Journal of Ubiquitous Systems & Pervasive Networks
Volume 17, No. 2 (2022) pp. 59-65

An Optimized Kappa Architecture for IoT Data Management in Smart Farming†

Jean Bertin Nkamla Penka[a],*, Saïd Mahmoudi[a], Olivier Debauche[a,b,c]

[a] *University of Mons, Faculty of Engineering - ILIA Lab / InforTech, Place du parc 20, Mons, Belgium, 7000*
[b] *University of Liège - GxABT, Terra, Passage des Déportés 2, Gembloux, Belgium, 5030*
[c] *University of Liège - GxABT, BioDynE - DEAL, Passage des Déportés 2, Gembloux, Belgium, 5030*

† Data from this paper were partially presented and published in the proceedings of the 18th International Conference on Mobile Systems and Pervasive Computing (MobiSPC) [1].

Abstract

Agriculture 4.0 is a domain of IoT in full growth which produces large amounts of data from machines, robots, and sensors networks. This data must be processed very quickly, especially for the systems that need to make real-time decisions. The Kappa architecture provides a way to process Agriculture 4.0 data at high speed in the cloud, and thus meets processing requirements. This paper presents an optimized version of the Kappa architecture allowing fast and efficient data management in Agriculture. The goal of this optimized version of the classical Kappa architecture is to improve memory management and processing speed. the Kappa architecture parameters are fine tuned in order to process data from a concrete use case. The results of this work have shown the impact of parameters tweaking on the speed of treatment. We have also proven that the combination of Apache Samza with Apache Druid offers the better performances.

Keywords: *Agriculture 4.0, IoT, Internet of Things, Kappa Architecture, Smart Farming, Smart Agriculture*

1. Introduction

Agriculture 4.0 is a natural evolution of precision agriculture which saw the integration of ICT in the field of agriculture. Nowadays with Agriculture 4.0, the interaction between Wireless Sensors and Actuators Network (WSAN), agricultural machinery, robots (Milking, UGVs), UAVs (drones) [2], geo-services and external sources of data and services allowing to propose new services for farmers reducing the time spent on technical interventions and improving their well-being. These new applications automate a series of tasks that farmers previously performed and enlightens them in their decision-making. Automation and rapid decision-making require the fastest possible processing of data. These treatments are critical when they impact the control of the environmental conditions in which biological objects (animal or plant) evolve. For example, in agriculture without substratum (aeroponic system) a failure in water or nutrient supply system quickly causes dieback or even death of cultivated plants. A second example come from industrial henhouse, where the control of the ammonia level, the temperature or the CO_2 level are not finely controlled, this implies the appearance of diseases and / or an increase in mortality. A third example is the supply chain ensured between farms and agroindustry that must know the availability of biological material used in the composition of their products to plan their production. Lambda and Kappa architectures are conventionally used for processing IoT data in smart farming.

The Lambda architecture is composed of two processing steps. The first process data in real-time while the second is specifically dedicated to batch processing of data in deferred time or for large-scale treatments. The main drawback of the Lambda architecture is the need to maintain two separate processing branches which leads to an increase in costs. Indeed, this architecture is well adapted if different processing operations are carried out on the two branches, for example when the current data is processed in real time and the old data stored in files or databases are processed in batch processing [3]. The Kappa architecture is well suited for the online processing of data flows produced by IoT devices but can also process offline data in the form of micro batches [3, 4]. In this architecture, one way of processing is to ensure the treatment for the real-time and batch processing of data. This approach is advantageous because it uses the same code to achieve the treatment in batch and real-time processing. This approach is implemented when an estimated value must quickly calculate in

* Corresponding author. Tel.: +3265374059
Fax: +3271140095; E-mail:
JeanBertin.NKAMLAPENKA@student.umons.ac.be

DOI: 10.5383/JUSPN.17.02.002

real-time and in a second time, a more precise value is calculated by batch processing and replace the value obtained in first approach, in a second time.

Recently, other architectures have been proposed such as iXen [19], and Fog and Edge extensions i.e. [20, 21] but we do not know at this stage if they will be adopted in smart farming domain. A convergence between the IoT and the blockchain is also being put in place at the level of supply chains [22].

However, this generalist architecture is not specifically optimized for smart farming. Wingerath et al. mentioned that Kappa architecture is viable only with fine-tuned data retention or data compression or if high power computing is available. Referring us to Wingerath et al. [5], who attempt to optimize the performance of the Kappa architecture at message queue level namely where the data is temporarily stored before its processing. They noticed that the way with which the message queue which stores temporary data before their processing is configured directly impacts the speed of data ingestion and processing. They have also analyzed the influence of the allocated memory and the offset commit period at the message queue level on the global speed of treatment.

