Benchmarking management techniques for massive IIoT time series in a fog architecture

Sergio Di Martino* and Adriano Peron

DIETI,
University of Naples "Federico II",
Naples 80127, Italy
Email: sergio.dimartino@unina.it
Email: adriano.peron2@unina.it
*Corresponding author

Alberto Riccabone

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

Vincenzo Norman Vitale

DIETI,
University of Naples "Federico II",
Naples 80127, Italy
Email: vincenzonorman.vitale@unina.it

Abstract: Within the Industrial Internet of Things (IIoT) scenario, the online availability of a growing number of assets in factories enables the collection of vast amounts of data. Each asset produces time-series collections that must be handled with proper techniques while providing effective ingestion and retrieval performance in complex architectures, maintaining compliance with company and infrastructure boundaries. In this paper, we describe an experience in the management of massive time-series, conducted in a plant of Avio Aero. Firstly, we propose a fog-based architecture to ease the collection and analysis of these massive datasets. Then, we present the results of an empirical comparison of four DBMSs (PostgreSQL, Cassandra, MongoDB, and InfluxDB) in the ingestion and retrieval of gigabytes of real IIoT data. In particular, we tested different DBMS features under different types of queries. Results show that InfluxDB provides very good performance, but PostgreSQL can still be an interesting alternative.

Keywords: big data; time series; IIoT; fog architecture; TSMS; NoSQL database; relational database; benchmarking.

Reference to this paper should be made as follows: Di Martino, S., Peron, A., Riccabone, A. and Vitale, V.N. (2021) 'Benchmarking management techniques for massive IIoT time series in a fog architecture', *Int. J. Grid and Utility Computing*, Vol. 12, No. 2, pp.113–125.

Biographical notes: Sergio Di Martino received his MSc (2001) and PhD degree (2005) in Computer Science at the University of Salerno, Italy. He is currently Associate Professor in Computer Science at the University of Naples Federico II, Italy, since 2015. His research interests focus on knowledge discovery and visualization from complex spatio-temporal datasets.

Adriano Peron is Full Professor in Computer Science at the Department of Electrical Engineering and Information Technologies of the University of Napoli "Federico II". He got a PhD in Computer Science in the Department of Computer Science from the University of Pisa. He is currently the director of the BS/MS Computer Science Program. In the last years, the research activity has been mainly focused on formal techniques for system specification and verification; model checking for interval and point based temporal logics; time series storage and analysis; timeline-based planning.

Alberto Riccabone holds a PhD in Aerospace Engineering. He joined Avio Aero in 2007 and he covered different roles in the Digital Technology team, supporting both the Engineering and the Supply Chain functions. From the Product Lifecycle Management to Big Data and Industry 4.0, he has worked on different technologies and processes typical of the aerospace industry. He's now leading the Engineering Digital team and he's the focal point for the Digital R&D initiatives.

Vincenzo Norman Vitale is a PhD student in Information Technology and Electrical Engineering at the University of Naples Federico II, Italy, from which he also graduated cum laude with a MSc degree in Computer Science in 2017. His research interests include spatio-temporal data storage and analysis; Human-Computer Interaction application in cultural heritage; applications of computer science to Intelligent Transportation Systems, smart cities and smart factories.

This paper is a revised and expanded version of a paper entitled 'Industrial Internet of Things: persistence for time series with NoSQL databases' presented at the '2019 IEEE 28th International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE)', Capri (Naples), Italy, 12–14 June 2019.

1 Introduction

The application of most recent Information and Communication Technologies (ICT) to industrial production has led to the concept of *Industry 4.0* (Lasi et al., 2014; Hermann et al., 2016). Its key objective is to obtain a *Smart Factory* (Wang et al. 2016; Chen et al. 2017), aimed at optimising the production chain by means of the digital integration of all the entities, tools and steps involved in the production process. This is often referred to as the *Fourth Industrial Revolution*, after the introduction of Machines (First Revolution), Electrification (Second) and Automation (Third).

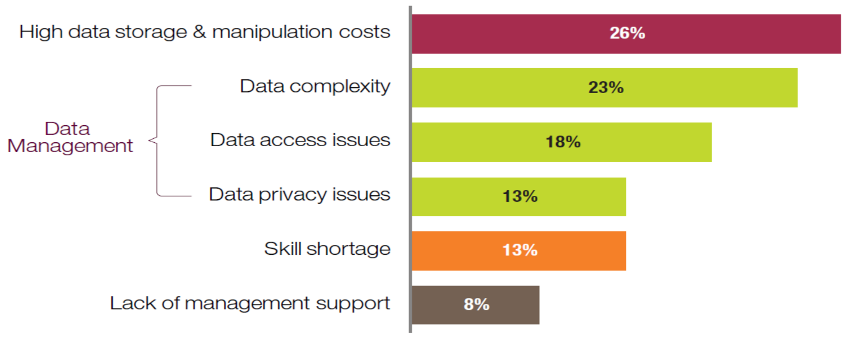
Although there is not a generally accepted definition of *Industry 4.0* (Hermann et al., 2016), there are anyhow some common underlying technologies, namely Cyber-Physical Production Systems (CPPS), Internet of Things (IoT), Cloud Computing, Big Data and Advanced Analytics Techniques (Zhou et al., 2015). Among them, the *Industrie 4.0* Working Group,[1] identified the IoT as the key enabler for the new revolution (Kagermann et al., 2013).

IoT is paradigm based on the digital interconnection of everyday objects, with the goal of letting a computer to sense information without human intervention (Atzori et al., 2010; Gubbi et al., 2013; Vermesan and Friess, 2013). When applied to an industrial setting, it is known as *Industrial IoT*, or IIoT, whose goal in to collect status data and control industrial machineries. The spreading of IoT/IIoT is leading to the constitution of a huge, connected and heterogeneous digital environment. Indeed, a total of 1 trillion interconnected sensors is foreseen by 2030 (Chen et al., 2014), generating data at an unprecedented rate. In an industrial context, over 4 trillion Gigabytes of data, could be generated in a year (GE, 2012) and this volume will increase (Baily and Manyika 2013). Moreover, these massive IIoT data sets are usually heterogeneous, with variable resolution, often asynchronous, stored in different formats (Yin and Kaynak, 2015), and generated as a stream of data. The combination of IoT/IIoT along with Big Data analytics showed great possibilities for innovation and for business growth (Bhattarai et al., 2019), but also a great number of ICT challenges, like data storage, retrieval, processing, visualisation and knowledge extraction (Marjani et al., 2017).

