  (cost=6712707.37..6712707.38..126.59 rows=1 width=33)
   ->  Nested Loop  (cost=0.00..6712694.32..115.00 rows=78460 width=33)
         Join Filter: (lineitem.l_partkey = part.p_partkey)
         ->  Seq Scan on lineitem  (cost=0.00..2010.32..115.55 rows=78460 width=16)
               Filter: ((l_shipdate >= '1995-09-01'::date) AND (l_shipdate < '1995-10-01 00:00:00
                   '::timestamp without time zone))
         ->  Seq Scan on part  (cost=0.00..60.00..125.84 rows=200000 width=25)
(6 rows)
```

FIGURE 5.10 – Le plan d'exécution de la requête TPC-H *Q14* retourné par la commande `EXPLAIN` de PostgreSQL.

5.5.3 La partie interface

L'interface graphique permet à l'utilisateur de manipuler les paramètres de *EnerQuery* et de voir leurs impacts sur le système. Les utilisateurs peuvent analyser les coûts de chaque opérateur algébrique en utilisant l'interface graphique, ils peuvent se rendre compte que les jointures ont un impact majeur sur la structure du plan d'exécution de requête. Cette interface est implémentée en utilisant le langage de programmation C++ et la bibliothèque Qt [20], qui offre une meilleure intégration avec le SGBD. Les modules principaux de l'interface graphique sont illustrés dans la Figure 5.11 et listés ci-dessous.

5.5.3.1 Configuration de surveillance

Ce module est responsable de l'établissement de la connexion avec le serveur SGBD. Les utilisateurs peuvent également spécifier le chemin pour le wattmètre afin de capturer la consommation d'énergie en temps réel. La partie la plus importante ici concerne les paramètres de puissance/performance, qui permet de configurer les objectifs d'optimisation : la performance, la puissance ou le compromis. Les utilisateurs peuvent également modifier les paramètres du planificateur de requête en forçant l'optimiseur d'évaluer d'autres plans (voir capture d'écran ① de la Figure 5.11), comme la technique des *hints* utilisée par le SGBD Oracle. Ces paramètres couvrent les modes d'optimisation suivants : les types d'accès (*séquentiel*, *index*, *index-only*, *bitmap*, et *TID*), les types de jointure (*hash*, *tri-fusion*, et *boucle imbriquée*), et enfin le *tri* et *l'agrégation*. *EnerQuery* offre aux utilisateurs finaux la possibilité de quantifier la puissance consommée d'une requête donnée en faisant varier les modes d'optimisation.

5.5.3.2 Charge de requêtes SQL

Dans ce module, les utilisateurs peuvent entrer soit une seule requête SQL ou une charge de requêtes à exécuter. Les requêtes prises en charge varient de simples opérations transactionnelles aux opérations de traitement analytiques plus complexes impliquant plusieurs tables de grandes dimensions. La charge est composée de requêtes générées à partir des outils des benchmarks et peut être exécuté en concurrence à un niveau de multi-programmation prédéfini. L'exécution se fait dans un processus (thread) séparé pour chaque requête et les résultats sont affichés dans un widget de table. Un exemple de requête que les utilisateurs peuvent introduire est *Q7* du TPC-H. I s'agit d'une requête imbriquée de deux niveaux et qui joint 7 tables ; elle contient également des opérations complexes de regroupement et de tri.

5.5.3.3 Chronologie de la consommation

Lorsque l'utilisateur exécute une requête, la consommation d'énergie en temps réel est affichée dynamiquement via le wattmètre. Quand l'exécution de la requête est terminée, l'énergie totale consommer sera calculée et affichée à l'utilisateur. Cela peut donner aux utilisateurs une observation réelle de l'énergie enregistrée, en utilisant les paramètres de compromis souhaités. Par exemple, dans la Figure 5.4 (c), pour une dégradation des performances de 1,7%, on obtient un gain de 12,4% d'économie d'énergie. En outre, les utilisateurs peuvent comparer les valeurs estimées et réelles pour vérifier la précision des modèles de coût ou affiner leurs paramètres.

20. https://www.qt.io/

5.5.3.4 Plan d'exécution

Lorsque l'utilisateur soumet une requête, l'optimiseur de requête sélectionne le meilleur plan d'exécution par rapport au compromis prédéfini. Une option d'affichage de ce plan d'exécution est disponible. Cette dernière permet de donner diverses informations, telles que le coût estimé, la consommation d'énergie, et les coûts d'E/S et de CPU pour chaque opérateur physique, grâce à l'événement défini au survole de la souris. Les utilisateurs peuvent identifier l'opérateur qui consomme plus d'énergie, par exemple, les opérateurs à forte intensité de CPU telles que le tri et l'agrégation sont gourmandes en puissance dans les serveurs traditionnels. De plus, les utilisateurs peuvent également voir dans quel cas les opérateurs intensifs en E/S conduisent à une forte consommation d'énergie. En outre, la notation du pipeline est visualisée comme le montre la Figure 5.4, les arbres de pipelines sont regroupés avec la *même couleur*. Cette partie d'interface montre comment les paramètres de compromis influencent les plans générés. Cela aide à l'interprétation des optimisations d'exécution et les différents pipelines.

5.6 Évaluation et résultats

Pour évaluer l'efficacité de nos propositions, nous menons plusieurs expérimentations. Dans la section suivante, nous présentons notre architecture expérimentale, y compris, la machine exploitée pour calculer l'énergie et l'ensemble de données utilisés.

5.6.1 Architecture d'expérimentations

Lors des expérimentations, on s'est basé sur la même architecture employé dans le chapitre précédant. Un wattmètre « Watts UP ? Pro ES » est utilisé pour mesurer la puissance électrique du système. Le dispositif est placé directement entre la prise d'alimentation et la station de travail dans laquelle la base de données à tester est stockée. Les valeurs de puissance sont enregistrées pour être traitées dans une machine de contrôle séparée.

Nous avons utilisé une station de travail *haut de gamme* Dell PowerEdge R310 ayant un processeur Xeon X3430 de 2,40 GHz et 32 Go de mémoire DDR3. Pour valider notre modèle par rapport à une nouvelle configuration matérielle *bas de gamme*, nous avons créé une autre configuration avec une station de travail Dell Precision T1500 équipée d'un processeur Intel Core i5 de 2.27GHz et 4 Go de mémoire. Les machines de travail sont installées avec *EnerQuery*, notre version modifiée de SGBD PostgreSQL 9.4.5 sous Ubuntu 14.04 LTS avec le noyau 3.13.

L'ensemble de données et des requêtes de TPC-H est utilisé dans nos expérimentations, avec un facteur d'échelle de 10 Go et 100 Go. Dans nos expériences, nous considérons trois types de configuration de *EnerQuery* : (1) Puissance-PG, qui est la configuration qui donne le coût d'énergie minimale, (2) Temps-PG est la configuration avec un coût de temps minimal, (3) Compromis-PG, en utilisant la méthode de la somme pondérée avec $\omega_1 = 0,5$ et $\omega_2 = 0,5$.

5.6.2 Résultats

Dans cette section, l'ensemble des résultats est commenté.

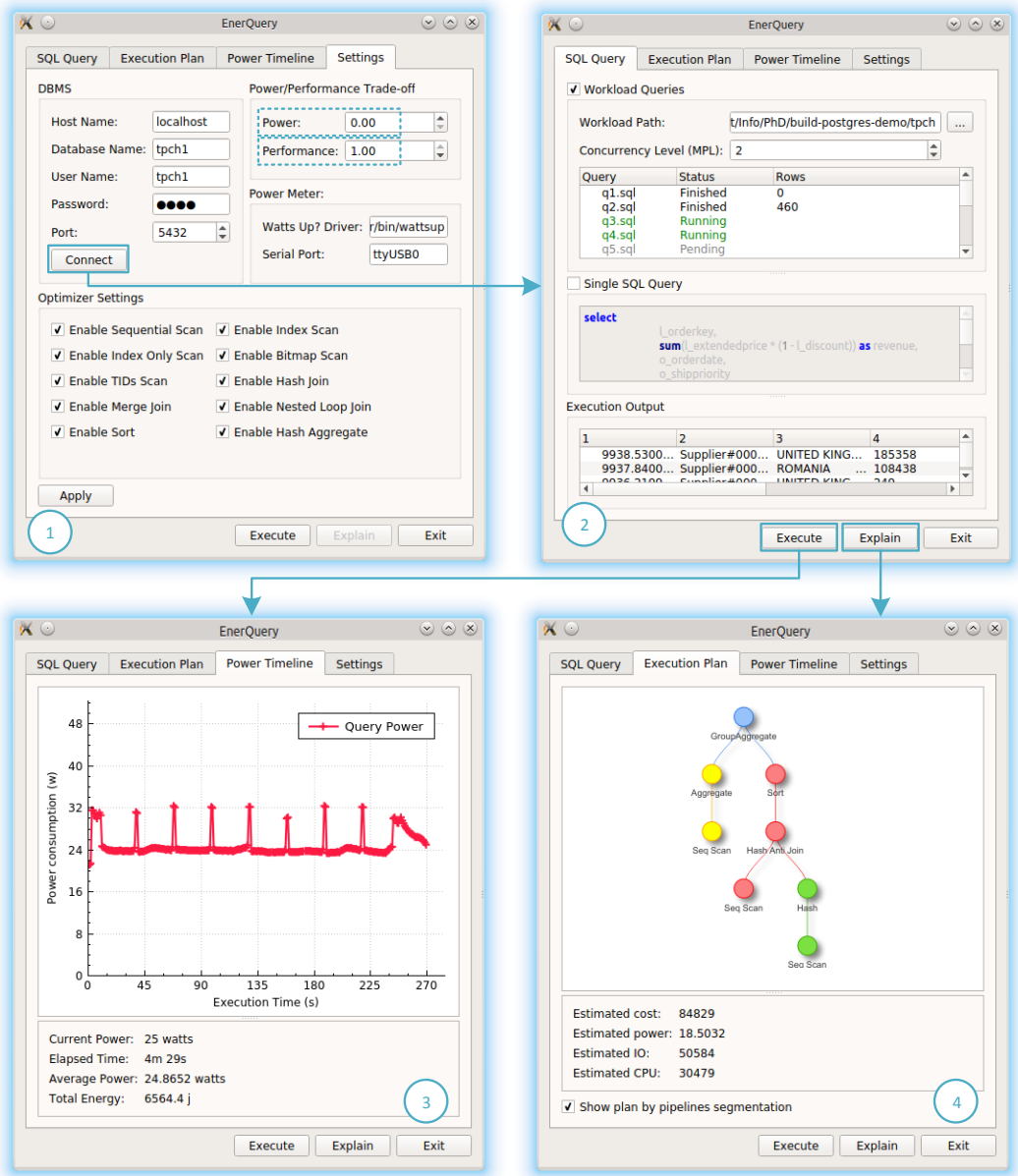
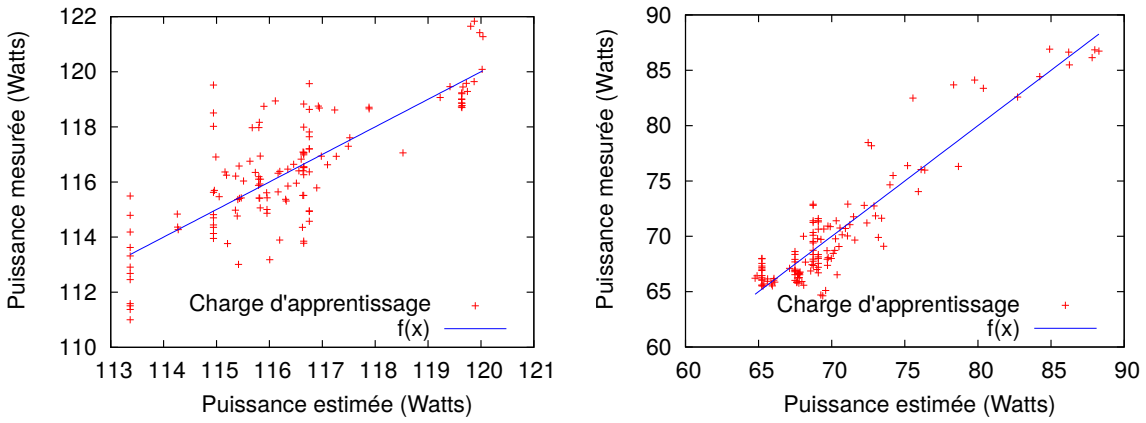


FIGURE 5.11 – L'interface principale de *EnerQuery* et ses modules.



(a) Modèle de régression en utilisant la configuration haut de gamme

(b) Modèle de régression en utilisant la configuration bas de gamme

FIGURE 5.12 – La qualité du modèle de coût énergétique avec différent type de machine.

5.6.2.1 Qualité du modèle de coût

Pour vérifier la portabilité de notre modèle de coût proposé contre les changements dans l'environnement matériel, nous menons cette expérimentation. Les résultats de la phase d'apprentissage de notre modèle de coût pour les deux configurations sont représentés sur la Figure 5.12. Comme nous pouvons le constater, la consommation d'énergie estimée et réelle, se rapprochent des lignes diagonales dans les deux configurations. Par ailleurs, dans la configuration du serveur, nous pouvons voir une variance entre la puissance estimée et observée pour certaines requêtes d'apprentissage. Une grande partie de ce résultat est due aux erreurs commises par l'optimiseur de requête du SGBD, dans l'estimation des coûts d'E/S et de CPU pour ces requêtes quand la taille du mémoire est grande. Ce phénomène classique a été rencontré pour une longue période par les optimiseurs de requêtes, et tous les modèles de performance proposés dans la littérature souffrent de ce problème, qui est lié à des erreurs d'estimation de cardinalités. En fait, les erreurs d'estimation dans les pipelines de bas niveau sont propagées aux niveaux supérieurs, et peuvent dégrader de manière significative la précision de la prédiction.

5.6.2.2 Erreurs d'estimations du modèle de coût

Dans ce type d'expérimentations, étant donné le coût de puissance estimé par notre modèle *(E)*, on le compare avec la consommation de puissance active par le système *(M)* réellement observée. Pour quantifier la précision du modèle, nous avons utilisé la métrique de taux d'erreur suivante : $Erreur = \frac{|M-E|}{M}$. Pour tester notre modèle avec un grand ensemble de données, nous exécutons les 22 requêtes du benchmark TPC-H contre deux bases de données, avec un facteur d'échelle de 10 Go et de 100 Go. La plupart des requêtes contiennent plus de 4 pipelines. Les résultats sont présentés dans le Tableau 5.5. Notez que certaines requêtes ont été arrêtés car ils ont dépassé les 72 heures d'exécution dans notre environnement de test en cours.

Comme nous pouvons le voir dans le tableau, l'erreur moyenne est généralement petite (0,1% dans les deux ensembles de données 100 Go et 10 Go), et l'erreur maximale est habituellement inférieure à 5%.

Tableau 5.5 – Erreurs d'estimation d'énergie dans les requêtes du benchmark TPC-H avec différentes tailles de base de données.

Requête	Erreur (%)		Requête	Erreur (%)	
	10 Go	100 Go		10 Go	100 Go
$Q1$	1,03	0,2	$Q11$	4,2	-
$Q2$	-	-	$Q12$	0,9	0,02
$Q3$	1,5	1,2	$Q13$	4,7	4,4
$Q4$	0,6	0,5	$Q14$	2,8	2,4
$Q5$	1,2	3,07	$Q15$	0,4	2,7
$Q6$	4,1	2,7	$Q16$	5,4	0,03
$Q7$	0,4	1,4	$Q18$	0,4	-
$Q8$	0,07	1,09	$Q19$	1,6	0,9
$Q10$	0,6	0,3	$Q22$	1,2	0,4

L'expérimentation montre la précision de notre modèle de prédiction, indiquant qui est suffisamment précise pour l'employer dans un SGBD réel.

5.6.2.3 Caractéristiques de requêtes

Pour étudier les caractéristiques des 22 requêtes de TPC-H, nous procédons à une série de tests en utilisant *EnerQuery*. Dans ces tests et pour chaque configuration (Temps-PG, Puissance-PG, Compromis-PG), nous exécutons tous les requêtes de TPC-H et nous collectons les estimations des coûts de performance et de puissance retournées par l'optimiseur de requêtes. Dans la Figure 5.13 (les valeurs sont reportées sur une échelle logarithmique), nous pouvons constater que 16 des 22 requêtes ont le potentiel d'économie d'énergie dans la configuration Puissance-PG. Théoriquement, ce gain en économie d'énergie pour ces requêtes à un impact négatif sur le temps de traitement, comme indiqué dans la même figure. Cependant, le choix de la configuration compromis peut conduire à de bonnes valeurs d'économie d'énergie, avec moins de dégradation en terme de performances. Ces requêtes sont caractérisées par un nombre important d'opérateurs SQL et diverses opérations d'E/S et de CPU, ce qui donne à l'optimiseur de requêtes une variété de plans à choisir. Par conséquent, nous pouvons obtenir des requêtes avec une bonne économie d'énergie à partir de ces plans. D'autre part, le reste des requêtes qui ne peuvent pas bénéficier d'une économie d'énergie, sont des requêtes simples avec quelques tables et opérateurs SQL. Cela conduit l'optimiseur de requêtes à choisir le même plan dans toutes les configurations *EnerQuery*, en raison du faible espace de recherche des plans.

5.6.2.4 Économie d'énergie et de puissance électrique

Le but de cette série d'expérimentations est d'étudier les avantages de notre approche en termes d'efficacité énergétique. Nous avons configuré le SGBD pour évaluer les coûts de performance et de consommation d'énergie pour les trois configurations (Temps-PG, Puissance-PG, Compromis-PG). Nous répétons les mêmes expérimentations avec deux tailles de bases de données différentes : 10 Go et 100 Go en utilisant le benchmark TPC-H.

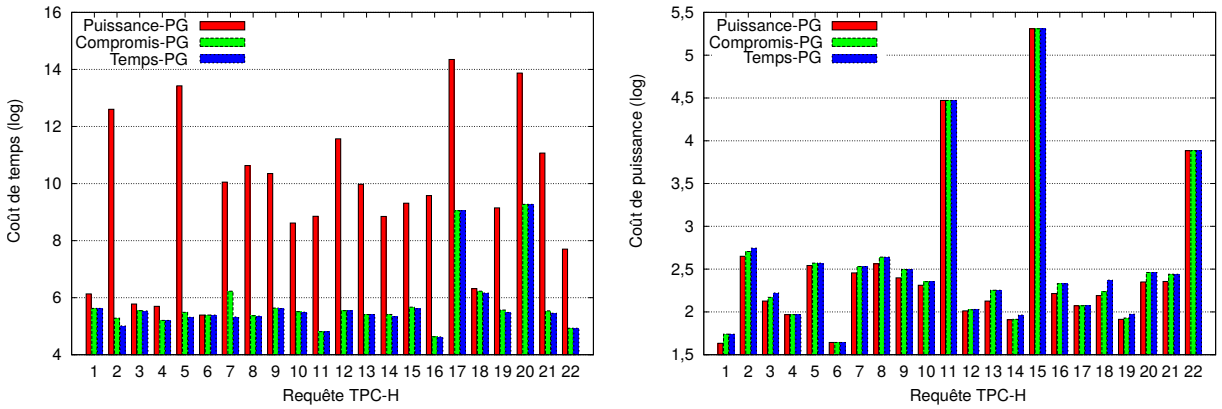


FIGURE 5.13 – Performance et puissance pour les requêtes de TPC-H en utilisant différentes configurations de *EnerQuery*.

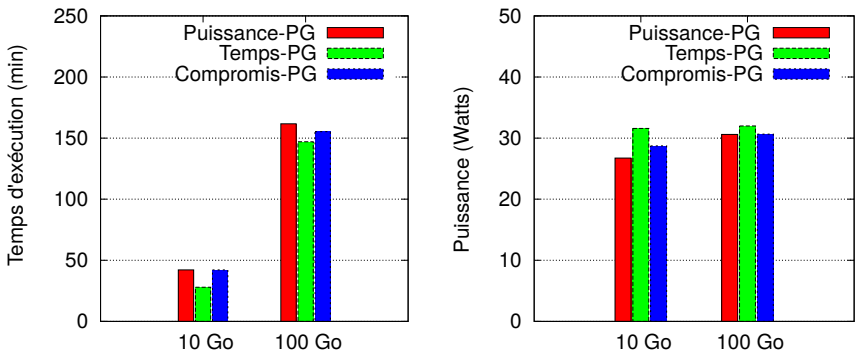


FIGURE 5.14 – Performance et économie d'énergie avec différentes configurations de *EnerQuery* en utilisant TPC-H.

Dans la Figure 5.14 nous présentons les résultats des expérimentations. On peut constater rapidement que la consommation est nettement plus faible quand on choisit une configuration d'optimiseur de requêtes favorisant les plans de faible puissance. Lorsque on compare les résultats d'une optimisation de puissance (Puissance-PG) avec une optimisation de performance (Temps-PG), on observe une grande marge en économie d'énergie, le bénéfice est remarquablement considérable dans la base de données de petite taille, peut-être cela est dû à la grande quantité des opérations d'E/S et le traitement des données requises par les requêtes exécutées dans cette base par rapport au bases de données de grande taille, qui se traduisent par plus de consommation d'énergie quel que soit le plan d'exécution choisi par l'optimiseur de requêtes. Comme prévu, les économies de la configuration Compromis-PG sont plus petites que celles obtenues par la configuration favorisant la puissance, mais elles sont encore acceptables, en particulier dans des ensembles de données de 100 Go, elles sont approximatives. D'autre part, la configuration de puissance prend plus de temps pour terminer l'exécution de toutes les requêtes, qui se traduit par une dégradation significative des performances. Ceci n'est pas surprenant, car si on gagne en puissance, on perd automatiquement en performances. Dans la configuration Compromis-PG, la dégradation des performances est en fait acceptable si on considère le gain de puissance atteint.

Notez que tous les résultats de nos expérimentations considèrent uniquement des économies d'énergie directes, dans un seul serveur de base de données. Ce nombre pourrait être encore plus élevé si on considère des centres de données à grande échelle avec des milliers de serveurs et de systèmes de refroidissement.

5.7 Conclusion

Dans ce chapitre, nous avons proposé un module de traitement de requête éco-énergétique *EnerQuery* implémenté sur un SGBD réel qui est PostgreSQL. Avant de l'implémenter, une vérification a été effectuée pour identifier les composants du module de traitement de requêtes sensibles à l'énergie. En fonction de cette vérification, une méthodologie de construction d'un tel module a été donnée. Notre méthodologie est prise en charge par un outil open source disponible à la forge de notre laboratoire, pour permettre aux chercheurs, industriels et étudiants d'en tirer des profits. Des expérimentations intensives ont été menées pour démontrer l'efficacité et l'utilité de notre proposition. Les résultats obtenus sont très encourageants puisqu'ils démontrent des gains important en termes d'économie d'énergie et leurs influences sur les plans d'exécution étudiés.

Chapitre

6 Intégration de l'énergie dans la conception physique : Cas des vues matérialisées

« Knowledge is power. »
— Francis Bacon

Sommaire

6.1 Introduction

Dans l'ère des Big data, la gestion de la consommation d'énergie par les serveurs et centres de données est devenue un enjeu majeur pour les entreprises, les institutions et les pays. Dans les applications centrées sur les données, les systèmes de gestion de base de données sont l'un des principaux consommateurs d'énergie lors de l'exécution des requêtes complexes impliquant de très grandes masse de données. Plusieurs initiatives ont été proposées pour traiter cette problématique, couvrant à la fois les dimensions matérielles et logicielles. Dans ce chapitre, nous nous concentrons sur la conception physique des bases de données, où des structures d'optimisation (par exemple, des index, des vues matérialisées) sont sélectionnées pour satisfaire un ensemble donné de besoins non fonctionnels tels que la performance pour une charge de requêtes. Dans ce chapitre, nous proposons d'abord une initiative, qui intègre la dimension énergétique dans la conception physique lors de la sélection des vues matérialisées, l'une des structures d'optimisation redondantes. Deuxièmement, nous fournissons une formalisation multi-objective du problème de sélection de vue matérialisée, en tenant compte de deux besoins non fonctionnels : la performance et la consommation d'énergie lors de l'exécution d'une charge de requêtes donnée. En troisième lieu, un algorithme évolutionnaire a été développé pour résoudre le problème. Cet algorithme diffère de ceux qui existent déjà en étant interactif, afin que les administrateurs de base de données peuvent ajuster certains paramètres sensibles de l'énergie à l'étape finale d'exécution de l'algorithme en fonction de leurs spécifications. Enfin, des expérimentations intensives sont menées en utilisant notre modèle de coût mathématique et un dispositif réel pour les mesures d'énergie. Les résultats obtenus sont très satisfaisants, ce qui rend l'approche qu'on a développé un moyen très efficace pour économiser l'énergie tout en optimisant les requêtes par le biais de vues matérialisées.

La suite de ce chapitre est organisée comme suit. Dans la première section, nous discutons notre initiative de conception éco-énergétique qui concerne la phase de conception physique du cycle de vie des bases de données. Cette section donne une formulation plus détaillée du problème de sélection de vue sur la base de graphe globale de requêtes, ainsi qu'un exemple de support pour illustrer les relations entre la performance et le comportement de la puissance électrique. Cette section présente également les modèles de coûts utilisés dans notre étude pour estimer les performances et la puissance des requêtes SQL. La section suivante décrit notre méthodologie qui se base sur des algorithmes évolutionnaires pour résoudre notre problème d'optimisation multi-objectifs. L'algorithme et ses paramètres sont expliqués en détail, y compris les étapes de sélection, croisement, mutation et la méthode de pondération. La section suivante présente et discute les résultats expérimentaux basés sur plusieurs scénarios de matérialisation de vues avec différents paramètres et tailles de données. Enfin, la dernière section conclut le chapitre en résumant les principaux résultats.

6.2 Pourquoi la conception physique ?

Notre initiative concerne la phase de conception physique. Récemment, plusieurs chercheurs bien connus du milieu universitaire (Stavros Harizopoulos *et al.* [116], MIT) et de l'industrie (Goetz Graefe [104], HP) ont recommandé de revoir la conception physique en impliquant la dimension d'énergie. Il convient de noter que la configuration physique de la base de données sur le support de stockage est spécifiée dans la phase de conception physique. Plusieurs techniques (structures) d'optimisation

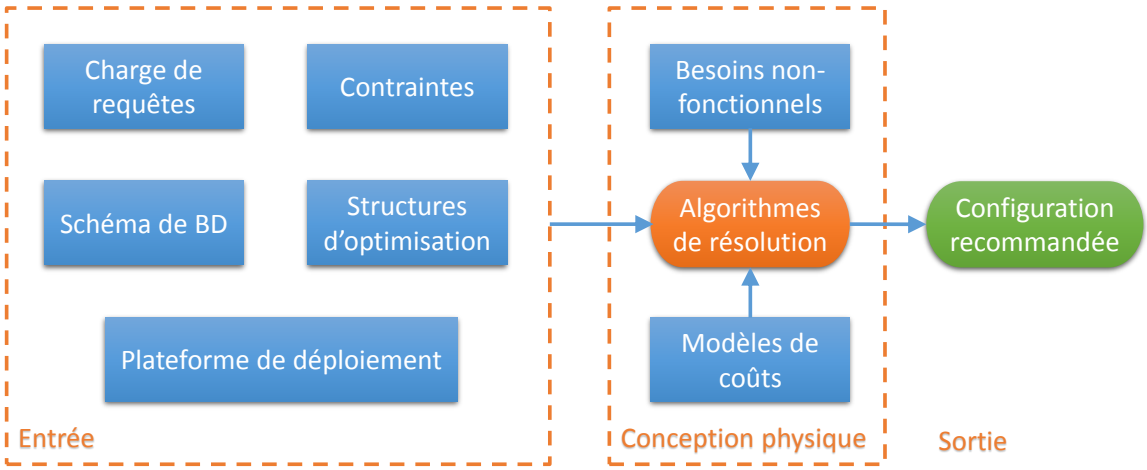


FIGURE 6.1 – Schéma de la conception physique des bases de données.

telles que les vues matérialisées, les indexes et le partitionnement sont sélectionnées à cette phase pour améliorer les performances d'exécution des requêtes [87]. L'importance de la conception physique a été amplifiée avec l'avance des optimiseurs de requêtes qui sont devenus assez sophistiqués pour faire face aux demandes d'aide à la décision complexes [59]. Habituellement, la sélection d'une structure physique est effectuée par le biais des algorithmes utilisant des modèles d'estimation de coûts d'un ou de plusieurs fonctions objectives [174]. Notez que ces modèles de coûts utilisent des paramètres liés à des schémas de base de données, le matériel, la requête, la charge de requêtes et les contraintes (coûts de stockage, les coûts de maintenance, etc.). La Figure 6.1 illustre le schéma de la conception physique des bases de données. La sélection de ces structures d'optimisation impacte fortement la base de données finale, car elles deviennent une partie de celui-ci. En conséquence, elles peuvent contribuer de manière positive ou négative à l'augmentation de la consommation d'énergie. Par conséquent, l'exploration de la conception physique éco-énergétiques est cruciale pour éviter le gaspillage d'énergie lors de la sélection et le maintien de ces structures d'optimisation. Pour surveiller cette énergie, nous recommandons son intégration dans le processus de conception physique.

Dans cette étude, nous proposons une initiative qui intègre l'énergie dans le processus de conception physique en considérant la technique des *vues matérialisées* qui est une structure d'optimisation redondante dont les résultats des requêtes sont stockés et maintenus afin d'optimiser globalement la charge de requêtes [110].

Notre initiative visant à réduire la dissipation d'énergie lors de la sélection des structures d'optimisations s'appuie sur la considération de l'énergie comme l'un des objectifs à optimiser. Par conséquent, le processus de sélection de vue matérialisée devient un problème d'optimisation multi-objectifs qui peut impliquer plusieurs objectifs potentiellement contradictoires (par exemple, l'amélioration de performance et d'économie d'énergie). Ce problème dispose d'un ensemble de Pareto [299] qui représente l'ensemble des solutions optimales dans les deux objectifs. Les algorithmes évolutionnaires multi-objectifs (AEMOs) sont une approche appropriée pour s'approcher de la *frontière d'efficacité de Pareto* en une seule exécution [299]. Les AEMOs ont montré leurs efficacités pour résoudre plusieurs problèmes (par exemple, problème du voyageur de commerce). Comme il est impossible d'avoir une solution unique qui

optimise simultanément les deux objectifs, l'utilisation d'un algorithme qui offre aux administrateurs de bases de données (DBA) de nombreuses solutions alternatives situées sur ou près de la frontière d'optimalité de Pareto est d'une grande valeur pratique [82]. A cet effet, nous utilisons l'algorithme élitiste : *Non-Dominated Sorting Genetic Algorithm II* (NSGA-II) [82]. De plus, nous offrons aux DBAs la possibilité de favoriser une fonction objective sur une autre. Pour satisfaire cette exigence, nous avons appliqué la méthode d'agrégation des sommes pondérées qui attribue un poids à chaque fonction objective afin d'avoir une seule solution à déployer par le DBA.

6.3 Démarche d'intégration de l'énergie dans le problème de sélection des vues

Dans cette section, nous présentons en détail notre initiative qui concerne la phase de conception physique du cycle de vie des bases de données. L'initiative vise à proposer des structures d'optimisation qui répondent aux besoins et aux exigences des utilisateurs en termes de performances des requêtes et de l'énergie consommée. La phase de conception physique est comme un entonnoir pour les autres phases du cycle de vie car elle intègre plusieurs paramètres à partir de ces phases. En raison de la diversité des structures d'optimisation, nous considérons le cas des vues matérialisées. Dans la section suivante, nous discutons les différentes possibilités d'intégrer l'énergie dans le processus de sélection de vue.

Rappelons que le $\mathcal{PSV}$ consiste à trouver un ensemble approprié de vues pour la matérialisation tout en satisfaisant un ensemble donné de $\mathcal{BNF}$ (comme la performance des requêtes, la fiabilité, l'utilisabilité, etc). Les vues matérialisées sélectionnées doivent répondre à un ensemble de contraintes telles que le coût de stockage, les coûts de maintenance, etc.

Si on veut intégrer l'énergie dans la sélection des vues matérialisées, on doit l'attribuer à l'une des entrées des formalisations suivantes :

1. **La base de données comme référentiel pour stocker et gérer l'énergie** : La consommation d'énergie peut être stockée dans une base de données. Projet MIRABEL [239] est un exemple de ce scénario.

2. **L'énergie comme une contrainte** : Ce scénario est possible ; par exemple, un DBA veut sélectionner un ensemble de vues matérialisées sous une contrainte d'énergie bien connue. En règle générale, ce n'est facile de l'estimer a priori.

3. **Énergie en tant que $\mathcal{BNF}$** : Si le DBA est plus sensible à l'énergie, il peut la considérer comme le principal objectif de sa sélection, comme il peut la combiner avec un autre objectif, tel que le coût de traitement des requêtes.

Dans notre étude d'intégration d'énergie, on considère les scénarios 2 et 3. Pour le reste de notre étude, nous considérons principalement le scénario 3 et nous donnons quelques résultats préliminaires en considérant le scénario 2 dans la partie expérimentale (Section 6.6.2.8 [page 174]).

6.4 $\mathcal{PSV}$ avec le scénario d'énergie comme un $\mathcal{BNF}$

Dans cette section, nous présentons tous les ingrédients nécessaires pour mener à bien le processus de sélection des vues matérialisées. Ce processus se résume en trois étapes : (i) la formalisation du $\mathcal{PSV}$, (ii) la définition des modèles de coût d'estimation de chaque fonctions objectives et (iii) la méthodologie de sélection.

6.4.1 Formalisation du $\mathcal{PSV}$

Dans notre étude, le $\mathcal{PSV}$ est formalisé comme suit : étant donné (i) un schéma de base de données DB ; (ii) une charge de requêtes W ; (iii) une contrainte stockage S ; (iv) une contrainte de temps de maintenance T ; (v) deux $\mathcal{BNF}$ représentant la consommation d'énergie et les coûts de traitement des requêtes. Le $\mathcal{PSV}$ consiste à sélectionner un ensemble de vues matérialisées $MV = \{V_1, V_2, \cdots, V_m\}$, tout en réduisant le coût de traitement des requêtes, et en économisant la consommation d'énergie, de telle sorte qu'on aurait une taille des vues sélectionnées inférieure à S et une durée totale de maintenance inférieure à T. Les contraintes indiquent que le $\mathcal{PSV}$ est étudié suivant des ressources limitées, par exemple, l'espace de stockage et le temps de maintenance.

Comme mentionné précédemment, le $\mathcal{PSV}$ est prouvé être NP-complet, en raison de la grande taille d'espace de recherche des solutions possibles. Dans notre cas, le $\mathcal{PSV}$ devient encore plus complexe, puisque nous considérons deux fonctions objectives (minimisation de l'énergie et du temps). Par conséquent, trouver de bonnes solutions dans un délai acceptable est un défi supplémentaire à ce problème. Ceci nous motive, dans cette première étude, de traiter les requêtes de type OLAP en lecture seule. Ces derniers consomment une quantité d'énergie majeure dans les entrepôts de données, car elles sont caractérisées par une longue durée d'exécution et une utilisation intense de ressources [152].

6.4.2 Modèle de coût

Notre objectif est de sélectionner des vues matérialisées qui assurent un bon compromis entre deux coûts : les performances des requêtes et la consommation d'énergie. En conséquence, nous avons besoin de modèles mathématiques pour estimer ces coûts. Dans cette section, nous présentons les étapes nécessaires pour créer les modèles de coûts. Dans le Chapitre 4, nous avons proposé une approche logicielle basée sur une étude empirique pour trouver les paramètres clés ayant un impact sur la performance et l'énergie lors de l'exécution d'une requête, et pour déterminer le niveau approprié afin de construire notre modèle. L'étude empirique sélectionne également l'algorithme mathématique approprié qui décrit le comportement des requêtes SQL en fonction du $\mathcal{BNF}$. Ces choix sont cruciaux parce qu'ils affectent la précision du modèle final. Dans ce chapitre, nous nous sommes basés sur le modèle proposé dans le Chapitre 4 pour la sélection des vues matérialisées.

Les paramètres d'entrée qui peuvent être pris en compte dans la construction de modèles de coût sont énumérés dans le Tableau 6.1. Dans la section qui suit, nous allons détailler le développement de nos modèles de coûts.