In this paper, we propose a fine-tuned Kappa architecture based on a concrete use case in Precision Livestock of behaviors classification. We will study the impact of each parameter on the overall performance of the architecture. The rest of this paper is organized as follow: In section 2, we summarize the works related to Kappa architecture. In section 3, we present the proposed modified Kappa architecture. Afterwards, in section 4, we present our experiments applied to a concrete use case, then the results are presented and analyzed. Finally, we conclude the paper and draw the future research perspectives.

2. Related works

In the domain of Internet of Things in particular in Smart Farming, different architectures exist which allow to collect, process and store data. One of the major architectures used is the Kappa Architecture [3]. In the following paragraph, we will summarize the principal contributions about Kappa Architecture and stream processing. Persico et al. [6] benchmarked Lambda and Kappa architecture with three different configurations (horizontal scalability with standard deployment and optimized deployment, and vertical scalability). Experimentation was achieved on the Yahoo Flickr Creative Commons 100 million (YFFC100M) divided in subsets Small (1M), Medium (10M), Large (60M) and Extra-Large (100M) of tuples. Results show that Lambda architecture perform better than Kappa architecture on all datasets, on horizontal and vertical scalability tests [6]. Bixio et al. presented an architecture based on proxy, adapter, and data processing microservices to manage stream data from IoT at edge and cloud level and able to manage dynamically and relocate microservices. This proposed architecture extends the IoT platform Senseioty [1], use Java OSGi microservice framework to develop microservices, and Siddhi[2] and Apache Flink as stream processing engines [7]. Zschörnig et al. suggested a personal analytics IoT platform based on a Kappa architecture where Kafka is the log data storage, Kafka stream is used for stream processing deployed as microservices developed in Java, Druid is the database for the serving layer, API Services are written in Python, Metabase allows the visualization of data, and a data lake allows long term storage of data [8]. Persson et al. proposed a Kappa architecture based on Serverless deployment for IoT to push computation to the very edge of the network. The framework used to design this architecture is the distributed IoT-framework Calvin [9]. Feick et al. presented the state-of-the-art real-time architectures Lambda and Kappa for data processing. After a short description of each architecture, they made an experimentation with both architectures with a case study based on the Twitter's streaming API as data source. Their conclusion shown that the choice between these architectures depends on the use case and the constraints defined by the application [10]. Sanla et al. presented a comparative performance between the Lambda and the Kappa architectures for real-time Big Data analytic. Experimentation has been done with data size 3 MB, 30 MB and 300 MB. The results shown that Lambda architecture outperforms Kappa architecture for around 9% of accuracy tests but it takes approximately 2.2 times more than Kappa architecture. They concluded also that Lambda architecture uses more 10-20% of CPU usage and 0.5 GB of RAM usage more than Kappa architecture. Therefore, they recommended to use Lambda architecture when accuracy is needed for the business and Kappa architecture when it is not the case, but quick results is required [11]. Roukh et al. developed WalleSmart, an architecture dedicated to Smart Farming and based on an adapted Lambda Architecture and a datalake. Indeed, in contrary to classical Lambda, they implemented different code for batch and real-time analysis [12]. Fote et al. presented a review of Big Data storage and analysis tools applied to Smart Farming. They proposed a Smart Farming architecture defined by Data sources (sensors, IoT devices, robots, etc.), by process tools (MQTT, Kafka and Storm), by storage point (Cassandra, PostgreSQL) and by a view dashboard built on NodeJS and Python [13].

These related works focused on optimized deployment and scalability, on comparison between Kappa and Lambda architectures performances or on a different Kappa architecture implementation based on Serverless deployment. This paper will focus on the impact of the message processing queue optimization on the global architecture performances in terms of ingesting speed.

Moreover, the related works focused on processing components choice, but rarely on the message queue optimization and its impact on global performances of the architecture. This part of the architecture is generally under studied. That is why; we proposed to investigate it in this work.

3. The modified Kappa architecture

The analysis of the related works presented in the previous section shows that there is no works that clearly attempts to optimize the message processing queue located upstream of the data processing component.

According to our study of the literature review, we propose in this paper a new optimized data processing pipeline based on a classical Kappa architecture and composed of four majors' parts: a message queue (**1**) which stores temporary data before

[1] https://senseioty.com/

[2] https://siddhi.io/

their treatment by the data processing software (**2**) which produces a result stored in database (**3**). While an orchestrator (**4**) ensures the coordination of the operation of the different software and monitors their operating status. In this proposition, the memory usage and processing speed were fine-tuned by optimizing Kafka parameters used as data log storage. Different combinations of data processing software and database have also been tested to choose the better ones. The Fig. 1 presents the conceptual organization of the proposed architecture composed of a message queue which allows to collect data coming from external services, databases, files (CSV, TSV, and so on), agriculture machinery, UAVs, UGVs or sensors [2]. Afterwards, the data is ingested by data processing software and finally, the result of the treatment is stored in a database where the data can be queried by applications. The duration of data storage depends on its nature and value. Indeed, some data immediately lose its value after consumption while others can retain a value over time [2].