To date, these challenges are slowing the diffusion of Industry 4.0 technologies, as reported by Capgemini Consulting (2015). The key issues limiting the implementation of Big Data Analytics solutions in industry are reported in Figure 1, where we can note that the high costs of storage and manipulation of massive data sets is a key impediment.

Some previous works reported that handling these heterogeneous and unstructured data sets might be difficult with traditional storage solutions based on Relational DataBase Management Systems (RDBMS) (e.g., Mourtzis et al., 2016). On the other hand, NoSQL DBMS (Davoudian et al., 2018; Gessert et al., 2017) were reported to be more effective in handling these Big Data (Van der Veen et al., 2012; Bhogal and Choksi, 2015), thanks to the relaxation of some constraints of the relational data model. Moreover, since data streams collected from machinery are often represented as Time Series, another kind of DBMS was investigated in IIoT, namely the Time Series Management Systems (TSMS) (Jensen et al., 2017), specifically devised to handle time series data. Nevertheless, even if in the literature there are many benchmarks and researches on DBMS performances for storage and retrieval (e.g., TCP, n.d.), but, as reported also by Liu and Yuan (2019), none of them is representative of the typical workloads induced by IIoT data streams. Moreover, with the introduction of Fog computing paradigm, part of computation and storage moves closer to data source (i.e., IoT devices), reducing network latencies, making data transfer to storage nodes more efficient (Bonomi et al. 2012; Dastjerdi and Buyya, 2016) and increasing responsiveness in time-critical manufacturing tasks. As a consequence, nowadays an IIoT solution architect, in charge of choosing an architecture and a DBMS to handle IIoT data streams, might have trouble identifying the most suitable storage solution for his/her needs, due to the lack of related works in the literature.

Figure 1 Top challenges in implementing big data analytics for industry 4.0 (Capgemini Consulting, 2015)



To fill this gap, in this work we present the results of an empirical study we conducted to benchmark different storage technologies, within a real IIoT Fog-based architecture we deployed in a factory producing parts of aircraft engines. More in detail, by exploiting a data set of about 60 Gigabytes of real IIoT data, collected from vibrational sensors of a grinding machine, we compared the storage and retrieval performances of four DBMS: one based on the relational model, i.e., PostgreSQL, two based on NoSQL paradigm, namely MongoDB[2], Apache Cassandra[3] and one specifically designed for Time Series, namely InfluxData[4]. The experimental protocol for the assessment included the measurement of performances about ingestion of massive amount of data, together with typical queries significant for subsequent Data Analytics for the specific industrial context.

The main contributions of this paper are:

- A description of the Fog-based architecture we deployed to collect massive sensor data from multiple industrial machinery;
- An empirical performance comparison, among three data storage paradigms, namely Relational, NoSQL and TSMS, on a real, massive IIoT data set composed of Time Series.

We believe that practitioners facing similar challenges in the field of Industry 4.0 can exploit our empirical results to make more informed decisions in the definition of the architecture and on the storage technique to employ.

The rest of the paper is structured as follows: in Section 2, we present some preliminary definitions on IIoT Time Series, and an overview of well-known DBMS technologies used to handle them. In Section 3, we describe the industrial use case we faced. In Section 4, we describe the investigated technologies and the research question. Section 5 contains a description of the experimental protocol and Section 6 results obtained from the protocol application, in the mentioned use case. Some final remarks conclude the paper.

2 Definitions and state of art on IIoT time series management

In the majority of IIoT scenarios, connected machineries generate huge amount of data, mostly as streams, collected from (heterogeneous) monitoring sensors (Jensen et al., 2017). These streams are usually represented as *Time Series*, a widely used solution to model *changing-over-time* data (Brockwell and Davis, 2016; Madsen, 2007).

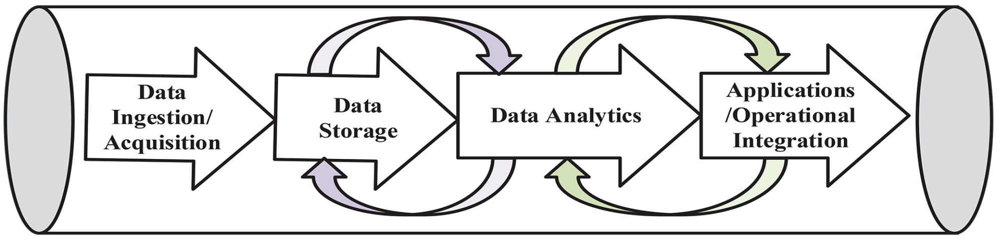
Formally, a Time Series is a collection of couples $\left[(p_1,t_1),(p_2,t_2),...,(p_n,t_n)\right]$, where p_i is the information collected at the time instant t_i (Das, 1994). In the context of IIoT, Time Series are mostly used to bind a measure collected from a sensor to the time instant it was acquired (Jensen et al., 2017). Thus, each data stream coming from a specific sensor can be considered a time-ordered collection, or a Time Series, of measures from that sensor. These Time Series are then the basis to perform advanced analytic tasks in the context of Industry 4.0, like *Anomaly Detection*, looking for spurious patterns, or *Predictive Maintenance*, aiming at predicting machinery breakage (Esling and Agon, 2012).