Tableau 6.1 – Les notations des paramètres du modèle de coût.

Paramètre	Définition
α_{cpu}	le temps d'effectuer une opération d'E/S
α_{es}	le temps d'effectuer une opération CPU
α_{net}	le temps d'effectuer une opération de communication
β_{cpu}	la puissance électrique d'effectuer une opération d'E/S
β_{es}	la puissance électrique d'effectuer une opération CPU
β_{net}	la puissance électrique d'effectuer une opération de communication
m	taille de la mémoire tampon pour l'opération de tri
$block$	taille d'une page dans le SGBD
L	bande passante du réseau
T_i	la taille de la table i
f_q	la fréquence d'exécution de la requête q
f_v	la fréquence de mise à jour de la vue v
t_i	nombre de tuples d'entrée pour l'opérateur i
p_i	nombre de pages d'entrée pour l'opérateur i
f	sélectivité d'index
s	la taille de la relation d'entrée pour l'opérateur de tri
n_{hash}	nombre de clauses de hachage en phase de construction
p_{hash}	nombre de partitions de hachage en phase de sondage
$p_{outer/inner}$	nombre de pages pour l'opérateur de jointure
$t_{outer/inner}$	nombre de tuples pour l'opérateur de jointure
n_{group}	nombre de colonnes de groupement

6.4.2.1 Modèle de coût d'énergie

La consommation d'énergie peut être réduite soit par la réduction de la consommation moyenne d'énergie soit le temps d'exécution des requêtes. Puisque les optimisations visant l'amélioration des performances des requêtes sont largement étudiées, dans ce travail, nous nous concentrons sur la partie puissance de l'Équation (3.2) (page 67). La première étape consiste à modéliser la puissance afin d'estimer sa consommation par les requêtes. À cette fin, nous nous sommes basés sur notre modèle de coûts d'énergie proposé dans le Chapitre 4. Le processus de construction de ce modèle est décrit dans la suite de cette section.

6.4.2.1.1 Paramètres du modèle de coût

Dans un SGBD traditionnel, le coût d'exécution de la requête est traité comme une combinaison linéaire de trois composantes : le coût de CPU, le coût d'E/S et le coût de communication [282]. Nous suivons la même logique pour proposer un modèle de coût de l'énergie pour une charge de requêtes. Pour traiter les tuples, chaque opérateur dans une requête doit effectuer des tâches de CPU et/ou d'E/S. Le coût de ces tâches est la puissance électrique active consommée afin de les terminer. Dans cette étude, nous nous concentrons sur une configuration de serveur centralisée. Ainsi, le coût de communication est ignoré. Plus formellement, pour une charge de requêtes donnée W de n requêtes $\{Q_1, Q_2, \ldots, Q_n\}$. Le modèle de coût de l'énergie pour l'ensemble de requêtes est défini comme suit :

$$Puissance(W) = \frac{\sum_{i=1}^{n} Puissance(Q_i) \times Temps(Q_i)}{Temps(W)} \tag{6.1}$$

$Temps(Q_i)$ et $Temps(W)$ représentent respectivement, le temps d'exécution de la requête i et de la charge W. Nous obtenons ces facteurs à partir de notre modèle de coût de performance que nous allons décrire dans la section suivante. Comme mentionné précédemment, la $Puissance(Q_i)$ dépend du nombre d'opérations algébriques de la requête Q_i. Pour illustrer cela, soit m_i le nombre de ces opérations $\{OP_1, OP_2, \ldots, OP_{m_i}\}$. Le coût $Puissance(Q_i)$ d'une requête i est donné par l'équation suivante :

$$Puissance(Q_i) = \beta_{cpu} \times \sum_{j=1}^{m_i} COUT\_CPU_j + \beta_{es} \times \sum_{j=1}^{m_i} COUT\_ES_j \tag{6.2}$$

Où β_{cpu} et β_{es} sont les paramètres du modèle (à savoir, les unités du coût d'énergie) pour les opérateurs. Le *COUT_ES* est le nombre prévu de *E/S* requis pour l'exécution d'un opérateur spécifique. Le *COUT_CPU* est le nombre prévu de *Cycle de CPU* et de *lecture du mémoire cache* que le SGBD doit exécuter pour un opérateur spécifique. Ces paramètres sont calculés par rapport aux nœuds (opérateurs) actuellement matérialisés. Un résumé des formules utilisées pour calculer les coûts d'E/S et CPU pour chaque opérateur de base peut être trouvée dans le Tableau 6.2 avec une partie des symboles déjà introduits dans le Tableau 6.1. Ces formules sont simplifiées à partir du modèle de coût utilisé par l'optimiseur de PostgreSQL.

Tableau 6.2 – Les paramètres du modèle de coût pour les opérateurs SQL.

Paramètre	Coût d'E/S	Coût de CPU
Accès séquentiel	p_{seq}	t_{seq}
Accès par index	p_{index}	$t_{index} \cdot f$
Accès par bitmap	p_{bitmap}	$t_{bitmap} \cdot f$
Jointure par boucle imbriquée	$p_{outer} + p_{inner}$	$t_{outer} \cdot t_{inner}$
Jointure par tri-fusion	$p_{outer} + p_{inner}$	$t_{sort(outer)} + t_{sort(inner)}$
Jointure par hachage	$p_{outer} + p_{inner}$	$t_{outer} \cdot n_{hash} + t_{inner} \cdot p_{hash}$
Tri	$p_{sort}, s < m$; $0, else$	$t_{sort} \cdot log_2(t_{sort})$
Agrégation	0	t_{agg}
Groupement	0	$t_{group} \cdot n_{group}$

6.4.2.1.2 Régression polynomiale

Maintenant, il reste à estimer les paramètres β_{cpu} et β_{es} de l'Équation (6.2). Notre méthodologie utilise la technique de régression pour calculer ces paramètres de coût d'énergie. Nous avons employé la méthode de régression polynomiale multiple, avec un ordre de *m=4*. Le coût de l'énergie $Puissance(Q_i)$ de la requête Q_i est calculé comme suit :

$$\begin{aligned} Puissance(Q_i) = \beta_0 + \beta_1(COUT\_ES) + \beta_2(COUT\_CPU) + \\ \beta_3(COUT\_ES^2) + \beta_4(COUT\_CPU^2) + \\ \beta_5(COUT\_ES \times COUT\_CPU) + \\ \cdots + \\ \beta_{13}(COUT\_ES^4) + \beta_{14}(COUT\_CPU^4) + \epsilon \end{aligned} \tag{6.3}$$

Où $COUT\_ES$, $COUT\_CPU$ représentent la somme des coûts d'E/S et CPU des opérateurs de la requête respectivement, ces coûts sont fournis par le module de statistiques du SGBD, et ϵ est un terme de bruit qui fait référence à l'erreur de mesure. Les paramètres β sont des coefficients de régression qui seront estimés durant de la phase d'apprentissage. Ainsi, les modèles de régression sont résolus par l'estimation des paramètres du modèle β, qui est fait avec la méthode des moindres carrés (voir Chapitre 4).

6.4.2.2 Modèle de coût de performance

De la même manière, nous utilisons les opérateurs algébriques pour estimer le temps d'exécution de la requête. Soit LV l'ensemble des vues candidat disponible à partir des nœuds de $\mathcal{GGR}$, et MV ($MV \in LV$) l'ensemble des vues matérialisées. Chaque vue $v \in LV$ possède une fréquence de mise à jour f_v associée. f_{Q_i} est la fréquence de la requête, Ainsi, le modèle de coût de performance pour la charge de requêtes est défini comme suit :

$$Temps(W) = \sum_{i=1}^{n} f_{Q_i} \times Temps(Q_i) \tag{6.4}$$

Le coût $Temps(Q_i)$ d'une requête i est calculé par l'équation suivante :

$$Temps(Q_i) = \alpha_{cpu} \times \sum_{j=1}^{m_i} COUT\_CPU_j + \alpha_{es} \times \sum_{j=1}^{m_i} COUT\_ES_j \tag{6.5}$$

Où $COUT\_ES$, $COUT\_CPU$ sont déjà décrits. Les paramètres α sont des coefficients spécifiés à notre machine de test, utilisés pour convertir les coûts de requête à des valeurs de temps. α_{cpu} est le temps *CPU* requis pour exécuter un *Cycle de CPU* et α_{es} est le temps *d'E/S* du périphérique pour exécuter une opération d'E/S.

En raison des changements qui peuvent survenir dans les relations de base, les vues matérialisées doivent être tenues à jour. Par conséquent, le coût de maintenance de la vue doit être considéré lors du $\mathcal{PSV}$. Dans notre travail, nous supposons une stratégie de maintenance incrémentale, lorsque les relations de base changent, nous calculons les changements produits dans les vues, puis nous étendons ces changements à d'autres vues matérialisées. Le coût de la mise à jour de MV est calculé comme suit :

$$U(MV) = \sum_{v \in MV} f_v \times Coût(v) \tag{6.6}$$

Où $Coût(v)$ est le coût de maintenance de la vue v. Il est calculé en additionnant le nombre de changements dans les relations de base à partir de laquelle v est mis à jour. Pour calculer ce changement, nous exploitons les techniques décrites dans [181]. Dans notre étude, nous considérons le cas du $\mathcal{PSV}$ statique, dans laquelle la charge de requêtes est statique, le cas où la charge de requêtes change, également appelé $\mathcal{PSV}$ dynamique, n'est pas traité dans cette thèse. Cependant, il est important de noter que nos algorithmes peuvent être facilement adaptés en utilisant, par exemple, le modèle de coût proposé dans [161] pour capturer ce changement.

6.4.3 Méthodologie de sélection des vues matérialisées

Dans cette section, nous décrivons notre méthode pour résoudre le $\mathcal{PSV}$. Comme dans les études d'état de l'art, le processus de sélection des vues matérialisées commence principalement par la construction du graphe global de requêtes ($\mathcal{GGR}$) qui fusionne tous les plans d'exécution des requêtes de la charge (appelé *Multi-Views Processing Plan* dans [286]). Le $\mathcal{GGR}$ est un graphe acyclique orienté. Il comporte quatre niveaux : le premier niveau contient les nœuds de feuille qui représentent les tables de la base de données. Au deuxième niveau, on trouve les nœuds qui représentent les résultats des opérations algébriques unaires telles que la sélection et la projection. Le troisième niveau contient des nœuds représentant des opérations binaires telles que la jointure, l'union, etc. Le dernier niveau représente les résultats de chaque requête. Chaque nœud intermédiaire du graphe est étiqueté avec son coût d'énergie et de traitement. Un exemple d'un $\mathcal{GGR}$ est représenté sur la Figure 6.2.

Notant que chaque nœud intermédiaire du $\mathcal{GGR}$ est un candidat pour la matérialisation. Pour choisir le meilleur $\mathcal{GGR}$ qui a le coût minimum, Yang *et al.* ont proposé deux algorithmes dans [286]. Le premier algorithme génère tous les $\mathcal{GGR}$ possibles et le plan qui a le coût minimum sera choisi. Le second algorithme est basé sur l'optimisation linéaire en nombres entiers, il génère les vues matérialisées candidates qui ont des gains positifs entre le traitement des requêtes et la maintenance des vues, seuls

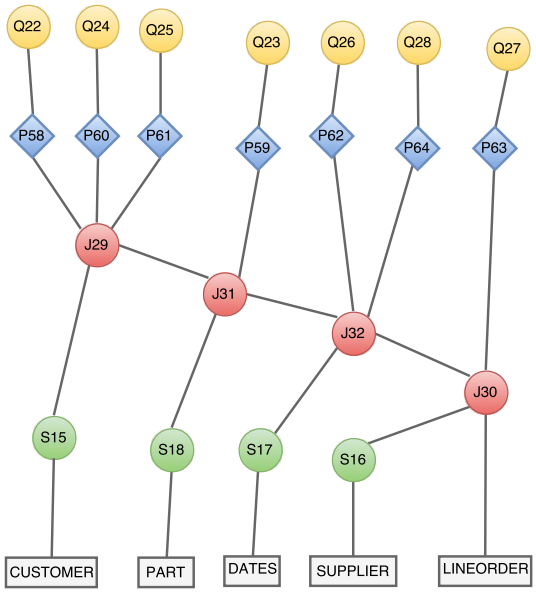


FIGURE 6.2 – Exemple d'un $\mathcal{GGR}$ de 7 requêtes.

les nœuds candidats avec des gains positifs sont sélectionnés pour être matérialisées. La principale limitation du travail de Yang *et al.* est la scalabilité de l'algorithme utilisé pour construire le $\mathcal{GGR}$, car la génération de tous les plans globaux possibles est impossible pour les applications qui contiennent un grand nombre de requêtes.

Pour faire face à ce problème, Boukorca *et al.* [40] ont proposé une nouvelle approche basée sur la structure de données de graphe inspiré du domaine de la conception assistée par ordinateur pour l'électronique (Electronic Design Automation). Le processus de génération est composé de cinq étapes : (1) l'analyse syntaxique de la requête SQL pour identifier les opérations logiques (nœuds), (2) la modélisation des nœuds de jointure par un hypergraphe, où les sommets représentent l'ensemble des nœuds de jointure et l'ensemble des hyper-arêtes représentent la charge de requêtes, (3) le calcul des composants connectés en utilisant les algorithmes de partition des hypergraphes du domaine de conception des circuits électroniques, résultant en un ensemble de composants disjoints de requêtes, chaque composant est représenté par un sous-hypergraphe, (4) la transformation de chaque sous-hypergraphe en un graphe orienté en utilisant un modèle de coût et un ensemble d'algorithmes, (5) la fusion des graphes produits pour générer le graphe global de requêtes ($\mathcal{GGR}$). Pour illustrer ces points, considérons un exemple en utilisant les algorithmes de Boukorca *et al.* [40].

Exemple 1.

Supposant une charge composée de 7 requêtes de type OLAP générées à partir du Star Schema Benchmark (SSB) [196]. Le benchmark contient une table de fait `Lineorder` *(notée $\mathcal{L}$), et quatre tables de dimension :* `Customer` *($\mathcal{C}$),* `Supplier` *($\mathcal{S}$),* `Part` *($\mathcal{P}$) et* `Dates` *($\mathcal{D}$). Le $\mathcal{GGR}$ correspondant à ces 7 requêtes est représenté dans la Figure 6.2. À partir de ce plan de requête global, deux alternatives principales sont possibles pour choisir des vues matérialisées pertinentes : (i) matérialiser tous les nœuds ou (ii) matérialiser certains nœuds intermédiaires. Dans cet exemple, nous considérons les nœuds de jointure désignés par J_i comme des vues candidates, le nombre total de configurations possibles est de 2^n, où n est le nombre de*

Tableau 6.3 – Les performances et la puissance active de la charge de requêtes pour différentes configurations de vues.

Vues matérialisées	Temps (min)	Puissance (watts)
$\mathcal{C}, \mathcal{P}, \mathcal{D}, \mathcal{S}, \mathcal{L}$	10,83	16,07
J_{32}	3,2	19,73
J_{31}	5,13	18,06
J_{31}, J_{32}, J_{30}	2,28	21,17
J_{29}	6,18	17,66
J_{29}, J_{31}, J_{30}	2,45	21,01
$J_{29}, J_{30}, J_{31}, J_{32}$	1,9	23,11

nœuds de jointure ($n = 4$). Le fait que nous avons seulement 16 configurations, nous utilisons une évaluation exhaustive. Pour chaque configuration, nous mesurons le temps d'exécution de la charge de requêtes totale et la consommation de puissance active moyenne. Les ensembles de données, la configuration du système, ainsi que les outils de mesure utilisés dans nos expérimentations seront regroupés dans la Section 6.6.1 (page 166). Les résultats obtenus sont résumés dans le Tableau 6.3. Notant que la configuration peut contenir une ou plusieurs vues matérialisées, aussi, les contraintes de stockage et de maintenance sont relaxées pour des raisons de simplicité.

Plusieurs notes peuvent être tirées de cette expérimentations :

- *le scénario qui consiste à matérialiser toutes les vues candidates donne la meilleure performance de requête avec la consommation d'énergie la plus élevée. Ainsi, le traitement des requêtes aussi rapide que possible n'est pas toujours la façon la plus économe en énergie, ce qui a déjà été mentionné dans les recherches précédentes, telles que dans [155, 283, 152] ;*
- *le scénario dans lequel aucune vue n'est sélectionnée offre la pire performance et la plus faible consommation d'énergie. Dans la pratique, cette situation n'est pas favorable puisque l'amélioration de la performance est primordiale pour toute base de données;*
- *enfin, matérialisant un certain nombre de vues parmi les candidats est le meilleur compromis. Par exemple, matérialisant J_{32} donne un bon scénario assurant le compromis. Cet exemple nous motive à définir tout d'abord un algorithme de sélection des configurations pertinentes qui permettent d'éviter la recherche exhaustive, puis en choisissant la meilleure solution répondant aux exigences des DBA.*

6.5 Optimisation multi-objectifs

Dans les problèmes d'optimisation multi-objectifs, il est essentiel de faire des compromis entre les objectifs. Les fonctions objectives de notre problème sont l'énergie (Équation (6.2) [page 155]) et la performance (Équation (6.5) [page 157]) respectivement, sous la contrainte de la taille de stockage pour garantir que l'espace total de vues matérialisées est au plus égale à la capacité de l'espace de stockage (S), et la contrainte des coûts de maintenance pour garantir que les coûts de mise à jour des vues à matérialiser est inférieure à la durée de maintenance disponible (U). Plus formellement, le $\mathcal{PSV}$

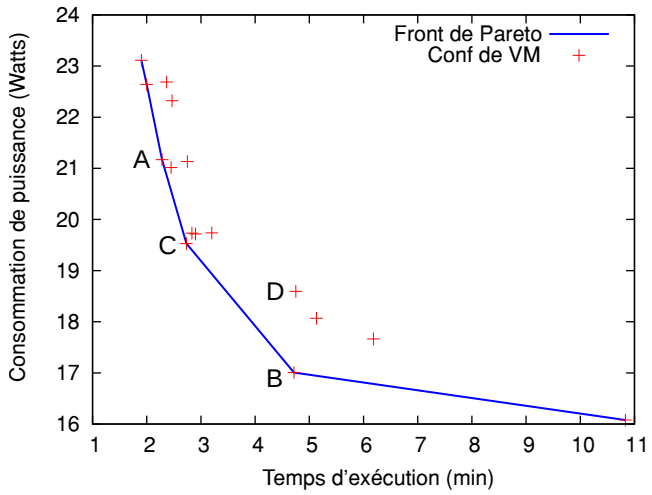


FIGURE 6.3 – Exemple de frontière de Pareto lors de la minimisation de deux objectifs (performance et énergie).

multi-objectif pour une solution MV peut être exprimé comme suit :

$$\text{minimiser } f(x) = (Temps(MV_x), Puissance(MV_x))$$
$$\text{avec } e(x) = S(MV_x) \leq S,\ U(MV_x) \leq U$$

Où $S(MV)$ et $U(MV)$ représentent respectivement les coûts de stockage et de maintenance de MV.

Rappelons que dans l'optimisation multi-objectifs, il n'existe typiquement pas de solution qui minimise toutes les fonctions objectives simultanément. Par conséquent, des solutions qui ne peuvent pas être améliorées dans l'un des objectifs sans dégrader au moins l'un des autres objectifs sont appelés *des solutions d'optimalité Pareto*. Plus formellement, une solution x_1 est dit *domine* une autre solution x_2 si $f_i(x_1) \leq f_i(x_2)$ pour tous les indices $i \in \{1, 2, \ldots, k\}$ et $f_j(x_1) < f_j(x_2)$ pour au moins une fonction objective j. Une solution x_1 est appelé optimale Pareto s'il n'existe pas une autre solution qui la domine. L'ensemble des résultats optimaux de Pareto est appelé la *frontière de Pareto*. Si l'on considère notre exemple précédent, les valeurs de performance/puissance de toutes les configurations de vues possibles sont tracées dans la Figure 6.3. Le point D n'est pas sur la frontière de Pareto car il est dominé par les deux points A, B, et le point C. Les points A, B et C ne sont pas strictement dominées par les autres, et donc se trouvent sur la frontière. Le but ultime est d'identifier des solutions dans l'ensemble d'optimalité de Pareto. Cependant, l'identification de la totalité de cette ensemble, pour notre problème, est pratiquement impossible en raison de sa NP-complétude. Par conséquent, une approche efficace est de rechercher un ensemble de solutions qui représentent l'ensemble d'optimalité de Pareto aussi bien que possible.

6.5.1 Techniques de résolutions

Comme nous l'avons indiqué dans la section ci-dessus, l'optimisation multi-objectifs renvoie un ensemble de solutions Pareto. Cependant, la sélection d'une solution unique pour être déployer dans le système par l'administrateur de base de données peut être très difficile à cause de la multiplicité des solutions retournées. Par conséquent, le problème du choix d'un bon compromis entre les fonctions

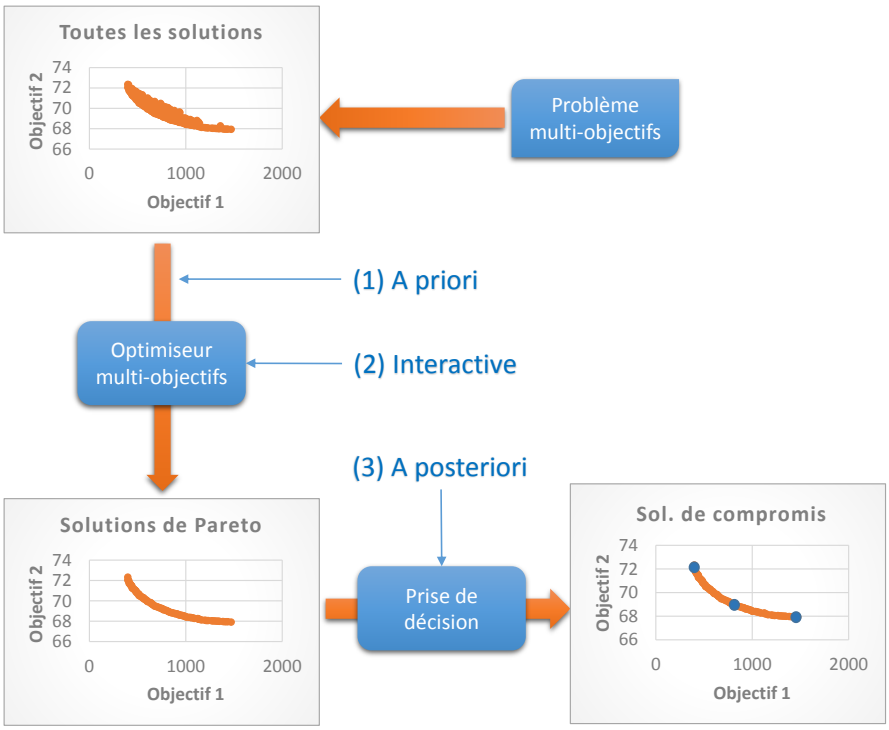


FIGURE 6.4 – Procédé d'application d'optimisation multi-objectifs.

objectives a été abordé dans la littérature et résolu à l'aide de trois méthodes : (i) méthodes *a priori* (ii) méthodes interactives (iii) et méthodes *a posteriori* [127]. Ces approches sont intégrées aux étapes de traitement de l'optimisation multi-objectifs comme le montre la Figure 6.4. La description de chaque approche est la suivante :

1. **Méthodes a priori**. Dans ces méthodes, le DBA intervient *avant* l'exécution du processus d'optimisation. Le DBA définit le compromis entre les fonctions objectives et effectue une seule étape de recherche pour obtenir la solution désirée. Cette méthode est intéressante car elle nécessite une seule étape de recherche. Cependant, il n'est pas simple pour le DBA pour définir les valeurs de compromis sans une connaissance du problème et sans voir des solutions à l'avance, surtout dans notre cas où nous traitons une nouvelle exigence non-fonctionnelle, qui est l'énergie.
2. **Méthodes interactives**. Dans le deuxième type de méthodes, le DBA intervient *pendant* l'exécution du processus d'optimisation. Le DBA est invité à définir les valeurs de compromis afin de réorienter progressivement l'espace de recherche vers une solution préférée. Bien que cette méthode peut parvenir à une bonne solution respectant la préférence de DBA, elle requiert toute l'attention de DBA pour évaluer la qualité de chaque solution pendant l'ensemble du processus d'optimisation, ce qui prend du temps pour un grand espace de recherche. En outre, les connaissances spécifiques au domaine du DBA est nécessaire pour prendre de bonnes décisions.
3. **Méthodes a posteriori**. Dans le dernier type de méthodes, le DBA intervient *après* l'exécution du processus d'optimisation. Cette méthode fournit aux DBA un ensemble de solutions et le choix de meilleur compromis est reporté à la fin du processus d'optimisation. L'objectif est de montrer ces solutions aux DBA en lui permettant de juger et sélectionner la solution la plus

préférée parmi les différentes solutions proposées. Autrement dit, le processus d'optimisation est entièrement automatisé et ne nécessite aucune attention du DBA ou de connaissances de problème. Toutefois, si l'ensemble de résultats est très grand, le processus peut prendre beaucoup de temps pour terminer.

Dans notre étude, nous avons opté pour une approche *a posteriori* pour donner la possibilité aux DBA de *prioriser ou non* l'énergie sur les performances des requêtes. Comme nous l'avons mentionné dans la section précédente, l'évaluation de toutes les solutions candidates est impossible dans notre cas, même pour un petit nombre de relations, en raison de la NP-Complétude du problème. Motivés par ce fait, nous utilisons des algorithmes évolutionnaires pour générer uniquement l'ensemble d'optimum de Pareto, puis nous appliquons la méthode des sommes pondérées à cet ensemble afin d'offrir aux DBA une solution unique. Dans la section suivante, nous discutons en détail les algorithmes proposés pour résoudre notre problème.

6.5.2 Algorithmes évolutionnaires

Les algorithmes évolutionnaires (AE) sont adaptés à des problèmes d'optimisation multi-objectifs, à travers lesquels de grands espaces de recherche peuvent être manipulés, et multiples compromis alternatifs peuvent être générés en une seule phase d'optimisation [299]. L'idée générale derrière les AE est de simuler le processus évolutif naturel dans lequel les individus les plus forts survivront après plusieurs générations. Les détails de la mise en œuvre de notre approche en utilisant les AE sont fournis dans les sections suivantes.

6.5.2.1 Représentation des solutions

Pour représenter des solutions d'un $\mathcal{GGR}$, toute les vues candidates sont étiquetées. Les solutions sont représentées comme une chaîne binaire de 1 et de 0 de sorte que si une vue particulière est sélectionnée pour la matérialisation, le bit correspondant dans la chaîne de solution prendra 1, et 0 sinon. Par exemple, dans la Figure 6.2, la solution $\{1, 1, 1, 0\}$ indique que les nœuds $\{J_{29}, J_{30}, J_{31}\}$ sont choisis pour la matérialisation. Pour chaque solution, les fonctions de coût renvoient le coût d'exécution estimé et la consommation d'énergie de la charge de requêtes, ainsi que la taille et le coût de maintenance des vues. La meilleure solution est celle qui a la plus petite valeur des coûts en respectant les contraintes de taille et de maintenance.

6.5.2.2 Fonction de fitness

La fonction de fitness mesure la qualité d'une solution (un ensemble de vues sélectionnées pour matérialiser). Nous utilisons l'approche de classement de Pareto comme une fonction de fitness, basée sur l'algorithme génétique NSGA-II proposé par Deb *et al.* [82], qui utilise explicitement le concept de Pareto dominance dans l'évaluation de la qualité des solutions, où une probabilité de sélection est affectée à chaque solution. Cet algorithme utilise des techniques pour améliorer la convergence et de garantir la diversité et la diffusion des solutions. Nous utilisons la sélection du tournoi binaire dans lequel les individus sont choisis au hasard de la même manière qu'un « tournoi ». La population est

classée selon la règle de dominance, puis chaque solution est attribuée une valeur de fitness en fonction de son rang dans la population. Puisque nous voulons minimiser les fonctions objectives, par conséquent, un rang minimal correspond à une meilleure solution. La solution infaisable, à savoir, une solution qui viole les contraintes est traitée comme suit : essentiellement, une solution réalisable domine toujours une solution infaisable, et si toutes les solutions sont infaisables, ceux avec des violations minimales de contrainte survivent.

6.5.2.3 Croisement

Le croisement encourage les permutations des informations entre les différents individus. Il favorise l'héritage de bons gènes d'une génération à une autre et le regroupement de meilleurs individus. Le croisement moitié uniforme (Half uniform crossover) est utilisé dans notre algorithme évolutionnaire ; dans cette technique, la moitié des bits qui sont différents entre les parents seront échangés. A cet effet, la technique calcule le nombre de bits différents en utilisant la distance de Hamming entre les parents. La moitié de ce nombre est le nombre de bits échangés entre les parents pour former des fils. Par exemple, étant donné deux individus P_1 et P_2, les résultats du croisement sont C_1 et C_2 :

$$P_1 = \{0, 1, 0, 1, 1, 1, 0, 1, 0\}\ P_2 = \{0, 1, 1, 1, 1, 0, 1, 1, 0\}$$
$$C_1 = \{0, 1, 0, 1, 1, 0, 1, 1, 0\}\ C_2 = \{0, 1, 1, 1, 1, 1, 0, 1, 0\}$$

Pour générer C_1 et C_2, l'algorithme compte le nombre de bits différents entre P_1 et P_2, qui est égale 4, la moitié de ce nombre (donc 2) seront échangées pour créer les nouveaux individus (les bits numéro 6 et 7).

6.5.2.4 Mutation

La mutation est la modification aléatoire occasionnelle d'une valeur dans la chaîne de bits. Elle introduit de nouvelles fonctionnalités qui peuvent ne pas être présentes dans aucun membre de la population. La mutation est effectuée en utilisant la méthode de « Bit flip » dans notre travail. Chaque bit est basculé (commuté d'un 0 à 1 ou vice-versa) en utilisant une probabilité spécifiée. Par exemple, supposons que le 7^{me} bits de l'individu L_1 est choisi pour la mutation, la valeur de la chaîne de bits est 1, il sera retourné à 0 avec le taux de mutation spécifiée, la nouvelle personne est L'_1 :

$$L_1 = \{0, 1, 0, 1, 1, 0, 1, 1, 0\}$$
$$L'_1 = \{0, 1, 0, 1, 1, 0, 0, 1, 0\}$$

6.5.3 Prise de décision

Notons que les algorithmes évolutionnaires multi-objectifs produisent un ensemble de solutions. Pour donner aux DBA la possibilité de choisir la meilleure solution parmi cet ensemble selon un compromis fixe (entre les performances des requêtes et l'énergie consommée), nous proposons d'utiliser la méthode des *sommes pondérées des fonctions objectives* (SPFO). Dans cette méthode d'agrégation, nous calculons la somme pondérée des fonctions objectives normalisées pour agréger les objectifs en une fonction

objectif unique équivalente pour être optimisée. Cette méthode est définie comme suit [299] :

$$\begin{aligned} &\text{minimiser } y = f(x) = \sum_{i=1}^{k} \omega_i \cdot f_i(\overrightarrow{x}) \\ &\text{tel que } \sum_{i=1}^{k} \omega_i = 1 \end{aligned} \tag{6.7}$$

Où ω_i sont les coefficients de pondération représentant l'importance relative des k fonctions objectives du problème. Pour normaliser le vecteur des valeurs de fonctions objectives, nous utilisons la formule suivante :

$$x' = \frac{x - min(x)}{max(x) - min(x)} \tag{6.8}$$

Où x est la valeur d'origine et x' est la valeur normalisée. La méthode des SPFO est bien adaptée lorsque la frontière de Pareto est convexe, tel est le cas dans notre problème. Un exemple de compromis est le point C dans la Figure 6.3 ($\omega_1 = 0.5$, $\omega_2 = 0.5$).

L'Algorithme 6.1 résume le processus de résolution du $\mathcal{PSV}$. Il identifie un ensemble de vues (MV) qui respecte deux contraintes : la taille de stockage et le temps de maintenance, puis il applique la méthode des SPFO pour obtenir la configuration de vues qui a le plus petit coût de traitement possible de la charge de requêtes.

Algorithme 6.1 $\mathcal{PSV}$ Orienté Énergie

Entrée : $\mathcal{GGR}$, un graphique global de requête
Sortie : MV, l'ensemble de vues sélectionnées pour la matérialisation

$LV \leftarrow \emptyset$;
$IterationActuel \leftarrow 0$;

tant que $IterationActuel < MaxEvaluations$ et $aPlusNoeudCandidats(\mathcal{GGR})$ **faire**
    $CV \leftarrow GAGetNoeudCandidats(\mathcal{GGR})$;
    $CoûtTemps \leftarrow GetCoûtTemps(CV)$;
    $CoûtPuissance \leftarrow GetCoûtPuissance(CV)$;
    $Contrainte_i \leftarrow GetContrainte_i(CV)$;
    si $Contrainte_i \leq MaxContrainte_i$ **alors**
        $LV \leftarrow CV$;
    fin si
    $IterationActuel \leftarrow IterationActuel + 1$;
fin tant que

$WF \leftarrow \emptyset$;
pour tout $v \in LV$ **faire**
    $WF \leftarrow GetPoidFonctions(GetCoûtTemps(v), GetCoûtPuissance(v))$;
fin pour
$MV \leftarrow GetMinimum(WF)$;

retourner MV;

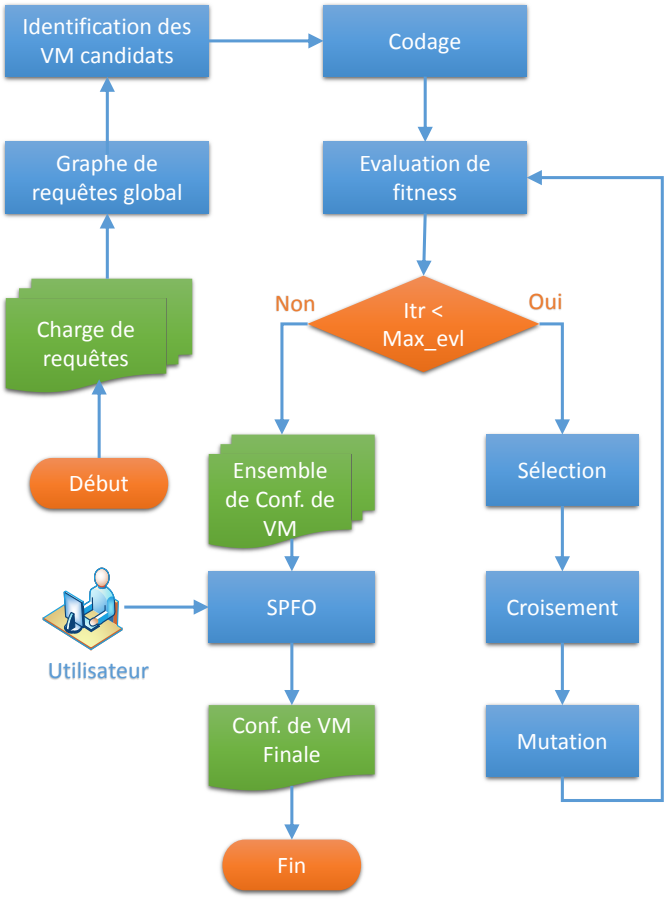


FIGURE 6.5 – Le processus de sélection des vues.