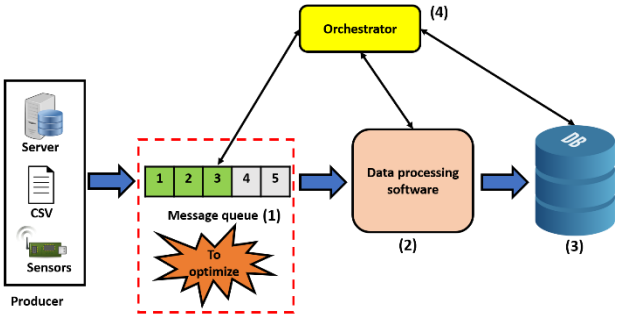


Fig. 1. General scheme of a Kappa architecture

In order to optimize the memory usage and the processing speed of our Kappa, we decided to focus on the message queue which is Apache Kafka in our case.

According to the Kafka's documentation, we found that we can improve its performances and therefore the processing time of the Kappa architecture, by using the *OffsetcommitPeriodMs* parameter. The offset in Apache Kafka represents the number assigned to each message (see message queue in Fig. 1. The *OffsetcommitPeriodMs* is the delay in milliseconds (ms) before the update of the offset in Kafka as treated by the consumer of the message. The Apache Kafka's documentation points out that Kafka needs 8 bytes of the RAM per offset to store its messages. Therefore, we have decided to analyze the impact of the RAM's memory allowed to Kafka on the processing time of the messages. Considering the existing software that can be used to build the Kappa, we have compared the well-known software for data processing and for data storage with a most recent software. They are described in the following section.

4. Experimentation

The use case implemented to achieve our experimentation is the behavior of farm animal's analysis in pasture based on IMU and GPS data [15-18]. This analysis is particularly important to early detect signs of illness, distress, and fertility periods. It allows also to detect especially bogged down animals, an escaped animals from its enclosure, lameness which is the source of economic losses and suffering for the animal, etc.

In the experimentation, the impact on processing time of the Kappa architecture was evaluated in function of RAM allocated and the offset commit period of the Kafka message queue.

The experimentation has been achieved on data produced by IMU 9-DOF and GPS posting logged by means of an iPhone 5s placed upside the neck of a cow. The iPhone 5s thanks to Data Sensor v1.26 [3] (Wavefront Labs) allowing to collect 41 parameters at a rate up to 100Hz. Collected data are (1) Acceleration on x, y, z; (2) Euler angles (pitch, roll, yaw); (3) Attitude quaternion on x, y, z; (4) Rotation matrix (3x3); (5) Gravitational component of acceleration; (6) User component of acceleration; (7) Rotation rate; (8) Magnetic data; (9) Magnetic and true heading; (10) Latitude and longitude; (11) Altitude and accuracies; (12) Course; (13) Speed; (14) Sensor proximity. We have implemented a Decision Tree based algorithms described in [14] to classify cow feeding behaviors. Fig. 2 shows the algorithm used for identifying cow behaviors. The cows' behaviors values GRA (grass intake), RUM (ruminating) or OTHERS are determined on the basis of means and standard deviations levels of gravitational acceleration (Gx) and rotation rate (Rx). These signals related to the head (HM) and jaw movements (JM). Means and standard deviations are calculated on interval of 1s which correspond to a mean of 100 data at 100 Hz. Hence, this algorithm allows to evaluate animal behaviors each second.

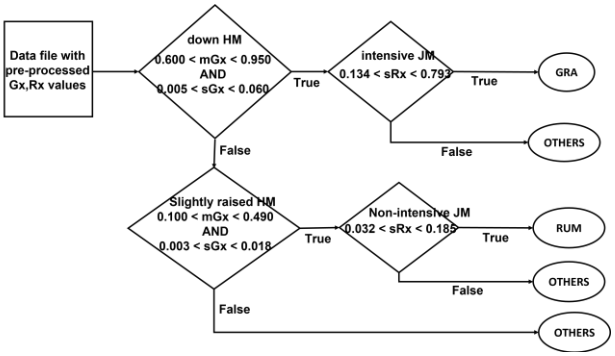


Fig. 2. Implemented algorithm of cows' behaviors determination

Hardware configuration used for the implementation of the architecture was a High-Performance VPS XXL Contabo with followings characteristics: 10 vCPU Cores, 60 GB RAM, 1.6 TB SSD, 1 Gbit/s Port [4]. Four different configurations of our architectures were benchmarked. All these combinations implement Kafka as log data storage. Kafka temporary stores data before their processing by Apache Storm or Apache Samza. Two kinds of databases were tested Apache HBase and Apache Druid in combinations with Storm and Samza to obtains 4 configurations (see Fig. 3).

