2019 IEEE 28th International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE)

Industrial Internet of Things: Persistence for Time Series with NoSQL Databases

Sergio Di Martino, Luca Fiadone, Adriano Peron, Vincenzo N. Vitale
DIETI
University of Naples "Federico II"
80127, Naples, Italy
sergio.dimartino-adriano.peron2-vincenzonorman.vitale@unina.it
luca.fiadone@gmail.com

Alberto Riccabone
Avio Aero
a GE Aviation Business
80038, Pomigliano D'Arco (NA), Italy
Alberto.Riccabone@avioaero.it

***Abstract*—With the advent of Internet of Things (IoT) technologies, there is a rapidly growing number of connected devices, producing more and more data, potentially useful for a large number of applications. The streams of data coming from each connected device can be seen as collections of Time Series, which need proper techniques to guarantee their persistence. In particular, these solutions must be able to provide both an effective data ingestion and data retrieval, which are challenging tasks. This problem is particularly sensible in the Industrial IoT (IIoT) context, given the potentially great number of equipment that could be instrumented with sensors generating time series. In this study we present the results of an empirical comparison of three NoSQL Database Management Systems, namely Cassandra, MongoDB and InfluxDB, in maintaining and retrieving gigabytes of real IIoT data, collected from an instrumented dressing machine. Results show that, for our specific Time Series dataset, InfluxDB is able to outperform Cassandra in all the considered tests, and has better overall performance respect to MongoDB.**

***Index Terms*—Big Data, Time Series, IoT, Database**

I. Introduction

The spreading of the Internet of Things (IoT) is leading to the constitution of a big, connected and heterogeneous environment. A report by Gartner [1] forecasts that, within 2020, there will be 20 billion connected devices.

In the context of a modern factory, the application of IoT in the industrial context, also known as Industrial IoT or IIoT, implies the usage of many sensors, actuators, control systems, and so on [2] to optimize the production chain. This is often referred to as the Fourth Industrial Revolution. The IIoT allows to take advantage of Cyber Physical Systems (CPS) with many applications, like Anomaly Detection and Predictive Maintenance. Beyond the great advantages of IIoT, there is also a great number of challenges, like data storage, retrieval, processing, visualization and knowledge extraction. In an industrial context, over 4 trillion Gigabytes of data, could be generated in a year [3], and this volume will increase, as observed by McKinsey Institute [4] [5]. As a consequence, many companies entered the Big Data era, characterized by an enormous increment of volume, variety and velocity of data to manage [2] [3]. In such context, the Big Data are known as Industrial Big Data [2] [3]. Beyond the great volume of data to store and retrieve, in the industrial manufacturing, there is a further challenge, namely the data processing. In the Big Data field in addition to the high data production rate, we have to note that great volumes of data are quickly processed. An important aspect to note is that, the information has a *useful time*, i.e. a time range in which it is useful. After that period it becomes useless, or at least not useful for certain tasks. It is known that handling Industrial Big Data with traditional storage solutions, like RDBMSs, may be difficult, as this kind of data can usually be heterogeneous and unstructured [2] [6]. Thanks to the relaxation of some constraints enforced by RDBMSs, NoSQL DBMSs [7] [8] might turn out to be more suitable to handle these Big Data [9], allowing Decision Makers to extract greater value from Big Data [10]. These systems are appreciated for their flexibility, in some kind of applications where great amount of heterogeneous data needs to be handled. NoSQLs are usually classified according to four kinds of data models, with different advantages and limitations [7] [8]. Among them, two types of systems, namely *Wide-Column* and *Document* store, are usually considered more suitable to store and also retrieve huge volumes of data [7]. Furthermore, in an IIoT context,we look at data gathered from each device, as a collection of streams. Each stream can be associated to a specific sensor, so we treat each stream as a Time Series, obtaining collections of Time Series, associated to each device. Looking at data streams of this form, we can choose the best system to handle them, namely the best with respect to storage and retrieval phases, for a specific application. As a consequence, another kind of DBMS can be considered, namely the Time Series Management Systems (TSMS) [11], specifically devised to handle time series data. To evaluate these systems there are some benchmarking, like TPC [12]. Unfortunately there are no standard benchmarks specifically suited for TSMS [13], neither for the specific format in the proposed industrial context.