Le processus global de notre méthodologie est décrit dans la Figure 6.5. Ce processus prend une charge de requêtes en entrée, puis il crée le graphique unifié correspondant à cette charge en utilisant les algorithmes de Boukorca *et al.* [40]. Les nœuds intermédiaires de ce graphique sont considérés comme des candidats à la matérialisation. Enfin, l'AE est appliqué jusqu'à ce que le nombre maximum d'itérations est atteint. Comme mentionné précédemment, la solution est un ensemble de points de Pareto qui représente la meilleure configuration de vues sur les deux objectifs (la performance et l'énergie). Pour obtenir une solution unique, le DBA définit les paramètres de compromis souhaités dans la méthode des SPFO. La sortie est une configuration de vues unique.

6.6 Évaluation et résultats

Pour évaluer l'efficacité de notre proposition, nous avons mené plusieurs expérimentations. Dans la section suivante, nous présentons notre machine expérimentale pour calculer l'énergie et le jeu de données utilisé ainsi que le simulateur développé.

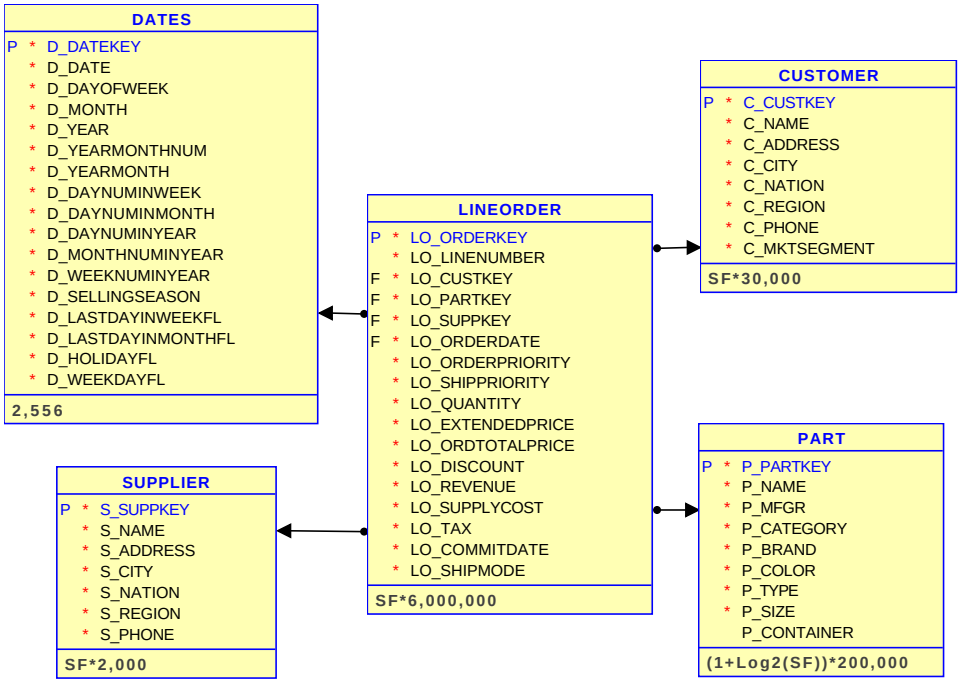


FIGURE 6.6 – Schéma du benchmark SSB.

6.6.1 Architecture des expérimentations

Nous nous sommes basés sur une configuration similaire à celle utilisée dans le Chapitre 4. Nous avons utilisé une station de travail Dell Precision T1500 ayant un processeur Intel Core i5 de 2.27GHz, 4 Go de mémoire DDR3. À noter que des techniques comme tension-fréquence dynamique du processeur (DVFS) ne sont pas appliquées dans nos expérimentations. Nous répétons chaque expérience plusieurs fois pour assurer la fiabilité des valeurs observées.

Notre station de travail est dotée de la dernière version du SGBD Oracle 11gR2 sous Ubuntu Server 14.04 LTS avec le noyau Linux 3.13, afin de minimiser les influences indésirables, nous avons désactivé les tâches de fond inutiles, nous avons également vidé le cache du système et le buffer de Oracle dans chaque exécution d'une requête.

Dans les expérimentations, nous utilisons les jeux de données et les requêtes issus du benchmark SSB et TPC-H, avec une taille (facteur d'échelle) de 10 Go et 100 Go. Les données et les requêtes sont générées à l'aide des outils disponibles dans les benchmarks. Les deux benchmarks illustrent des systèmes décisionnels qui examinent de grands volumes de données et exécutent différents types de requêtes avec un degré de complexité élevé. Le SSB est conçu pour le requêtage ad-hoc, le schéma illustre un modèle produits-commandes-fournisseurs contenant une table de faits et 4 tables de dimensions (cf. Figure 6.6), avec une charge de 13 requêtes décisionnelles caractérisées par un large volume de données. Les requêtes sont exécutées d'une façon isolée dans les expérimentations.

Nous avons développé un outil de simulation utilisant la plate-forme Java. Ce simulateur implémente les algorithmes de Boukorca *et al.* qui génèrent le $\mathcal{GGR}$. L'outil peut extraire automatiquement les

Tableau 6.4 – Caractéristiques des configurations de vues matérialisées.

Configuration de VM	Caractéristiques de VM
VM-Puissance	Consommation de puissance minimale
VM-Temps	Temps d'exécution minimale
VM-Compromis	Compromis entre la puissance et le temps
Origine	Sans vue matérialisée

Tableau 6.5 – Les paramètres de l'algorithme évolutionnaire.

Paramètre	Valeur
Type d'encodage	chaîne de bits
Méthode de sélection	tournoi binaire
taille de la sélection du tournoi	2
Type de croisement	Half Uniform
Probabilité de croisement	1,0
Type de mutation	Flip Bit
Probabilité de mutation	0,01
Nombre d'évaluations	10,000

caractéristiques de méta-données de la base de données. De plus, il intègre le Framework MOEA [21], qui est une bibliothèque Java pour les algorithmes évolutionnaires multi-objectifs. Pour chaque configuration de vue matérialisée produite par le framework MOEA, le simulateur calcule les performances et la consommation d'énergie de la charge de requêtes en utilisant les modèles de coûts. Dans notre expérience, nous considérons trois types de configuration de vues matérialisées par : (1) VM-Puissance qui est la configuration qui donne le coût d'énergie minimale, (2) VM-Temps est la configuration avec un coût minimum de temps, (3) VM-Compromis, en utilisant la méthode des SPFO avec $\omega_1 = 0,5$, $\omega_2 = 0,5$. Les détails de vues matérialisées candidates pour chaque charge de requêtes utilisée dans les expérimentations sont regroupés dans le Tableau 6.6. Nous supposons que la fréquence d'exécution des requêtes et la fréquence de mises à jour des vues sont uniformes.

Les paramètres de configuration de l'algorithme évolutionnaire sont résumés dans le Tableau 6.5. La taille de sélection du tournoi, du croisement, et les probabilités de mutation utilisés dans nos expérimentations sont les valeurs par défaut du framework MOEA. Le nombre maximum d'itération est fixé à 10,000, ce qui permet d'obtenir de bons résultats.

6.6.2 Résultats

Dans cette section, nous présentons les résultats de nos diverses expérimentations.

21. http ://www.moeaframework.org

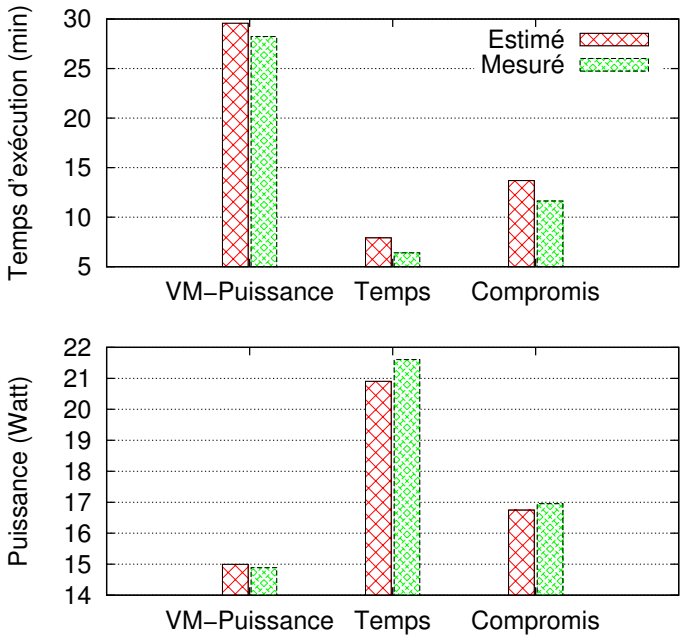


FIGURE 6.7 – Les erreurs d'estimation des performances et de la consommation d'énergie de la charge de requêtes.

6.6.2.1 Erreurs d'estimations des modèles de coût

Dans ce type d'expérimentations, étant donné le coût de performance et d'énergie prédit par nos modèles de coût pour une charge de requêtes, on le compare avec le temps d'exécution réellement observé et la consommation d'énergie du système. Pour évaluer notre proposition, nous exécutons 30 requêtes de type OLAP générées aléatoirement du benchmark SSB, avec une taille de base de données de 10 Go utilisant les trois configurations de vues matérialisées. Nous optons pour 30 requêtes au lieu des treize requêtes par défaut du SSB pour garantir un grand $\mathcal{GGR}$ et un ensemble riche de vues candidates à la matérialisation.

Les résultats des erreurs de prédiction du modèle de coût sont représentés dans la Figure 6.7. L'expérience démontre l'exactitude des prédictions de notre modèle. L'erreur d'estimation moyenne est de 12% pour le temps d'exécution et de 1,7% pour la puissance électrique, ce qui indique que le modèle est suffisamment précis pour les applications envisagées.

6.6.2.2 Qualité des solutions

Pour vérifier la qualité des solutions proposées par nos algorithmes, nous créons une simple charge de 15 requêtes. Nous générons d'abord son $\mathcal{GGR}$ et ensuite nous exécutons deux algorithmes différents pour obtenir des configurations de vues matérialisées : (i) l'algorithme de recherche exhaustive et (ii) notre algorithme évolutionnaire multi-objectifs (AEMO). Pour obtenir les points de frontière de Pareto depuis l'espace de recherche total, nous utilisons le fameux algorithme Block-Nested Loops (BNL) [36]. L'algorithme BNL lit à plusieurs reprises l'ensemble des solutions et maintient dans la mémoire principale une fenêtre de solutions incomparables. Quand une solution de s est lue à partir des données

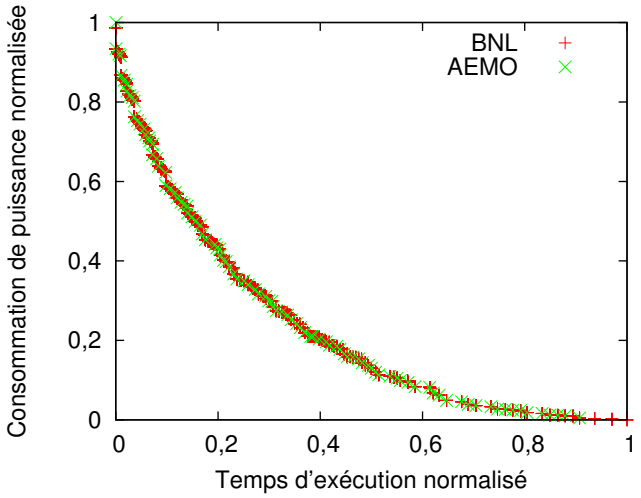


FIGURE 6.8 – Comparaison entre les solutions d'algorithme évolutionnaire et exhaustive.

Tableau 6.6 – Temps d'exécution de l'AE en fonction de la taille des charges de requêtes SSB.

Nombre de requêtes	Vues candidates	Temps d'exécution (sec)
15	2^{23}	2
30	2^{34}	2
100	2^{70}	4
200	2^{88}	7

d'entrée, elle sera comparée à toutes les solutions de la fenêtre. Sur la base de cette comparaison, s est soit éliminée ou placée dans la fenêtre en fonction de la règle de dominance. Nous avons opté pour une petite charge de 15 requêtes pour accélérer la recherche exhaustive de l'algorithme BNL. Les résultats de l'expérience des deux algorithmes sont tracés sur la Figure 6.8. Comme nous pouvons le voir sur la figure, les solutions résultantes de l'algorithme évolutionnaire et de l'algorithme BNL sont presque identiques avec une bonne répartition dans l'espace, ce qui signifie que l'AE peut trouver des solutions présentant un haut degré d'optimalité. De plus, le temps d'exécution de l'AE est très faible comparé à celui de l'algorithme de recherche exhaustif, pour un $\mathcal{GGR}$ de 200 requêtes, le premier prend 7 secondes tandis que le second prend 4 jours sans avoir terminé son exécution. Le Tableau 6.6 montre le temps d'exécution que l'AE a dépensé pour la génération des points de frontière de Pareto lorsque le nombre de requêtes de requêtes et de vues augmente. L'algorithme BNL n'est pas pris en compte car il nécessite plusieurs jours pour terminer son exécution, même avec une implémentation multi-thread exécutée sur un serveur haut de gamme. Les résultats obtenus montrent que notre approche donne rapidement les meilleures solutions.

6.6.2.3 Impact des coûts d'E/S et CPU

Le but de cette expérimentation est d'étudier les profils de performance et d'énergie des configurations de vues matérialisées. Cela nous aide à mieux comprendre l'impact des coûts d'E/S et CPU de vues et leurs relations avec la performance et la consommation d'énergie. Sur la base de ces connaissances, nous

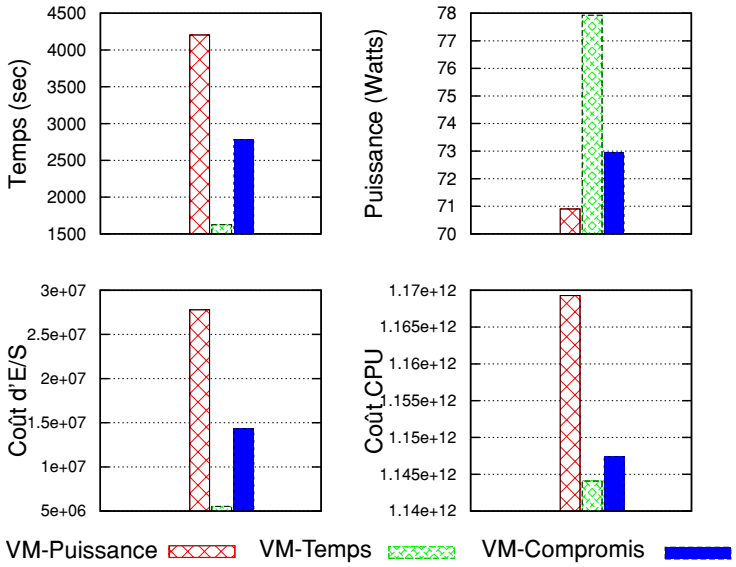
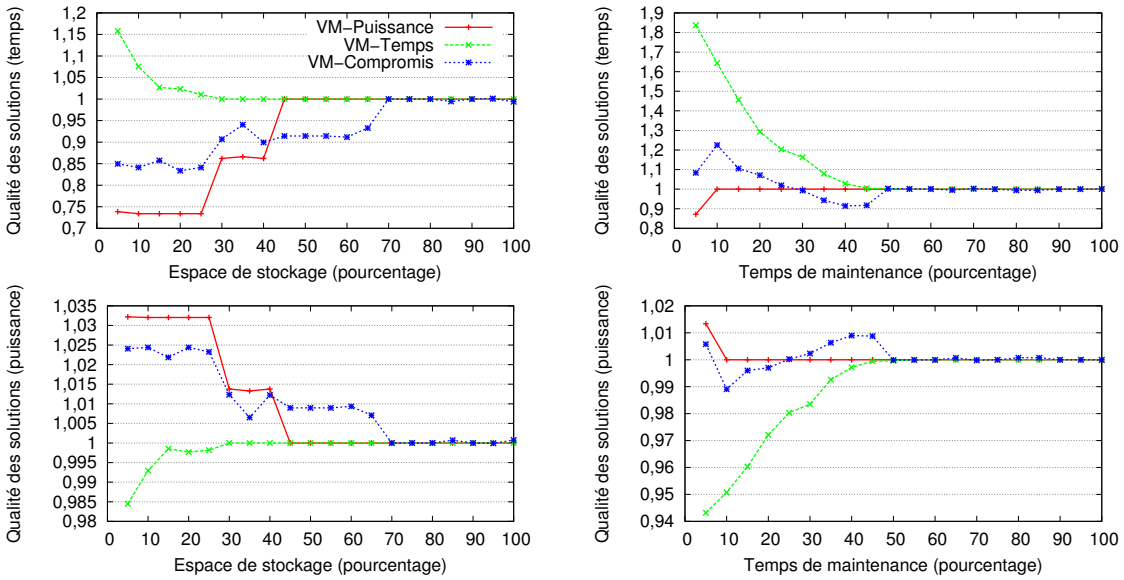


FIGURE 6.9 – Les caractéristiques des vues matérialisées et leurs impacts sur la performance et la consommation d'énergie.

pouvons concevoir les bases de données en tenant compte des stratégies d'économies et d'efficacité énergétique. Nous avons utilisé notre charge de 30 requêtes SSB pour générer le $\mathcal{GGR}$ correspondant, puis en utilisant l'algorithme évolutionnaire, en obtient les configurations de vues matérialisée dominantes. Ensuite, nous prenons les configurations VM-Temps, VM-Puissance, VM-Compromis et nous calculons leurs temps d'exécution, consommation d'énergie, le coût total d'E/S, et le coût total de CPU. Les résultats sont présentés dans la Figure 6.9. À partir de cette dernière, nous remarquons que : (1) la configuration VM-Temps est caractérisée par un nombre minimal des coûts d'E/S et de CPU, qui conduisent à un temps d'exécution court, mais une forte consommation d'énergie, (2) VM-Puissance a des coûts d'E/S et de CPU élevés, conduisant à une faible consommation d'énergie avec un temps d'exécution long, (3) dans la configuration VM-Compromis, les coûts et le temps/puissance sont dans des valeurs de compromis. L'observation immédiate est que le temps d'exécution et le coût d'E/S sont presque identiques dans toutes les configurations, ce qui est déjà connu dans l'optimisation des bases de données traditionnelles en raison du temps de traitement CPU rapide par rapport aux périphériques d'E/S. La seconde observation est que ni le coût d'E/S, ni le coût de CPU contribuent directement à la consommation d'énergie. Notre interprétation est que pour certaines requêtes, l'opération de lecture de gros fichiers a besoin de tâches d'E/S supplémentaires. Lorsque les résultats intermédiaires ne peuvent pas tenus dans la mémoire, ils seront écrits sur le disque et lus plus tard. Cette surcharge est traduite par une augmentation du temps d'attente de CPU. En d'autres termes, l'optimiseur de requête passe plus de temps dans les opérations de lecture/écriture au lieu de traiter les données. Ainsi, les requêtes dominées par des coûts d'E/S ont une moindre consommation de puissance électrique. D'un autre côté, quand la taille des fichiers est petite, l'opération de lecture de données se termine rapidement et en son temps restant, la requête est dominée par le processeur. Cela peut se traduire par une consommation de puissance électrique élevée. Ainsi, pour produire des vues matérialisées avec un compromis entre la performance et la puissance électrique, nous recommandons de choisir des vues avec des coûts moyens d'E/S et de CPU.



(a) L'impact du contrainte d'espace de stockage.

(b) L'impact du contrainte du coût de maintenance.

FIGURE 6.10 – Qualité des solutions en terme de performance et de consommation d'énergie avec la contrainte des ressources.

6.6.2.4 Impact d'espace de stockage

Dans cette série d'expérimentations, nous considérons la contrainte de l'espace de stockage. Soit S l'espace de stockage nécessaire pour sauvegarder tous les vues candidates ; S se varie de 5% à 100% ; la charge de requêtes est la même que celle utilisée dans les expérimentations précédentes (30 requêtes OLAP basées sur le benchmark SSB). Nous calculons la qualité des solutions (Q_s), qui est exprimée par le ratio entre le coût de chaque solution et les meilleurs coûts des solutions que notre algorithme peut obtenir lorsque les contraintes sont ignorées. L'optimalité de Pareto (cf. Section 6.5 [page 159]) implique que, dans certains cas, Q_s est supérieur à 1. Cependant, plus la valeur du Q_s est proche à 1, plus la qualité de la solution est meilleure. Pour chaque type de configuration, nous calculons la variation des qualités des solutions pour la charge de requêtes en fonction de temps d'exécution et de la consommation d'énergie, par rapport à l'espace de stockage alloué pour les vues matérialisées. Les résultats sont tracés dans la Figure 6.10a. Nous notons que la qualité des solutions s'améliore quand S est plus relaxé. Ceci est dû au fait qu'il y aura plus d'espace disque pour stocker les vues matérialisées. À partir de la même figure, on remarque que l'algorithme évolutionnaire trouve la meilleure solution pour VM-Temps rapidement, exactement à partir du point $S = 30\%$ (c-à-d $Q_s = 1$), ce n'est pas le cas pour la VM-Puissance, où la meilleure solution est obtenue quand S est à 45%. En conséquence, la configuration VM-Compromis est devenue stable jusqu'à $S = 70\%$. Ceci est dû à la diversité des solutions apportées par l'algorithme évolutionnaire ; un bon compromis peut être vu quand S prend une valeur comprise entre 70% et 100% de l'espace total. La Figure 6.10a montre également que les valeurs de performance et de puissance évoluent en sens inverses (lorsque la puissance est élevée, le temps est court et vice versa), cela correspond au principe des frontières de Pareto discutés précédemment.

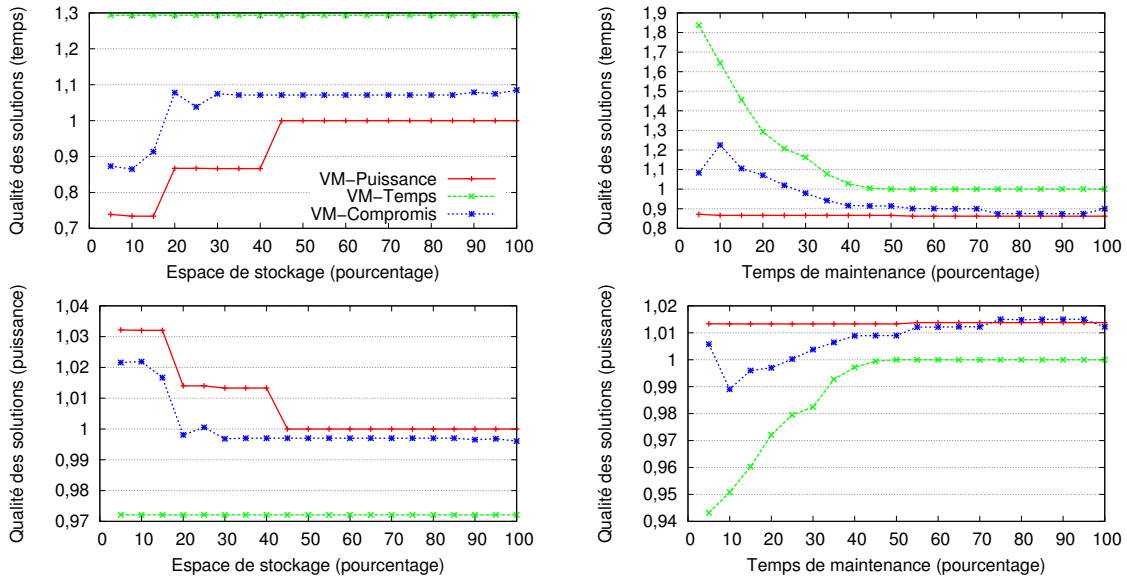
6.6.2.5 Impact du coût de maintenance

Le but de cette expérimentation est de sélectionner un ensemble de vues, tout en respectant la contrainte des coûts de maintenance. Cette contrainte est évaluée comme suit. Soit T le temps de maintenance de vue totale, lorsque le résultat de chaque nœud candidat de la charge de requêtes est matérialisé ; T se varie entre 5% et 100%. Dans la Figure 6.10b, la qualité des solutions trouvées par notre approche en termes de temps d'exécution et de consommation d'énergie est illustrée. De même que pour la contrainte d'espace, la qualité des solutions s'améliore quand T devient plus relaxé, car ça nécessite plus de temps pour maintenir les vues matérialisées. Comme nous pouvons le voir sur la même figure, la meilleure solution pour VM-Puissance est trouvée quand T est à 10%, tandis que les autres configurations ont convergé vers la meilleure solution quand T atteint 50%. Cela est dû à la haute intensité énergétique de la tâche de maintenance, car elle implique de nombreuses opérations de comparaison pour calculer le changement dans la relation de base, ce qui impliquent plus de tâches de CPU supplémentaires et une consommation d'énergie élevée. La meilleure solution de compromis est trouvée rapidement par rapport au cas de la contrainte de l'espace, bien que les deux contraintes semblent similaires, elles se distinguent par une différence significative. L'espace occupé par un ensemble de vues augmente toujours quand une nouvelle vue est insérée, alors ce n'est pas le cas pour le coût de maintenance ; il est possible de diminuer le temps de mise à jour d'un ensemble de vues après l'ajout d'une nouvelle vue [174]. Ceci est clairement reflété par la Figure 6.10b, quand T est entre 5% et 25% de l'intervalle de temps total, les solutions sont plus efficaces en terme de performances. D'autre part, quand T est entre 25% et 50% les solutions sont plus efficaces en terme d'énergie. Quand T dépasse 50%, l'intervalle de temps devient suffisant pour obtenir la meilleure solution de compromis.

6.6.2.6 Impact d'espace de stockage et du coût de maintenance

Afin d'examiner l'impact des contraintes d'espace de stockage et du coût de maintenance sur la qualité des solutions, nous procédons à cette série d'expérimentations. Dans le premier cas, nous avons fixé l'intervalle de temps T à 20% et nous avons varié les valeurs de la contrainte de l'espace S de 5% à 100%. Dans la Figure 6.11a, nous illustrons la qualité des solutions fournies par l'AE pour les différentes valeurs de S. Nous observons que seule la configuration VM-Puissance converge vers la meilleure solution, tandis que la configuration VM-Temps ne converge pas dans les deux objectifs à savoir le temps d'exécution et la consommation d'énergie (c'est à dire, $Q_s \approx 1,3$, $Q_s \approx 0,97$ respectivement). Cela conduit au fait que dans cette situation, les valeurs restrictives de la contrainte de maintenance produisent des solutions qui sont efficaces en terme d'énergie, mais avec un temps d'exécution important. Par conséquent, la solution de compromis n'est pas atteinte, même lorsque le S est complètement relaxé, parce que la contrainte de coût de maintenance devient le facteur dominant.

Dans le deuxième cas, nous avons fixé l'espace de stockage S à 40% et nous avons varié les valeurs de la contrainte de maintenance T de 5% à 100%. La Figure 6.11b montre les résultats de la qualité des solutions lorsque T est varié. Encore une fois, une seule configuration qui est VM-Temps converge vers la meilleure solution, les deux autres ne parviennent pas à converger même quand T est totalement relaxé (c'est à dire, $Q_s \approx 0,9$ pour le temps et $Q_s \approx 1.015$ pour l'énergie), ceci est dû au fait que la contrainte d'espace est un facteur dominant dans cette situation. En outre, les résultats indiquent que les valeurs restrictives des solutions favorisent d'espace de stockage avec un faible temps d'exécution,



(a) L'impact du contrainte d'espace de stockage (T = 20%).

(b) L'impact du contrainte du coût de maintenance (S = 40%).

FIGURE 6.11 – Qualité des solutions en terme de performance et de consommation d'énergie avec différentes combinaisons de contraintes.

mais la consommation d'énergie est importante. De même à la Figure 6.10b, les solutions de compromis changent de façon significative avec la variation du temps de maintenance, cela étaie nos constatations dans l'expérimentation de la Section 6.6.2.5, la meilleure valeur ne peut être vu qu'à T = 25%.

Les résultats présentés ci-dessus montrent clairement l'impact des contraintes d'espace de stockage et de maintenance sur la performance/énergie dans le processus de conception physique de la base de données. Par conséquent, le DBA doit définir les paramètres des contraintes de stockage et de maintenance attentivement pour obtenir un bon compromis entre les deux objectifs ou donner la priorité à des solutions orientées performance ou énergie.

6.6.2.7 Impact des fréquences de mises à jour des vues

Dans les expérimentations précédentes, nous avons supposé que la fréquence de mises à jour des vues f_v était uniforme. Dans cet ensemble d'expérimentations, nous varions les valeurs de f_v de 0,01 à 25 et nous fixons l'espace de stockage S à 40 % afin d'étudier l'impact de la fréquence de mises à jour des vues sur la qualité des solutions. La Figure 6.12 illustre la relation entre la qualité des solutions et la fréquence de mises à jour des vues en terme de performance et de puissance. À partir de cette figure, nous observons que des valeurs faibles de f_v (entre 0,01 et 1) produisent des solutions de bonne qualité pour toutes les configurations de vues matérialisées en terme de performance et de puissance. Cependant, lorsque f_v est élevé (entre 1 et 25), la qualité des solutions des configurations VM-Temps et VM-Compromis est devenue mauvaise, seule la configuration VM-Puissance donne toujours les solutions les plus proches aux meilleurs ensembles. Rappelons que la fréquence de mises à jour a un impact sur le coût de maintenance (c'est-à-dire l'Équation (6.6)) et le coût total de la charge. Par conséquent, des

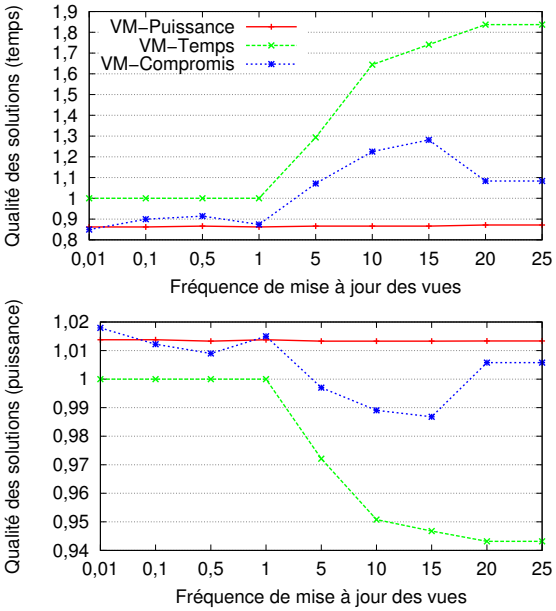


FIGURE 6.12 – Qualité des solutions en terme de performances et de consommation d'énergie avec différentes fréquences de mises à jour ($S = 40\%$).

valeurs élevées de f_v augmentent les coûts des requêtes et conduisent à la réduction des ensembles de solutions satisfaisant les contraintes. De ce fait, on produit des solutions de mauvaise qualité, notamment pour la configuration VM-Temps. A partir des résultats ci-dessus, nous concluons que des valeurs très élevées de fréquence de mises à jour ont un impact majeur sur la qualité des solutions de vues matérialisées. Par conséquent, ce cas devrait être pris en considération par le DBA.

6.6.2.8 Évaluation avec le scénario d'énergie comme contrainte

Supposons un scénario où un fournisseur de services tel qu'un centre de calcul ou un opérateur de Cloud Computing veut optimiser une charge de requêtes, en sélectionnant un ensemble de vues matérialisées qui réduisent la consommation d'énergie, tout en conservant un objectif de performance prédéfini. Cette situation est commune dans un environnement de Cloud où la tendance actuelle est « base de données en tant que de service » (Database as a Service (DBaaS)). Dans telles situations, l'objectif est de répondre à des contrats de service (Service Level Agreement (SLA)) et d'optimiser l'utilisation des ressources telles que la consommation d'énergie, afin de maximiser les marges bénéficiaires. Nous proposons d'utiliser notre méthodologie pour aider les opérateurs à identifier la valeur minimale d'énergie qui assure une performance conforme au SLA établi. Dans ce scénario, la consommation d'énergie devient une contrainte à satisfaire. Le nouveau problème $\mathcal{PSV}$ est formalisé comme suit : pour (i) un schéma de base de données BD ; (ii) une charge de requêtes $\mathcal{W}$; (iii) une contrainte de consommation d'énergie P ; (iv) un $\mathcal{BNF}$ représentant le coût de traitement des requêtes. Le $\mathcal{PSV}$ consiste à sélectionner un ensemble de vues matérialisées $VM = \{V_1, V_2, \cdots, V_m\}$ réduisant le coût de traitement des requêtes et garantissant que la consommation d'énergie ne dépasse pas le seuil P.

Soit P la puissance électrique de la charge de requêtes lorsque tous les vues sont matérialisées moins

Tableau 6.7 – Impact de la contrainte de puissance électrique.

Limite de puissance (%)	Violation de SLA (%)	Économie d'énergie (%)
5	46,59	28,95
10	32,79	27,32
15	27,45	26,58
20	24,91	26,18
25	16,73	24,75
30	11,73	23,77
35	11,73	23,77
40	11,73	23,77
45	0	20,81

la puissance de l'exécution de la charge requêtes sans matérialisation ; P se varie de 5% à 100%. Le SLA est défini comme étant le temps d'exécution de la meilleure solution de compromis que notre AE peut trouver. Pour chaque valeur de P, on définit la violation d'une solution du contrat SLA comme le rapport de la durée d'exécution de cette solution sur le temps d'exécution de la meilleure solution. Notez que nous relaxons les deux autres contraintes. Les résultats de l'expérience sont récapitulés dans le Tableau 6.7. De la table, nous pouvons voir qu'en utilisant notre technique, la violation du SLA descend à 0 lorsque $P = 45\%$. Ainsi, le réglage de la puissance électrique à cette valeur garantit le respect de SLA et produit un bénéfice de 20,8% d'économie d'énergie.

Les résultats confirment l'efficacité de notre méthode pour aider les opérateurs de service à trouver les bons paramètres pour un meilleur compromis entre la performance et l'efficacité énergétique. En outre, il peut être utilisé comme un provisionnement dynamique des ressources par les fournisseurs de services pour allouer dynamiquement un nombre minimal de ressources nécessaires pour satisfaire une qualité de service spécifique. Par la suite, nous allons évaluer les économies d'énergie que l'approche proposée peut atteindre.

6.6.2.9 Économie d'énergie et de puissance électrique

Le but de cette série d'expérimentations est d'étudier les avantages de notre approche en termes d'efficacité énergétique dans le scénario « énergie comme un $\mathcal{BNF}$ » . Nous allons étudier deux cas en ce qui concerne la puissance électrique et l'économie d'énergie : (i) par requête et (ii) par la charge de requêtes. Dans le premier cas, en utilisant notre simulateur, nous exécutons tous les 30 requêtes OLAP du benchmark SSB avec une taille de base de données de 100 Go utilisant les configurations de vues matérialisées VM-Temps et VM-Puissance. Nous collectons les coûts de performances et de consommation d'énergie pour chaque configuration. Nous calculons la puissance/économie d'énergie et de la dégradation des performances de VM-Puissance par rapport à VM-Temps en utilisant les formules suivantes (à savoir, le cas d'économie d'énergie) pour la requête Q_i :

$$EconomieEnergie_i = \frac{Energie_i - Energie_{VM-Temps}}{Energie_{VM-Temps}} \times 100 \tag{6.9}$$

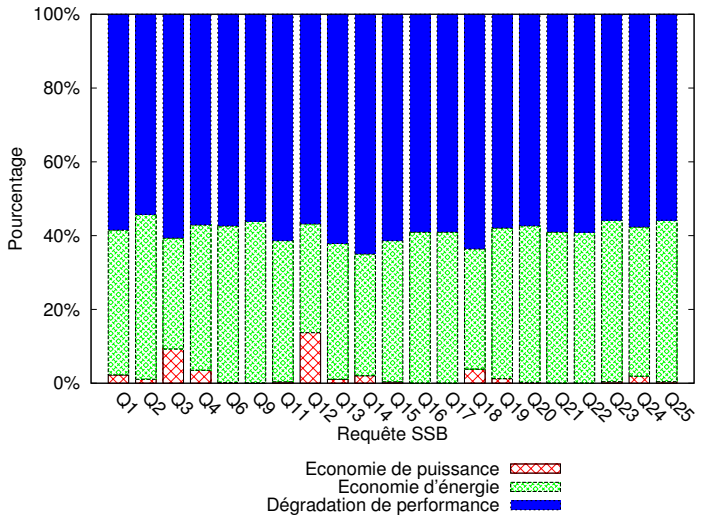


FIGURE 6.13 – Impact des configurations de vues matérialisées sur les performances et la consommation d'énergie des requêtes SSB.

Dans la Figure 6.13, les résultats d'économie d'énergie et la dégradation de performance en pourcentage sont affichés. Comme nous pouvons le remarquer, 21 des 30 requêtes ont le potentiel d'économie d'énergie dans la configuration VM-Puissance. En principe, le bénéfice de la conservation de l'énergie pour ces requêtes à un impact négatif sur le temps de traitement comme indiqué dans la même figure. Cependant, comme nous allons le démontrer, le choix de la configuration de compromis (VM-Compromis) peut conduire à de bonnes valeurs d'économie d'énergie avec moins de dégradation des performances. Ces requêtes sont caractérisées par un nombre important d'opérateurs SQL, diverses opérations d'E/S et de CPU, et partagent beaucoup d'opérateurs SQL communs. Il en résulte un riche $\mathcal{GGR}$ et offre au algorithme évolutionnaire une variété de choix de configurations de vues. Par conséquent, nous pouvons avoir des requêtes offrant une bonne économie d'énergie à partir de ces vues, car les composants les plus intensifs en puissance électrique seront matérialisés. D'autre part, le reste de requêtes qui ne présentent pas de possibilités d'économie d'énergie, sont des requêtes avec un petit nombre d'opérateurs SQL et un ensemble limité de composants partagés. Cela conduit l'algorithme évolutionnaire à choisir les mêmes vues dans les deux configurations VM-Temps et VM-Puissance. Nous concluons qu'il existe des possibilités de conservation de l'énergie au *niveau de requête* en utilisant une technique d'optimisation orientée-énergétique dans la conception physique.

Dans le second cas, nous nous concentrons notre attention sur le *niveau de la charge de requêtes*. Plus précisément, nous créons trois charges de requêtes à partie de SSB : (1) WL30 contient 30 requêtes, (2) WL100 contient 100 requêtes, et (3) WL200 contient 200 requêtes. Pour chaque jeu de requête, nous générons leurs $\mathcal{GGR}$ en utilisant notre simulateur pour obtenir les coûts de performances, de puissance et la consommation d'énergie pour les trois configurations de vues matérialisées (VM-Temps, VM-Puissance, VM-Compromis du Tableau 6.4) avec la relaxation des contraintes. En outre, nous calculons les coûts initiaux de la charge de requêtes sans optimisation afin de les exploiter dans la partie de comparaison. Nous exécutons les expérimentations sur deux tailles de bases de données différentes : 10 Go et 100 Go. Nous calculons l'économie de la puissance/d'énergie et de la dégradation des performances de VM-Puissance et VM-Compromis par rapport à la configuration VM-Temps, en utilisant l'Équation (6.9).

Tableau 6.8 – Économies de puissance/énergie et dégradation de performance dans différentes configurations des vues matérialisées.

Taille BD	Charge Req	Conf MV	Temps (min)	Puissance (W)	Énergie (kJ)	Économie Puissance (%)	Économie Énergie (%)	Deg Perf (%)