In a Smart Factory, as long as an instrumented machinery is running, its sensors generate these Time Series, which must be stored somewhere, to enable the above cited analytical tasks (Ren et al., 2011). Since sensor sampling rates may be very high, depending on the physical phenomenon to monitor, the acquired streams may lead to very large data sets (Bao et al., 2012), whose nature matches with the definition of *Big Data*, due to Volume, Velocity and Variety (Lee et al., 2015). Indeed, it is not uncommon that each instrumented machinery generates data in the range of hundreds of Megabytes, or even Gigabytes per hour. Since a Smart Factory may have hundreds of these machineries, this becomes a challenging scenario for data storage and retrieval (Lee et al., 2015; Van der Veen et al., 2012; Bhogal and Choksi, 2015). Moreover, in the context of Industry 4.0, there are also further aspects to deal with (Yin and Kaynak, 2015; Mourtzis et al., 2016), like company constraints, data usage limitations due to sensitive nature of data, network security policies and more.

Moving on, in order to enable analytic tasks, there are some key steps described by the general scheme in Figure 2.

The *Data Ingestion/Acquisition* is about data collection from IIoT devices, since each device has different format and sensing rates. Then *Data Storage* phase deals with making persistent the acquired data, to enable its usage for subsequent *Data Analytics* phases.

Figure 2 Key stages of big data analytics (Bhattarai et al., 2019)


Data Ingestion/ Acquisition
Data Storage
Data Analytics
Applications /Operational Integration

Focusing on the Data Storage step, the database can be located locally in the production plant, with all the related management commitments, or remotely, in a company Data Lake, or in the Cloud. Cloud computing (Mell et al., 2011) has been for years the main computing paradigm associated with IoT (Biswas and Giaffreda, 2014). Nevertheless, it can introduce some network related problems, that can be critical in the context of IIoT. Indeed, data transmission to a remote Cloud from the production plant may be subject to delays, also due to the distance to the data centres, to the routing through multiple gateways, and so on (Yi et al., 2015). Thus, even if the Cloud relieves a company from the burden of ICT management, still some problems might arise for latency-sensitive applications (Bonomi et al., 2012), like anomaly trends detection. In such cases, to solve these issues, a system architecture based on the *Fog* computing can be favourable. Indeed, Fog computing is an architectural paradigm based on the exploitation of additional ICT resources, lying between IoT/IIoT devices and remote data centres (Simmhan, 2018; Bonomi et al., 2012; Dastjerdi and Buyya, 2016). Usually, a Fog node processes and optionally stores data locally, sending mostly the results of data aggregations (usually by far smaller than the raw sensed data) to the Cloud (Kudo, 2019). Clearly, a Fog node has significantly less computational power and storage than the Cloud, but, being closer to the data sources, it has also lower network latency, enabling time-sensitive applications. With such architectural paradigm, if the Fog node is also devoted to store sensed data, due to the limited computing resources, the choice of the most suited DBMS is a crucial point, as reported in some previous investigations (e.g., Di Martino et al., 2019). In the following we provide a description of some common storage solutions for IIoT.

2.1 Data management solutions for IIoT

In the literature, many works report the use of NoSQL DBMS to handle Time Series, also for IoT/IIoT (e.g., Lavin and Klabjan, 2015; Kang et al., 2016; Ramesh et al., 2016; Chebotko et al., 2015). NoSQL solutions are a family of DBMS based on data models different from the relational one. In contrast to traditional RDBMS, NoSQL ones can operate also without structuring data into a fixed schema, such as tables and references Davoudian et al. (2018) and Gessert et al. (2017). This can ease handling heterogeneous data, that can also be effectively distributed over multiple nodes. On the downside, they mostly lack in supporting ACID properties, providing instead a concept called *eventual consistency*, i.e., a delay in the propagation of data updates across multiple nodes might exists (Leavitt, 2010; Stonebraker, 2010). Within this family of DBMS, there is also a new category, namely the *Time Series Management Systems* (TSMS) (Jensen et al., 2017). Designed specifically to handle Time Series data, they have been recently proved to be effective in IoT scenario (Liu and Yuan, 2019).

Some studies report a comparison of DBMS performances on large data set, in the context of IoT/IIoT, with a particular focus on NoSQL systems (Hendawi et al., 2019; Syafrudin et al., 2018; Pereira et al., 2018).

Anyhow, to the best of our knowledge, no vendor-independent benchmark, nor studies in literature are available to support a system architect of an IIoT context in the choice of the most suitable DBMS for managing data collected from instrumented machinery, in a Fog-based Industry 4.0 scenario.

3 The deployed Fog-based architecture for massive IIoT time series management

In this section, we describe the investigated Industry 4.0 scenario and the system architecture we deployed.