In this work we presents the results of an empirical study conducted on storage technologies in a real IIoT context. More in detail, by exploiting a dataset of about 60 Gigabytes of real IIoT data, collected from vibrational sensors of a grinding machine, we compared the performances of three NoSQL

978-1-7281-0676-2/19/$31.00 ©2019 IEEE
DOI 10.1109/WETICE.2019.00076

DBMSs, namely MongoDB[1], Apache Cassandra[2] and InfluxData InfluxDB[3]. The experimental protocol for the assessment included the measurement of performances about ingestion of massive amount of data and different types of query which might be used for Predictive Maintenance. Results show that MongoDB is able to provide the best performances for queries on non-temporal indexed attributes. On the other hand, InfluxDB turns out to be on average the more balanced solution, outperforming the competitors in all the other tests, while Cassandra is surpassed by the two competitors in almost all the tests.

The main contribution of this paper to the body of knowledge consists of a practical experience on real IIoT data, on which storage technology is the most suited to handle vibrational data. Thus, practitioners having to face similar challenges can exploit these empirical results in the definition of their technological IIoT architecture.

The rest of the paper is structured as follows: in section II we present some preliminary definitions on Time Series, and an overview of well known DBMS technologies used to handle the Time Series. In section III we describe the industrial use case, together with the investigated technologies and the research question. The section IV contains a description of the experimental protocol. We identified some aspects to look at, during the evaluation of a system, suitable to handle time series. Eventually in section V we present the results obtained from the protocol application, in the mentioned use case. Some final remarks conclude the paper.

II. Definitions and State of Art on IIoT Time Series

Time Series are a widely used format for *changing-over-time* data representation and analysis. Formally, a Time Series is a description of a stochastic process, consisting of a collection of couples $[(p_1, t_1), (p_2, t_2), ..., (p_n, t_n)]$, where p_i is the information collected at the time instant t_i [14]. Thanks to their nature, Time Series are widely used to express data coming from sensors or assets in the IIoT context [11], in order to bind a sensor reading to time. More in detail, since each data stream acquired by a specific sensor has fixed schema, it can be considered a time-ordered collection, or a Time Series, of measures from that sensor. Industrial Time Series are the basis to perform advanced analytic tasks, such as *Anomaly Detection*, in order to check for spurious patterns, or *Predictive Maintenance*, in order to predict failures [15].

In literature, some related works report the use of two NoSQL DBMSs to handle Time Series, namely Cassandra and MongoDB. The former is a Column store system while the latter is a Document store. Both DBMSs can handle great volumes of data, from IoT devices [16] [17], and both can model a discrete Time Series [18] [19]. We have to take into account a last kind of systems, Time Series Management Systems (TSMS) [11]. Designed to handle Time Series data, they have been recently studied [13] to establish a way to evaluate them. In particular we will focus on InfluxDB [20], a recent system optimized for fast, high-availability storage and retrieval of time series, designed for fields like Internet of Things sensor data, real-time analytics and so on. Anyhow, to the best of our knowledge, no vendor-independent benchmark is available, to support a Decision Maker of an IIoT context in the choice of the most suited DBMS for managing data collected from instrumented machinery, in a Industry 4.0 scenario.

[1]https://www.mongodb.com/
[2]http://cassandra.apache.org/
[3]https://www.influxdata.com/

III. The Investigated Scenario of Time Series Storage Systems in IIoT

This study has been carried out within a Predictive Maintenance pilot project of Avio Aero[1]. Avio Aero designs, manufactures and maintains components and systems in the field of civil and military aviation engines. The factory, located in Pomigliano D'Arco, Italy, is involved in a modernization process in the Industry 4.0 direction, thanks to the deployment of a significant number of IIoT devices. The project aims to minimize the downtime of critical assets, by detecting trends in sensed attributes which might indicate an imminent breakage of specific components.