10 Go	WL30	Origine	45,83	15,29	42,04	-	-	-
		VM-Temps	7,92	20,91	9,93	-	-	-
		VM-Puissance	29,58	15	26,62	28,26	168,02	273,62
		VM-Compromis	13,69	16,75	13,75	19,9	38,48	72,89
	WL100	Origine	174,42	18,95	198,33	-	-	-
		VM-Temps	87,23	23,82	124,66	-	-	-
		VM-Puissance	222,15	16,87	224,85	29,17	80,38	154,66
		VM-Compromis	138,69	19,41	161,53	18,49	29,59	58,99
	WL200	Origine	350,86	19,03	400,55	-	-	-
		VM-Temps	179,78	23,69	255,51	-	-	-
		VM-Puissance	439,01	17,05	449,04	28,03	75,74	144,2
		VM-Compromis	290,25	19,31	336,22	18,5	31,59	61,45
100 Go	WL30	Origine	676,44	17,45	708,3	-	-	-
		VM-Temps	190,81	25,61	293,19	-	-	-
		VM-Puissance	1536,84	13,55	1249,14	47,1	326,04	705,43
		VM-Compromis	241,17	22,79	329,8	11	12,49	26,39
	WL100	Origine	2225,99	17,27	2306,09	-	-	-
		VM-Temps	582,04	25,96	906,63	-	-	-
		VM-Puissance	3963,11	13,67	3251,25	47,33	258,61	580,9
		VM-Compromis	1487,87	16,8	1499,61	35,3	65,4	155,63
	WL200	Origine	4459,49	17,4	4654,45	-	-	-
		VM-Temps	1231,82	25,63	1894,56	-	-	-
		VM-Puissance	8977,86	13,66	7358,2	46,71	288,39	628,83
		VM-Compromis	3664,77	15,89	3493,13	38,03	84,38	197,51

Dans le Tableau 6.8 nous présentons les résultats des expérimentations (par exemple, l'économie d'énergie de la configuration de VM-Compromis avec une charge de requêtes 30WL et une taille de base de données de 10 Go est : $|\frac{16,75-20,91}{20,91}| \times 100\% = 19,9\%$). Nous pouvons voir clairement que les charges de requêtes consomment beaucoup moins d'énergie lorsque nous choisissons une configuration des vues matérialisées qui favorise des vues de faibles puissances électriques. Quand on compare les configurations puissance seule (VM-Puissance) avec performance seule (VM-Temps) des résultats, nous observons une marge importante des économies d'énergie, allant de 28% à 47%, le gain est remarquable quand nous allons de petite à grande taille de base de données et de charges de requêtes, cela est dû à la diversité des configurations de vues possibles en $\mathcal{GGR}$ d'une grande taille et leur impact sur la performance/puissance des systèmes. Ainsi, les expérimentations montrent des niveaux comparables d'économie d'énergie. Comme on s'y attendait, les économies de la configuration VM-Compromis sont plus petites que celles obtenues par la configuration de puissance seule, mais elle réalise encore 11 à 38% des valeurs d'économie. D'autre part, la configuration de puissance prend plus de temps pour terminer l'exécution de toutes les requêtes, ce qui se traduit par une dégradation notable des performances. Ce résultat n'est pas étonnant puisque, si l'on gagne en puissance, nous perdons automatiquement en performance. Cependant, dans un scénario plus réaliste, un DBA est censé déployer la configuration VM-Compromis, dans laquelle la dégradation des performances est réellement acceptable si on la compare avec le cas d'optimisation (la colonne *Origine* dans le Tableau 6.8). En effet, par rapport au cas origine (aucune optimisation), nous gagnons toujours en puissance et économie d'énergie sans aucune dégradation de performance, en utilisant une configuration de compromis. Nous notons également que l'économie d'énergie totale de 12% à 84% dans le domaine de la recherche éco-énergétique, ce sont des taux intéressants et très prometteurs pour la conception des SGBDs plus efficaces en énergie. En outre, pour la configuration de vues matérialisées en compromis, nous avons proposé un poids de 0,5 pour ω_1 et ω_2 pour les fonctions objectifs de performance et de puissance, mais dans les serveurs de base de données en monde réel, l'administrateur peut ajuster ces valeurs pour obtenir le compromis souhaité entre la conservation d'énergie et la dégradation de performance.

En résumé, les résultats obtenus prouvent que notre approche est valable. Avec notre configuration, nous obtenons des résultats encourageants pour économiser à la fois la puissance électrique et l'énergie. Si l'on considère un serveur de haute performance avec une puissance moyenne de 380 W, et en service continu, le serveur pourrait consommer pendant un an environ 3283 KWh d'énergie [20]. Le coût de l'énergie total pour ce serveur unique serait de 328 dollars US par an (pour 10 centimes par kWh) en ignorant les coûts des systèmes d'infrastructure, d'alimentation et de refroidissement [20]. En utilisant les techniques proposées (12% - 84% d'économie d'énergie réalisée), en peut économiser 39 - 275 dollars US par serveur par an. Ce nombre pourrait augmenter pour les centres de calculs à grande échelle avec des milliers de serveurs.

6.7 Conclusion

Dans ce chapitre, nous avons proposé une initiative visant à intégrer l'énergie dans la phase physique du cycle de vie de la conception de base de données, en considérant le cas des vues matérialisées, une structure d'optimisation redondante. Différents scénarios d'intégration de l'énergie dans le processus de sélection des vues matérialisées sont discutés. Dans cette étude, nous nous sommes concentrés

sur deux scénarios : *l'énergie comme un besoin non-fonctionnel* et *l'énergie comme une contrainte*. Ces deux scénarios sont formalisés et résolus en utilisant des algorithmes évolutionnaires. Pour évaluer la qualité des solutions finales, nous avons développé des modèles de coûts mathématiques pour estimer les objectifs utilisés : consommation d'énergie et performances des requêtes. Des techniques d'apprentissage automatique ont été utilisées pour estimer les paramètres du modèle de coût dédié à la consommation d'énergie et le temps d'exécution. Ces modèles de coûts ont été utilisés pour évaluer notre proposition, et leurs résultats ont été comparés à ceux obtenus par la validation réelle. En outre, nous avons souligné l'importance d'intégrer l'énergie dans la phase de conception physique pour avoir des bases de données éco-énergétiques. L'étude expérimentale confirme notre revendication, car l'approche proposée permet de réduire considérablement la consommation d'énergie globale de la charge de requêtes, en réalisant une *économie de puissance active jusqu'à 38%* et une *économie d'énergie totale jusqu'à 84 %*. En outre, nous avons étudié un autre facteur très important dans les environnements de Cloud actuels, qui visent à réduire la consommation d'énergie tout en respectant un certain objectif de performance.

Quatrième partie

Conclusion et perspectives

Chapitre

7

Conclusion générale et Perspectives

« Research is to see what everybody else has seen, and to think what nobody else has thought. »
— Albert Szent-Gyorgyi

Sommaire

7.1 Conclusion

Nous présentons dans ce chapitre un bilan du travail que nous avons effectué ainsi qu'un ensemble d'ouvertures et de perspectives pour ce travail.

À l'heure actuelle, la plupart des gens sont conscients de la gravité du problème de l'énergie : nos sources d'énergie primaires sont en train de s'écouler, tandis que la demande d'énergie dans les environnements commerciaux et domestiques ne cesse d'augmenter. En outre, les effets secondaires de la consommation d'énergie ont d'importantes considérations environnementales à l'échelle mondiale. L'émission de gaz à effet de serre, comme le CO_2, que la plupart des climatologues considèrent aujourd'hui comme étant liée au réchauffement climatique, n'en est qu'un exemple. Les scientifiques et les politiciens du monde mettent l'accent sur une solution stratégique : Le développement de nouvelles sources d'énergies propres et renouvelables est nécessaire et vital pour surmonter le problème énergétique. Steve Chu, l'ancien secrétaire américain à l'énergie, a placé cette situation dans son contexte [69] : « Une double stratégie est nécessaire pour résoudre le problème énergétique : (1) maximiser l'efficacité énergétique et diminuer la consommation d'énergie ; (2) développer de nouvelles sources d'énergies propres. La stratégie (1) restera la plus simple à réaliser pour les prochaines décennies. »

Le facteur clé à considérer avec les systèmes informatiques d'aujourd'hui est que la quantité de puissance électrique qu'ils consomment ne s'ajuste pas en fonction de la quantité de travail que le système réalise. Jusqu'à présent, le principal objectif de la plupart des systèmes informatiques à usage général était de maximiser les performances sans tenir compte de la consommation d'énergie. Cela évolue rapidement à mesure que nous approchons du point où le coût d'acquérir un équipement informatique sera dépassé par le coût de l'énergie pour le faire fonctionner, sauf si nous accordons une importance à la conception de systèmes qui consomment moins d'énergie. Cependant, la puissance et l'énergie ont commencé à restreindre sérieusement la conception des équipements, des systèmes, des clusters, des centres de données et des applications. Une meilleure conception de l'équipement et de meilleures politiques de gestion de l'énergie sont nécessaires pour répondre à ces préoccupations.

Nous nous sommes intéressés dans cette thèse au problème de l'intégration de l'énergie dans les bases de données avancées afin de minimiser leur consommation. Les BDs sont aux cœur des centres de données, et elles sont responsables de la majeure partie de leurs consommations énergétiques. Notre solution proposée pour diminuer la consommation d'énergie des BDs, est de formaliser le problème comme un problème d'optimisation. Dans cette formalisation, il faut ajuster les ressources logicielles afin de minimiser leurs consommation en gardant un certain niveau de performance désirée. En analysant l'état de l'art et à notre connaissance, nous avons constaté qu'il n'existait pas de démarche pour intégrer l'énergie dans les bases de données. En se basant sur les travaux existants, et d'après notre étude de ce problème, nous avons proposé au cours des chapitres de cette thèse des lignes directrices détaillées qui établissent un référentiel à la communauté des bases de données pour concevoir des BDs vertes. Les points essentiels de cette démarche sont : (1) la proposition des modèles pour quantifier la consommation des taches exécutées par un SGBD, (2) le développement des techniques pour résoudre le problème d'optimisation d'énergie sans sacrifier la performance et (3) l'intégration de ces modèles et techniques dans le fonctionnement interne des BD et SGBD.

7.1.1 État de l'art

Notre travail a été guidé par une étude préalable des travaux existants à fin d'examiner les dimensions principales qu'ils faut prendre en charge lors de l'intégration de l'énergie. Nous avons divisé notre étude sur trois axes : le premier se focalise sur les problèmes de conception et de traitement des bases de données. Nous avons étudié le cycle de vie d'une base de données avec un accent sur les structures d'optimisation dans la phase de conception physique. Nous avons également décrit le processus du traitement de requêtes dans un SGBD relationnel. Le deuxième axe traite le problème d'optimisation multi-objectifs. Nous avons présenté le problème dans sa forme générale avec les techniques de résolution. Ensuite, nous avons étudié les travaux sur les problèmes multi-objectifs dans les bases de données. En fin, le troisième axe concerne l'efficacité énergétique dans les systèmes informatiques. Nous avons passé en revue les travaux visant à minimiser l'énergie au niveau matériel, système d'exploitation et application. Nous avons détaillé les études sur ce dernier qui comprend les bases de données. Nous avons identifié les lacunes des approches existantes et nous avons proposé nos démarches pour les améliorer.

7.1.2 Modèle de coût énergétique

Notre première contribution est la proposition d'un modèle de coût pour estimer la consommation d'énergie lors de l'exécution d'un ensemble de requêtes. Nous avons indiqué comment la modélisation au niveau des pipelines pourrait être un indicateur robuste de la prédiction d'énergie. Pour garantir la portabilité de notre approche, nous avons choisi les coûts d'E/S et CPU des pipelines comme paramètres de notre modèle. Alors qu'un certain nombre d'études récentes ont exploré ce problème, la majorité des travaux existants considère la prédiction de l'énergie pour une seule requête isolée. Dans cette thèse, nous avons également considéré le problème le plus général avec plusieurs requêtes concurrentes. Ceci est utile pour les tâches de la gestion de nombreuses bases de données, y compris le contrôle d'admission, l'ordonnancement des requêtes et le contrôle d'exécution avec l'efficacité énergétique comme un objectif primaire. Le modèle de coût est construit sur la base de résultats empiriques obtenus à partir d'une phase d'apprentissage des charges de requêtes qui ont été soigneusement créés. Les paramètres sont calculés par un algorithme de régression polynomiale multiple. De plus, nous avons effectué des tests sur notre framework avec des bases de données réelles, tel que SSB, TPC-H et TPC-DS, en exécutant un ensemble de requêtes et en comparant leurs coûts d'énergie avec ceux prédits par notre modèle de prédiction. Nos résultats montrent que le modèle peut prédire l'énergie avec une grande précision.

7.1.3 Compromis entre la performance et l'énergie

À fin d'incorporer l'énergie dans les bases de données, il est essentiel de choisir un compromis entre l'efficacité énergétique et la dégradation en performance. Contrairement au cas d'optimisation mono-objectif où la solution optimale est généralement unique, dans notre cas d'optimisation multi-objectif, il existe plutôt un ensemble de solutions de compromis alternatives. En analysant l'état de l'art, la plupart des travaux ignorent ce choix lors de l'intégration de l'implémentation de leurs approches. Dans cette thèse, nous avons proposé et développé des techniques d'optimisation multi-objectifs qui offrent le meilleur compromis entre le temps de réponse aux requêtes et la consommation d'énergie du système. Notamment, nous avons utilisé deux techniques, une classique connue sous le nom de la

méthode sommes pondérées, et la deuxième technique est basée sur les algorithmes évolutionnaires, connue sous le nom de NSGA-II. Ces deux techniques ont été utilisées d'une façon a posteriori, où nous donnons le choix aux administrateurs de bases de données et aux utilisateurs de choisir le meilleur compromis parmi une liste de frontière de Pareto.

7.1.4 L'énergie dans le traitement de requêtes

Jusqu'à présent, la conception du système de base de données s'est concentrée sur l'amélioration de la performance lors de la phase de traitement des requêtes. Nous avons exploré le potentiel de la conservation de l'énergie dans les systèmes de gestion des bases de données relationnelles, en étudiant chaque étape du processus de traitement des requêtes. Notre deuxième contribution consiste en la modification d'optimiseur de requêtes dans un SGBD pour prendre en compte le coût énergétique des plans de requêtes. À cette fin, nous avons pris le cas du SGBD open-source PostgreSQL, son module de traitement de requêtes a été étudié et son code a été modifié pour inclure la dimension d'énergie. Un outil, nommé EnerQuery, a été développé pour permettre aux DBA et aux utilisateurs de diagnostiquer le comportement de la consommation d'énergie des systèmes et des BDs. Cet outil permet aussi de visualiser le plan de requêtes avec les segmentations de pipelines correspondantes et diverses informations de coût pour chaque opérateur, il permet également de définir des valeurs de compromis entre l'énergie et le temps d'exécution. Les résultats d'expérimentations affirment que les techniques proposées peuvent réduire la consommation d'énergie des serveurs de base de données et contrôler les compromis entre la consommation et la performance du système en choisissant des plans d'exécution alternatifs.

7.1.5 L'énergie dans la conception physique des BD

Comme troisième contribution, nous avons proposé une initiative visant à intégrer l'énergie dans la phase de conception physique du cycle de vie des bases de données, en considérant le cas des vues matérialisées - une structure d'optimisation redondante. Avant de présenter notre méthodologie pour résoudre le problème de la sélection des vues matérialisées, nous avons analysé et discuté les différents scénarios d'intégration de l'énergie dans le processus de sélection des vues. Dans cette thèse, nous nous sommes concentrés sur deux scénarios : l'énergie comme un besoin non-fonctionnel et l'énergie comme contrainte. Ces deux scénarios ont été formalisés et résolus à l'aide d'algorithmes évolutionnaires multi-objectifs. Pour évaluer la qualité des solutions finales, nous avons développé des modèles de coûts mathématiques pour estimer les coûts des objectifs de notre problème : la consommation d'énergie et la performance des requêtes. Les résultats d'évaluation de notre proposition ont été confrontés à ceux obtenus par la validation réelle avec un matériel de mesure d'énergie. En outre, nous avons souligné l'importance de la conception physique de la base de données vers les SGBD à haut rendement énergétique. L'étude expérimentale confirme notre affirmation car l'approche proposée réduit de manière significative la consommation d'énergie globale de la charge de requêtes, en permettant des économies de puissance active jusqu'à 38% et des économies d'énergie totale jusqu'à 84%. En outre, nous avons étudié un autre facteur important dans les environnements cloud d'aujourd'hui qui vise à réduire la consommation d'énergie tout en répondant à un seuil de performance sous un contrat de SLA.

7.2 Perspectives

Les travaux initiés dans cette thèse peuvent se poursuivre dans de nombreuses directions. Nous esquissons ici quelques pistes de perspectives à court et à long terme.

7.2.1 Extension du modèle de coût

Nous avons proposé un modèle de coût énergétique basé sur les paramètres d'E/S et de CPU des pipelines et la régression polynomiale. En premier lieu, il serait intéressant d'augmenter le modèle avec d'autres paramètres lié au matériels, tel que les compteurs de performance [74]. Ces derniers donnent l'accès aux différents types d'événements générés sur le processeur et la mémoire, ces événements permettent de comprendre le comportement d'énergie et les types d'actions qui influencent sur la consommation du système. Également, il est important de comprendre comment développer des modèles précis pour les autres composants du système, comme les équipements de réseau et la carte graphique. En deuxième lieu, l'utilisation des techniques d'apprentissage automatique avancées, telles que la méthode d'analyse canonique des corrélations [96], Gradient boosting [166, 148], machines à vecteurs de support [11] et la méthode de lasso [298] est l'une des directions pour des futures recherches. Ces techniques sont capables de donner des résultats d'estimation avec une grande précision, mais la complexité de la modélisation et le temps de calcul sont les principaux inconvénients de ces méthodes.

7.2.2 Généralisation de la formalisation multi-objectifs avec d'autres structures d'optimisation

Dans notre thèse, nous avons pris le problème de sélection des vues matérialisées comme un cas d'étude sur l'intégration de l'énergie dans la phase de conception physique des bases de données. Ainsi, nous avons proposé une formalisation multi-objectifs pour résoudre ce problème. Cette formalisation peut être généralisée pour d'autres structures d'optimisation, comme le problème de sélection d'index binaires, ou le problème de fragmentation, du fait que le processus de recherches des données à partir des indexes binaires peut avoir un impact important sur la consommation d'énergie du système. De plus, notre formalisation peut être combinée avec d'autres besoins non-fonctionnels, tel que le coût monétaire dans les environnements de cloud, on auras en résultats plus de deux $\mathcal{BNF}$. En outre, dans ce cas, nous pouvons exploiter les techniques émergentes des algorithmes évolutionnaire scalables nommés : optimisation « many-objectifs » [50].

7.2.3 Le cas du traitement de requêtes parallèles ou distribuées

Nous avons étudié le problème d'efficacité énergétique dans le contexte de bases de données centralisées. Il serait judicieux d'adapter les approches proposées dans le cadre de base de données parallèles ou distribuées particulièrement pour la phase de traitement de requêtes. Dans ce contexte, il faut adapter, en première étape, le modèle de coût avec les caractéristiques du réseau à fin d'estimer le coût de transfert des données entre les sites. En deuxième étape, il faut développer des approches pour étendre le module de traitement de requêtes parallèles ou distribuées avec la dimension d'énergie, la formalisation du nouveau problème sera : choisir le meilleur plan d'exécution des requêtes qui minimise à la fois les

coûts du temps de réponse, du transfère réseau et de la consommation d'énergie. Pour avoir ce plan, il faut employer des techniques avancées lors de la phase de génération et comparaison des plans, tels que les algorithmes évolutionnaires ou bien les algorithmes aléatoires [258]. De plus, dans le cas des BDs parallèles ou distribuées, nous pouvons exploiter notre modèle de coût pour les requêtes concurrentes comme dans [206], chose qui n'est pas abordée dans cette thèse.

7.2.4 L'ordonnancement de requêtes et la gestion du cache

Une technique intéressante pour minimiser l'énergie dans une base de données est l'ordonnancement de requêtes. Cette technique est très utile dans les environnements de cloud Database-as-a-Service (DBaaS) ou il faut avoir une politique de contrôle d'admission de requêtes [281, 66, 67]. Dans une charge de requêtes, souvent il existe des requêtes qui partagent des sous-expressions et de données communes. Cependant, au lieu d'exécuter les requêtes d'une manière séquentielle ou aléatoire, il faut proposer des méthodes pour planifier l'ordre d'exécution et sauvegarder les résultats intermédiaires des opérations tout en exploitant les parties communes des requêtes, afin de minimiser le temps d'exécution et la dissipation d'énergie. La sauvegarde de résultats intermédiaires est généralement sous la contrainte de la taille de mémoire ou de disque. Ainsi, il existe deux problèmes : d'abord le problème de trouver le meilleur ordre d'évaluation et d'exécution des expressions, appelé le problème *d'ordonnancement des requêtes*, et le problème de décider à quel moment admettre la sauvegarde d'un résultat commun dans la mémoire cache, et quand le retirer, appelé le problème de *gestion du cache*. La méthode courante pour résoudre ces problèmes est la méthode d'optimisation multi-requêtes [230, 106, 83], il faut revoir les algorithmes basés sur cette méthode pour rajouter la dimension d'énergie.

7.2.5 Proposition des approches orientées matériel

Le travail réalisé au cours de cette thèse a principalement consisté à élaborer des techniques d'amélioration d'efficacité énergétique orientées logicielles. Toutefois, la deuxième partie des techniques, orientées matériel, est connue pour avoir des résultats de minimisation d'énergie très intéressants grâce à la propriété d'énergie proportionnel du matériel [242]. Par contre, nous proposons de combiner les deux branches de solutions dans le cas des bases de données. Par exemple, au lieu d'utiliser la technique de DVFS pour changer la fréquence du CPU d'une manière isolée, il serait préférable de développer une méthode qui permet d'ajuster les valeurs de DVFS suivant la charge de travail actuel directement à partir du SGBD. Le cas le plus évident pour introduire cette méthode est le traitement des requêtes, une EE significative peut être achevée si nous pouvons trouver des plans d'exécution qui ne nécessitent pas autant de temps CPU. Pour les équipements de stockage multi-disque, au lieu de se focaliser uniquement sur le problème *d'équilibrage de charge* pour améliorer les performances d'E/S. Cependant dans notre cas, nous proposons de développer une technique de *consolidation de charge* en plaçant les charges d'E/S les plus accédées dans les mêmes portions de disques [243]. De cette façon, des possibilités d'économie d'énergie sont créées car les disques à faible charge d'E/S peuvent basculer vers des modes de faible consommation/performance.

7.2.6 Benchmark pour l'énergie

Dans la phase d'apprentissage lors de la création de notre modèle de coût énergétique, nous avons opté pour crée nos propres charges de requêtes manuellement pour collecter les paramètres d'E/S, de CPU et les mesures d'énergie à fin d'appliquer la méthode de régression. Ce processus était le même pour les travaux précédents connexes [282, 152, 155, 215], car il n'existe pas un benchmark énergétique standard avec des requêtes SQL pour examiner le comportement d'énergie des composantes matériels. L'approche classique est soit de crée des charges de requêtes personnalisées soit d'utiliser des outils séparés (comme Phoronix Test Suite [22]) pour examiner l'ensemble d'équipements matériels. Les deux approches souffrent des problèmes de portabilités entre les systèmes, et ne garantissent pas un bon examen des équipements. Cependant, il serait très bénéfique de développer un benchmark avec des requêtes SQL caractérisées par des opérations gourmandes en CPU, par des opérations exhaustives en terme de ressources de stockage (gourmandes en E/S) et par des requêtes avec un taux élevé de transfère réseau.

22. http://www.phoronix-test-suite.com/

Cinquième partie

Annexes

Annexe

A L'estimation de cardinalités et de coûts d'opérateurs dans l'optimisation des requêtes

Comme nous l'avons décrit précédemment dans la Section 2.2.2.3 du Chapitre 2 (page 40), l'optimiseur de requêtes peut créer une estimation du coût global d'exécution des requêtes à partir de sa connaissance des coûts des opérateurs individuels, des paramètres du système et des distributions de données, ce qui lui permet de choisir le plan de coût minimal estimé à partir de son espace de d'énumération de plan. Naturellement, l'estimation de la cardinalité est l'un des aspects les plus essentiels du processus d'estimation des coûts : l'estimateur prédit la cardinalité du résultat étant donné un ensemble de relations d'entrée (qui peuvent être stockées sur disque ou dérivées par la requête) et une opération. Cette cardinalité estimée fournit alors une prédiction de la taille d'une des entrées à l'opération suivante, et ainsi de suite, jusqu'à la fin de la requête.

Dans cette annexe, nous décrirons le processus d'estimation des coûts dans la phase d'optimisation des requêtes dans un SGBD. Premièrement, nous donnons des fonctions d'estimation de cardinalité des résultats intermédiaires dans le plan de requête physique. Deuxièmement, nous détaillons les formules d'estimation du coût de traitement des requêtes représentant le nombre d'entrées/sorties.

A.1 Estimation de la cardinalité des résultats intermédiaires

L'estimation de cardinalité des résultats intermédiaires dépend du type de l'opérateur encours dans le plan de requête physique. Dans cette section, nous présentons les formules de base utilisées par un SGBD pour estimer la cardinalité des résultats intermédiaires des opérateurs. Notez que l'objectif de l'estimation de cardinalité n'est pas de donner une information exacte, mais plutôt une estimation pour sélectionner un bon plan de requête physique. Pour une relation donnée ou un résultat intermédiaire R, on note sa cardinalité par $|R|$. Les notations utilisées les formules sont démontrées le Tableau A.1. Les formules de coût sont basées sur les travaux de Selinger *et al.* [229].

A.1.1 Cardinalité de la sélection

La sélection s'appliquait à une relation utilisant un ou plusieurs prédicats. Soit p un prédicat de sélection appliqué à une relation R. On note $\sigma_p(R)$ comme résultat de cette sélection. Premièrement, nous avons besoin d'un paramètre supplémentaire appelé *facteur de sélectivité* du prédicat p, qui est défini comme le nombre de tuples satisfaisant le prédicat, et sa valeur est comprise entre 0 et 1. En d'autres termes, c'est la probabilité de choisir chaque tuple à partir de relations à utiliser comme arguments des