Apache Kafka is an open-source distributed event streaming platform which is mainly used as temporary log storage. Apache Storm and Apache Samza are two open source distributed real-time computation systems. Apache Storm is also scalable, fault-tolerant which guarantees that the data will be processed in case

[3] Available in the Apple App store: https://itunes.apple.com/us/app/sensor-data/id397619802?mt=8

[4] https://contabo.com/en/vps/

of incident. An Apache Storm topology ingests streams of data and processes those streams in arbitrarily complex ways.

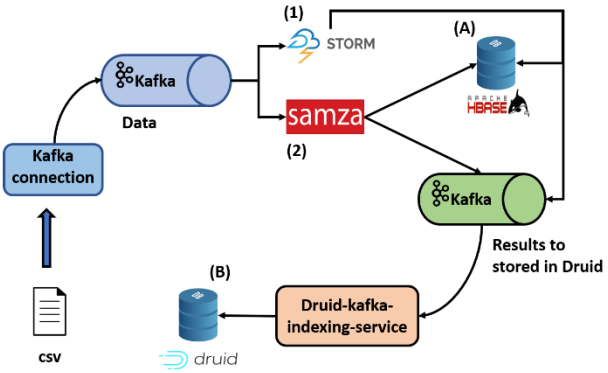


Fig. 3. Implementation of the Kappa architecture to be optimized

Apache Samza is a scalable data processing engine that allows to process and analyze data in real-time. Apache HBase is an open source, distributed, versioned, non-relational database built to provide random, real-time read/write access to Big Data with large tables composed of billions of rows and millions of columns. Apache Druid is an open-source distributed data store with high performance real-time analytics capabilities and combining concepts of data warehouses, time series databases, and search systems. We have chosen Apache Storm because it is a well-known and used tool for data processing while Apache Samza is a promising stream processing software. We have also decided to work with NoSQL databases to be more flexible and adaptable to the evolution of data structure over the time. Tested configurations are: (*1A*) Apache Kafka - Apache Storm - HBase; (*1B*) Apache Kafka - Apache Storm - Apache Kafka - Druid; (*2A*) Apache Kafka - Apache Samza - Apache HBase; (*2B*) Apache Kafka - Apache Samza - Apache Kafka - Apache Druid. In configurations 1B and 2B results of stream processing is store in a new Kafka topic and then ingested by the service Druid-Kafka-Indexing service which ingests data directly in the Kafka. While in configuration 1A and 1B data is directly store in HBase. All the software components used for the experiments are listed in Table 1.

Table 1. Software used for experiments

Software	Version	Reference
Java	1.8	https://www.java.com
Apache Kafka	2.12-2.5.0	https://kafka.apache.org
Apache Samza	1.3.1	https://samza.apache.org
Apache Storm	2.1.0	https://storm.apache.org
Apache HBase	2.2.4	https://hbase.apache.org
Apache Druid	0.18.0	https://druid.apache.org
Apache Cassandra	3.11.4	https://cassandra.apache.org
Apache Zookeeper	3.6.0	https://zookeeper.apache.org

5. Results

Results of the experimentation allows us to demonstrate the impact of the memory (RAM) allocated and offset commit at Kafka level on global performances of Kappa architecture.

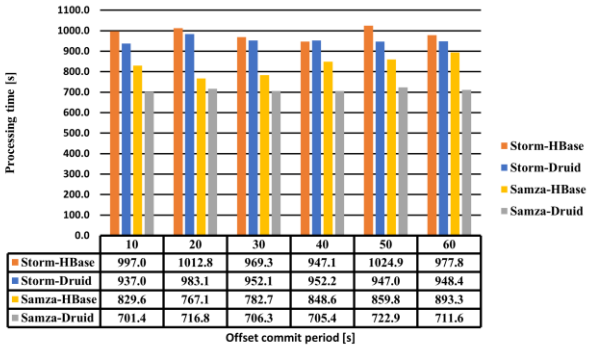


Fig. 4. Impact of the offset commit period on the processing time

The Fig. 4 presents the results of the benchmark used. For each Kappa's configurations defined on Fig. 3, the processing time per Kafka's offset commit period has been measured. The processing time measured is the time elapsed between data reading from csv file, the treatment by Apache Storm or Apache Samza and the storage into one NoSQL database. The experiments were repeated ten times per offset commit period and the values are the mean values.

On Fig. 4, by comparing the performances, we can notice that the combinations with Apache Druid gave better processing time than the others with Apache HBase. The comparison between the data processing tools shows that Apache Samza gave better performance than Apache Storm.