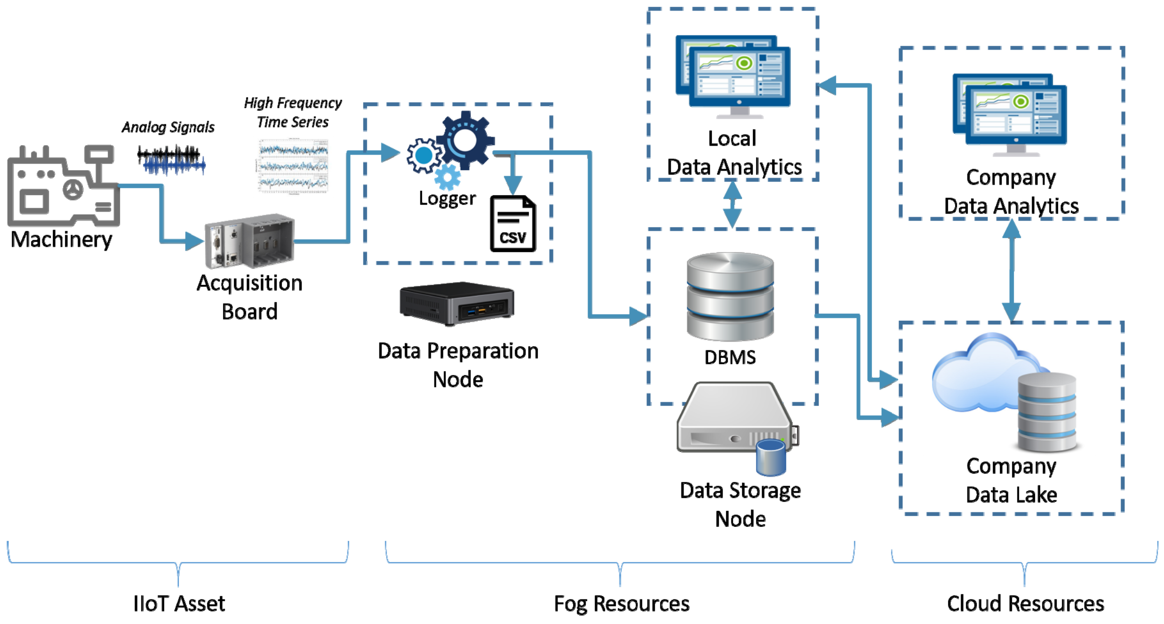
3.1 The investigated IIoT scenario

The current study has been carried out within a pilot project of *Avio Aero*, part of *General Electric Aviation*. It is a company that designs, manufactures and performs maintenance of components and systems in the field of civil and military aviation engines. The company is undergoing many initiatives within the Industry 4.0 context in many of its production plants, including instrumentation of machineries with IIoT sensors/devices.

The current investigation is part of a project for Predictive Maintenance, aimed at minimising the downtime of critical machineries by detecting trends that might anticipate a forthcoming breakage. In particular, we focused on grinder machines, used to smooth the surface of aircraft engine blades. These grinders have been retrofitted with some sensors, mainly to monitor vibrations of core parts, rounds per minute of the spindle, and oil temperature/ pressure. This use case turned to be challenging from a data management perspective, since it features the four characteristics of Industrial Big Data:

- *Velocity*: Data is acquired at high frequency. The vibration sensors have a sampling rate of about 13 KHz, to allow the identification of rapid and anomalous changes. Each instrumented machinery has at least four of these high frequency sensors, thus generating more than 52,000 records per second.
- *Volume*: In a typical working day, composed of three shifts, each vibrational sensor can produce up to 300 million data points, corresponding to roughly 30 Gigabyte per day.
- *Variety*: Data are collected from heterogeneous sources, like high-frequency vibrations, low-frequency temperature, process-specific information from numeric control, etc., Data need to be integrated, to enable identification of critical points in subsequent Data Analytics steps.
- *Veracity*: Reliability of data interpretation heavily depends on the temporal alignment of all the data sources. For example, since the duration of a subroutine inside a working program might be a few milliseconds to seconds, a temporal misalignment of data might lead to potentially wrong phenomena understanding.

Figure 3 The deployed Fog-based architecture for massive time series management



Furthermore, there are key Company requirements for the data management:

1 The Company has a distributed data management protocol, featuring both local analytics activities in each production plant, and remote general analytics tasks, in the Company Headquarters.

2 Data collected from IIoT devices are crucial, and redundancy of their storage is required.

3 Data collected from IIoT devices must be locally stored, to feed Data Analytics tools in the production plant.

4 Data collected from IIoT devices should be available to remote Company Data Analysts, implying that it must be suitably replicated in a Company *Data Lake*. This is usually done off-line, with some notable delay (e.g., data copied during the night).

3.2 *Architecture description*

We started with the design of a proper distributed system architecture, compliant with the above-described requirements. The resulting solution, depicted in Figure 3, falls in the category of *Fog Architectures*, as some preliminary processing is done close to the IIoT data sources.

With reference to Figure 3, we have the *Instrumented Machinery* under monitoring, whose sensors produce multiple analogue signals. Those signals are fed to an *Acquisition Board*, physically wired to the sensors, whose role is to discretise received signals, producing high frequency Time Series. The Board is connected via an Ethernet cable to a *Data Preparation Node*, which is in our case a simple and economic *mini Personal Computer* with basic computational and storage resources. The Mini PC receives the Time Series and performs two tasks: (I) it pushes the collected data to the *Data Storage Node*, and (II) it keeps a local copy of the received Time Series, in the CSV format, avoiding the risk of a single point of failure, thus fulfilling Requirement 2 of Sub-section 3.1. Given the limited computing capabilities of a Mini PC, one Data Preparation Node is able to serve usually a single instrumented machine.

The Data Preparation Node and Data Storage Node are located inside the factory LAN, in order to reduce latency for local analytics tasks. The Data Storage Node includes a DBMS to store, in a more structured fashion, the collected data. This node makes available the data in the local network of the plant, so any local *Data Analytics Tool* can access it at any moment.

Data are then replicated to the *Company Data Lake* for further massive elaboration tasks. This step is done when the load of the network is low, like during the night. After this replication, the CSV files are deleted from the Data Preparation Nodes, since now they are replied on the Data Storage Node and on the Data Lake. One Data Storage Node can serve multiple Data Preparation Nodes.

4 The research question and the investigated DBMS

A potential bottleneck of the proposed architecture, is the DBMS of the Data Storage Node, which is flooded by massive amounts of sensed data. Indeed, such a DBMS should be able to store data in a way that is efficient for massive ingestions and also for fast retrieve to empower local Data Analytics. Considering the potential volume of the data streams, there are three critical aspects to deal with: *ingestion performance*, *disk space usage*, and *retrieval performance* (Jensen et al., 2017).

In a preliminary attempt, a solution based on a free relational DBMS, namely MySQL, was implemented. This solution unfortunately failed, since the instance of MySQL, was not able to write data at the rate generated by the sensors. This motivated us to empirically assess which DBMS is able to provide better performances in our IIoT environment, and thus to formulate the following Research Question: *What is the most effective DBMS in terms of ingestion, retrieval and storage space, for massive IIoT Time Series?*

To identify the DBMS to benchmark, the Company posed us two constraints on the pool of candidates: (I) it must run on Windows platforms, for IT policy compliance and (II) at least in a first assessment, the costs are a important factor. Thus, ideally, they must be free solutions, able to be executed on a single node.