Tableau A.1 – Notation pour l'estimation de cardinalité.

Paramètres	**Description**
R, S	Deux relations ou résultat intermédiaire
$\|R\|$	Nombre de tuples de la relation R
$\|\|R\|\|$	Nombre de pages sur lesquelles la relation R est stockée
$\|a\|$	Nombre de valeurs distinctes de l'attribut a
max_a	Valeur maximale pour un attribut a dans la relation R
min_a	Valeur minimale pour un attribut a dans la relation R
$L_a(a_R)$	Longueur moyenne de la valeur de l'attribut a de la relation R (en octets)
$L_t(R)$	Longueur moyenne d'un tuple de relation R (en octets)

Tableau A.2 – Estimation de la sélectivité des prédicats complexes.

Prédicat p	**Sélectivité** f_p
$\neg(p)$	$1 - f_p$
$p_1 \wedge p_2$	$f_{p_1} \times f_{p_2}$
$p_1 \vee p_2$	$f_{p_1} + f_{p_2} - f_{p_1} \times f_{p_2}$
$a = val$	$\frac{1}{\|a\|}$
$a = b$	$\frac{1}{max(\|a\|, \|b\|)}$
$a > val$	$\frac{(max_a - val)}{(max_a - min_a)}$
$a < val$	$\frac{(val - min_a)}{(max_a - min_a)}$
$val_1 \leqslant a \leqslant val_2$	$\frac{(val_2 - val_1)}{(max_a - min_a)}$

opérateurs. Soit f_p le facteur de sélectivité du prédicat p. f_p est défini comme :

$$f_p = \frac{|\sigma_p(R)|}{|R|} \tag{A.1}$$

Par conséquent, la cardinalité du résultat de sélection peut être définie comme suit :

$$|\sigma_p(R)| = f_p \times |R| \tag{A.2}$$

Pour estimer la cardinalité, nous supposons que nous avons des valeurs uniformes dans les colonnes. Un prédicat p peut impliquer plus d'un attribut ou valeur. Ainsi, le prédicat p peut être simple lorsqu'il a une valeur ou complexe lorsqu'il implique des opérateurs *booléens* reliant plusieurs prédicats simples (comme p_1 et p_2). La sélectivité des prédicats complexes peut être calculée facilement en fonction de leurs prédicats simples. Dans la Table Tableau A.2, nous résumons les formules d'estimation de la sélectivité, avec a et b désignent des attributs, val, val_1 et val_2 dénotent des constantes, max_a est la valeur maximale dans la colonne a, $|a|$ est le nombre de valeurs distinctes dans l'attribut a. Notez que l'optimiseur peut utiliser des équivalents logiques pour détecter si la condition est fausse, et dans ce cas la cardinalité est égale à 0.

A.1.2 Cardinalité de projection

La cardinalité du résultat d'une projection sur la relation R est en réalité la cardinalité de R elle-même :

$$|\pi_a(R)| = |R| \tag{A.3}$$

Tel que a, est un attribut ou un ensemble d'attributs. Parfois, l'optimiseur de requête élimine les tuples en double pour améliorer le plan de requête, car lorsque ceux-ci sont utilisés par d'autres opérateurs comme la jointure, le résultat est le même. Dans ce cas, la cardinalité est égale à :

$$|\pi_a(R)| = |a| \tag{A.4}$$

A.1.3 Cardinalité du produit vectoriel

La cardinalité du résultat d'un produit vectoriel de deux relations est la multiplication de leurs cardinalités :

$$|R \times S| = |R| \times |S| \tag{A.5}$$

A.1.4 Cardinalité de jointure

Dans cette discussion, nous considérons seulement la jointure naturelle. La jointure naturelle entre deux relations implique seulement l'égalité des deux attributs. Nous étudions donc la jointure $R(a, b) \bowtie S(b, c)$, oùb est l'attribut du prédicat de jointure, a et c désignent un ensemble d'attributs. La cardinalité du résultat de jointure $|R \bowtie S|$ dépend des valeurs de l'attribut b. Par exemple :

- Si R et S ont des ensembles disjoints de valeurs b, alors $|R \bowtie S| = 0$;
- Si b est la clé primaire de S et la clé étrangère de R, chaque tuple de R joint exactement un tuple de S, et $|R \bowtie S| = |R|$.
- Si tous les tuples de R et S ont la même valeur b, dans ce cas $|R \bowtie S| = |R| \times |S|$.

En général, on suppose que b_R et b_S sont l'attribut b de R et S respectivement, et r est un tuple de R et s de S. Si $|b_R| > |b_S|$, alors la valeur b de s est une valeur qui apparaît dans la valeur b de R. Par conséquent, la probabilité que r et s partagent la même valeur b est $1/|b_R|$. De même, si $|b_R| < |b_S|$, alors la probabilité que r et s partagent la même valeur b est $1/|b_S|$. Dans l'opération de jointure, le nombre de comparaison possible des paires r et s est $|R| \times |S|$, donc la cardinalité du résultat d'une jointure naturelle est :

$$|R \bowtie S| = max(|R|, |S|)/max(|b_R|, |b_S|) \tag{A.6}$$

Le même principe peut s'appliquer pour les attributs de jointure multiples. Nous supposons que B est un ensemble des attributs impliqués dans la jointure, la cardinalité devient :

$$|R \bowtie S| = max(|R|, |S|)/max(|b_{i_R}|, |b_{i_S}|), \forall b_i \in B \tag{A.7}$$

Si l'on suppose que J_{RS} est la probabilité que deux lignes vérifient les conditions de jointure, la cardinalité du résultat de cette jointure peut être calculée comme suit :

$$|R \bowtie S| = max(|R|, |S|) \times J_{RS} \tag{A.8}$$

A.1.5 Cardinalité de l'agrégation

Le nombre de tuples générés à partir d'une agrégation correspond au nombre des groupes résultants. Ce dernier peut être compris entre 1 et $|R|$, donc la cardinalité du résultat de l'agrégation est égale à :

$$|\gamma(R)| = \frac{|R|}{2} \tag{A.9}$$

A.1.6 Cardinalité de l'union

Le nombre maximal de tuples généré par l'union de deux relations R et S ($R \cup S$) est la somme de leurs cardinalités $|R|$ et $|S|$, et le minimum est le maximum entre $|R|$ et $|S|$. Ainsi, la cardinalité du l'union peut être la moyenne des valeurs maximales et minimales.

$$|R \cup S| = max(|R|, |S|) + \frac{min(|R|, |S|)}{2} \tag{A.10}$$

A.1.7 Cardinalité de l'intersection

La cardinalité du résultat de l'intersection de deux relations R et S ($R \cap S$) est comprise entre 0 et la cardinalité minimale des deux relations. Ainsi, la cardinalité peut être considérée comme la valeur moyenne :

$$|R \cap S| = \frac{min(|R|, |S|)}{2} \tag{A.11}$$

A.1.8 Cardinalité de la différence

Le maximum de tuples généré à partir de la différence de deux relations R et S ($R-S$), est la cardinalité de R ($|R|$) et le minimum est $|R| - |S|$. Ainsi, la cardinalité de la différence peut être considérée comme la moyenne :

$$|R - S| = \frac{(|R| - |S|)}{2} \tag{A.12}$$

A.2 Modèles de coût des opérateurs physique

Comme il a été mentionné précédemment dans le Paragraphe 2.2.2.3.2 du Chapitre 2 (page 41), le coût d'exécution d'une requête peut être représenté par plusieurs paramètres ($C_{\mathcal{E}/S}$, $C_{\mathcal{CPU}}$, $C_{\mathcal{COM}}$, etc.). Pour les grandes bases de données, l'accent est souvent mis sur la minimisation des coûts d'accès au stockage secondaire. Les fonctions de coûts ignorent d'autres facteurs et comparent les plans d'exécution de requêtes en termes de nombre de transferts de blocs entre le disque et les tampons mémoire. Pour les bases de données plus petites, où la plupart des données impliqués dans la requête peuvent être stockées en mémoire, l'accent est mis sur la minimisation des coûts de calcul du processeur. Dans les bases de données distribuées, où de nombreux sites sont impliqués, les coûts de communication doivent être minimisés. Il est difficile d'inclure tous les paramètres du coût dans une fonction de coût (pondérée) en raison de la difficulté d'attribuer des pondérations appropriées aux paramètres de coût.

Tableau A.3 – Paramètres des fonctions de coûts.

Paramètres	Description
PS	Taille de la page
t_{disque}	Temps de transfert du disque d'une page
M	Le nombre de blocs dans le tampon de mémoire principale
$\|\|R\|\|$	Taille sur les pages de la relation R
$\|R\|$	Cardinalité de la relation R (nombre de tuples)
$L_t(R)$	Longueur de tuple pour la relation R
$L_a(C_R)$	La longueur moyenne de la colonne C dans la relation R
$\|R_C\|$	Nombre de valeurs distinctes de la colonne C dans la relation R
σ_R	facteur de sélectivité pour la sélection f sur la relation R
π_R	Facteur de sélectivité pour la projection π sur la relation R
$\bowtie_{RS}$	Facteur de sélectivité pour la jonction J sur les relations R et S

C'est pourquoi certaines fonctions de coût considèrent un seul facteur d'accès au disque. Cependant, dans cette discussion, nous supposons que le coût qui domine l'exécution d'une requête est le $C_{\mathcal{E}/\mathcal{S}}$.

A.2.1 Préliminaires

Nous supposons que chaque opérateur est responsable de transmettre son résultat à l'opérateur suivant via la mémoire principale. Pour le modèle de coût, nous considérons certains paramètres, comme le montre le Tableau A.3. Le coût de traitement concerne la mesure du temps d'exécution, qui est directement influencée par le nombre de pages d'E/S. Ainsi, nous représentons le $C_{\mathcal{E}/\mathcal{S}}$ de tout opérateur (op) soit par numéro de pages chargées soit par le nombre secondes écoulé, comme suit :

$$C_{\mathcal{E}/\mathcal{S}}(op) = \begin{cases} P_0 & \text{, si le coût est représenté par le nombre de pages d'E/S} \\ T_0 & \text{, si le coût est représenté par le temps écoulé.} \end{cases} \tag{A.13}$$

Où P_{op} est le nombre de pages lues ou écrites pendant l'exécution de l'opération op, et T_{op} est le temps écoulé pendant l'exécution de l'opérateur op. P_{op} et T_{op} sont définis comme :

$$P_{op} = P_{entre} + P_{interm} + P_{sortie} \tag{A.14}$$

$$T_{op} = T_{entre} + T_{interm} + T_{sortie} \tag{A.15}$$

Où P_{entre} et T_{entre} sont le nombre de pages et le temps consommé dans la lecture de(des) relation(s) d'entrée, respectivement. P_{sortie} et T_{sortie} sont respectivement le nombre de pages et le temps consommé dans la transmission de la relation de sortie à l'opérateur père suivant ou au disque, en supposant que le flux de sortie final est écrit sur le disque pour une utilisation ultérieure. P_{interm} et T_{interm} sont le nombre de pages et le temps consommé dans le stockage des résultats intermédiaires sur le disque. Pour certains opérateurs, P_{interm} et T_{interm} sont nuls. T_{op} peut être directement estimé à partir de P_{op} en utilisant des paramètres de la base de données pour définir le temps moyen pour charger des pages à partir du réseau ou du disque. Sachant que les opérateurs unaires ont besoin d'une relation à lire comme

entrée à la différence des opérateurs binaires, qui ont besoin de deux relations ; Le coût d'entrée d'un opérateur est défini comme suit :

$$T_{entre} = \begin{cases} P_{entre} \times t_{disque} & \text{, si un opérateur unaire} \\ P_{entre_R} \times t_{disque} + P_{entre_S} & \text{, si un opérateur binaire sans pipelines} \\ max(P_{entre_R} \times t_{disque}, P_{entre_S} \times t_{disque}) & \text{, si un opérateur binaire utilisant les pipelines.} \end{cases} \tag{A.16}$$

Les coûts intermédiaires et de sortie restent les mêmes pour les opérateurs unaires ou binaires :

$$T_{sortie} = P_{sortie} \times t_{disque} \tag{A.17}$$

$$T_{interm} = P_{interm} \times t_{disque} \tag{A.18}$$

Suivant, nous donnons le coût d'E/S pour chaque type d'opérateur.

A.2.2 Coût de la sélection

Une sélection sur une relation R réduit la taille de R : horizontalement par un facteur de sélectivité σ_R et verticalement par un facteur de filtrage $\varphi_{R,q}$ (seuls les attributs q sont retenus). Nous définissons φ_q^R comme le rapport de la taille de q colonnes d'un tuple dans la relation R, φ_q^R peut être défini comme :

$$\varphi_q^R = \sum_{c \in q} L_a(R_c)/L_t(R) \tag{A.19}$$

Le coût de sortie d'une opération de sélection est défini comme suit :

$$P_{sortie} = \sigma_R \times ||R|| \times \varphi_q^R \tag{A.20}$$

Pour le coût d'entrée, tous les tuples de R sont scannés, sauf si R est trié par des attributs de restriction, alors seulement les tuples satisfaisant la condition de sélection seront analysés. Ainsi, le coût d'entrée peut être calculé comme suit :

$$P_{entre} = \begin{cases} ||R|| & \text{, si } R \text{ n'est pas trié} \\ \sigma_R \times ||R|| & \text{, si } R \text{ est trié.} \end{cases} \tag{A.21}$$

L'opérateur de sélection ne produit aucun résultat intermédiaire ($P_{interm} = 0$).

A.2.3 Coût de la projection

Dans une projection, il n'y a pas d'attributs redondants à supprimer, c'est-à-dire que l'entrée est exactement les attributs à transmettre à l'opérateur parent.

$$P_{sortie} = \pi_R \times ||R|| \tag{A.22}$$

Si la projection doit trier son entrée, le P_{interm} devient :

$$P_{interm} = \begin{cases} 0 & \text{, si } P_R \leqslant M-1 \text{ ou aucun tri n'est effectué} \\ P_R \times log_{M-1}(P_R) & \text{, sinon.} \end{cases} \tag{A.23}$$

A.2.4 Coût de la jointure

Nous considérons l'opérateur de jointure classique qui est appliqué sur deux relations R an S et produit une relation de sortie RS équivalente à :

$$P_{sortie} = \bowtie_{RS} \times PS \times \frac{L_t(RS)}{L_t(R) \times L_t(S)} \times ||R|| \times ||S|| \tag{A.24}$$

A.2.4.1 Jointure par boucle imbriquée

Dans l'algorithme de la boucle imbriquée, la relation externe doit être la plus petite, afin de réduire le nombre d'itérations. Cependant, si une seule relation entre dans la mémoire principale, elle est utilisée comme relation interne pour éviter les accès répétés au disque. Soit R et S les relations impliquées dans l'opérateur de jointure, où R est la relation extérieure. Donc, $||S|| \leqslant M \leqslant ||R||$ Comme nous l'avons montré précédemment, P_{entre} est défini comme :

$$P_{entre} = \begin{cases} ||R|| + ||S|| & \text{, si sans pipeline} \\ max(||R||, ||S||) & \text{, sinon.} \end{cases} \tag{A.25}$$

Pour le coût intermédiaire, il diffère si la relation interne (R) peut être allouée en mémoire ou non. Si c'est le cas, le coût de la jointure est égal au coût du traitement du CPU. Sinon, R doit être stockée sur le disque et être lue à plusieurs reprises pour chaque tuple de la relation externe (S). Ainsi, le P_{interm} peut être défini comme :

$$P_{interm} = \begin{cases} 0 & \text{, si } ||R|| \leqslant (M - ||S||) \\ |R| \times ||S|| & \text{, sinon.} \end{cases} \tag{A.26}$$

Mais si la relation interne est déjà triée, la comparaison ne sera répétée que pour les tuples satisfaisant le prédicat de condition.

A.2.4.2 Jointure par fusion

L'algorithme de jointure par fusion est utilisé uniquement lorsque les deux relations sont triées sur l'attribut de jointure. Les relations sont retrouvées en parallèle. Ainsi, P_{interm} est nul, et P_{entre} est calculé comme suit :

$$P_{entre} = max(||R||, ||S||) \tag{A.27}$$

A.2.4.3 Jointure par hachage

Nous supposons que R est la relation extérieure. Ainsi, la table de hachage est construite pour S et $||S|| \leqslant ||R||$. Le coût de création de la table de hachage h_{constr} est le coût de chargement de S et de l'écriture de la table de hachage sur le disque. Ensuite h_{constr} et R sont joints suivant une jointure par boucle imbriquée de sorte que h_{constr} est la relation interne. Si l'on suppose que $P_{\bowtie_S}$ est le pourcentage de tuples de S qui dépend du facteur de sélectivité de jointure et de la fonction de hachage, alors le coût d'entrée P_{entre} et intermédiaire P_{interm} seront défini comme suit :

$$||h_{constr}|| = P_{\bowtie_S} \times ||S|| \tag{A.28}$$

$$P_{entre} = \begin{cases} ||R|| + ||S|| + ||h_{constr}|| & \text{, si aucun pipeline} \\ max(||R||, ||S||) + ||h_{constr}|| & \text{, sinon.} \end{cases} \tag{A.29}$$

$$P_{interm} = \begin{cases} 0 & \text{, si } ||h_{constr}|| \leqslant (M - ||R||) \\ |R| \times ||h_{constr}|| & \text{, sinon.} \end{cases} \tag{A.30}$$

A.2.5 Coût de tri

Lors du tri de relations qui peuvent être allouées entièrement en mémoire principale, des techniques de tri standard telles que le tri rapide peuvent être utilisées. Le tri des relations qui ne peuvent pas être allouées en mémoire est appelé *tri externe*. La technique la plus couramment utilisée pour le tri externe est l'algorithme de tri-fusion externe. L'idée principale de cet algorithme est d'apporter des portions de la relation (R) dans la mémoire principale, de les trier, puis de retourner le résultat sur le disque. Cela crée des segments de fichiers triés, qui doivent être fusionnés afin de créer un seul fichier trié. La première étape lit chaque bloc de la relation et les écrit à nouveau.

$$P_{entre} = 2 \times ||R|| \tag{A.31}$$

Le nombre initial d'exécutions (segment de fichier trié produit par une itération) est $\lceil ||R||/M \rceil$. Puisque le nombre d'exécutions diminue d'un facteur $M - 1$ dans chaque passe de fusion, le nombre total de passes de fusion requises est :

$$P_{interm} = \lceil log_{M-1}(||R||/M) \rceil \tag{A.32}$$

Chacune de ces passes lit chaque bloc de la relation une fois et l'écrit une fois, donc le P_{sortie} est :

$$P_{sortie} = \lceil log_{M-1}(||R||/M) \rceil \tag{A.33}$$

A.2.6 Coût des opérations ensemblistes

Nous pouvons implémenter les opérations union, intersection et différence en triant d'abord les deux relations, puis en balayant une fois par chacune des relations triées pour produire le résultat. Pour $R \cup S$, lorsqu'un balayage simultané des deux relations révèle le même tuple dans les deux fichiers, seul

l'un des tuples est conservé. Le résultat de $R \cap S$ ne contiendra que les tuples qui apparaissent dans les deux relations. Nous appliquons une différence des relations, $R - S$, de la même manière, en retenant des tuples en R seulement s'ils sont absents dans S.

Pour toutes ces opérations, un seul balayage des deux relations d'entrée triées est nécessaire, donc le coût est $||R|| + ||S||$ transferts de blocs si les relations sont triées dans le même ordre. Si non, le coût du tri doit être inclus.

A.2.7 Coût d'agrégation

Les algorithmes permettant de calculer les agrégats sont basés soit sur le tri, soit sur une combinaison du hachage et du tri. Un tri sur les attributs de groupement (argument du `GROUP BY` de SQL) permet de créer des segments correspondant à chaque valeur de ces attributs. Pour chaque segment, on applique les fonctions d'agrégat (`COUNT`, `SUM`, `MIN`, `MAX`, etc.) demandées. Un hachage préalable sur l'attribut de groupement permet de ramener le problème à un calcul d'agrégat dans chaque partition de hachage. L'approche est donc ici analogue à celle de la jointure par hachage et tri présentée ci-dessus.

Annexe

B Bornes de confiance de modèle de coût énergétique

Dans cette annexe, nous allons étudier les bornes de confiance dans les données et les résultats de notre modèle de coût, en utilisant les données de configuration d'expérimentation haut de gamme introduite dans le Chapitre 5 (page 125). Pour trouver les bornes inférieures et supérieures de la population, nous utilisons l'inégalité de Chebyshev (6). L'inégalité est basé sur la moyenne de la population (notée μ) et l'écart type de la population (notée σ). Malheureusement, μ et σ sont des paramètres inconnus. Par conséquent, nous devons trouver leurs limites supérieures et inférieures avec un certain degré de confiance pour calculer l'inégalité de Chebyshev. Pour le faire, nous (a) testons ci-dessus si les échantillons proviennent d'une population qui suit la distribution normale ; (b) trouvons les bornes inférieures et supérieures de la moyenne de population, avec le degré de confiance étant de 99% ; (c) trouvons les bornes inférieures et supérieures de l'écart-type de la population, avec le degré de confiance étant de 99%.

B.1 Testons si la population suit la distribution normale

L'échantillon moyen est égal à 116,4554, qui est désignée par $\bar{x}$; tandis que l'écart type d'échantillon est égal à 2,1822, qui est désignée par s. Le nombre d'échantillons est égal à 131 et est notée n. Nous effectuons des tests d'hypothèses pour déterminer si les échantillons proviennent d'une population normalement distribuée ou non. Le test d'hypothèse est effectuée en appliquant le test du chi carré pour la distribution normale. L'hypothèse nulle (H_0) est défini comme "La distribution de probabilité de la population est normal". À l'autre extrême, l'hypothèse alternative (H_a) est défini comme "La distribution de probabilité de la population n'est **pas** normale".

Nous divisons d'abord la distribution standard normale $N(0, 1)$ dans un ensemble A_1 contenant huit parties proportionnellement égales, chaque partie étant égale à 1/8 ; puis nous trouvons un ensemble B_1 de contenant huit parties dans une correspondance un-à-un avec ceux appartenant à A_1 tel que chaque échantillon est affecté sur l'une des parties appartenant à B_1. Nous trouvons le point de distribution normale, ce qui équivaut à 1,15, le plus à droite divisée en soustrayant 1/8 = 0,125 de 1, puis regardant dans la table de distribution normale. En suivant la procédure ci-dessus, nous obtenons l'ensemble suivant (nommé A_2) des points de division -1.15, -0,675, -0.32, 0, 0,32, 0,675, 1,15 représentant les valeurs Z. Selon les points de division ci-dessus, on trouve sept nouveaux points intermédiaires en correspondance un-à-un avec les précédentes. Plus précisément, nous appliquons l'Équation (B.1) pour chaque point de A_2, ce qui donne l'ensemble suivant (nommé B_2) des points de division 113,946, 114,9825, 115,7572, 116,4554, 117,1537, 117,9284, 118,9649.

Tableau B.1 – Information sur les calculs.

Intervalle	**Points observés** (O)	**Points estimés** (E)	$(O-E)$	$(O-E)^2/E$
≤ 113,946	15	16,37	-1,37	0,114655
(113,946, 114,9825]	18	16,37	1,63	0,162303
(114,9825, 115,7572]	18	16,37	1,63	0,162303
(115,7572, 116,4554]	19	16,37	2,63	0,422535
(116,4554, 117,1537]	18	16,37	1,63	0,162303
(117,1537, 117,9284]	6	16,37	-10,37	6,569145
(117,9284, 118,9649]	20	16,37	3,63	0,804942
≥ 118,9649	17	16,37	0,63	0,024246

En prenant en considération ce qui précède, on peut formuler l'hypothèse nulle et l'hypothèse alternative comme suit :

— H_0 : Toutes les parties de B_1 sont proportionnellement égale.

— H_a : Au moins l'une des parties de B_1 n'est pas proportionnellement égale avec le reste des parties.

$$Z = \frac{x - \bar{x}}{s} \Leftrightarrow x = \bar{x} + Z \times s \tag{B.1}$$

$$\tilde{X^2} = \sum_{i=1}^{8} \frac{(O_i - E_i)^2}{E_i} \tag{B.2}$$

Ensuite, nous trouvons le nombre de points (estimés) appartiennent à chacune des parties de A_1. Pour la première partie de A_1, nous constatons qu'il contient $np_1 = 131 \times 1/8 = 16,375$ points (estimés), avec p_1 représentant la probabilité de la première partie. Puisque toutes les parties ont la même probabilité, nous concluons que chaque partie contient 16,375 points (estimés). Les points observés appartenant aux parties de B_1 sont trouvés comme suit. Chaque point est inférieure ou égale au point le plus à gauche de B_2 appartient à la première partie de B_1. Les points qui sont supérieurs que le point le plus à gauche de B_2 et inférieur ou égal au second point le plus à gauche de B_2 appartiennent à la deuxième partie de B_1. Le processus d'affectation se déroule d'une manière similaire pour le reste des parties. Le Tableau B.1 contient les informations ci-dessus ainsi que des informations pour le calcul de l'Équation (B.2) qui se rapproche asymptotiquement chi carré de distribution X^2.

La région critique de l'hypothèse nulle représente la région que l'hypothèse nulle est rejetée. Pour calculer cette région, nous avons d'abord besoin de choisir (a) le niveau de signification a, ce qui est normalement comprise entre 0,5% et 5% ; et (b) les degrés de liberté $df = k - m - 1$, avec k et m dénotant le nombre de groupes et le nombre de paramètres du modèle, respectivement. Dans notre cas, nous choisissons $a = 0,05$; le nombre de groupes est égal à huit ; tandis que le nombre de paramètres du modèle est égal à deux (moyenne et écart-type). Par conséquent, $df = 8 - 2 - 1 = 5$. La région critique est la région au-delà de $X^2_{0,5} = 11.07$, qui est désignée en tant que valeur critique et calculée à partir de la table de distribution chi carré. Selon l'Équation (B.2) et le Tableau B.1, nous observons que $\tilde{X^2} = 8,42$, ce qui est inférieur à la valeur critique 11,07. En conséquence, l'hypothèse nulle n'est pas rejetée et nous pouvons supposer que la population suit la distribution normale. Notez que plus que $\tilde{X^2}$ est élevé, plus est la preuve pour rejeter l'hypothèse nulle.

B.2 Trouvons les bornes inférieures et supérieures de la moyenne de population avec 99% de confiance

La moyenne de population est notée μ, tandis que l'écart-type de la population est définie comme σ. Nous trouvons les bornes inférieures et supérieures (μ_l et μ_u) de la population signifient moins de $P\%$ probabilité ou équivalente $P\%$ degré de confiance. Nous devons d'abord préciser le niveau de signification $a = (100 - P)/100$. On notera que plus le degré de confiance est élevé, plus l'intervalle entre les bornes inférieures et supérieures est grand. Pour notre problème, nous allons calculer les bornes inférieures et supérieures de la population signifient avec 99% degré de confiance, ce qui est une valeur commune. Par conséquent, le niveau de signification est de $a = (100 - 99)/100 = 0,01$. Puisque la population suit la distribution normale et l'écart σ n'est pas connue, nous utilisons l'Équation (B.3) pour calculer les limites de la moyenne de la population. Notez que $t_{a/2}$ représente la distribution t de puissance, avec a indiquant le niveau de signification. Étant donné que les degrés de liberté df égale $n - 1$, on calcule $t_{0,005} = 2,58$. Par conséquent, selon l'Équation (B.3), nous pouvons observer que, avec 99% degré de confiance, $\mu_l = 115,96$, et $\mu_u = 116,94$.

$$\bar{x} \pm t_{a/2} \times \frac{s}{\sqrt{n}} \tag{B.3}$$

B.3 Trouvons les bornes inférieures et supérieures de l'écart type de population avec 99% de confiance

Puisque nous exigeons 99% de confiance, le niveau de signification est $a = 0,01$. Les bornes inférieures et supérieures de la population écart-type sont exprimés par l'Équation (B.4) et l'Équation (B.5), respectivement. En regardant dans le tableau de distribution chi carré nous observons que $X^2_{0,005} = 175,3$ et $X^2_{0.995} = 92,2$ (notez que df est égal à $n - 1$). Par conséquent, $\sigma_l = 1,88$ et $\sigma_u = 2,59$.

$$\sigma_l = s \times \sqrt{\frac{n-1}{X^2_{a/2}}} \tag{B.4}$$

$$\sigma_u = s \times \sqrt{\frac{n-1}{X^2_{1-a/2}}} \tag{B.5}$$

B.4 Trouvons avec la probabilité les bornes inférieures et supérieures de la population

De l'inégalité de Chebyshev (Équation (B.6)) nous pouvons trouver avec la probabilité les bornes pour la population. Plus précisément, pour trouver la limite inférieure de la population, nous supposons que $X - \mu \leq 0$, $\mu = \mu_l$, et $\sigma = \sigma_u$. Selon le ci-dessus, l'inégalité de Équation (B.6) devient l'inégalité de l'Équation (B.7). Pour $k = 3$, nous avons ce que $Pr(X \leq 108,19) \leq 0,11$. Pour trouver la limite supérieure

de la population, nous supposons que $X - \mu \geq 0$, $\mu = \mu_u$, et $\sigma = \sigma_u$. Selon le ci-dessus, l'inégalité de l'Équation (B.6) devient inégalité de l'Équation (B.8). Pour $k = 3$, nous avons $Pr(X \geq 124,71) \leq 0,11$. Pour trouver le niveau de confiance pour la limite inférieure de la population, il faut multiplier la probabilité de X pour être supérieure à 108,19 (c-à-d $: 1 - 0,11 = 0,89$) par (a) le niveau de confiance pour la limite inférieure de la population moyenne et (b) par le niveau de confiance pour la limite supérieure de la norme de la population type. En conséquence, la limite inférieure de la population est égale à 108,9 avec 87% ($0.99 \times 0.99 \times 0.89$) degré de confiance. En opérant de manière analogue, la limite supérieure de la population est égale à 124,71 avec un degré de confiance de 87%. Notez que nous pouvons augmenter le degré de confiance (en augmentant k) au coût de la diminution/augmentation de la partie inférieure/supérieure de la population.

$$Pr(|X - \mu| \geq k\sigma) \leq 1/k^2 \tag{B.6}$$

$$Pr(X \leq \mu_l - k\sigma_u) \leq 1/k^2 \tag{B.7}$$

$$Pr(X \geq \mu_u + k\sigma_u) \leq 1/k^2 \tag{B.8}$$

Annexe

C Requêtes d'apprentissage du modèle de coût

Les requêtes suivantes sont les requêtes utilisées pour construire notre modèle de coût énergétique lors de la phase d'apprentissage. Il sont crées à partir du schéma de benchmark TPC-H [5] avec une taille de 10 Go.