In particular, we focused on a grinder machine, used to smooth the surface of engine blades. The machine was retrofitted with many sensors, mainly monitoring vibrations, RPM, oil temperature and pressure. This is a typical Industrial Big Data scenario. There is *velocity*, since the data is acquired with very high frequency of about 13KHz. Then the *volume*, as data we collect about 2GB of data per hour, per instrumented machine. And the *variety* comes from the context, as the heterogeneous nature of the data is a consequence of the amount of installed sensors.

Within this IIoT environment, data acquired by sensors on board of instrumented machineries can be stored in many ways inside a company data lake. For example, they can be stored in CSV files or in a DBMS, depending on the nature of data, on the available technologies, and on the intended use of these data. The main motivation for the present study came from the problems encountered with the use of a free relational DBMS, namely MySQL, in the management of this stream of heterogeneous data. Indeed, in an initial set-up, also due to the presence of some indexes to improve retrieval performances, the instance of MySQL running on a personal computer with a standard configuration, was not able to write data at the rate generated by the sensors. As a consequence, we identified a set of potential alternative DBMSs, and run an empirical evaluation, using our real IIoT data on them. The selected technologies and the goals of the empirical study are detailed in the following.

A. The Investigated DBMSs

In the following we provide an overview of the tested DBMSs. The selection was done according to some non-

[1]Avio Aero is part of General Electric Aviation

functional requirements posed by *Avio Aero* (e.g. the possibility to run on computers with Windows), integrated by an analysis of the related literature. Indeed, as discussed before, many papers report a successful adoption of Apache Cassandra or MongoDB to handle Time Series (e.g. [16], [18]). Moreover, we were interested in assessing the potential benefits and drawbacks of DBMSs specifically designed for this specific type of data, namely the TSMS [11]. Among them, InfluxDB was the only one able to fulfill all the non-functional requirements. For example, the TSMS Apache Druid is not available for Windows operating systems.

1) Apache Cassandra: Apache Cassandra is a NoSQL DBMS belonging to the Column Store class, created to operate in a distributed environment, providing high availability, fault tolerance and capable to manage huge amount quantity of data. According to the consequences of the CAP Theorem [21], and being a NoSQL store, Cassandra is an AP DBMS, because it allows horizontal partitioning and favouring high availability over data consistency. In this sense, Cassandra wasn't designed to operate in master-slave configuration, then removing any risk derived from the Single Point Of failure. It also balances workloads among peers, i.e. write/read requests can be sent to any node in the cluster: when a client sends a request, it gets connected to the node in the cluster that manages the communication. Unlike some other NoSQL DBMSs, Cassandra has a *SQL-like* querying language, named *CQL* Cassandra Query Language, since its flexible data model can be mapped on a relational one. Actually, the data model underlying Cassandra is represented by a well-defined hierarchy of components: within each node, data are firstly organized in *keyspaces*, that resemble the relational concept of *database*. Keyspaces are the outermost containers for *column-families*, that are similar to a relational *table*. Of course, the atomic component for Cassandra is not the *row*, but rather the *column*, being a *column-oriented* DBMS.

Cassandra can be used for storing Time Series too [18]. In particular, it offers optimizations for storing and retrieving temporal data, such as its reordering according to the timestamp key. In fact, while declaring a table (i.e. the *column-family*), the designer can declare a *composite* primary key that involves *n* attributes (with $n \geq 2$). The first $n-1$ attributes form the *partition key*, used for partition data across nodes, while the last attribute forms the *clustering* key, used to sort data **within** a node. Using the time key as the clustering key will sort data by time, thus favouring retrieval tasks based on time ranges.