The combination Apache Samza - Apache Druid gave the best result of our benchmark. This result could be explained by the fact that Apache Druid is designed and optimized to ingest and read data. Apache Samza is also designed to optimize data treatment by its internal parallelism strategy.

The combination Apache Storm - Apache HBase presents less good result of our benchmark. Indeed, Apache Storm implements an internal mechanism to avoid congestion during the treatment of data. This mechanism could introduce a delay in the data treatment.

Another reason for this bad performance in comparison to the previous combination could be explained by the fact that Apache HBase is not optimized to quickly ingest data but to store a large amount of data.

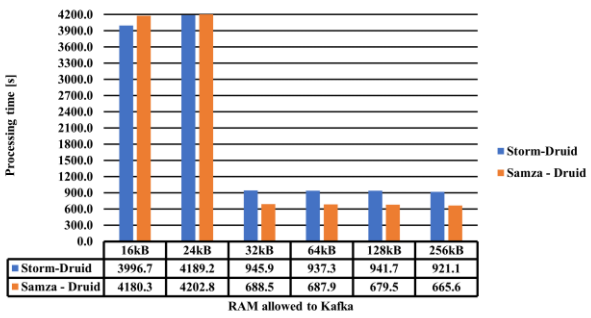


Fig. 5. Impact of the RAM allowed to Kafka on the processing time

The Fig. 5 presents the impact of the RAM allowed to the Kafka on the processing time. Regarding to our previous benchmark

results, this choice was made to analyze the impact only for the combinations Apache Storm - Apache Druid and Apache Samza - Apache Druid. The experiments were repeated ten times per ram values and the represented values are the mean values.

On the Fig. 5, we can see that the processing time is only impacted with the RAM values lower than 32KB. This behavior could be explained by the fact that the memory allowed to Apache Kafka is a kind of buffer for messages processing. Therefore, when the memory allowed to Apache Kafka is insufficient to process incoming messages, Kafka stores the messages into his buffer with a delay greater than expected. The consequence is the increase of the data processing time.

The analysis of the impact on the processing time of the Kappa architecture in function of the offset commit period and the RAM allocated to the Kafka message queue was presented in the use case. At the end, we can conclude that the best optimized combination is the Apache Kafka as the message queue, Apache Samza as the data processing and Apache Druid as NoSQL database.

Additionally, Apache Storm implements a particular parameter called *MaxUncommitedOffsets* which define the maximum time period during which data can be temporary stored between this processing. In other terms, micro batches are simulated, and their size is intrinsically controlled through the *MaxUncommitedOffsets* parameter which makes it possible to regularize the size of the data sets to be processed.

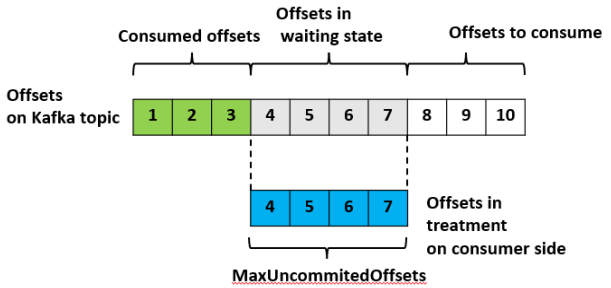


Fig. 6. Offsets on Kafka topic and consumer

The Fig. 6 is a simple representation of the mechanism to process and update the offsets on Kafka topic by a consumer. On this figure, the offsets on Kafka topics are numbers 1 to 10. The numbers from 1 to 3 in green are already consumed. The numbers 4 to 7 in grey are in waiting state because they are in treatment on the consumer side, where the same numbers are represented in blue. When using Apache Storm for processing data, the offsets in blue are also known as the *UncommitedOffsets* because it is the responsibility to the consumer to commit an offset as consumed on the Kafka topic. This mechanism enables Kafka topic to have multiple consumers at the same time. Apache Storm has defined an internal parameter named *MaxUncommitedOffsets* which is the maximum *UncommitedOffsets* that a consumer can commit on Kafka topic. The commit occurs during a period of time called *OffsetCommitPeriodMs (OCP)* which is also an internal parameter defined by Apache Storm for processing data from an Apache Kafka topic. The offsets number 8 to 10 in white will be consumed later.

Afterwards, we decided to analyze the impact of the *MaxUncommitedOffsets* parameter on the processing time of our Kappa architecture define by the tested configurations: Apache Kafka – Apache Storm – Apache HBase, Apache Kafka – Apache Storm – Apache Druid and Apache Kafka – Apache Storm – Apache Cassandra. The aim of this experimentation is to study the link between the *MaxUncommitedOffsets* and the processing time for value of *OCP* comprised between 0.5s and 3s with a step of 0.5s.
We decided to add Apache Cassandra as a tested non-relational database because Apache Cassandra is a well-known open-source non-relational database which can store a huge amount of data and we wanted to know if we can use it for our purpose.