```
-- Q1
SELECT SUM(L_QUANTITY), AVG(L_QUANTITY), SUM(L_QUANTITY - L_EXTENDEDPRICE),
       SUM(L_QUANTITY * L_EXTENDEDPRICE - 1), AVG(L_DISCOUNT * (L_TAX +1)),
       AVG(L_QUANTITY / (L_TAX +1)), SUM(L_TAX)/SUM(L_EXTENDEDPRICE), SUM(
       L_QUANTITY * 0.5), AVG(L_QUANTITY * 0.2), SUM(O_TOTALPRICE), AVG(
       O_SHIPPRIORITY), SUM(O_TOTALPRICE - L_EXTENDEDPRICE), AVG(
       O_SHIPPRIORITY * L_EXTENDEDPRICE - 1), SUM(O_SHIPPRIORITY)/SUM(
       O_TOTALPRICE), COUNT(*)
FROM LINEITEM, ORDERS
WHERE L_SUPPKEY <= 100000 AND L_SUPPKEY >= 5000 AND O_ORDERSTATUS = 'F' AND
       L_ORDERKEY = O_ORDERKEY;

-- Q2
SELECT L_SHIPMODE, L_SHIPINSTRUCT, O_ORDERSTATUS
FROM LINEITEM, ORDERS
WHERE L_SUPPKEY <= 100000 AND L_SUPPKEY >= 5000 AND O_ORDERSTATUS = 'F' AND
       L_ORDERKEY = O_ORDERKEY
GROUP BY L_SHIPMODE, L_SHIPINSTRUCT, O_ORDERSTATUS;