In the end, we selected four DBMS: we considered a general-purpose relational solution, i.e., PostgreSQL, two general purpose NoSQL solutions, i.e., Apache Cassandra and MongoDB (reported to be efficient in handling Time Series (Lavin and Klabjan, 2015; Ramesh et al., 2016)), and a native Time Series Management System (TSMS), i.e., InfluxDB (the only TSMS able to fulfil all the non-functional requirements) (Jensen et al., 2017).

To better describe the characteristics of these DBMS, we start by briefly recalling the well-known *CAP theorem* (Brewer, 2000). It states that there are three fundamental properties of a DBMS, namely *Consistency* (as defined in ACID properties), *Availability* ("data is considered highly available if a given consumer of the data can always reach some replica") (Fox and Brewer, 1999) and *Partitioning* ("the system as whole can survive a partition between data replicas") (Fox and Brewer, 1999). Given these three properties, only two can be guaranteed, meaning that it is possible to have CA, CP or AP systems, but not all three. Relational DBMS are classified as AC, meaning that a RDBMS offers high Availability and strong Consistency, while NoSQL systems are mostly classified as AP or CP.

In the following, we provide a brief description of the assessed DBMS.

4.1 PostgreSQL

PostgreSQL (n.d.) is an Object Relational DBMS (ORDBMS), sparkling from a project of the University of Berkley. It is currently open source and multiplatform. To date, it is widely used in many different contexts and applications, for industrial, government and academic purposes (e.g., Kaur and Rani, 2015; Liu and Nielsen, 2016; Di Martino et al., 2019). According to *CAP Theorem*, PostgreSQL is classified as AC, since it provides high Availability and strong Consistency. Being based on the Relational data model, PostgreSQL supports a large part of the ISO/IEC SQL standard, as stated in the official on-line documentation, offering also a total compliance with ACID properties.

4.2 Apache Cassandra

Apache Cassandra is a NoSQL DBMS, belonging to a *Column Store* category, designed to operate in a distributed environment with a masterless configuration. It provides high availability, fault tolerance and is able to manage huge amount of data. According to the CAP Theorem, it is classified as AP, allowing horizontal Partitioning and preferring high Availability over Consistency. Cassandra has a SQL-like querying language, named *Cassandra Query Language (CQL)*.

In Ramesh et al. (2016), Cassandra has been reported to be suitable for storing Time Series, since it offers some optimisations for storing and retrieving temporal data, such as reordering according to the timestamp key. Indeed, by declaring a *column-family*, it is possible to declare a composite primary key that involves n attributes (with $n \geq 2$). The first $n-1$ attributes forming the partition key, will be used for *data partitioning* across nodes. The last attribute, representing the clustering key, will be used to sort data within a node. Using time as clustering key, data are sorted by time, thus favouring time-based queries.

4.3 MongoDB

MongoDB is a NoSQL DBMS, belonging to the *Document-Oriented* category. Since data can have heterogeneous structure, it is stored in a JSON-like format. According to the CAP Theorem, Mongo is a CP database, where strong Consistency and Partition tolerance are effectively provided. The data model of MongoDB is composed of key-value pairs, saved in *BSON* documents, i.e., binary JSON, a binary serialisation schema for documents. This format is often used for Time Series storage (Ramesh et al., 2016), using different techniques (i.e., one document per record, or a document for values recorded in a given time frame). Also, MongoDB provides optimisations for Time Series data. The interesting feature is *data sharding*, namely data distribution across nodes by a defined key value, the time in our case. Sharding in MongoDB is conceptually close to Cassandra partitioning. MongoDB has no querying language, as each query must be expressed in JSON format, through an API.

4.4 InfluxDB

InfluxDB is a native Time Series Management System, specifically designed to handle Time Series (Naqvi et al., 2017). In InfluxDB, data are physically ordered by time. Another key feature of InfluxDB is the ability to define *retention policies*, namely specific rules to manage ageing of data. As an example, it is possible to specify that all the data older than a given number of months must be deleted, or replaced by an aggregated representation. InfluxDB offers also the ability to define *continuous queries*, namely a query tool able to filter continuously a stream of input data. This DBMS is based on a *hybrid* data model, where values, referred as *measurements*, are stored like in a row-based DBMS, and indexes, named *tags*, are stored like in a column-based one.

5 The experimental setup

In order to answer the Research Question defined in Section 4, we conducted an empirical study to assess the performances of the four identified DBMS in terms of ingestion/retrieval performances and storage space, using a massive data set of real IIoT data we collected within the system architecture described in Section 3. In the following, we describe the experimental setup, in terms of employed data set metrics and DBMS configurations.

5.1 The considered data set

For the experiments, we used real data collected from an instrumented grinder machinery. The Acquisition Board (see Sub-section 3.2) feeds the Data Preparation Node with a stream of data, that is then organised in records, each with the following attributes:

1 *Timestamp* (`data type: timestamp`): the time instant of the acquisition, expressed as standard Unix epoch time in nanoseconds;

2 *Sensor* (`data type: string`): the name of sensor collecting the measurement;

3 *Value* (`data type: floating point`): the value of the measurement;

4 *Program* (`data type: integer`): the Id of the *Part Program*, i.e., the processing procedure executed by the grinder;

5 *Subprogram* (`data type: integer`): the Id of the specific subroutine associated to the Part Program;

6 *Tool* (`data type: integer`): the Id of the tool type on which the sensor is mounted.

The data set used in the present work consists of 600 million records, according to the above-described schema, corresponding to about 2 full days of logging. The grinder was used to perform three types of Part Programs, whose time duration is respectively 2, 6 and 20 minutes.