2) MongoDB: MongoDB is a NoSQL DBMS, in the *document-oriented* category: data, which can have heterogeneous structure, is stored in a JSON-like format. According to Brewer's CAP Theorem [21], Mongo is an AP database: high Availability and Partition tolerance are effectively provided. Data Model can be easily mapped to the application domain. Indeed, each record is composed of key-value pairs, contained in **BSON** documents, i.e. *binary* JSON, a binary serialization schema for documents. This format is often used for Time Series storage [18], using different techniques (i.e. one document per record, or a document for values recorded in a given time frame). Furthermore, MongoDB provides some optimization for temporal data, such as *data sharding*, i.e. data distribution, across nodes, by a defined key value, like time. Mongo's sharding is conceptually not far from Cassandra *partitioning*. In fact, the sharding only defines how data is split across nodes, while using a distributed cluster for data storing. This provides no guarantee on data sorting inside each single node of the cluster.

3) InfluxDB: Within the wide family of NoSQL DBMSs, a new branch is defined specifically to handle Time Series. Among them, InfluxDB [20] is one of the most promising, being designed from the scratch to handle this specific kind of data, rather than being an adoption of a generic data schema to this particular case. Since in a Time Series the primary aspect is ordering, in Influx data is physically ordered by time. Another key feature is the ability to define, for a given data group, some *retention policies*, i.e. specific rules to manage old data, like for example deleting all the data older than three months or replacing them with some aggregated values. Another key feature of InfluxDB is the possibility to define *continuous queries*, i.e. a query tool able to work continuously on a stream of input data, rather than in a batch fashion. As a final remark, InfluxDB uses an *hybrid* data model, where the values are stored like in a row-based DBMS, while indexes, named **tags** in InfluxDB, are stored like in a column-based DBMS.

B. The Research Questions

A modern factory is a challenging scenario for data storage, due to the great number of instrumented machines producing huge data volumes. Since the acquired information will be used for critical business tasks, the time needed to store, retrieve and process the data is limited. Storage and retrieval are thus crucial phases in the IIoT [9] [10], and are the very first steps after the acquisition phase.

In our investigation, the first choice concerned the aspects to consider, in order to identify the more appropriate DBMS [11]. Right after the acquisition, there is the *storage* phase for which, given the potential volume of the data stream, there are two critical aspects: *ingestion time* and *Disk usage*. On the other side, the *retrieval* phase can be an expensive operation, and since it is crucial for subsequent analytics tasks, an important aspect to consider are *Query performances*, in terms of recovery time. The first kind of queries to look at are the *time-based* ones, since the time is the primary aspect in this kind of data, the time based recovery needs to be as fast as possible. Then we have the *attribute specific* queries, another fundamental aspect, since beside time-value there are often other attributes enriching the information given by each reading. Given those observation, the research question we investigated is *Which storage technology is able to provide the better performances in our IIoT environment?*

IV. Experimental Protocol

Given the high frequency data streams collected from the machineries, we aimed at experimentally understand benefits and drawbacks of Time Series DBMS over general purpose NoSQL ones, already reported to be successfully used to handle this kind of data.

Consequently, we tested various aspects of the above mentioned DBMSs, with real data we got from an instrumented machine. As done in similar works (but in other contexts), we defined an experimental protocol aimed at assessing three key performance dimensions:

- **Batch-Ingestion time** i.e. the time need to load a massive amount of data in the database, measured in seconds. Through this measure, we can understand the volume of data that each DBMS is capable to handle during the ingestion phase.
- **Retrieval time** in seconds, measured on some queries we used for subsequent data analyses. Considering both ingestion and retrieval performances is fundamental, since it is very common the risk to strongly optimize one of the two (for example defining indexes), slowing down the other.
- **Disk usage** in Gigabytes, intended to measure the ability of the DBMS to efficiently store massive amounts of data generated by the IIoT continuous data streams.