-- Q3
SELECT COUNT(*)
FROM LINEITEM, ORDERS
WHERE L_SUPPKEY <= 100000 AND L_SUPPKEY >= 5000 AND L_ORDERKEY = O_ORDERKEY;

-- Q4
SELECT L_SHIPMODE, L_SHIPINSTRUCT, O_ORDERSTATUS
FROM LINEITEM, ORDERS
WHERE L_SUPPKEY <= 1000 AND L_ORDERKEY = O_ORDERKEY
GROUP BY L_SHIPMODE, L_SHIPINSTRUCT, O_ORDERSTATUS;

-- Q5
SELECT L_SHIPMODE, L_SHIPINSTRUCT, O_ORDERSTATUS, COUNT(*)
FROM LINEITEM, ORDERS
WHERE L_SUPPKEY <= 10000 AND L_SUPPKEY >= 500 AND O_ORDERSTATUS = 'F' AND
       L_ORDERKEY = O_ORDERKEY
GROUP BY L_SHIPMODE, L_SHIPINSTRUCT, O_ORDERSTATUS
ORDER BY L_SHIPMODE, L_SHIPINSTRUCT, O_ORDERSTATUS;

-- Q6
SELECT L_SHIPMODE, L_SHIPINSTRUCT, O_ORDERSTATUS, SUM(L_QUANTITY), AVG(
       L_QUANTITY), SUM(O_TOTALPRICE), COUNT(*)
FROM LINEITEM, ORDERS
WHERE L_SUPPKEY <= 1000 AND L_SUPPKEY >= 500 AND L_ORDERKEY = O_ORDERKEY
GROUP BY L_SHIPMODE, L_SHIPINSTRUCT, O_ORDERSTATUS
ORDER BY L_SHIPMODE, L_SHIPINSTRUCT, O_ORDERSTATUS;

-- Q7
SELECT L_SHIPMODE, L_SHIPINSTRUCT, O_ORDERSTATUS, SUM(L_QUANTITY), AVG(
       L_QUANTITY), SUM(O_TOTALPRICE), COUNT(*)
FROM LINEITEM, ORDERS
WHERE L_SUPPKEY <= 100000 AND L_SUPPKEY >= 5000 AND O_ORDERSTATUS = 'F' AND
       L_ORDERKEY = O_ORDERKEY
GROUP BY L_SHIPMODE, L_SHIPINSTRUCT, O_ORDERSTATUS
ORDER BY L_SHIPMODE, L_SHIPINSTRUCT, O_ORDERSTATUS;

-- Q8
SELECT L_SHIPMODE, L_SHIPINSTRUCT, O_ORDERSTATUS, O_ORDERPRIORITY, SUM(
       L_QUANTITY), AVG(L_QUANTITY), SUM(L_QUANTITY - L_EXTENDEDPRICE), SUM(
       L_QUANTITY * L_EXTENDEDPRICE - 1), SUM(O_TOTALPRICE), AVG(
       O_SHIPPRIORITY), COUNT(*)
FROM LINEITEM, ORDERS
WHERE L_SUPPKEY <= 1000 AND L_SUPPKEY >= 500 AND L_ORDERKEY = O_ORDERKEY
GROUP BY L_SHIPMODE, L_SHIPINSTRUCT, O_ORDERSTATUS, O_ORDERPRIORITY
ORDER BY L_SHIPMODE, L_SHIPINSTRUCT, O_ORDERSTATUS, O_ORDERPRIORITY;

-- Q9
SELECT L_SHIPMODE, L_SHIPINSTRUCT, O_ORDERSTATUS, O_ORDERPRIORITY, SUM(
       L_QUANTITY), AVG(L_QUANTITY), SUM(L_QUANTITY - L_EXTENDEDPRICE), SUM(
       L_QUANTITY * L_EXTENDEDPRICE - 1), SUM(O_TOTALPRICE), AVG(
       O_SHIPPRIORITY), COUNT(*)
FROM LINEITEM, ORDERS
WHERE L_SUPPKEY <= 100000 AND L_SUPPKEY >= 5000 AND O_ORDERSTATUS = 'P' AND
       L_ORDERKEY = O_ORDERKEY
GROUP BY L_SHIPMODE, L_SHIPINSTRUCT, O_ORDERSTATUS, O_ORDERPRIORITY
ORDER BY L_SHIPMODE, L_SHIPINSTRUCT, O_ORDERSTATUS, O_ORDERPRIORITY;

-- Q10
SELECT L_SHIPMODE, L_SHIPINSTRUCT, O_ORDERSTATUS, O_ORDERPRIORITY, SUM(
       L_QUANTITY), AVG(L_QUANTITY), SUM(L_QUANTITY - L_EXTENDEDPRICE), SUM(
       L_QUANTITY * L_EXTENDEDPRICE - 1), AVG(L_DISCOUNT * (L_TAX +1)), AVG(
       L_QUANTITY / (L_TAX +1)), SUM(O_TOTALPRICE), AVG(O_SHIPPRIORITY), SUM
       (O_TOTALPRICE - L_EXTENDEDPRICE), AVG(O_SHIPPRIORITY *
       L_EXTENDEDPRICE - 1), COUNT(*)
FROM LINEITEM, ORDERS
WHERE L_SUPPKEY <= 1000 AND L_SUPPKEY >= 500 AND L_ORDERKEY = O_ORDERKEY
GROUP BY L_SHIPMODE, L_SHIPINSTRUCT, O_ORDERSTATUS, O_ORDERPRIORITY
ORDER BY L_SHIPMODE, L_SHIPINSTRUCT, O_ORDERSTATUS, O_ORDERPRIORITY;

-- Q11
SELECT L_SHIPMODE, L_SHIPINSTRUCT, O_ORDERSTATUS, O_ORDERPRIORITY, SUM(
       L_QUANTITY), AVG(L_QUANTITY), SUM(L_QUANTITY - L_EXTENDEDPRICE), SUM(
       L_QUANTITY * L_EXTENDEDPRICE - 1), AVG(L_DISCOUNT * (L_TAX +1)), AVG(
       L_QUANTITY / (L_TAX +1)), SUM(O_TOTALPRICE), AVG(O_SHIPPRIORITY), SUM
       (O_TOTALPRICE - L_EXTENDEDPRICE), AVG(O_SHIPPRIORITY *
       L_EXTENDEDPRICE - 1), COUNT(*)
FROM LINEITEM, ORDERS
WHERE L_SUPPKEY <= 100000 AND L_SUPPKEY >= 5000 AND O_ORDERSTATUS = 'P' AND
       L_ORDERKEY = O_ORDERKEY
GROUP BY L_SHIPMODE, L_SHIPINSTRUCT, O_ORDERSTATUS, O_ORDERPRIORITY
ORDER BY L_SHIPMODE, L_SHIPINSTRUCT, O_ORDERSTATUS, O_ORDERPRIORITY;

-- Q12
SELECT L_SHIPMODE, L_SHIPINSTRUCT, O_ORDERSTATUS, O_ORDERPRIORITY, SUM(
       L_QUANTITY), AVG(L_QUANTITY), SUM(L_QUANTITY - L_EXTENDEDPRICE), SUM(
       L_QUANTITY * L_EXTENDEDPRICE - 1), AVG(L_DISCOUNT * (L_TAX +1)), AVG(
       L_QUANTITY / (L_TAX +1)), SUM(L_TAX)/SUM(L_EXTENDEDPRICE), SUM(
       L_QUANTITY * 0.5), AVG(L_QUANTITY * 0.2), SUM(O_TOTALPRICE), AVG(
       O_SHIPPRIORITY), SUM(O_TOTALPRICE - L_EXTENDEDPRICE), AVG(
       O_SHIPPRIORITY * L_EXTENDEDPRICE - 1), SUM(O_SHIPPRIORITY)/SUM(
       O_TOTALPRICE), COUNT(*)
FROM LINEITEM, ORDERS
WHERE L_SUPPKEY <= 1000 AND L_SUPPKEY >= 500 AND L_ORDERKEY = O_ORDERKEY
GROUP BY L_SHIPMODE, L_SHIPINSTRUCT, O_ORDERSTATUS, O_ORDERPRIORITY
```

```
ORDER BY L_SHIPMODE, L_SHIPINSTRUCT, O_ORDERSTATUS, O_ORDERPRIORITY;

-- Q13
SELECT COUNT(*)
FROM LINEITEM, ORDERS
WHERE L_SUPPKEY <= 1000 AND L_ORDERKEY = O_ORDERKEY;

-- Q14
SELECT L_SHIPMODE, L_SHIPINSTRUCT, O_ORDERSTATUS, O_ORDERPRIORITY, SUM(
       L_QUANTITY), AVG(L_QUANTITY), SUM(L_QUANTITY - L_EXTENDEDPRICE), SUM(
       L_QUANTITY * L_EXTENDEDPRICE - 1), AVG(L_DISCOUNT * (L_TAX +1)), AVG(
       L_QUANTITY / (L_TAX +1)), SUM(L_TAX)/SUM(L_EXTENDEDPRICE), SUM(
       L_QUANTITY * 0.5), AVG(L_QUANTITY * 0.2), SUM(O_TOTALPRICE), AVG(
       O_SHIPPRIORITY), SUM(O_TOTALPRICE - L_EXTENDEDPRICE), AVG(
       O_SHIPPRIORITY * L_EXTENDEDPRICE - 1), SUM(O_SHIPPRIORITY)/SUM(
       O_TOTALPRICE), COUNT(*)
FROM LINEITEM, ORDERS
WHERE L_SUPPKEY <= 100000 AND L_SUPPKEY >= 5000 AND O_ORDERSTATUS = 'P' AND
       L_ORDERKEY = O_ORDERKEY
GROUP BY L_SHIPMODE, L_SHIPINSTRUCT, O_ORDERSTATUS, O_ORDERPRIORITY
ORDER BY L_SHIPMODE, L_SHIPINSTRUCT, O_ORDERSTATUS, O_ORDERPRIORITY;

-- Q15
SELECT SUM(O_TOTALPRICE)
FROM LINEITEM, ORDERS
WHERE L_SUPPKEY <= 10000 AND L_SUPPKEY >= 500 AND L_ORDERKEY = O_ORDERKEY;

-- Q16
SELECT SUM(L_QUANTITY), AVG(L_QUANTITY), SUM(O_TOTALPRICE), COUNT(*)
FROM LINEITEM, ORDERS
WHERE L_SUPPKEY <= 1000 AND L_SUPPKEY >= 500 AND L_ORDERKEY = O_ORDERKEY;

-- Q17
SELECT SUM(L_QUANTITY), AVG(L_QUANTITY), SUM(O_TOTALPRICE), COUNT(*)
FROM LINEITEM, ORDERS
WHERE L_SUPPKEY <= 100000 AND L_SUPPKEY >= 5000 AND O_ORDERSTATUS = 'F' AND
       L_ORDERKEY = O_ORDERKEY;

-- Q18
SELECT SUM(L_QUANTITY), AVG(L_QUANTITY), SUM(L_QUANTITY - L_EXTENDEDPRICE),
       SUM(L_QUANTITY * L_EXTENDEDPRICE - 1), SUM(O_TOTALPRICE), AVG(
       O_SHIPPRIORITY), COUNT(*)
FROM LINEITEM, ORDERS
WHERE L_SUPPKEY <= 1000 AND L_SUPPKEY >= 500 AND L_ORDERKEY = O_ORDERKEY;

-- Q19
SELECT SUM(L_QUANTITY), AVG(L_QUANTITY), SUM(L_QUANTITY - L_EXTENDEDPRICE),
       SUM(L_QUANTITY * L_EXTENDEDPRICE - 1), SUM(O_TOTALPRICE), AVG(
       O_SHIPPRIORITY), COUNT(*)
FROM LINEITEM, ORDERS
WHERE L_SUPPKEY <= 100000 AND L_SUPPKEY >= 5000 AND O_ORDERSTATUS = 'F' AND
       L_ORDERKEY = O_ORDERKEY;

-- Q20
SELECT SUM(L_QUANTITY), AVG(L_QUANTITY), SUM(L_QUANTITY - L_EXTENDEDPRICE),
       SUM(L_QUANTITY * L_EXTENDEDPRICE - 1), AVG(L_DISCOUNT * (L_TAX +1)),
       AVG(L_QUANTITY / (L_TAX +1)), SUM(O_TOTALPRICE), AVG(O_SHIPPRIORITY),
        SUM(O_TOTALPRICE - L_EXTENDEDPRICE), AVG(O_SHIPPRIORITY *
       L_EXTENDEDPRICE - 1), COUNT(*)
FROM LINEITEM, ORDERS
WHERE L_SUPPKEY <= 1000 AND L_SUPPKEY >= 500 AND L_ORDERKEY = O_ORDERKEY;

-- Q21
SELECT SUM(L_QUANTITY), AVG(L_QUANTITY), SUM(L_QUANTITY - L_EXTENDEDPRICE),
       SUM(L_QUANTITY * L_EXTENDEDPRICE - 1), AVG(L_DISCOUNT * (L_TAX +1)),
       AVG(L_QUANTITY / (L_TAX +1)), SUM(O_TOTALPRICE), AVG(O_SHIPPRIORITY),
        SUM(O_TOTALPRICE - L_EXTENDEDPRICE), AVG(O_SHIPPRIORITY *
       L_EXTENDEDPRICE - 1), COUNT(*)
FROM LINEITEM, ORDERS
WHERE L_SUPPKEY <= 100000 AND L_SUPPKEY >= 5000 AND O_ORDERSTATUS = 'F' AND
       L_ORDERKEY = O_ORDERKEY;

-- Q22
SELECT SUM(L_QUANTITY), AVG(L_QUANTITY), SUM(L_QUANTITY - L_EXTENDEDPRICE),
       SUM(L_QUANTITY * L_EXTENDEDPRICE - 1), AVG(L_DISCOUNT * (L_TAX +1)),
       AVG(L_QUANTITY / (L_TAX +1)), SUM(L_TAX)/SUM(L_EXTENDEDPRICE), SUM(
       L_QUANTITY * 0.5), AVG(L_QUANTITY * 0.2), SUM(O_TOTALPRICE), AVG(
       O_SHIPPRIORITY), SUM(O_TOTALPRICE - L_EXTENDEDPRICE), AVG(
       O_SHIPPRIORITY * L_EXTENDEDPRICE - 1), SUM(O_SHIPPRIORITY)/SUM(
       O_TOTALPRICE), COUNT(*)
FROM LINEITEM, ORDERS
WHERE L_SUPPKEY <= 1000 AND L_SUPPKEY >= 500 AND L_ORDERKEY = O_ORDERKEY;

-- Q23
SELECT SUM(L_QUANTITY), AVG(L_QUANTITY), SUM(L_QUANTITY - L_EXTENDEDPRICE),
       SUM(L_QUANTITY * L_EXTENDEDPRICE - 1), AVG(L_DISCOUNT * (L_TAX +1)),
       AVG(L_QUANTITY / (L_TAX +1)), SUM(L_TAX)/SUM(L_EXTENDEDPRICE), SUM(
       L_QUANTITY * 0.5), AVG(L_QUANTITY * 0.2), COUNT(*)
FROM LINEITEM
WHERE L_SUPPKEY <= 100000 AND L_SUPPKEY >= 5000;

-- Q24
SELECT L_SHIPMODE, L_SHIPINSTRUCT
FROM LINEITEM
WHERE L_SUPPKEY <= 100000 AND L_SUPPKEY >= 5000
GROUP BY L_SHIPMODE, L_SHIPINSTRUCT;

-- Q25
SELECT COUNT(*)
FROM LINEITEM
WHERE L_SUPPKEY <= 100000 AND L_SUPPKEY >= 5000;

-- Q26
SELECT L_SHIPMODE, L_SHIPINSTRUCT
FROM LINEITEM
WHERE L_SUPPKEY <= 1000
GROUP BY L_SHIPMODE, L_SHIPINSTRUCT;

-- Q27
SELECT L_SHIPMODE, L_SHIPINSTRUCT, COUNT(*)
FROM LINEITEM
WHERE L_SUPPKEY <= 10000 AND L_SUPPKEY >= 500
GROUP BY L_SHIPMODE, L_SHIPINSTRUCT
ORDER BY L_SHIPMODE, L_SHIPINSTRUCT;

-- Q28
SELECT L_SHIPMODE, L_SHIPINSTRUCT, SUM(L_QUANTITY), AVG(L_QUANTITY), COUNT
       (*)
FROM LINEITEM
WHERE L_SUPPKEY <= 1000 AND L_SUPPKEY >= 500
GROUP BY L_SHIPMODE, L_SHIPINSTRUCT
ORDER BY L_SHIPMODE, L_SHIPINSTRUCT;

-- Q29
SELECT L_SHIPMODE, L_SHIPINSTRUCT, SUM(L_QUANTITY), AVG(L_QUANTITY), COUNT
       (*)
FROM LINEITEM
WHERE L_SUPPKEY <= 100000 AND L_SUPPKEY >= 5000
GROUP BY L_SHIPMODE, L_SHIPINSTRUCT
ORDER BY L_SHIPMODE, L_SHIPINSTRUCT;

-- Q30
SELECT L_SHIPMODE, L_SHIPINSTRUCT, SUM(L_QUANTITY), AVG(L_QUANTITY), SUM(
       L_QUANTITY - L_EXTENDEDPRICE), SUM(L_QUANTITY * L_EXTENDEDPRICE - 1),
        COUNT(*)
FROM LINEITEM
WHERE L_SUPPKEY <= 1000 AND L_SUPPKEY >= 500
GROUP BY L_SHIPMODE, L_SHIPINSTRUCT
ORDER BY L_SHIPMODE, L_SHIPINSTRUCT;

-- Q31
SELECT L_SHIPMODE, L_SHIPINSTRUCT, SUM(L_QUANTITY), AVG(L_QUANTITY), SUM(
       L_QUANTITY - L_EXTENDEDPRICE), SUM(L_QUANTITY * L_EXTENDEDPRICE - 1),
        COUNT(*)
FROM LINEITEM
WHERE L_SUPPKEY <= 100000 AND L_SUPPKEY >= 5000
GROUP BY L_SHIPMODE, L_SHIPINSTRUCT
ORDER BY L_SHIPMODE, L_SHIPINSTRUCT;

-- Q32
SELECT L_SHIPMODE, L_SHIPINSTRUCT, SUM(L_QUANTITY), AVG(L_QUANTITY), SUM(
       L_QUANTITY - L_EXTENDEDPRICE), SUM(L_QUANTITY * L_EXTENDEDPRICE - 1),
        AVG(L_DISCOUNT * (L_TAX +1)), AVG(L_QUANTITY / (L_TAX +1)), COUNT(*)
FROM LINEITEM
WHERE L_SUPPKEY <= 1000 AND L_SUPPKEY >= 500
GROUP BY L_SHIPMODE, L_SHIPINSTRUCT
ORDER BY L_SHIPMODE, L_SHIPINSTRUCT;

-- Q33
SELECT L_SHIPMODE, L_SHIPINSTRUCT, SUM(L_QUANTITY), AVG(L_QUANTITY), SUM(
       L_QUANTITY - L_EXTENDEDPRICE), SUM(L_QUANTITY * L_EXTENDEDPRICE - 1),
        AVG(L_DISCOUNT * (L_TAX +1)), AVG(L_QUANTITY / (L_TAX +1)), COUNT(*)
FROM LINEITEM
WHERE L_SUPPKEY <= 100000 AND L_SUPPKEY >= 5000
GROUP BY L_SHIPMODE, L_SHIPINSTRUCT
ORDER BY L_SHIPMODE, L_SHIPINSTRUCT;

-- Q34
```

```
SELECT L_SHIPMODE, L_SHIPINSTRUCT, SUM(L_QUANTITY), AVG(L_QUANTITY), SUM(
       L_QUANTITY - L_EXTENDEDPRICE), SUM(L_QUANTITY * L_EXTENDEDPRICE - 1),
        AVG(L_DISCOUNT * (L_TAX +1)), AVG(L_QUANTITY / (L_TAX +1)), SUM(
       L_TAX)/SUM(L_EXTENDEDPRICE), SUM(L_QUANTITY * 0.5), AVG(L_QUANTITY *
       0.2), COUNT(*)
FROM LINEITEM
WHERE L_SUPPKEY <= 1000 AND L_SUPPKEY >= 500
GROUP BY L_SHIPMODE, L_SHIPINSTRUCT
ORDER BY L_SHIPMODE, L_SHIPINSTRUCT;

-- Q35
SELECT COUNT(*)
FROM LINEITEM
WHERE L_SUPPKEY <= 1000;

-- Q36
SELECT L_SHIPMODE, L_SHIPINSTRUCT, SUM(L_QUANTITY), AVG(L_QUANTITY), SUM(
       L_QUANTITY - L_EXTENDEDPRICE), SUM(L_QUANTITY * L_EXTENDEDPRICE - 1),
        AVG(L_DISCOUNT * (L_TAX +1)), AVG(L_QUANTITY / (L_TAX +1)), SUM(
       L_TAX)/SUM(L_EXTENDEDPRICE), SUM(L_QUANTITY * 0.5), AVG(L_QUANTITY *
       0.2), COUNT(*)
FROM LINEITEM
WHERE L_SUPPKEY <= 100000 AND L_SUPPKEY >= 5000
GROUP BY L_SHIPMODE, L_SHIPINSTRUCT
ORDER BY L_SHIPMODE, L_SHIPINSTRUCT;

-- Q37
SELECT COUNT(*)
FROM LINEITEM
WHERE L_SUPPKEY <= 10000 AND L_SUPPKEY >= 500;

-- Q38
SELECT SUM(L_QUANTITY), AVG(L_QUANTITY), COUNT(*)
FROM LINEITEM
WHERE L_SUPPKEY <= 1000 AND L_SUPPKEY >= 500;

-- Q39
SELECT SUM(L_QUANTITY), AVG(L_QUANTITY), COUNT(*)
FROM LINEITEM
WHERE L_SUPPKEY <= 100000 AND L_SUPPKEY >= 5000;

-- Q40
SELECT SUM(L_QUANTITY), AVG(L_QUANTITY), SUM(L_QUANTITY - L_EXTENDEDPRICE),
       SUM(L_QUANTITY * L_EXTENDEDPRICE - 1), COUNT(*)
FROM LINEITEM
WHERE L_SUPPKEY <= 1000 AND L_SUPPKEY >= 500;

-- Q41
SELECT SUM(L_QUANTITY), AVG(L_QUANTITY), SUM(L_QUANTITY - L_EXTENDEDPRICE),
       SUM(L_QUANTITY * L_EXTENDEDPRICE - 1), COUNT(*)
FROM LINEITEM
WHERE L_SUPPKEY <= 100000 AND L_SUPPKEY >= 5000;

-- Q42
SELECT SUM(L_QUANTITY), AVG(L_QUANTITY), SUM(L_QUANTITY - L_EXTENDEDPRICE),
       SUM(L_QUANTITY * L_EXTENDEDPRICE - 1), AVG(L_DISCOUNT * (L_TAX +1)),
       AVG(L_QUANTITY / (L_TAX +1)), COUNT(*)
FROM LINEITEM
WHERE L_SUPPKEY <= 1000 AND L_SUPPKEY >= 500;

-- Q43
SELECT SUM(L_QUANTITY), AVG(L_QUANTITY), SUM(L_QUANTITY - L_EXTENDEDPRICE),
       SUM(L_QUANTITY * L_EXTENDEDPRICE - 1), AVG(L_DISCOUNT * (L_TAX +1)),
       AVG(L_QUANTITY / (L_TAX +1)), COUNT(*)
FROM LINEITEM
WHERE L_SUPPKEY <= 100000 AND L_SUPPKEY >= 5000;