5.2 Database configurations

The database schemas reflect the data set structure described above, with the appropriate adjustments for each system and data model. Moreover, significant tuning activities were done to obtain better performances from each DBMS, on the given data schema and query type. In particular, we operated mainly on indexing methods, to increase the speed of data retrieval. The choice of the right indexes is a key part of database tuning. A lack of suitable indexes will lead to the scan of the entire files from the disk during the execution of a query, with deleterious impacts on the performances. On the other hand, non-essential indexes will slow down data ingestion, due to the time required to build/update each index, and waste space on the disk, without significant benefits for the retrieval tasks. Moreover, when dealing with Time Series, there are non-trivial issues in indexing, due to the fact that often data is feed as a continuous stream over the time (John et al., 2016). In the investigation, for each involved DBMS, we considered two configurations of the schema, based on the type and amount of indexes:

Non-Indexed: Each DBMS is configured to index only the *timestamp* attribute.

Indexed: Each DBMS is configured to index the *timestamp* and additional attributes, to optimise queries.

In the Not-Indexed configuration, only the temporal attribute is indexed, for sake of uniformity across the DBMS. Indeed, since an index on the timestamp is native in InfluxDB, the same index was defined also for the other DBMSs. On the other hand, the Indexed configuration is meant to assess the impact of secondary indexes both on ingestion and retrieval performances. In particular, we defined some indexes on the *Program*, *Subprogram* and *Tool* attributes, to evaluate trade-offs in presence of secondary indexes. In particular, such indexes are differently selective: Subprogram is highly selective, with a domain of about 2200 different values, while Program and Tools are lowly selective, with a domain of just 3 and 2 values, respectively.

Furthermore, PostgreSQL has been evaluated with two different settings of physical ordering of the files. Indeed, this DBMS has a special command, i.e., CLUSTER, instructing the DBMS to *cluster* a table according to a specified index. From the DBMS documentation, this means that: "*When a table is clustered, it is physically reordered based on the index information. Clustering is a one-time operation: when the table is subsequently updated, the changes are not clustered.*" (PostgreSQL, n.d.). In a first setting, we did not execute the cluster command, while in the other one, we triggered this command immediately after the ingestion of each day of data, to sort the data set on the basis of timestamp.

In Table 1 are reported the main indexing setting for each configuration.

Table 1 Indexing setting for each configuration

Configuration	*Timestamp*		*Secondary index type*
	Index Type	*Physical Ordering*	
InfluxDB	*Time Series Index*	*Yes*	*Tags*
Cassandra	Sorted-String-Table	Yes	SSTable attached secondary indexes[5]
MongoDB	B-Tree	No	B-Tree
PostgreSQL	B-Tree	No	B-Tree
Clustered PostgreSQL	B-Tree	Yes	B-Tree

5.3 Metrics

As performance indicators, we used the following three metrics, widely used in DBMS benchmarking (e.g., John et al., 2016; Cooper et al., 2010):

- *Batch-Ingestion time* in seconds, to load a massive amount of data in the database. This measure is useful to evaluate the volume of data that a DBMS is capable to handle during the ingestion phase.
- *Retrieval time* in seconds, to execute a query. This measure is useful to evaluate the retrieval performances of a DBMS.

- *Disk usage* in Gigabytes, to measure the storage space required by a DBMS to store massive amounts of data.

A combined view of ingestion and retrieval performances is fundamental, since it is very high the risk of optimising one aspect while penalising the other (e.g., defining indexes to optimise retrieval, slowing down the ingestion). Moreover, considering the disk usage, gives an important insight on storage costs to ensure data and indexes persistence.

5.4 The hardware infrastructure

Referring to the architecture in Figure 3, the *Data Preparation Node* was equipped with an Intel i5 7300U, 16 GB RAM and 256 GB SSD, while the *Data Storage Node* was equipped with a hyper-threaded exa-core Intel Xeon E-217 6M, 64 GB RAM, 1 TB SSD. The two nodes were directly connected through a gigabyte LAN, and the tests were performed in isolation to estimate the performance of each chosen system, without outer interferences. To avoid temporal variability due to the sensors, for all the experiments we simulated the acquisition phase by replaying a stream of previously recorded real data.

5.5 The defined ingestion tests

In order to evaluate the ingestion performances of each DBMS, we investigate two scenarios. In the first one, the insertion is performed on an empty database, so data corresponding to a single day of registration (about 300 million points) for a single high frequency sensor, is inserted. In the second scenario, we consider a database already containing data generated in one day, and we fed it with another day of data. These two scenarios allowed us to evaluate the costs of index building and update.

To minimise biases due to external factors, the experimental evaluation consisted of five executions, for each of the mentioned configuration. After each run a reboot was performed, to minimise biases due to caching. A script measured the required ingestion time, in seconds, for each run. In the results Section, we will report the average values of these five runs.

5.6 The defined retrieval tests

To evaluate the retrieval performances, we define two query formats, relevant for the considered industrial context.

The first one, exemplified in Listing 1, is used to evaluate the performance of *Time-based* data retrieval. This is a common kind of analysis performed in the industrial plant, retrieving all the collected data, given the starting and ending time instants. We will refer to this type of query as *Time Query*.

Listing 1: *Time Query*, in SQL language

```
SELECT *
FROM recordings
WHERE Timestamp > StartTS
AND Timestamp < EndTS;
```

We defined two sets of values for the StartTS and EndTS parameters of the *Time Query* so that it can retrieve data collections representing the three mentioned *Part Programs* (see Sub-section 5.1). This because it is a very common query done by the Data Analysts in the plant, to monitor what happened during time-frames corresponding to a working program. Thus, we had queries retrieving 2 minutes of data, 6 minutes and 20 minutes, leading to result-sets with, respectively, ≈1.5, 5 and 16 millions points.

The second one, exemplified in Listing 2 is used to evaluate the performances of queries filtering by some *Process Parameters*. This is also a very common industrial scenario, where the Data Analyst may be interested in getting information about a specific working process. We will refer to this type of query as *Process Query*.