For all the experiments, we used records with the following fixed data schema:

1) **Timestamp** (timestamp): the time instant of the acquisition, expressed as standard Unix epoch time in nanoseconds;
2) **Sensor** (string): the name of sensor collecting the measurement;
3) **Value** (floating point): the value of the measurement;
4) **Program** (integer): the *Part Program*, i.e. the processing procedure executed by the grinder, represented by an Id;
5) **Subprogram** (integer): the id of a specific subroutine associated to the part program;
6) **Tool** (integer): the tool type, on which the sensor was mounted.

Despite the fact that three analysed DBMSs have different technical characteristics, we tried to optimize them in a comparable way. The details of the configurations for each of the tests is detailed in the following.

A. Configurations for Ingestion Tests

We performed two types of ingestion tests. In the first one, we considered an empty database in which we inserted 300 million points, representing about one day of registration for a single high frequency sensor. In the second one, we simulated a scenario of a database already containing a massive amount of data, and, on it, we performed again an ingestion of 300 million points. The rationale behind these configurations is to stress the way indexes are managed.

The ingestion tests were performed with two configurations of indexes for each DBMS. In both the configurations, we defined in MongoDB and Cassandra an index on the time attribute, since an index on time is native in InfluxDB. In the first configuration, we forced the DBMS to index also attributes Program, Subprogram and Tool of the previous list, while in the second one, only the timestamp is indexed.

We ran all the experiments five times for each DBMS, rebooting each time the computer to minimize biases due to caching. For each experiment, we had a script measuring the time, in seconds, required for the ingestion.

B. Configurations for Retrieval Tests

To analyse the retrieval performances, we employed three real queries. The first one includes a **Time-based filtering** to retrieve all the data regarding a specific processing procedure, giving its starting and ending time, which are known from an external application. The second query retrieves data for all the processing procedure belonging to a given type of Program. The last one does the same, for a SubProgram. In all the cases, no projection is applied, as the three queries return all the six attributes above described. Let us note that, as the Programs for the considered machine are just three, its selectivity is extremely low.

Before executing these queries, the DBs were populated with two days of data, corresponding to about 600M points of real data. We ran all the retrieval experiments twenty times for each DBMS, rebooting each time the computer to minimize biases due to caching. For each experiment, we used a script to measure the time, in seconds, required for the retrieval.

C. Configurations for Disk Usage Tests

At the end of each ingestion test, we measured also the *Disk Usage*, in Gigabytes for both the configuration, with four and with one index. This is a useful measure on how the DBMS can store in an efficient way, both data and indexes.

D. Hardware Configurations

To eliminate temporal variability (and thus biases) due to the way sensors can produce data, for all the experiments we simulated the acquisition phase by replaying a stream of recorded data. Thus, we used two computers. The first one is the DB server, equipped with a hyper-threaded exa-core Intel Xeon, 64GB RAM, 1TB SSD. The computer replaying the stream is equipped with an Intel i5, 16GB RAM, and 256GB SSD. The two computers are directly connected through a gigabyte LAN. The tests were performed in isolation to estimate the performance of each chosen systems, without outer interferences, on a single node configuration.

V. Result and Discussion

In this section we discuss the results obtained in the experiments.

A. Ingestion Performances

In Figure 1 are reported the average ingestion times for InfluxDB, Cassandra and MongoDB, for each mentioned configuration and load. From the image, we can see that InfluxDB clearly outperforms the competitors, showing the

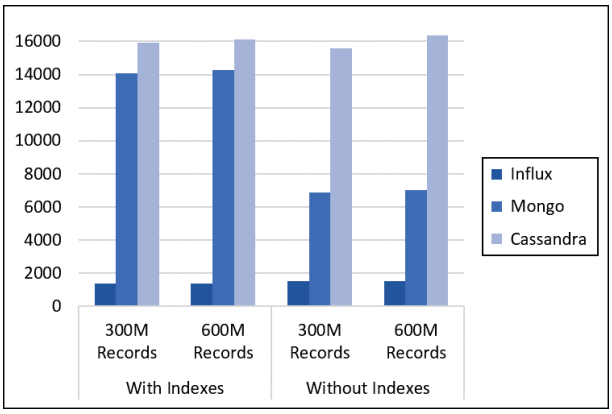

Fig. 1. Time, in seconds, to import 300M and 600M records for each DBMS, with and without indexes.