-- Q44
SELECT SUM(L_QUANTITY), AVG(L_QUANTITY), SUM(L_QUANTITY - L_EXTENDEDPRICE),
       SUM(L_QUANTITY * L_EXTENDEDPRICE - 1), AVG(L_DISCOUNT * (L_TAX +1)),
       AVG(L_QUANTITY / (L_TAX +1)), SUM(L_TAX)/SUM(L_EXTENDEDPRICE), SUM(
       L_QUANTITY * 0.5), AVG(L_QUANTITY * 0.2), COUNT(*)
FROM LINEITEM
WHERE L_SUPPKEY <= 1000 AND L_SUPPKEY >= 500;
```

Abréviations

AE Algorithme Évolutionnaire. 55

AEMO Algorithme Évolutionnaire Multi-Objectifs. 56

AOL Approches Orientées Logicielles. 75

AOM Approches Orientées Matérielles. 75

BD Base de Données. 18

BF Besoin Fonctionnel. 23

BNF Besoin Non-Fonctionnel. 23

BNL Block-Nested Loops. 168

DAG Graphe Orienté Acyclique. 29

DBA Administrateur de Bases de Données. 85

DBaaS Database as a Service. 174

DVFS Ajustement Dynamique de la Tension et de la Fréquence. 9

E/S Entrée/Sortie. 9

EDP Energy Delay Product. 81

EE Efficacité Énergétique. 67

GGR Graphe Globale de Requêtes. 157

GPU Unités de Traitement Graphique. 73

LHS Latin Hypercube Sampling. 119

MPL Multiprogramming Level. 97

MVPP Traitement de Plan Multi-Vues. 30

NSGA-II Non-Dominated Sorting Genetic Algorithm II. 57

OLAP Online Analytical Processing. 20

OLTP Online Transaction Processing. 126

OMO Optimisation Mono-objectifs. 49

PMO Problème d'Optimisation Multi-objectifs. 49

PRMO Problème d'Optimisation de Requêtes Multi-Objectifs. 134

PSV Problème de Sélection de Vues. 28

SE Système d'Exploitation. 6

SGBD Système de Gestion de Base de Données. 6

SLA Service Level Agreement. 174

SO Structure d'Optimisation. 26

SPC Storage Performance Council. 70

SPEC Standard Performance Evaluation Corporation. 70

SPFO Somme Pondérée des Fonctions Objectifs. 163

SQL Structured Query Language. 10

SSD Solid-State Drive. 9

TPC Transaction Processing Performance Council. 70

UML Unified Modeling Language. 22

VM Vues Matérialisées. 27

Bibliographie

[1] http://www.tpc.org/tpc_energy/ (cf. p. 8, 70).

[2] http://www.spec.org/power_ssj2008/ (cf. p. 71).

[3] http://www.storageperformance.org/specs/ (cf. p. 71).

[4] https://www.top500.org/green500/ (cf. p. 71, 73, 75).

[5] http://www.tpc.org/tpch/ (cf. p. 94, 113, 129, 207).

[6] http://www.tpc.org/tpcds/ (cf. p. 95, 113).

[7] D. Abadi et al. « The beckman report on database research ». In : *Communications of the ACM* 59.2 (2016), p. 92–99 (cf. p. 126).

[8] R. Agrawal, A. Ailamaki et al. « The Claremont report on database research ». In : *ACM SIGMOD Record* 37.3 (2008), p. 9–19 (cf. p. 126).

[9] S. Agrawal, S. Chaudhuri et V. R. Narasayya. « Automated Selection of Materialized Views and Indexes in SQL Databases. » In : *VLDB*. T. 2000. 2000, p. 496–505 (cf. p. 30, 32).

[10] M. Ahmad, S. Duan et al. « Predicting completion times of batch query workloads using interaction-aware models and simulation ». In : *EDBT*. ACM. 2011, p. 449–460 (cf. p. 119).

[11] M. Akdere et al. « Learning-based query performance modeling and prediction ». In : *2012 IEEE 28th International Conference on Data Engineering*. IEEE. 2012, p. 390–401 (cf. p. 187).

[12] D. Ameller et al. « How do software architects consider non-functional requirements : An exploratory study ». In : *20th International Requirements Engineering Conference (RE)*. IEEE. 2012, p. 41–50 (cf. p. 23).

[13] V. Anagnostopoulou et al. « Barely alive memory servers : Keeping data active in a low-power state ». In : *ACM Journal on Emerging Technologies in Computing Systems (JETC)* 8.4 (2012), p. 31 (cf. p. 74, 75).

[14] R. Appuswamy, M. Olma et A. Ailamaki. « Scaling the Memory Power Wall With DRAM-Aware Data Management ». In : *Proceedings of the 11th International Workshop on Data Management on New Hardware*. ACM. 2015, p. 3 (cf. p. 78).

[15] D. Atwood et J. G. Miner. *Reducing data center cost with an air economizer*. White Paper. 2008. URL : https://www-ssl.intel.com/content/dam/doc/technology-brief/data-center-efficiency-xeon-reducing-data-center-cost-with-air-economizer-brief.pdf (cf. p. 6).

[16] H. Aydin et al. « Dynamic and aggressive scheduling techniques for power-aware real-time systems ». In : *Real-Time Systems Symposium, 2001.(RTSS 2001). Proceedings. 22nd IEEE*. IEEE. 2001, p. 95–105 (cf. p. 73, 75).

[17] C. W. Bachman. « The programmer as navigator ». In : *Communications of the ACM* 16.11 (1973), p. 653–658 (cf. p. 18, 19).

[18] J. Bader et E. Zitzler. « HypE : An algorithm for fast hypervolume-based many-objective optimization ». In : *Evolutionary computation* 19.1 (2011), p. 45–76 (cf. p. 58, 59).

[19] G. L. Barbose et al. « The future of utility customer-funded energy efficiency programs in the USA : projected spending and savings to 2025 ». In : *Energy Efficiency* 6.3 (2013), p. 475–493 (cf. p. 4).

[20] S. Barielle. *Calculating TCO for Energy*. IBM Systems Magazine, http://www.ibmsystemsmag.com/mainframe/Business-Strategy/ROI/energy_estimating/. 2011. URL : http://www.ibmsystemsmag.com/mainframe/Business-Strategy/ROI/energy_estimating/ (cf. p. 178).

[21] M. Barr. « Bi-Objective Optimization Based on Compromise Method for Horizontal Fragmentation in Relational Data Warehouses ». In : *International Journal of Machine Learning and Computing* 3.3 (2013), p. 250 (cf. p. 60).

[22] P. Behzadnia et al. « Dynamic Power-Aware Disk Storage Management in Database Servers ». In : *DEXA*. Springer International Publishing, 2016, p. 315–325 (cf. p. 77, 78).

[23] L. Bellatreche et al. « Horizontal partitioning of very-large data warehouses under dynamically-changing query workloads via incremental algorithms ». In : *Proceedings of the 28th Annual ACM Symposium on Applied Computing*. ACM. 2013, p. 208–210 (cf. p. 60).

[24] L. Bellatreche et al. « Incremental Algorithms for Selecting Horizontal Schemas of Data Warehouses : The Dynamic Case ». In : *International Conference on Data Management in Cloud, Grid and P2P Systems*. Springer. 2013, p. 13–25 (cf. p. 60).

[25] A. Beloglazov et al. « A taxonomy and survey of energy-efficient data centers and cloud computing systems ». In : *Advances in Computers* 82.2 (2011), p. 47–111 (cf. p. 69, 76, 78, 85).

[26] L. Benini et G. d. Micheli. « System-level power optimization : techniques and tools ». In : *ACM Transactions on Design Automation of Electronic Systems (TODAES)* 5.2 (2000), p. 115–192 (cf. p. 74).

[27] K. Bennett, M. C. Ferris et Y. E. Ioannidis. *A genetic algorithm for database query optimization*. Computer Sciences Department, University of Wisconsin, Center for Parallel Optimization, 1991 (cf. p. 45).

[28] R. A. Bergamaschi et al. « Data center power and performance optimization through global selection of p-states and utilization rates ». In : *Sustainable Computing : Informatics and Systems* 2.4 (2012), p. 198–208 (cf. p. 73, 75).

[29] N. Beume, B. Naujoks et M. Emmerich. « SMS-EMOA : Multiobjective selection based on dominated hypervolume ». In : *European Journal of Operational Research* 181.3 (2007), p. 1653–1669 (cf. p. 58, 59).

[30] N. Beume et al. « On the complexity of computing the hypervolume indicator ». In : *IEEE Transactions on Evolutionary Computation* 13.5 (2009), p. 1075–1082 (cf. p. 58).

[31] K. Bilal et al. « A survey on green communications using adaptive link rate ». In : *Cluster Computing* 16.3 (2013), p. 575–589 (cf. p. 74, 75).

[32] K. Bilal et al. « A taxonomy and survey on green data center networks ». In : *Future Generation Computer Systems* 36 (2014), p. 189–208 (cf. p. 74, 75).

[33] K. Bilal et al. « Quantitative comparisons of the state-of-the-art data center architectures ». In : *Concurrency and Computation : Practice and Experience* 25.12 (2013), p. 1771–1783 (cf. p. 74, 75).

[34] K. S. Bøgh, I. Assent et M. Magnani. « Efficient GPU-based skyline computation ». In : *Proceedings of the Ninth International Workshop on Data Management on New Hardware*. ACM. 2013, p. 5 (cf. p. 63).

[35] P. Bohrer et al. « The case for power management in web servers ». In : *Power aware computing*. Springer, 2002, p. 261–289 (cf. p. 74, 75).

[36] S Borzsony, D. Kossmann et K. Stocker. « The skyline operator ». In : *ICDE*. 2001, p. 421–430 (cf. p. 60, 62, 63, 168).

[37] S. Bouarar, L. Bellatreche et A. Roukh. « Eco-Data Warehouse Design Through Logical Variability ». In : *International Conference on Current Trends in Theory and Practice of Informatics*. Springer. 2017, p. 436–449 (cf. p. 13).

[38] R. Bouchakri et L. Bellatreche. « On simplifying integrated physical database design ». In : *Advances in Databases and Information Systems*. Springer. 2011, p. 333–346 (cf. p. 104).

[39] K. Boukhalfa. « De la conception physique aux outils d'administration et de tuning des entrepôts de données ». Thèse de doct. Citeseer, 2009 (cf. p. 104).

[40] A. Boukorca et al. « Coupling Materialized View Selection to Multi Query Optimization : Hyper Graph Approach ». In : *International Journal of Data Warehousing and Mining (IJDWM)* 11.2 (2015), p. 62–84 (cf. p. 29, 30, 32, 104, 158, 165).

[41] B. Bowers. « The Science of Energy : A Cultural History of Energy Physics in Victorian Britain (review) ». In : *Technology and Culture* 41.2 (2000), p. 362–363 (cf. p. 66).

[42] J. Branke et al. *Multiobjective optimization : Interactive and evolutionary approaches*. T. 5252. Springer, 2008 (cf. p. 50, 52–55).

[43] D. Brooks, V. Tiwari et M. Martonosi. *Wattch : a framework for architectural-level power analysis and optimizations*. T. 28. 2. ACM, 2000 (cf. p. 68).

[44] N. Bruno. *Automated Physical Database Design and Tuning*. CRC Press, 2011 (cf. p. 60).

[45] N. Bruno et S. Chaudhuri. « Constrained physical design tuning ». In : *Proceedings of the VLDB Endowment* 1.1 (2008), p. 4–15 (cf. p. 60).

[46] N. Bruno et al. « AutoAdmin Project at Microsoft Research : Lessons Learned ». In : *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering* (2011). URL : http://research.microsoft.com/apps/pubs/default.aspx?id=156989 (cf. p. 104).

[47] E. Capra, C. Francalanci et S. A. Slaughter. « Is software "green" ? Application development environments and energy efficiency in open source applications ». In : *Information and Software Technology* 54.1 (2012), p. 60–71 (cf. p. 75).

[48] S. Chakkappen et al. « Efficient and scalable statistics gathering for large databases in Oracle 11g ». In : *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*. ACM. 2008, p. 1053–1064 (cf. p. 43).

[49] D. D. Chamberlin et al. « A history and evaluation of System R ». In : *Communications of the ACM* 24.10 (1981), p. 632–646 (cf. p. 19).

[50] S. Chand et M. Wagner. « Evolutionary many-objective optimization : A quick-start guide ». In : *Surveys in Operations Research and Management Science* 20.2 (2015), p. 35–42 (cf. p. 187).

[51] A. Charnes, W. W. Cooper et R. O. Ferguson. « Optimal estimation of executive compensation by linear programming ». In : *Management science* 1.2 (1955), p. 138–151 (cf. p. 54, 59).

[52] A. Charnes et W. W. Cooper. « Goal programming and multiple objective optimizations : Part 1 ». In : *European Journal of Operational Research* 1.1 (1977), p. 39–54 (cf. p. 54, 59).

[53] J. S. Chase et al. « Managing energy and server resources in hosting centers ». In : *ACM SIGOPS Operating Systems Review* 35.5 (2001), p. 103–116 (cf. p. 74, 75).

[54] A. Chaudhri, R. Zicari et A. Rashid. *XML data management : native XML and XML enabled DataBase systems*. Addison-Wesley Longman Publishing Co., Inc., 2003 (cf. p. 19).

[55] S. Chaudhuri. « An overview of query optimization in relational systems ». In : *ACM SIGACT-SIGMOD-SIGART*. ACM. 1998, p. 34–43 (cf. p. 40, 104).

[56] S. Chaudhuri et U. Dayal. « An overview of data warehousing and OLAP technology ». In : *ACM Sigmod record* 26.1 (1997), p. 65–74 (cf. p. 28).

[57] S. Chaudhuri, V. Narasayya et R. Ramamurthy. « Estimating progress of execution for SQL queries ». In : *ACM SIGMOD*. ACM. 2004, p. 803–814 (cf. p. 46, 99).

[58] S. Chaudhuri et V. R. Narasayya. « An efficient, cost-driven index selection tool for Microsoft SQL server ». In : *VLDB*. T. 97. Citeseer. 1997, p. 146–155 (cf. p. 104).

[59] S. Chaudhuri et V. R. Narasayya. « Self-Tuning Database Systems : A Decade of Progress ». In : *VLDB*. 2007, p. 3–14 (cf. p. 27, 151).

[60] S. Chaudhuri et al. « Optimizing Queries with Materialized Views. » In : *icde*. T. 1995. Citeseer. 1995, p. 190–200 (cf. p. 31, 32).

[61] L. W. F. Chaves et al. « Towards materialized view selection for distributed databases ». In : *Proceedings of the 12th International Conference on Extending Database Technology : Advances in Database Technology*. ACM. 2009, p. 1088–1099 (cf. p. 29, 30, 32).

[62] P. P.-S. Chen. « The entity-relationship model—toward a unified view of data ». In : *ACM Transactions on Database Systems (TODS)* 1.1 (1976), p. 9–36 (cf. p. 19).

[63] Y. Chen et al. « Energy efficiency for large-scale mapreduce workloads with significant interactive analysis ». In : *Proceedings of the 7th ACM european conference on Computer Systems*. ACM. 2012, p. 43–56 (cf. p. 75).

[64] Y. Chen et al. « Managing server energy and operational costs in hosting centers ». In : *ACM SIGMETRICS performance evaluation review*. T. 33. 1. ACM. 2005, p. 303–314 (cf. p. 74, 75).

[65] S.-K. Cheong, C. Lim et B.-C. Cho. « Database processing performance and energy efficiency evaluation of DDR-SSD and HDD storage system based on the TPC-C ». In : *Cloud Computing and Social Networking (ICCCSN), 2012 International Conference on*. IEEE. 2012, p. 1–3 (cf. p. 77, 78).

[66] Y. Chi, H. J. Moon et H. Hacigümüş. « iCBS : incremental cost-based scheduling under piecewise linear SLAs ». In : *Proceedings of the VLDB Endowment* 4.9 (2011), p. 563–574 (cf. p. 188).

[67] Y. Chi et al. « Distribution-based query scheduling ». In : *Proceedings of the VLDB Endowment* 6.9 (2013), p. 673–684 (cf. p. 188).

[68] R. Chirkova et J. Yang. « Materialized views ». In : *Databases* 4.4 (2011), p. 295–405 (cf. p. 29).

[69] S Chu. « The energy problem and Lawrence Berkeley National Laboratory ». In : *Talk given to the California Air Resources Board* (2008) (cf. p. 184).

[70] L. Chung et al. *Non-functional requirements in software engineering*. T. 5. Springer Science & Business Media, 2012 (cf. p. 23).

[71] E. F. Codd. « A relational model of data for large shared data banks ». In : *Communications of the ACM* 13.6 (1970), p. 377–387 (cf. p. 19).

[72] Y. Collette et P. Siarry. *Multiobjective optimization : principles and case studies*. Springer Science & Business Media, 2013 (cf. p. 52, 55).

[73] T. M. Connolly et C. E. Begg. *Database systems : a practical approach to design, implementation, and management*. Pearson Education, 2005 (cf. p. 94).

[74] G. Contreras et M. Martonosi. « Power prediction for intel XScale® processors using performance monitoring unit events ». In : *ISLPED'05. Proceedings of the 2005 International Symposium on Low Power Electronics and Design, 2005*. IEEE. 2005, p. 221–226 (cf. p. 68, 187).

[75] D. W. Corne et al. « PESA-II : Region-based selection in evolutionary multiobjective optimization ». In : *Proceedings of the genetic and evolutionary computation conference (GECCO'2001*. Citeseer. 2001 (cf. p. 57, 59).

[76] C. Coronel, S. Morris et P. Rob. *Database Systems : Design, Implementation, and Management*. Bpa Series. Cengage Learning, 2009. isbn : 9780538469685 (cf. p. 21, 23, 25, 31).

[77] T. N.R. D. Council. *America's Data Centers Consuming and Wasting Growing Amounts of Energy*. Issue Paper, http://www.nrdc.org/. 2014. URL : http://www.nrdc.org/energy/files/data-center-efficiency-assessment-IB.pdf (cf. p. 5).

[78] C. Date. « When's an extension not an extension ? » In : *Intelligent Enterprise* 2.8 (1999) (cf. p. 19).

[79] M. Dayarathna, Y. Wen et R. Fan. « Data Center Energy Consumption Modeling : A Survey ». In : *IEEE Communications Surveys & Tutorials* 18.1 (2016), p. 732–794 (cf. p. 69).

[80] T. De Matteis, S. Di Girolamo et G. Mencagli. « Continuous skyline queries on multicore architectures ». In : *Concurrency and Computation : Practice and Experience* (2016) (cf. p. 63).

[81] K. Deb. *Multi-objective optimization using evolutionary algorithms*. T. 16. John Wiley & Sons, 2001 (cf. p. 49, 51, 52, 55).

[82] K. Deb et al. « A fast and elitist multiobjective genetic algorithm : NSGA-II ». In : *IEEE Transactions on Evolutionary Computation* 6.2 (2002), p. 182–197 (cf. p. 57, 59, 152, 162).

[83] A. Diwan, S Sudarshan et D. Thomas. « Scheduling and caching in multi-query optimization ». In : *International Conference on Management of Data COMAD, Delhi, India*. 2006 (cf. p. 188).

[84] J. Do, Y.-S. Kee et al. « Query processing on smart SSDs : opportunities and challenges ». In : *ACM SIGMOD*. ACM. 2013, p. 1221–1230 (cf. p. 77, 78).

[85] T. Dokeroglu, S. A. Sert et M. S. Cinar. « Evolutionary multiobjective query workload optimization of Cloud data warehouses ». In : *The Scientific World Journal* 2014 (2014) (cf. p. 60).

[86] S. Duan, V. Thummala et S. Babu. « Tuning database configuration parameters with iTuned ». In : *Proceedings of the VLDB Endowment* 2.1 (2009), p. 1246–1257 (cf. p. 43).

[87] I. Elghandour et al. « Recommending XML physical designs for XML databases ». In : *VLDB Journal* 22.4 (2013), p. 447–470 (cf. p. 151).

[88] R. Elmasri. *Fundamentals of database systems*. Pearson Education India, 2008 (cf. p. 25, 26, 33, 36, 37, 39, 40, 43–45).

[89] J. Fan et S. Kambhampati. « Multi-objective query processing for data aggregation ». In : *ASU CSE TR* (2006) (cf. p. 61).

[90] X. Fan, C. S. Ellis et A. R. Lebeck. « The synergy between power-aware memory systems and processor voltage scaling ». In : *International Workshop on Power-Aware Computer Systems*. Springer. 2003, p. 164–179 (cf. p. 73, 75).

[91] U. Fischer et al. « Towards integrated data analytics : Time series forecasting in DBMS ». In : *Datenbank-Spektrum* 13.1 (2013), p. 45–53 (cf. p. 83, 84).

[92] J. Flinn et M. Satyanarayanan. « Managing battery lifetime with energy-aware adaptation ». In : *ACM Transactions on Computer Systems (TOCS)* 22.2 (2004), p. 137–179 (cf. p. 74, 75).

[93] C. M. Fonseca, P. J. Fleming et al. « Genetic Algorithms for Multiobjective Optimization : Formulation-Discussion and Generalization. » In : *ICGA*. T. 93. Citeseer. 1993, p. 416–423 (cf. p. 57, 59).

[94] J. C. Freytag. *A rule-based view of query optimization*. T. 16. 3. ACM, 1987 (cf. p. 37).

[95] F. Fusco et al. « Data Management System for Energy Analytics and its Application to Forecasting ». In : *Joint EDBT/ICDT PhD workshop*. 2016 (cf. p. 83, 84).

[96] A. Ganapathi et al. « Predicting multiple metrics for queries : Better decisions enabled by machine learning ». In : *2009 IEEE 25th International Conference on Data Engineering*. IEEE. 2009, p. 592–603 (cf. p. 43, 187).

[97] H. Garcia-Molina, J. D. Ullman et J. Widom. *Database Systems : The Complete Book*. 2e éd. Upper Saddle River, NJ, USA : Prentice Hall Press, 2008. ISBN : 9780131873254 (cf. p. 33, 36, 37, 39, 40, 43).

[98] G. Gardarin. *Bases de données*. Editions Eyrolles, 2003 (cf. p. 21).

[99] S. Gass et T. Saaty. « The computational algorithm for the parametric objective function ». In : *Naval research logistics quarterly* 2.1-2 (1955), p. 39–45 (cf. p. 53, 59).

[100] C. Ge, Z. Sun et N. Wang. « A survey of power-saving techniques on data centers and content delivery networks ». In : *IEEE Communications Surveys & Tutorials* 15.3 (2013), p. 1334–1354 (cf. p. 69).

[101] J. Goldstein et P.-Å. Larson. « Optimizing queries using materialized views : a practical, scalable solution ». In : *ACM SIGMOD Record*. T. 30. 2. ACM. 2001, p. 331–342 (cf. p. 31, 32).

[102] R. Goswami, D. K. Bhattacharyya et M. Dutta. « Multiobjective Differential Evolution Algorithm Using Binary Encoded Data in Selecting Views for Materializing in Data Warehouse ». In : *International Conference on Swarm, Evolutionary, and Memetic Computing*. Springer. 2013, p. 95–106 (cf. p. 60).

[103] K. Govil, E. Chan et H. Wasserman. « Comparing algorithm for dynamic speed-setting of a low-power CPU ». In : *Proceedings of the 1st annual international conference on Mobile computing and networking*. ACM. 1995, p. 13–25 (cf. p. 73).

[104] G. Graefe. « Database servers tailored to improve energy efficiency ». In : *EDBT*. ACM. 2008, p. 24–28 (cf. p. 7, 79, 117, 150).

[105] G. Graefe. « Query evaluation techniques for large databases ». In : *ACM Computing Surveys (CSUR)* 25.2 (1993), p. 73–169 (cf. p. 42).

[106] A. Gupta, S Sudarshan et S. Vishwanathan. « Query scheduling in multi query optimization ». In : *Database Engineering and Applications, 2001 International Symposium on*. IEEE. 2001, p. 11–19 (cf. p. 188).

[107] A. Gupta, I. S. Mumick et K. A. Ross. *Adapting materialized views after redefinitions*. T. 24. 2. ACM, 1995 (cf. p. 31, 32).

[108] A. Gupta, I. S. Mumick et al. « Maintenance of materialized views : Problems, techniques, and applications ». In : *IEEE Data Eng. Bull.* 18.2 (1995), p. 3–18 (cf. p. 31, 32).

[109] H. Gupta. « Selection of views to materialize in a data warehouse ». In : *International Conference on Database Theory*. Springer. 1997, p. 98–112 (cf. p. 28–30, 32).

[110] H. Gupta et I. S. Mumick. « Selection of views to materialize under a maintenance cost constraint ». In : *International Conference on Database Theory*. Springer. 1999, p. 453–470 (cf. p. 28–30, 32, 151).

[111] S. Gurumurthi et al. « Using complete machine simulation for software power estimation : The soft-watt approach ». In : *High-Performance Computer Architecture, 2002. Proceedings. Eighth International Symposium on*. IEEE. 2002, p. 141–150 (cf. p. 68).

[112] L. M. Haas et al. « Seeking the truth about ad hoc join costs ». In : *The VLDB journal* 6.3 (1997), p. 241–256 (cf. p. 42).

[113] Y. Y. Haimes, L. Ladson et D. A. Wismer. *Bicriterion formulation of problems of integrated system identification and system optimization*. 1971 (cf. p. 54, 59).

[114] A. Y. Halevy. « Answering queries using views : A survey ». In : *The VLDB Journal* 10.4 (2001), p. 270–294 (cf. p. 30).

[115] V. Harinarayan, A. Rajaraman et J. D. Ullman. « Implementing data cubes efficiently ». In : *ACM SIGMOD Record*. T. 25. 2. ACM. 1996, p. 205–216 (cf. p. 30, 32).

[116] S. Harizopoulos et al. « Energy efficiency : The new holy grail of data management systems research ». In : *arXiv preprint arXiv :0909.1784* (2009) (cf. p. 7, 68, 77–79, 117, 150).

[117] A. Hassan, H. Vandierendonck et D. S. Nikolopoulos. « Energy-Efficient In-Memory Data Stores on Hybrid Memory Hierarchies ». In : *Proceedings of the 11th International Workshop on Data Management on New Hardware*. ACM. 2015, p. 1 (cf. p. 78).

[118] F. Helff, L. Gruenwald et L. d'Orazio. « Weighted Sum Model for Multi-Objective Query Optimization for Mobile-Cloud Database Environments ». In : *EDBT/ICDT Workshops*. 2016 (cf. p. 60).

[119] I. Hong, M. Potkonjak et M. B. Srivastava. « On-line scheduling of hard real-time tasks on variable voltage processor ». In : *Proceedings of the 1998 IEEE/ACM international conference on Computer-aided design*. ACM. 1998, p. 653–656 (cf. p. 73, 75).

[120] I. Hong et al. « Power optimization of variable-voltage core-based systems ». In : *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 18.12 (1999), p. 1702–1714 (cf. p. 73, 75).

[121] J. Horn, N. Nafpliotis et D. E. Goldberg. « A niched Pareto genetic algorithm for multiobjective optimization ». In : *Evolutionary Computation, 1994. IEEE World Congress on Computational Intelligence., Proceedings of the First IEEE Conference on*. Ieee. 1994, p. 82–87 (cf. p. 57, 59).

[122] C.-H. Hsu et U. Kremer. « The design, implementation, and evaluation of a compiler algorithm for CPU energy reduction ». In : *ACM SIGPLAN Notices*. T. 38. 5. ACM. 2003, p. 38–48 (cf. p. 75).

[123] P.-K. Huang et S. Ghiasi. « Efficient and scalable compiler-directed energy optimization for realtime applications ». In : *ACM Transactions on Design Automation of Electronic Systems (TODAES)* 12.3 (2007), p. 27 (cf. p. 75).

[124] E. J. Hughes. « Evolutionary many-objective optimisation : many once or one many ? » In : *2005 IEEE congress on evolutionary computation*. T. 1. IEEE. 2005, p. 222–227 (cf. p. 58, 59).

[125] E. J. Hughes. « Multiple single objective Pareto sampling ». In : *Evolutionary Computation, 2003. CEC'03. The 2003 Congress on*. T. 4. IEEE. 2003, p. 2678–2684 (cf. p. 58, 59).

[126] A. Hurson et H. Azad. *Energy Efficiency in Data Centers and Clouds*. Academic Press, 2016 (cf. p. 7, 77, 78).

[127] C.-L. Hwang et A. S. M. Masud. *Multiple Objective Decision Making - Methods and Applications : A State-of-the-Art Survey*. T. 164. Springer-Verlag Berlin Heidelberg, 1979 (cf. p. 52, 161).

[128] R. Ihaka et R. Gentleman. « R : a language for data analysis and graphics ». In : *Journal of computational and graphical statistics* 5.3 (1996), p. 299–314 (cf. p. 109).

[129] I. IMS. « 360–Application Description Manual ». In : *GH-20.0765, IBM, White Plains, New York* () (cf. p. 19).

[130] Intel et Oracle. *Oracle Exadata on Intel® Xeon® Processors : Extreme Performance for Enterprise Computing*. White Paper. 2011. url : http://www.intel.co.uk/content/dam/doc/white-paper/performance-xeon-7500-xeon-5600-oracle-exadata-extreme-performance-for-enterprise-computing-brief.pdf (cf. p. 76, 78).

[131] Intel Corporation. *Intel® 64 and IA-32 Architectures Optimization Reference Manual*. 248966-030. 2014 (cf. p. 108).

[132] Y. E. Ioannidis et Y. Kang. « Randomized algorithms for optimizing large join queries ». In : *ACM Sigmod Record*. T. 19. 2. ACM. 1990, p. 312–321 (cf. p. 45).

[133] Y. E. Ioannidis et E. Wong. *Query optimization by simulated annealing*. T. 16. 3. ACM, 1987 (cf. p. 45).

[134] H. Ishibuchi et T. Murata. « A multi-objective genetic local search algorithm and its application to flowshop scheduling ». In : *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 28.3 (1998), p. 392–403 (cf. p. 58).

[135] M. Jarke et J. Koch. « Query optimization in database systems ». In : *ACM Computing surveys (CsUR)* 16.2 (1984), p. 111–152 (cf. p. 36).

[136] Y. Jin, T. Okabe et B. Sendho. « Adapting weighted aggregation for multiobjective evolution strategies ». In : *International Conference on Evolutionary Multi-Criterion Optimization*. Springer. 2001, p. 96–110 (cf. p. 58).

[137] R. Joseph et M. Martonosi. « Run-time power estimation in high performance microprocessors ». In : *Proceedings of the 2001 international symposium on Low power electronics and design*. ACM. 2001, p. 135–140 (cf. p. 68).

[138] I. Kadayif et al. « Compiler-directed high-level energy estimation and optimization ». In : *ACM Transactions on Embedded Computing Systems (TECS)* 4.4 (2005), p. 819–850 (cf. p. 75).

[139] P. Kalnis, N. Mamoulis et D. Papadias. « View selection using randomized search ». In : *Data & Knowledge Engineering* 42.1 (2002), p. 89–111 (cf. p. 29, 30, 32).

[140] W. Kang, S. H. Son et J. A. Stankovic. « Power-Aware Data Buffer Cache Management in Real-Time Embedded Databases ». In : *Embedded and Real-Time Computing Systems and Applications, 2008. RTCSA'08. 14th IEEE International Conference on*. IEEE. 2008, p. 35–44 (cf. p. 82, 84).

[141] K. Kant. « Data center evolution : A tutorial on state of the art, issues, and challenges ». In : *Computer Networks* 53.17 (2009), p. 2939–2965 (cf. p. 5, 6).

[142] S. W. Keckler et al. « GPUs and the future of parallel computing ». In : *IEEE Micro* 31.5 (2011), p. 7–17 (cf. p. 73, 75).

[143] Y. Kehua et A. Diasse. « A dynamic materialized view Selection in a Cloud-based Data Warehouse ». In : *IJCSI International Journal of Computer Science Issues* 11.2 (2014), p. 1694–0814 (cf. p. 29, 32).

[144] A. Kerkad, L. Bellatreche et D. Geniet. *Queen-bee : Query interaction-aware for buffer allocation and scheduling problem*. Springer, 2012 (cf. p. 104).

[145] M. E. Khalefa et al. « Model-based integration of past & future in TimeTravel ». In : *Proceedings of the VLDB Endowment* 5.12 (2012), p. 1974–1977 (cf. p. 83, 84).

[146] A. A. Khattab, A. Algergawy et A. Sarhan. « NNMonitor : performance modeling for database servers ». In : *Computer Engineering & Systems (ICCES), 2013 8th International Conference on*. IEEE. 2013, p. 301–306 (cf. p. 43).

[147] S. Kong, Y. Li et L. Feng. « Cost-performance driven resource configuration for database applications in IaaS cloud environments ». In : *Cloud Computing and Services Science*. Springer, 2012, p. 111–129 (cf. p. 60).

[148] A. C. König et al. « A statistical approach towards robust progress estimation ». In : *Proceedings of the VLDB Endowment* 5.4 (2011), p. 382–393 (cf. p. 187).

[149] J. Koomey. « Growth in data center electricity use 2005 to 2010 ». In : *A report by Analytical Press, completed at the request of The New York Times* (2011), p. 9 (cf. p. 5).

[150] M. Korkmaz et al. « Towards Dynamic Green-Sizing for Database Servers ». In : *International Workshop on Accelerating Data Management Systems Using Modern Processor and Storage Architectures (ADMS)*. 2015, p. 25–36 (cf. p. 78, 79).

[151] Y. Kotidis et N. Roussopoulos. « DynaMat : a dynamic view management system for data warehouses ». In : *ACM SIGMOD Record*. T. 28. 2. ACM. 1999, p. 371–382 (cf. p. 29, 32).

[152] M. Kunjir, P. K. Birwa et J. R. Haritsa. « Peak power plays in database engines ». In : *EDBT*. ACM. 2012, p. 444–455 (cf. p. 80, 82, 84, 97, 107, 108, 111, 122, 153, 159, 189).

[153] M. H. Kutner, C. Nachtsheim et J. Neter. *Applied linear regression models*. McGraw-Hill/Irwin, 2004 (cf. p. 105, 107).

[154] B. Ladjel. « Contributions à la Conception et l'Exploitation des Systèmes d'Intégration de Données ». Thèse de doct. 2009 (cf. p. 27).

[155] W. Lang, R. Kandhan et J. M. Patel. « Rethinking Query Processing for Energy Efficiency : Slowing Down to Win the Race. » In : *IEEE Data Eng. Bull.* 34.1 (2011), p. 12–23 (cf. p. 80, 82, 84, 107, 159, 189).

[156] W. Lang et J. Patel. « Towards eco-friendly database management systems ». In : *arXiv preprint arXiv :0909.1767* (2009) (cf. p. 76, 78, 81, 84).

[157] W. Lang et J. M. Patel. « Energy management for mapreduce clusters ». In : *Proceedings of the VLDB Endowment* 3.1-2 (2010), p. 129–139 (cf. p. 75).

[158] W. Lang et al. « Towards energy-efficient database cluster design ». In : *Proceedings of the VLDB Endowment* 5.11 (2012), p. 1684–1695 (cf. p. 81, 83, 84).

[159] J. H. Laros III et al. « Energy delay product ». In : *Energy-Efficient High Performance Computing*. Springer, 2013, p. 51–55 (cf. p. 81).

[160] M. Lawrence. « Multiobjective genetic algorithms for materialized view selection in olap data warehouses ». In : *Proceedings of the 8th annual conference on Genetic and evolutionary computation*. ACM. 2006, p. 699–706 (cf. p. 29, 32, 60).

[161] M. Lawrence et A. Rau-Chaplin. « Dynamic view selection for OLAP ». In : *International Conference on Data Warehousing and Knowledge Discovery*. Springer. 2006, p. 33–44 (cf. p. 157).

[162] A. R. Lebeck et al. « Power aware page allocation ». In : *ACM Sigplan Notices* 35.11 (2000), p. 105–116 (cf. p. 73, 75).

[163] J. Leverich et C. Kozyrakis. « On the energy (in) efficiency of hadoop clusters ». In : *ACM SIGOPS Operating Systems Review* 44.1 (2010), p. 61–65 (cf. p. 75).

[164] A. Y. Levy, A. O. Mendelzon et Y. Sagiv. « Answering queries using views ». In : *Proceedings of the fourteenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*. ACM. 1995, p. 95–104 (cf. p. 30–32).

[165] J. Li, R. Nehme et J. Naughton. « GSLPI : A cost-based query progress indicator ». In : *Data Engineering (ICDE), 2012 IEEE 28th International Conference on*. IEEE. 2012, p. 678–689 (cf. p. 99).

[166] J. Li et al. « Robust estimation of resource consumption for sql queries using statistical techniques ». In : *Proceedings of the VLDB Endowment* 5.11 (2012), p. 1555–1566 (cf. p. 43, 187).

[167] T. Li et L. K. John. « Run-time modeling and estimation of operating system power consumption ». In : *ACM SIGMETRICS Performance Evaluation Review*. T. 31. 1. ACM. 2003, p. 160–171 (cf. p. 68).

[168] E. Liebert. *Five Strategies for Cutting Data Center Energy Costs Through Enhanced Cooling Efficiency*. White Paper. 2007. url : http://www.emersonnetworkpower.com/documentation/en-us/brands/liebert/documents/white%20papers/data-center-energy-efficiency_151-47.pdf (cf. p. 5).

[169] S. S. Lightstone, T. J. Teorey et T. Nadeau. *Physical Database Design : the database professional's guide to exploiting indexes, views, storage, and more*. Morgan Kaufmann, 2010 (cf. p. 26, 31).

[170] S. Ligoudistianos, D. Theodoratos et T. Sellis. « Experimental evaluation of data warehouse configuration algorithms ». In : *Database and Expert Systems Applications, 1998. Proceedings. Ninth International Workshop on*. IEEE. 1998, p. 218–223 (cf. p. 30, 32).

[171] G. M. Lohman. « The DB2 Universal Database Optimizer ». In : *IBM Research Division, IBM Almaden Research Center* (2007) (cf. p. 43).

[172] G. Luo et al. « Toward a progress indicator for database queries ». In : *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*. ACM. 2004, p. 791–802 (cf. p. 47, 99).

[173] D. Maier et al. *Development of an object-oriented DBMS*. T. 21. 11. ACM, 1986 (cf. p. 19).

[174] I. Mami et Z. Bellahsene. « A survey of view selection methods ». In : *SIGMOD Record* 41.1 (2012), p. 20–29 (cf. p. 29, 151, 172).

[175] I. Mami, Z. Bellahsene et R. Coletta. « A Constraint Optimization Method for Large-Scale Distributed View Selection ». In : *Transactions on Large-Scale Data-and Knowledge-Centered Systems XXV*. Springer, 2016, p. 71–108 (cf. p. 29, 32).

[176] I. Mami, R. Coletta et Z. Bellahsene. « Modeling view selection as a constraint satisfaction problem ». In : *International Conference on Database and Expert Systems Applications*. Springer. 2011, p. 396–410 (cf. p. 29, 30, 32).

[177] I. Manotas, L. Pollock et J. Clause. « SEEDS : a software engineer's energy-optimization decision support framework ». In : *Proceedings of the 36th International Conference on Software Engineering*. ACM. 2014, p. 503–514 (cf. p. 75).

[178] R. T. Marler et J. S. Arora. « Survey of multi-objective optimization methods for engineering ». In : *Structural and multidisciplinary optimization* 26.6 (2004), p. 369–395 (cf. p. 52).

[179] D. Meisner, B. T. Gold et T. F. Wenisch. « PowerNap : eliminating server idle power ». In : *ACM SIGARCH Computer Architecture News* 37.1 (2009), p. 205–216 (cf. p. 74, 75).

[180] R. S. Michalski, J. G. Carbonell et T. M. Mitchell. *Machine learning : An artificial intelligence approach*. Springer Science & Business Media, 2013 (cf. p. 69).

[181] H. Mistry et al. « Materialized view selection and maintenance using multi-query optimization ». In : *ACM SIGMOD Record*. T. 30. 2. ACM. 2001, p. 307–318 (cf. p. 29, 32, 157).

[182] M. Mohania, S. Konomi et Y. Kambayashi. « Incremental maintenance of materialized views ». In : *International Conference on Database and Expert Systems Applications*. Springer. 1997, p. 551–560 (cf. p. 31, 32).

[183] D. C. Montgomery, E. A. Peck et G. G. Vining. *Introduction to linear regression analysis*. T. 821. John Wiley & Sons, 2012 (cf. p. 105).

[184] K. Mullesgaard et al. « Efficient skyline computation in MapReduce ». In : *17th International Conference on Extending Database Technology (EDBT)*. 2014, p. 37–48 (cf. p. 63).

[185] T. Murata, H. Ishibuchi et M. Gen. « Specification of genetic search directions in cellular multi-objective genetic algorithms ». In : *International Conference on Evolutionary Multi-Criterion Optimization*. Springer. 2001, p. 82–95 (cf. p. 58).

[186] R. O. Nambiar et M. Poess. « The making of TPC-DS ». In : *Proceedings of the 32nd international conference on Very large data bases*. VLDB Endowment. 2006, p. 1049–1058 (cf. p. 113).

[187] D. Narayanan et al. « Migrating server storage to SSDs : analysis of tradeoffs ». In : *Proceedings of the 4th ACM European conference on Computer systems*. ACM. 2009, p. 145–158 (cf. p. 74, 75).

[188] J. Neter et al. *Applied linear statistical models*. T. 4. Irwin Chicago, 1996 (cf. p. 105, 106, 108).

[189] R. Neugebauer et D. McAuley. « Energy is just another resource : Energy accounting and energy pricing in the Nemesis OS ». In : *Hot Topics in Operating Systems, 2001. Proceedings of the Eighth Workshop on*. IEEE. 2001, p. 67–72 (cf. p. 74, 75).

[190] D. Niu et al. « Impact of process variations on emerging memristor ». In : *Design Automation Conference (DAC), 2010 47th ACM/IEEE*. IEEE. 2010, p. 877–882 (cf. p. 73, 75).

[191] B. Nuseibeh et S. Easterbrook. « Requirements engineering : a roadmap ». In : *Proceedings of the Conference on the Future of Software Engineering*. ACM. 2000, p. 35–46 (cf. p. 24).

[192] Oracle. *MySQL Enterprise Monitor and Query Analyzer Feature*. White Paper. April 2010. URL : http://www.oracle.com/us/products/mysql/mysql-enterprise-m-q-a-wp-067591.pdf (cf. p. 100).

[193] A.-C. Orgerie, M. D. d. Assuncao et L. Lefevre. « A survey on techniques for improving the energy efficiency of large-scale distributed systems ». In : *ACM Computing Surveys (CSUR)* 46.4 (2014), p. 47 (cf. p. 69, 74, 75).

[194] A. Ouared, Y. Ouhammou et A. Roukh. « A Meta-advisor Repository for Database Physical Design ». In : *International Conference on Model and Data Engineering*. Springer. 2016, p. 72–87 (cf. p. 13).

[195] M. T. Özsu et P. Valduriez. *Principles of distributed database systems*. Springer Science & Business Media, 2011 (cf. p. 42, 47).

[196] P. O'Neil et al. « The star schema benchmark and augmented fact table indexing ». In : *Performance evaluation and benchmarking*. Springer, 2009, p. 237–252 (cf. p. 158).

[197] V. Pallipadi et A. Starikovskiy. « The ondemand governor ». In : *Proceedings of the Linux Symposium*. T. 2. sn. 2006, p. 215–230 (cf. p. 74, 75).

[198] C. H. Papadimitriou et M. Yannakakis. « Multiobjective query optimization ». In : *Proceedings of the twentieth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*. ACM. 2001, p. 52–59 (cf. p. 60, 61).

[199] Y. Park et al. « Power-aware memory management for hybrid main memory ». In : *Next Generation Information Technology (ICNIT), 2011 The 2nd International Conference on*. IEEE. 2011, p. 82–85 (cf. p. 73, 75).

[200] L. Pérez-Lombard, J. Ortiz et C. Pout. « A review on buildings energy consumption information ». In : *Energy and buildings* 40.3 (2008), p. 394–398 (cf. p. 4).

[201] P. Pillai et K. G. Shin. « Real-time dynamic voltage scaling for low-power embedded operating systems ». In : *ACM SIGOPS Operating Systems Review*. T. 35. 5. ACM. 2001, p. 89–102 (cf. p. 73, 75).

[202] M. Poess et R. O. Nambiar. « Energy cost, the key challenge of today's data centers : a power consumption analysis of TPC-C results ». In : *PVLDB* 1.2 (2008), p. 1229–1240 (cf. p. 7, 70).

[203] M. Poess, R. O. Nambiar et D. Walrath. « Why you should run TPC-DS : a workload analysis ». In : *Proceedings of the 33rd international conference on Very large data bases*. VLDB Endowment. 2007, p. 1138–1149 (cf. p. 116).

[204] M. Poess et al. « Energy benchmarks : a detailed analysis ». In : *Proceedings of the 1st International Conference on Energy-Efficient Computing and Networking*. ACM. 2010, p. 131–140 (cf. p. 70).

[205] V. Poosala et al. « Improved histograms for selectivity estimation of range predicates ». In : *ACM SIGMOD Record*. T. 25. 2. ACM. 1996, p. 294–305 (cf. p. 41).

[206] I. Psaroudakis, M. Athanassoulis et A. Ailamaki. « Sharing data and work across concurrent analytical queries ». In : *Proceedings of the VLDB Endowment* 6.9 (2013), p. 637–648 (cf. p. 188).

[207] I. Psaroudakis et al. « Dynamic fine-grained scheduling for energy-efficient main-memory queries ». In : *Proceedings of the Tenth International Workshop on Data Management on New Hardware*. ACM. 2014, p. 1 (cf. p. 82, 84).

[208] R. Rajkumar et al. « Resource kernels : A resource-centric approach to real-time and multimedia systems ». In : *Photonics West'98 Electronic Imaging*. International Society for Optics et Photonics. 1997, p. 150–164 (cf. p. 74, 75).

[209] R. Ramakrishnan et J. Gehrke. *Database Management Systems*. 3e éd. New York, NY, USA : McGraw-Hill, Inc., 2003. isbn : 0072465638, 9780072465631 (cf. p. 41).

[210] T. Rault, A. Bouabdallah et Y. Challal. « Energy efficiency in wireless sensor networks : A top-down survey ». In : *Computer Networks* 67 (2014), p. 104–122 (cf. p. 4).

[211] S. Reda et A. N. Nowroz. « Power modeling and characterization of computing devices : a survey ». In : *Foundations and Trends in Electronic Design Automation* 6.2 (2012), p. 121–216 (cf. p. 69).

[212] S. Rivoire, P. Ranganathan et C. Kozyrakis. « A Comparison of High-Level Full-System Power Models. » In : *HotPower* 8 (2008), p. 3–3 (cf. p. 68, 91).

[213] S. Rivoire et al. « JouleSort : a balanced energy-efficiency benchmark ». In : *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*. ACM. 2007, p. 365–376 (cf. p. 68, 71).

[214] S. M. Rivoire. « Models and metrics for energy-efficient computer systems ». Thèse de doct. Stanford University, 2008 (cf. p. 70, 90).

[215] M. Rodriguez-Martinez et al. « Estimating power/energy consumption in database servers ». In : *Procedia Computer Science* 6 (2011), p. 112–117 (cf. p. 81, 84, 111, 189).

[216] M. Rofouei et al. « Energy-aware high performance computing with graphic processing units ». In : *Workshop on power aware computing and system*. 2008 (cf. p. 76, 78).

[217] C. Ronneau. *Énergie, pollution de l'air et développement durable*. T. 1. Presses univ. de Louvain, 2004 (cf. p. 66).

[218] K. A. Ross, D. Srivastava et S Sudarshan. « Materialized view maintenance and integrity constraint checking : Trading space for time ». In : *ACM SIGMOD Record*. T. 25. 2. ACM. 1996, p. 447–458 (cf. p. 28, 32).

[219] A. Roukh. « Estimating power consumption of batch query workloads ». In : *International Conference on Model and Data Engineering*. Springer, 2015, p. 198–212 (cf. p. 12).

[220] A. Roukh et L. Bellatreche. « Eco-processing of olap complex queries ». In : *International Conference on Big Data Analytics and Knowledge Discovery*. Springer, 2015, p. 229–242 (cf. p. 12).

[221] A. Roukh, L. Bellatreche et C. Ordonez. « EnerQuery : Energy-Aware Query Processing ». In : *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*. ACM. 2016, p. 2465–2468 (cf. p. 13).

[222] A. Roukh et al. « Eco-DMW : Eco-design methodology for data warehouses ». In : *Proceedings of the ACM Eighteenth International Workshop on Data Warehousing and OLAP*. ACM. 2015, p. 1–10 (cf. p. 12).

[223] A. Roukh et al. « Eco-Physic : Eco-Physical Design Initiative for Very Large Databases ». In : *Information Systems* 68C (2017), p. 44–62 (cf. p. 12).

[224] A. Roukh et al. « Energy-Aware Query Processing on Parallel Database Cluster Nodes ». In : *International Conference on Algorithms and Architectures for Parallel Processing*. Springer, 2016, p. 260–269 (cf. p. 13).

[225] D. G. Sachs et al. *GRACE : A hierarchical adaptation framework for saving energy*. 2004 (cf. p. 74, 75).

[226] D. Schall et T. Härder. « WattDB-A Journey towards Energy Efficiency ». In : *Datenbank-Spektrum* 14.3 (2014), p. 183–198 (cf. p. 83, 84).

[227] D. Schall, V. Hoefner et M. Kern. « Towards an enhanced benchmark advocating energy-efficient systems ». In : *Technology Conference on Performance Evaluation and Benchmarking*. Springer. 2011, p. 31–45 (cf. p. 71).

[228] D. Schall, V. Hudlet et T. Härder. « Enhancing energy efficiency of database applications using SSDs ». In : *Proceedings of the Third C* Conference on Computer Science and Software Engineering*. ACM. 2010, p. 1–9 (cf. p. 7, 77, 78).

[229] P. G. Selinger et al. « Access path selection in a relational database management system ». In : *Proceedings of the 1979 ACM SIGMOD international conference on Management of data*. ACM. 1979, p. 23–34 (cf. p. 40, 42, 44, 45, 193).

[230] T. K. Sellis. « Multiple-query optimization ». In : *ACM Transactions on Database Systems (TODS)* 13.1 (1988), p. 23–52 (cf. p. 188).

[231] K. Sergey. *Oracle : Real-Time SQL Monitoring*. White Paper. December 2009. URL : http://www.oracle.com/technetwork/database/manageability/owp-sql-monitoring-128746.pdf (cf. p. 95, 99).

[232] H. Shafi et al. « Design and validation of a performance and power simulator for PowerPC systems ». In : *IBM Journal of Research and Development* 47.5.6 (2003), p. 641–651 (cf. p. 68).

[233] Y. Shang, D. Li et M. Xu. « Energy-aware routing in data center network ». In : *Proceedings of the first ACM SIGCOMM workshop on Green networking*. ACM. 2010, p. 1–8 (cf. p. 74, 75).

[234] V. Sharma et al. « Power-aware QoS management in web servers ». In : *Real-Time Systems Symposium, 2003. RTSS 2003. 24th IEEE*. IEEE. 2003, p. 63–72 (cf. p. 74, 75).

[235] M. B. Sheikh et al. « A bayesian approach to online performance modeling for database appliances using gaussian models ». In : *Proceedings of the 8th ACM international conference on Autonomic computing*. ACM. 2011, p. 121–130 (cf. p. 43).

[236] Y. Shin et K. Choi. « Power conscious fixed priority scheduling for hard real-time systems ». In : *Design Automation Conference, 1999. Proceedings. 36th*. IEEE. 1999, p. 134–139 (cf. p. 73, 75).

[237] J. Shuja et al. « Survey of Techniques and Architectures for Designing Energy-Efficient Data Centers ». In : *IEEE Systems Journal* 10.2 (2016), p. 507–519. issn : 1932-8184. doi : 10.1109/JSYST.2014.2315823 (cf. p. 73, 75).

[238] N. Siegmund et al. « Measuring non-functional properties in software product line for product derivation ». In : *2008 15th Asia-Pacific Software Engineering Conference*. IEEE. 2008, p. 187–194 (cf. p. 24).

[239] L. Siksnys, C. Thomsen et T. B. Pedersen. « MIRABEL DW : Managing Complex Energy Data in a Smart Grid ». In : *DaWaK*. 2012, p. 443–457 (cf. p. 83, 84, 152).

[240] R. Silipo et P. Winters. *Big Data, Smart Energy, and Predictive Analytics*. White Paper, https://www.knime.org/files/knime_bigdata_energy_timeseries_whitepaper.pdf. 2013 (accessed 06.04.16) (cf. p. 83, 84).

[241] M. Själander, M. Martonosi et S. Kaxiras. « Power-Efficient Computer Architectures : Recent Advances ». In : *Synthesis Lectures on Computer Architecture* 9.3 (2014), p. 1–96 (cf. p. 9).

[242] K. Skadron et al. « Temperature-aware microarchitecture : Modeling and implementation ». In : *ACM Transactions on Architecture and Code Optimization (TACO)* 1.1 (2004), p. 94–125 (cf. p. 188).

[243] S. Srikantaiah, A. Kansal et F. Zhao. « Energy aware consolidation for cloud computing ». In : *Proceedings of the 2008 conference on Power aware computing and systems*. T. 10. San Diego, California. 2008, p. 1–5 (cf. p. 188).

[244] N. Srinivas et K. Deb. « Muiltiobjective optimization using nondominated sorting in genetic algorithms ». In : *Evolutionary computation* 2.3 (1994), p. 221–248 (cf. p. 57, 59).

[245] M. Steinbrunn, G. Moerkotte et A. Kemper. « Heuristic and randomized optimization for the join ordering problem ». In : *The VLDB Journal—The International Journal on Very Large Data Bases* 6.3 (1997), p. 191–208 (cf. p. 45).

[246] M. Stephen Chu in Conrick. « Introducing databases, Health informatics : transforming healthcare with technology ». In : *JOURNAL OF TELEMEDICINE AND TELECARE* 13.1 (2007), p. 60 (cf. p. 18).

[247] M. Stillger et M. Spiliopoulou. « Genetic programming in database query optimization ». In : *Proceedings of the 1st annual conference on genetic programming*. MIT Press. 1996, p. 388–393 (cf. p. 45).

[248] M. Stonebraker. « NewSQL : An Alternative to NoSQL and Old SQL for New OLTP Apps ». In : *Communications of the ACM. Retrieved* (2012), p. 07–06 (cf. p. 20).

[249] M. Stonebraker. « SQL databases v. NoSQL databases ». In : *Communications of the ACM* 53.4 (2010), p. 10–11 (cf. p. 20).

[250] A. Sunil et al. *Troubleshooting Performance Problems in SQL Server 2008*. White Paper. March 2009. url : http://download.microsoft.com/download/D/B/D/DBDE7972-1EB9-470A-BA18-58849DB3EB3B/TShootPerfProbs2008.docx (cf. p. 100).

[251] G. e Sustainability Initiative et I. the Boston Consulting Group. *GeSI SMARTer 2020 : The Role of ICT in Driving a Sustainable Future*. Press Release. 2012 (cf. p. 5).

[252] S. H. Talebian et S. A. Kareem. « A weight based genetic algorithm for selecting views ». In : *2012 International Conference on Graphic and Image Processing*. International Society for Optics et Photonics. 2013, 87683O–87683O (cf. p. 60).

[253] D. Taniar et al. *High performance parallel database processing and grid databases*. T. 67. John Wiley & Sons, 2008 (cf. p. 47).

[254] D. Theodoratos, T. Sellis et al. « Data warehouse configuration ». In : *VLDB*. T. 97. 1997, p. 126–135 (cf. p. 30, 32).

[255] M. Thiele, A. Bader et W. Lehner. « Multi-objective scheduling for real-time data warehouses ». In : *Computer Science-Research and Development* 24.3 (2009), p. 137–151 (cf. p. 60).

[256] E. Tiakas, A. N. Papadopoulos et Y. Manolopoulos. « Skyline queries : An introduction ». In : *Information, Intelligence, Systems and Applications (IISA), 2015 6th International Conference on*. IEEE. 2015, p. 1–6 (cf. p. 62, 63).

[257] S. F. Tie et C. W. Tan. « A review of energy sources and energy management system in electric vehicles ». In : *Renewable and Sustainable Energy Reviews* 20 (2013), p. 82–102 (cf. p. 4).

[258] I. Trummer et C. Koch. « A Fast Randomized Algorithm for Multi-Objective Query Optimization ». In : *Proceedings of the 2016 International Conference on Management of Data*. SIGMOD '16. ACM, 2016, p. 1737–1752. isbn : 978-1-4503-3531-7 (cf. p. 62, 188).

[259] I. Trummer et C. Koch. « An incremental anytime algorithm for multi-objective query optimization ». In : *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*. ACM. 2015, p. 1941–1953 (cf. p. 62).

[260] I. Trummer et C. Koch. « Approximation schemes for many-objective query optimization ». In : *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*. ACM. 2014, p. 1299–1310 (cf. p. 62).

[261] I. Trummer et C. Koch. « Multi-objective parametric query optimization ». In : *Proceedings of the VLDB Endowment* 8.3 (2014), p. 221–232 (cf. p. 60, 62).

[262] I. Trummer et C. Koch. « Multiple query optimization on the D-Wave 2X adiabatic quantum computer ». In : *Proceedings of the VLDB Endowment* 9.9 (2016), p. 648–659 (cf. p. 62).

[263] I. Trummer et C. Koch. « Parallelizing query optimization on shared-nothing architectures ». In : *Proceedings of the VLDB Endowment* 9.9 (2016), p. 660–671 (cf. p. 62).

[264] I. Trummer et C. Koch. « Solving the Join Ordering Problem via Mixed Integer Linear Programming ». In : *arXiv preprint arXiv :1511.02071* (2015) (cf. p. 62).

[265] D. Tsirogiannis, S. Harizopoulos et M. A. Shah. « Analyzing the energy efficiency of a database server ». In : *sigmod*. 2010, p. 231–242 (cf. p. 5, 68, 83).

[266] Y.-C. Tu et al. « A system for energy-efficient data management ». In : *ACM SIGMOD Record* 43.1 (2014), p. 21–26 (cf. p. 76–78).

[267] V. Vardhan et al. « GRACE-2 : integrating fine-grained application adaptation with global adaptation for saving energy ». In : *international Journal of embedded Systems* 4.2 (2009), p. 152–169 (cf. p. 74, 75).

[268] S Vellev. « Review of Algorithms for the Join Ordering Problems in Database Query Optimization ». In : *Information Technologies and Control* 1 (2009), p. 32–40 (cf. p. 45).

[269] V. Venkatachalam et M. Franz. « Power reduction techniques for microprocessor systems ». In : *ACM Computing Surveys (CSUR)* 37.3 (2005), p. 195–237 (cf. p. 67).

[270] C. Vira et Y. Y. Haimes. *Multiobjective decision making : theory and methodology*. 8. North-Holland, 1983 (cf. p. 54, 59).

[271] T. Wagner, N. Beume et B. Naujoks. « Pareto-, aggregation-, and indicator-based methods in many-objective optimization ». In : *International conference on evolutionary multi-criterion optimization*. Springer. 2007, p. 742–756 (cf. p. 56, 58).

[272] J. Wang et al. « A survey on energy-efficient data management ». In : *ACM SIGMOD Record* 40.2 (2011), p. 17–23 (cf. p. 68, 83).

[273] X. Wang et al. « Spin torque random access memory down to 22 nm technology ». In : *IEEE transactions on magnetics* 44.11 (2008), p. 2479–2482 (cf. p. 73, 75).

[274] T. Warren. *Feature analysis of codasyl data base management systems*. Rapp. tech. DTIC Document, 1975 (cf. p. 19).

[275] M. Weiser et al. « Scheduling for reduced CPU energy ». In : *Mobile Computing*. Springer, 1994, p. 449–471 (cf. p. 72, 75).

[276] L. Woods, Z. István et G. Alonso. « Ibex : an intelligent storage engine with support for advanced SQL offloading ». In : *Proceedings of the VLDB Endowment* 7.11 (2014), p. 963–974 (cf. p. 76, 78).

[277] D. Wu et al. « RAMZzz : rank-aware dram power management with dynamic migrations and demotions ». In : *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. IEEE Computer Society Press. 2012, p. 32 (cf. p. 73, 75).

[278] W. Wu et al. « Predicting query execution time : Are optimizer cost models really unusable ? » In : *Data Engineering (ICDE), 2013 IEEE 29th International Conference on*. IEEE. 2013, p. 1081–1092 (cf. p. 42).

[279] W. Wu et al. « Towards predicting query execution time for concurrent and dynamic database workloads ». In : *Proceedings of the VLDB Endowment* 6.10 (2013), p. 925–936 (cf. p. 43).

[280] C. Xian, Y.-H. Lu et Z. Li. « A programming environment with runtime energy characterization for energy-aware applications ». In : *Low Power Electronics and Design (ISLPED), 2007 ACM/IEEE International Symposium on*. IEEE. 2007, p. 141–146 (cf. p. 75).

[281] P. Xiong et al. « ActiveSLA : a profit-oriented admission control framework for database-as-a-service providers ». In : *Proceedings of the 2nd ACM Symposium on Cloud Computing*. ACM. 2011, p. 15 (cf. p. 188).

[282] Z. Xu, Y.-C. Tu et X. Wang. « Dynamic Energy Estimation of Query Plans in Database Systems ». In : *ICDCS*. IEEE. 2013, p. 83–92 (cf. p. 79, 84, 96, 103–105, 107, 122, 155, 189).

[283] Z. Xu, Y.-C. Tu et X. Wang. « Exploring power-performance tradeoffs in database systems ». In : *ICDE*. 2010, p. 485–496 (cf. p. 78, 79, 82, 84, 111, 131, 159).

[284] Z. Xu, Y.-C. Tu et X. Wang. *Power Modeling in Database Mansgement Systems*. Rapp. tech. CSE/12-094. University of South Florida, Department of Computer Science et Engineering, 2012 (cf. p. 95).

[285] Z. Xu, X. Wang et Y.-C. Tu. « Power-Aware Throughput Control for Database Management Systems. » In : *ICAC*. 2013, p. 315–324 (cf. p. 76, 78).

[286] J. Yang, K. Karlapalem et Q. Li. « Algorithms for materialized view design in data warehousing environment ». In : *VLDB*. 1997, p. 25–29 (cf. p. 29, 30, 32, 157).

[287] F. Yao, A. Demers et S. Shenker. « A scheduling model for reduced CPU energy ». In : *Foundations of Computer Science, 1995. Proceedings., 36th Annual Symposium on*. IEEE. 1995, p. 374–382 (cf. p. 73, 75).

[288] A. G. Yoder. « Energy efficient storage technologies for data centers ». In : *WEED 2010-Workshop on Energy-Efficient Design*. 2010 (cf. p. 73, 75).

[289] E. Young, P. Cao et M. Nikolaiev. « First TPC-energy benchmark : lessons learned in practice ». In : *Technology Conference on Performance Evaluation and Benchmarking*. Springer. 2010, p. 136–152 (cf. p. 70).

[290] J. X. Yu et al. « Materialized view selection as constrained evolutionary optimization ». In : *Systems, Man, and Cybernetics, Part C : Applications and Reviews, IEEE Transactions on* 33.4 (2003), p. 458–467 (cf. p. 30, 32).

[291] L. Zadeh. « Optimality and non-scalar-valued performance criteria ». In : *IEEE transactions on Automatic Control* 8.1 (1963), p. 59–60 (cf. p. 53, 59).

[292] M. Zaharioudakis et al. « Answering complex SQL queries using automatic summary tables ». In : *ACM SiGMOD Record*. T. 29. 2. ACM. 2000, p. 105–116 (cf. p. 31, 32).

[293] P. Zave. « Classification of research efforts in requirements engineering ». In : *ACM Computing Surveys (CSUR)* 29.4 (1997), p. 315–321 (cf. p. 23).

[294] H. Zeng et al. « ECOSystem : Managing energy as a first class operating system resource ». In : *ACM SIGPLAN Notices*. T. 37. 10. ACM. 2002, p. 123–132 (cf. p. 74, 75).

[295] C. Zhang, X. Yao et J. Yang. « An evolutionary approach to materialized views selection in a data warehouse environment ». In : *Systems, Man, and Cybernetics, Part C : Applications and Reviews, IEEE Transactions on* 31.3 (2001), p. 282–294 (cf. p. 29, 30, 32).

[296] Q. Zhang et H. Li. « MOEA/D : A multiobjective evolutionary algorithm based on decomposition ». In : *IEEE Transactions on evolutionary computation* 11.6 (2007), p. 712–731 (cf. p. 58, 59).

[297] X. Zhao et al. « Optimizing security and quality of service in a Real-time database system using Multi-objective genetic algorithm ». In : *Expert Systems with Applications* 64 (2016), p. 11–23 (cf. p. 60).

[298] X. Zheng et al. « Learning-based analytical cross-platform performance prediction ». In : *Embedded Computer Systems : Architectures, Modeling, and Simulation (SAMOS), 2015 International Conference on*. IEEE. 2015, p. 52–59 (cf. p. 187).

[299] A. Zhou et al. « Multiobjective evolutionary algorithms : A survey of the state of the art ». In : *Swarm and Evolutionary Computation, Elsevier* 1.1 (2011), p. 32–49 (cf. p. 49, 151, 162, 164).

[300] P. Zhou et al. « A durable and energy efficient main memory using phase change memory technology ». In : *ACM SIGARCH computer architecture news*. T. 37. 3. ACM. 2009, p. 14–23 (cf. p. 73, 75).

[301] E. Zitzler et S. Künzli. « Indicator-based selection in multiobjective search ». In : *International Conference on Parallel Problem Solving from Nature*. Springer. 2004, p. 832–842 (cf. p. 58, 59).

[302] E. Zitzler, M. Laumanns, L. Thiele et al. « SPEA2 : Improving the strength Pareto evolutionary algorithm ». In : *Eurogen*. T. 3242. 103. 2001, p. 95–100 (cf. p. 57, 59).

[303] E. Zitzler et L. Thiele. « Multiobjective evolutionary algorithms : a comparative case study and the strength Pareto approach ». In : *IEEE transactions on Evolutionary Computation* 3.4 (1999), p. 257–271 (cf. p. 58).