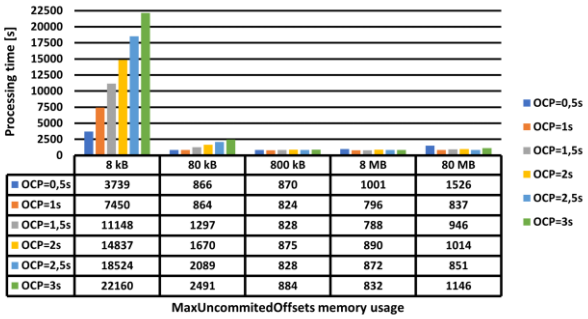


Fig. 7. Storm-HBase: Impact of *MaxUncommittedOffsets* parameter on processing time by memory usage

On the Fig. 7, we can see the impact of *MaxUncommittedOffsets* parameter on the processing time; with a too small size of 8Kb the processing time is highly impacted and conduct to an important increasing of the processing time. Subsequently, we can also observe a decreasing of the processing time with the size of the *MaxUncommitedOffsets* parameter up to 80kB and a stabilization of the processing time beyond 80kB. Moreover, we also see that the processing time, inside a same group, increase with the OCP values but the impact is less important for *MaxUncommittedOffsets* values in the range 800Kb to 80 Mb. These behaviors can be explained on one hand inside groups by the fact that for a smaller size of MaxUncommittedOffsets and high value of OCP, the algorithm of data treatment spend a lot of time in a waiting state to commit the treated offsets. On the other hand, between groups, we have more greater value for memory to store data and the increase of OCP does not really affect the processing time again.

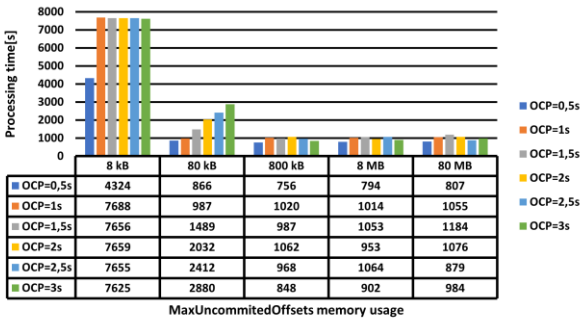
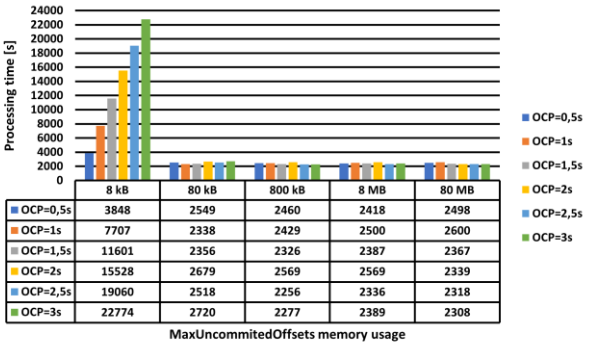


Fig. 8. Storm-Cassandra – Impact of *MaxUncommittedOffsets* parameter on processing time by memory usage

On the Fig. 8, we can see a similar behavior of the *MaxUncommittedOffsets* parameter as the couple Storm - HBase with the following differences: In the group 8kB the impact of

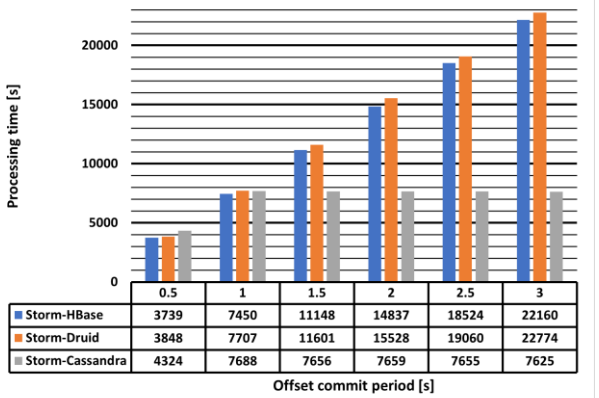
OCP values is not proportional as in the same group in the Storm-HBase experimentation. Furthermore, the proportional relation between processing time and OCP values can be observed on in the group 80kB. This impact is less visible for *MaxUncommittedOffset*s values upper than 80kB.