Listing 2: *Time Query*, in SQL language

```
SELECT *
FROM recordings
WHERE subprogram = SPVal
AND part_program = PPVal
AND tool = ToolVal
```

We defined three sets of values for the `SPVal`, `PPVal` and `ToolVal` parameters of the *Process Query*, so that it can retrieve data collections representing various combinations of working programs of the dressing machinery (see Sub-section 5.1). In particular, we had a query instance with all the three parameters set, one with only the SPVal set, thus being highly-selective, and one with only the `PPVal` set, being lowly-selective.

No projection is applied in these queries, so they will retrieve all the six attributes (see Sub-section 5.1).

Each retrieval test was performed on the databases populated with two days of data, corresponding to about 600M points of real data. The experimental evaluation consisted of 20 retrieval experiments, for each mentioned query and for each DBMS. After each run, again a reboot was performed to minimise biases due to caching. A script measured, in seconds, the required retrieval time for each run. In the results Section, we will report the average values of these twenty runs.

5.7 Disk usage tests

At the end of the ingestion tests, with the database containing two days of registration, we measured also the *Disk Usage* in Gigabytes, for both the Indexed and Not-Indexed configurations.

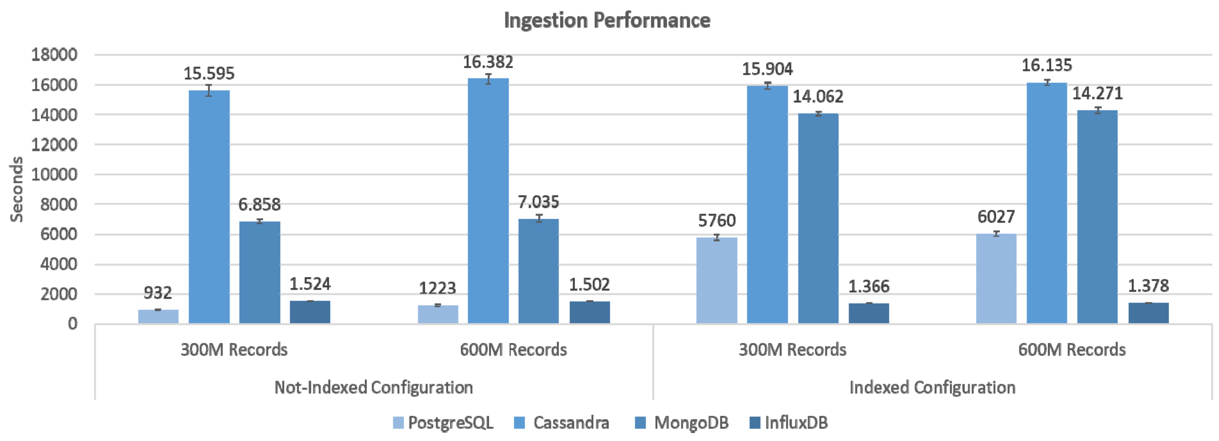
6 Results and discussion

In this section we discuss the results obtained with the experiments described in the previous section.

6.1 Data ingestion

In Figure 4, we report the histograms of the average ingestion times for the considered DBMS, while the bars represent the standard deviations.

Figure 4 Time, in seconds, to import 300 M and 600 M records for each DBMS, with and without indexes



In the *Not-indexed* configuration, PostgreSQL performs slightly better than InfluxDB. Probably this is due to the fact that, in such configuration, to avoid InfluxDB to index also the other fields, we defined all the non-temporal attributes as *measurement*. This leads to frequent compacting steps, resulting in higher ingestion time. MongoDB was pretty slower than the two previous DBMS, but still it required less than half of the time took by Cassandra. As expected, without indexes, there are no notable differences in performances between empty and pre-loaded configurations, for all the DBMS.

In the *Indexed* configuration, InfluxDB outperforms by far all the other DBMS. In particular, in this configuration InfluxDB behaves as a column store on indexed attributes (i.e., tags), and now it can save different values only once, storing only references to all records containing a value for a specific tag. Consequently, it obtains better performances than the Not-Indexed configuration. On the other hand, PostgreSQL dramatically slowed down its performances by a factor of about six. MongoDB slowed down by a factor of two, while Cassandra is practically unaffected, being anyhow still the slowest solution. Finally, let us note that, since PostgreSQL clustering is a recurring operation, to sort one day of data (300 Million points), it required on average 2100 additional seconds for the Not-Indexed configuration and 3800 seconds for the Indexed one. For sake of clarity, these numbers have not been reported in the histograms for PostgreSQL of Figure 4.

6.2 *Retrieval performances*

In this section, we report retrieval performances for Time and Process queries.

6.2.1 *Results for time query*

The results of the three Time queries are reported in Figure 5. Since these queries do not include any filtering on non-temporal attributes, the performances we obtained for the *Not-Indexed* and *Indexed* configurations are practically the same. For sake of brevity, we report only the first set of results. As for the most selective query (label *2 min* in the Figure), we can see that InfluxDB and Clustered PostgreSQL are roughly one order of magnitude faster than the other solutions. Nevertheless, InfluxDB has a peculiar behaviour, showing a required time to execute these queries that growth more or less linearly with the amount of data to retrieve (retrieving about 16 million points required about 10x more time than 1.5 million points). No other DBMS showed this behaviour, being on the contrary pretty stable among the three investigated queries. MongoDB and Cassandra show, again, lower performances in all the cases. PostgreSQL, with the basic setting, showed better performances than the two NoSQL solutions in all the tests, but by far worse than InfluxDB. On the other hand, it is interesting to note that the effect of the CLUSTER command of PostgreSQL is to reduce by about one order of magnitude the time required for the retrieval, thus being extremely advisable if these queries are frequently performed.

6.2.2 *Results for process query*

Results of the Process queries are reported in Figures 6 and 7, for both the Not-Indexed and Indexed configurations.