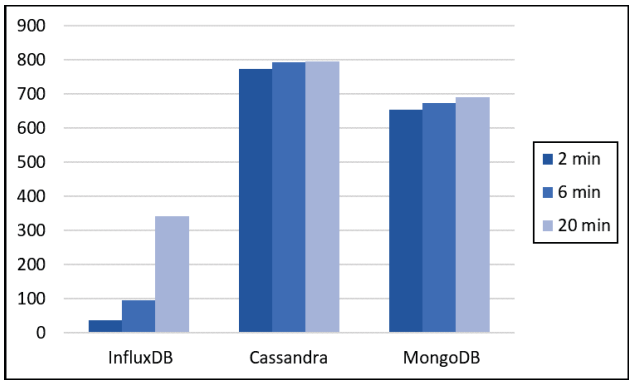

Fig. 2. Average Execution Time of Queries with Time-based Filtering, in Seconds, for three different amounts of data.

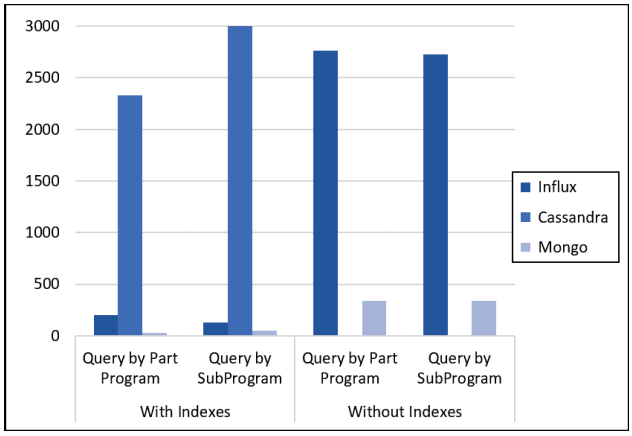

Fig. 3. Average Execution Time of Queries by Non-Temporal Attribute Filtering, in Seconds.

best performances for all the configurations. This is due to the fact that InfluxDB is highly optimized for ingestion tasks, natively including buffering algorithms. MongoDB show lower performances, and a very sensible increase of running time when indexes are defined. Even if this is a known trend, we did not expected that the use of indexes would have doubled the ingestion times. Cassandra, on the other hand, ingests $\approx$ 300 millions points in 4.5 hours, at least, no matter about parameters, being thus almost 10 times slower than InfluxDB in all the configurations.

It is also important to highlight that the configuration of InfluxDB without indexes has slightly worse performances: this is probably due to InfluxDB storage engine, that, being a columnar store on indexed attributes, i.e. tags, saves different tag values only once, then storing only references to all records containing that value for that tag. On the other hand, InfluxDB will allocate space for all the $\approx$ 600 millions records. In an IIoT context this is a very good result, since it shows how a specific DBMS optimized for Time Series data, as InfluxDB, is capable to handle a huge quantity of data, outperforming *general-purpose* solutions.

B. Retrieval Performances

From Figure 2, we can see the results for the time-based queries. Recorded data is about three principal work programs performed by the industrial asset, with a fixed length, namely 2, 6, and 20 minutes. Retrieval operations have then been performed for all these programs, thus resulting in result-sets with, respectively, $\approx$ 1.5, 5, 16 millions points. MongoDB and Cassandra show, again, lower performances in all cases, but pretty stable among the different configurations. On the other hand, InfluxDB takes advantage from its sorting of the data by timestamps, returning 16 millions records, for the longest program, in less than half the time required by the other two DBMSs.