	8 kB	80 kB	800 kB	8 MB	80 MB
OCP=0,5s	3848	2549	2460	2418	2498
OCP=1s	7707	2338	2429	2500	2600
OCP=1,5s	11601	2356	2326	2387	2367
OCP=2s	15528	2679	2569	2569	2339
OCP=2,5s	19060	2518	2256	2336	2318
OCP=3s	22774	2720	2277	2389	2308

Fig. 9. Storm-Druid – Impact of *MaxUncommitedOffsets* parameter on processing time by memory usage

The Fig. 9 shows a look like the graph of the Fig.7. associating Storm with HBase with relatively similar values. Indeed, we can observe an identical behavior on the two graphs for the *MaxUncommitedOffets* value equal to 8kB where the processing time evolves quasi linearly with the value of OCP. The processing time is slightly lower for the Storm-HBase combination than for the Storm-Druid combination.



	0.5	1	1.5	2	2.5	3
Storm-HBase	3739	7450	11148	14837	18524	22160
Storm-Druid	3848	7707	11601	15528	19060	22774
Storm-Cassandra	4324	7688	7656	7659	7655	7625

Fig. 10. Storm associated with HBase-Druid-Cassandra – Impact of *MaxUncommitedOffsets* parameter value of 1000 offsets (8kB) by offsetCommitPeriod

The Fig. 10 shows a focus for the three combinations above for an 8kB memory used by the *MaxUncommitedOffsets* parameter during the OCP ranged between 0.5s to 3s. The combination Apache Kafka - Apache Storm – Apache Cassandra seems to stack to a certain value. This behavior could be explained by an overload of the system which result by a non-treatment of all data. Unlike the two other combinations where the processing time increase when the offset commit period increase, but all the data are treated. Additional reading on Apache Cassandra points the fact that despite its ability to store a huge data value, its writing and reading speed are less good than other non-relational databases. This result shows that the non-relational database Apache Cassandra does not feet our purpose.

The results in Fig. 7. to 10 show that a too low size of the *MaxUncommitedOffsets* parameter directly impacts the performances of the architecture in terms of processing time. A value of OCP of 0.5s generally give better processing time independently of the size of the *MaxUncommiteOffsets* and tested configurations.

7. Conclusion and perspectives

Agriculture 4.0 is an IoT domain in full growth with needs to process large volumes of data in the shortest possible time. The performances of 4 combinations of software associating Kafka as message queue and mixing (Samza or Storm as processing software) with (HBase or Druid as database) to build the Kappa architecture have been evaluated. Afterwards, we have described the improvement of the Kappa architecture to optimize the speed of data ingestion and processing. This optimal Kappa architecture is implemented with Apache Samza which process data and Apache Druid to store them. Apache Kafka is used as a first message queue which play the role of temporary log storage before data ingesting by Samza. Then, data processing results are stored in a second message queue before its insertion in Druid database. This operation is achieved by the "Kafka-indexing-service", a Druid service that ingests data stored in Kafka message queue and insert it in Druid database. Moreover, in this paper, we have shown that this association of software outperforms classically associated Apache Storm with Apache HBase in Kappa Architecture. Afterwards, we have shown the impact on the overall performance of this optimal architecture of both RAM allocation and offset commit period at Kafka level. Finally, the specific *MaxUncommitedOffsets* parameter used by Kafka when configured with Storm has been tested. Three configurations have been assessed (Storm-HBase, Storm-Druid, Storm-Cassandra) to evaluate the impact of the *MaxUncommitedOffsets* parameter on the overall performance of the architecture.

In our future works, the architecture will be completed with a data lake to store raw data on long term and develop an edge computing complement to process data at fog level in order to improve performances.

Acknowledgments

This research is partially funded by InforTech and Numediart institutes. Authors would like thanks to Prof Jérôme Bindelle (ULiège - GxABT) which has provided us the dataset of dairy cows used to achieve our experimentation.

References

[1] Nkamla Penka JB, Mahmoudi S, Debauche O. A new Kappa Architecture for IoT Data Management in Smart Farming. Procedia Computer Science 2021; 191:17-24. https://doi.org/10.1016/j.procs.2021.07.006

[2] Debauche O, Trani J-P, Mahmoudi S, Manneback P, Bindelle J, Mahmoudi SA, Guttadauria A, Lebeau F. Data management and internet of things: A methodological

review in smart farming. Internet of Things 2021; 14:100378. https://doi.org/10.1016/j.iot.2021.100378

[3] Debauche O, Mahmoudi S, Manneback P, Lebeau F. Cloud and Distributed Architectures for Data Management in Agriculture 4.0 : Review and Future Trends. Journal of King Saud University - Computer and Information Sciences. In Press. https://doi.org/10.1016/j.jksuci.2021.09.015

[4] Kreps J. Questioning the lambda architecture. Online article, 2014:205.

[5] Wingerath W, Gessert F, Friedrich S, Ritter N. Real-time stream processing for big data. it-Information Technology 2016; 58:186-194. https://doi.org/10.1515/itit-2016-0002

[6] Persico V, Pescapé A, Picariello A, Sperlí G. Benchmarking big data architectures for social networks data processing using public cloud platforms. Future Generation Computer Systems 2018; 89:98-109. https://doi.org/10.1016/j.future.2018.05.068

[7] Bixio L, Delzanno G, Rebora S, Rulli M. A flexible iot stream processing architecture based on microservices. Information 2020; 11, 565. https://doi.org/10.3390/info11120565

[8] Zschörnig T, Wehlitz R, Franczyk B. A personal analytics platform for the internet of things-implementing kappa architecture with microservice-based stream processing, in: International Conference on Enterprise Information Systems. SCITEPRESS, 2017, pp. 733–738. https://doi.org/10.5220/0006355407330738

[9] Persson P, Angelsmark O. Kappa: serverless iot deployment, in: Proceedings of the 2nd International Workshop on Serverless Computing, 2017, pp. 16–21. https://doi.org/10.1145/3154847.3154853

[10] Feick M, Kleer N, Kohn M. Fundamentals of real-time data processing architectures lambda and kappa, in: Becker, M. (Ed.), SKILL2018 - Studierendenkonferenz Informatik, Gesellschaft für Informatik e.V., Bonn. 2018, pp. 55–66.

[11] Sanla A, Numnonda T. A comparative performance of real-time big data analytic architectures, in: 2019 IEEE 9th International Conference on Electronics Information and Emergency Communication (ICEIEC), IEEE, 2019, pp. 1–5. https://doi.org/10.1109/ICEIEC.2019.878458

[12] Roukh A, Fote FN, Mahmoudi SA, Mahmoudi S. Wallesmart: Cloud platform for smart farming, in: 32nd International Conference on Scientific and Statistical Database Management, 2020, pp. 1–4. https://doi.org/10.1145/3400903.340169

[13] Fote FN, Mahmoudi S, Roukh A, Mahmoudi SA. Big data storage and analysis for smart farming, in: 2020 5th International Conference on Cloud Computing and Artificial Intelligence: Technologies and Applications (CloudTech), IEEE, 2020, pp. 1–8. https://doi.org/10.1109/CloudTech49835.2020.936586

[14] Andriamandroso ALH, Lebeau F, Beckers Y, Froidmont, E, Dufrasne I, Heinesch B, Dumortier P, Blanchy G, Blaise Y, Bindelle J. Development of an open-source algorithm based on inertial measurement units (imu) of a smartphone to detect cattle grass intake and ruminating behaviors. Computers and electronics in agriculture 2017;139: 126–137. https://doi.org/10.1016/j.compag.2017.05.02

[15] Debauche O, Mahmoudi S, Andriamandroso, ALH, Manneback P, Bindelle J, Lebeau F. Web-based cattle behavior service for researchers based on the smartphone inertial central. Procedia Computer Science 2017;110: 110–116. https://doi.org/10.1016/j.procs.2017.06.127

[16] Debauche O, Mahmoudi S, Andriamandroso ALH, Manneback P, Bindelle J, Lebeau F. Cloud services integration for farm animals' behavior studies based on smartphones as activity sensors. Journal of Ambient Intelligence and Humanized Computing 2019;10: 4651–4662. https://doi.org/10.1007/s12652-018-0845-9

[17] Debauche, O, Mahmoudi S, Mahmoudi SA, Manneback P, Bindelle J, Lebeau F. Edge computing for cattle behavior analysis, in: 2020 Second International Conference on Embedded & Distributed Systems (EDiS), IEEE. 2020, pp. 52–57. https://doi.org/10.1109/EDiS49545.2020.9296471

[18] Debauche, O, Mahmoudi, S, Manneback, P, Tadrist, N, Bindelle J, Lebeau F. Improvement of battery life of iphones inertial measurement unit by using edge computing application to cattle behavior. In: ISCSA2017, 2017.

[19] Petrakis EGM, Koundourakis X. iXen: Secure Service Oriented Architecture and Context Information Management in the Cloud. Journal of Ubiquitous Systems & Pervasive Networks 2021; 14(2): 1-10. https://doi.org/10.5383/JUSPN.14.02.001

[20] Abdelaziz J, Adda M, Mcheick H. An Architectural Model for Fog Computing. Journal of Ubiquitous Systems & Pervasive Networks 2018; 10(1): 21-25. https://doi.org/10.5383/JUSPN.10.01.003

[21] Badidi E. QoS-Aware Placement of Tasks on a Fog Cluster in an Edge Computing Environment. Journal of Ubiquitous Systems & Pervasive Networks 2020; 13(1): 11-19. https://doi.org/10.5383/JUSPN.13.01.002

[22] Pal K, Yansar A-U-H. Convergence of Internet of Things and Blockchain Technology in Managing Supply Chain. Journal of Ubiquitous Systems & Pervasive Networks 2021; 14(2): 11-19. https://doi.org/10.5383/JUSPN.14.02.002