These results have been obtained filtering data using the time attribute only. In this sense, since timestamps are always indexed, there is no *no-indexes* case to report. Moving on, in figure 3 are reported the *Program-based* queries results. In particular, data has been searched while fixing the value of some attributes, or of all of them, thus increasing query selectivity. In 4 cases (with the lowest selectivity rates) Cassandra service crashed, being unable to return any result. The other two cases, i.e. when all *non-vibrational* fields have been used as filters, or the most selective one only (i.e. *subprogram*), Cassandra was able to return correct results, but showing very low performances. On the other hand, as we can see, Mongo and InfluxDB have better performances in all the measured cases, even if, for InfluxDB especially, with no definition of indexes/tags there is a considerable performance decrease. Indeed, in this configuration, MongoDB is able to provide the best performances for retrieval from 600 millions points, outperforming the two other competitors.

C. Disk Performances

In Figure 4 are reported the *Disk Occupancy* measurements from both configurations, *index* and *no-index*, for each system. Let us note that for InfluxDB we talk about ”tags” not ”index”, since tag values set is limited, while an index can have a very big value set.

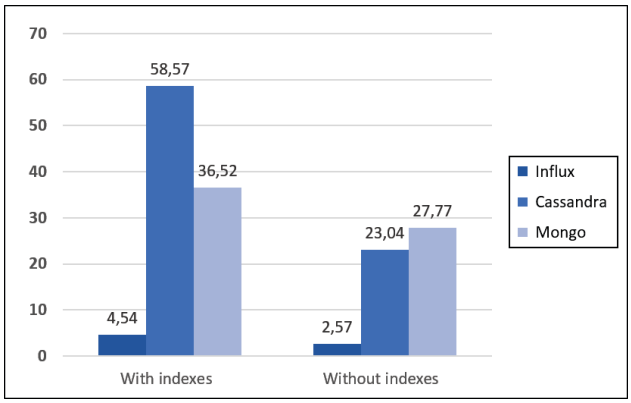

Fig. 4. Disk space need to store 600M points, in Gigabytes.

As we can see, InfluxDB has the best performances also in terms of space. This makes it more suitable to handle high frequency data in contexts where the sources number is very high. The great compression rate is due to the presence, within the InfluxDB storage engine, of several compression techniques and algorithms, for each different data type. In this way, needed space for 600 millions of points is $\approx$ 4.5 GB, or 2.7 GB if tags are stored as fields. Cassandra and MongoDB need by far more space on disk for data storage, that increases when indexes are defined. This is true especially for Cassandra, whose indexes need more space than the data itself.

D. Discussion

Although Cassandra is a successful general purpose NoSQL DBMS, capable to handle great volumes of data and, already used for Time Series storage, on a single node configuration it was always by far outperformed by InfluxDB. While MongoDB showed good performances during the retrieval phase, performing better than InfluxDB in some cases, during the ingestion phase, InfluxDB performed better than MongoDB. In general, InfluxDB seems to be able to provide more balanced performances, between ingestion and retrieval, than MongoDB

VI. Conclusion

In the IIoT scenario, the amount of data generated by instrumented machinery can be huge, clearly falling in the Big Data class. The most of this data is often composed of heterogeneous Time Series, that can be challenging for a Decision Maker interested in adopting a suitable DBMS to handle them.

In this paper we presented an empirical analysis we conducted on three NoSQL DBMSs, to investigate the achievable performances in terms of ingestion, retrieval and required storage space, for IIoT data. In particular, we measured the performances of two widely employed DBMSs, namely Apache Cassandra and MongoDB, and of a Time Series Db, i.e. InfluxDB, in handling a dataset of about 600 million records (about 60 GB), collected from an instrumented grinding machine. With our dataset, MongoDB gave the best performances for queries on non-temporal indexed attributes, while Cassandra is outperformed by the two competitors in almost all the tests and turned out to be unstable on a single node configuration. Lastly InfluxDB turns out to be on average the more balanced solution, outperforming both competitors under storage aspects, and providing the best performances on ingestion and time-based queries. In conclusion, given our IIoT use case, InfluxDB turned out to be the most advisable solution.

Since this is a first experimental set-up, many possible evolutions of this evaluation are envisioned. We need to run experiments on a cluster, rather than on a single node, to measure the impact of data distribution and parallelization. It would also be interesting to include in the evaluation a Relational DBMS. Finally, it would also be interesting to consider different data schemas.

References

[1] Gartner, "Leading the iot," 2017.
[2] D. Mourtzis, E. Vlachou, and N. Milas, "Industrial big data as a result of iot adoption in manufacturing," *Procedia Cirp*, vol. 55, pp. 290–295, 2016.
[3] GE, "The rise of industrial big data," 2012.
[4] M. G. Institute, "Big data report," *Mckinsey Global Institute*, 2011.
[5] M. Baily and J. Manyika, "Is manufacturing "cool" again," *Project Syndicate*, vol. 21, 2013.
[6] S. Yin and O. Kaynak, "Big data for modern industry: challenges and trends [point of view]," *Proceedings of the IEEE*, vol. 103, no. 2, pp. 143–146, 2015.
[7] A. Davoudian, L. Chen, and M. Liu, "A survey on nosql stores," *ACM Computing Surveys (CSUR)*, vol. 51, no. 2, p. 40, 2018.
[8] F. Gessert, W. Wingerath, S. Friedrich, and N. Ritter, "Nosql database systems: a survey and decision guidance," *Computer Science-Research and Development*, vol. 32, no. 3-4, pp. 353–365, 2017.
[9] J. S. Van der Veen, B. Van der Waaij, and R. J. Meijer, "Sensor data storage performance: Sql or nosql, physical or virtual," in *2012 IEEE fifth international conference on cloud computing*. IEEE, 2012, pp. 431–438.
[10] J. Bhogal and I. Choksi, "Handling big data using nosql," in *2015 IEEE 29th International Conference on Advanced Information Networking and Applications Workshops*. IEEE, 2015, pp. 393–398.
[11] S. K. Jensen, T. B. Pedersen, and C. Thomsen, "Time series management systems: A survey," *IEEE Transactions on Knowledge and Data Engineering*, vol. 29, no. 11, pp. 2581–2600, 2017.
[12] TPC. TPC. [Online]. Available: http://www.tpc.org/
[13] R. Liu and J. Yuan, "Benchmark time series database with iotdb-benchmark for iot scenarios," *arXiv preprint arXiv:1901.08304*, 2019.
[14] S. Das, *Time Series analysis*. Princeton University Press, Princeton, NJ, 1994.
[15] P. Esling and C. Agon, "Time-Series Data Mining," *ACM Computing Surveys*, vol. vol. 45, November 2012.
[16] A. Lavin and D. Klabjan, "Clustering time-series energy data from smart meters," *Energy efficiency*, vol. 8, no. 4, pp. 681–689, 2015.
[17] Y.-S. Kang, I.-H. Park, J. Rhee, and Y.-H. Lee, "Mongodb-based repository design for iot-generated rfid/sensor big data," *IEEE Sensors Journal*, vol. 16, no. 2, pp. 485–497, 2016.
[18] D. Ramesh, A. Sinha, and S. Singh, "Data modelling for discrete time series data using cassandra and mongodb," in *2016 3rd International Conference on Recent Advances in Information Technology (RAIT)*. IEEE, 2016, pp. 598–601.
[19] A. Chebotko, A. Kashlev, and S. Lu, "A big data modeling methodology for apache cassandra," in *Big Data (BigData Congress), 2015 IEEE International Congress on*. IEEE, 2015, pp. 238–245.
[20] S. N. Z. Naqvi, S. Yfantidou, and E. Zimányi, "Time series databases and influxdb," *Studienarbeit, Université Libre de Bruxelles*, 2017.
[21] S. Gilbert and N. Lynch, "Perspectives on the cap theorem," *Computer*, vol. 45, no. 2, pp. 30–36, Feb 2012.