In these tests, Cassandra again performed worse than any other system. In all the tests of the Not-Indexed configuration, plus the one with the lowest selectivity rate in the Indexed configuration, Cassandra service crashed, being unable to return any result. In the remaining two cases, it was able to return correct results, but showing very poor performances, going by far out of scale of the Figure 7. As for InfluxDB, the configuration without secondary indexes led to very poor results, being by far the worst (working) solution. With the indexes, temporal requirements for the execution of the query were significantly lower, but still higher than MongoDB or the two configurations of PostgreSQL. MongoDB turned out to be better than InfluxDB and Cassandra in both configurations, showing a performance decrease only on attribute with lower selectivity in the not-indexed configuration. Finally, both the configurations of PostgreSQL outperformed all the other solutions, in all the Not-Indexed tests, and in most of the indexed one, were anyhow was pretty close to MongoDB, that showed the best results.

Figure 5 Average execution three time-based queries, in seconds, for the not-indexed configurations

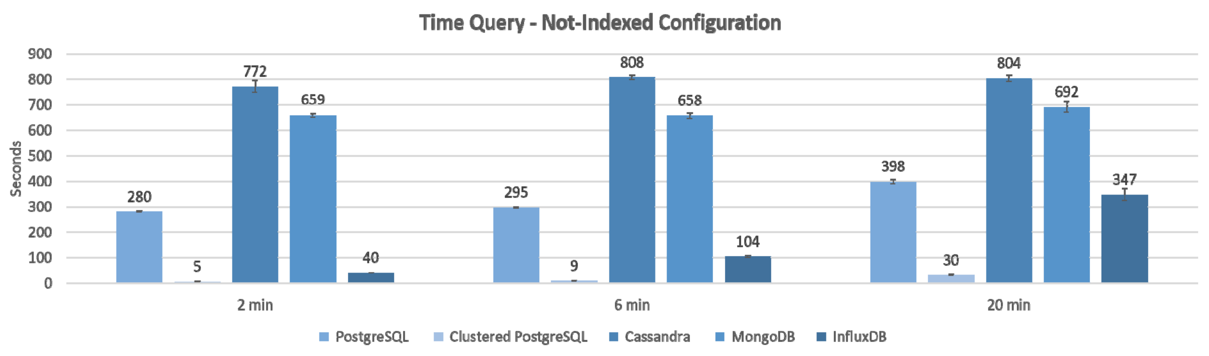


Figure 6 average execution three non-time-based queries, in seconds, for the not-indexed configurations

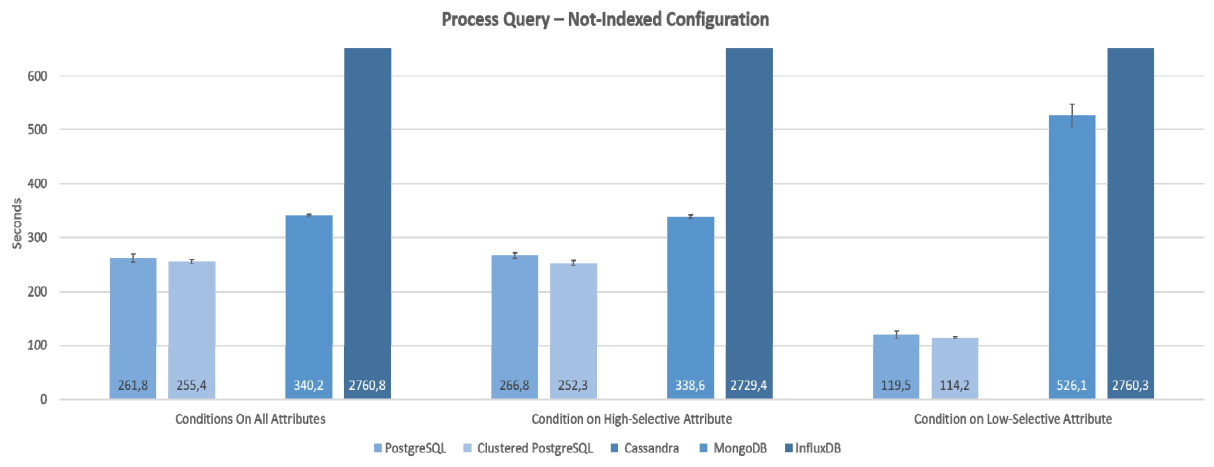
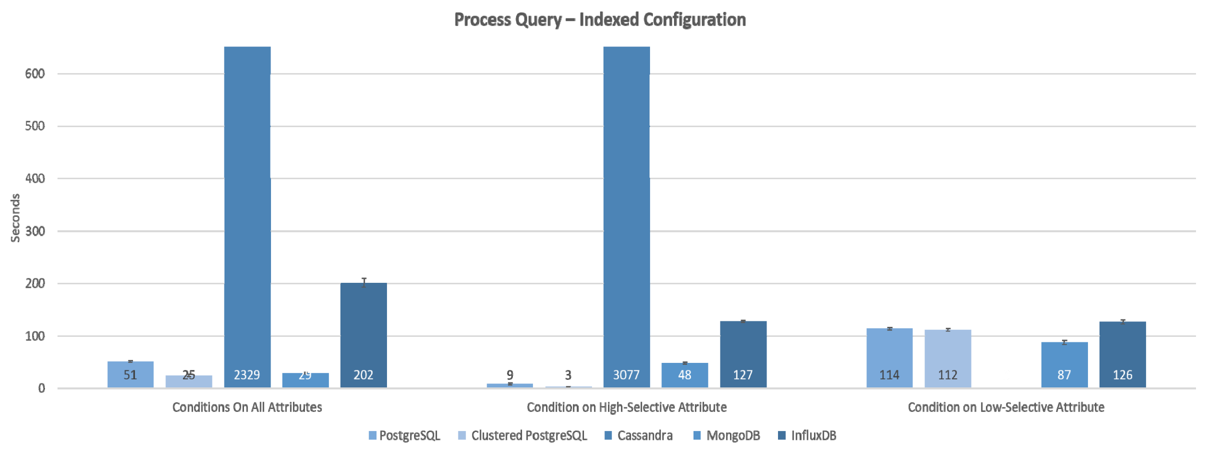


Figure 7 Average execution three non-time-based queries, in seconds, for the indexed configurations

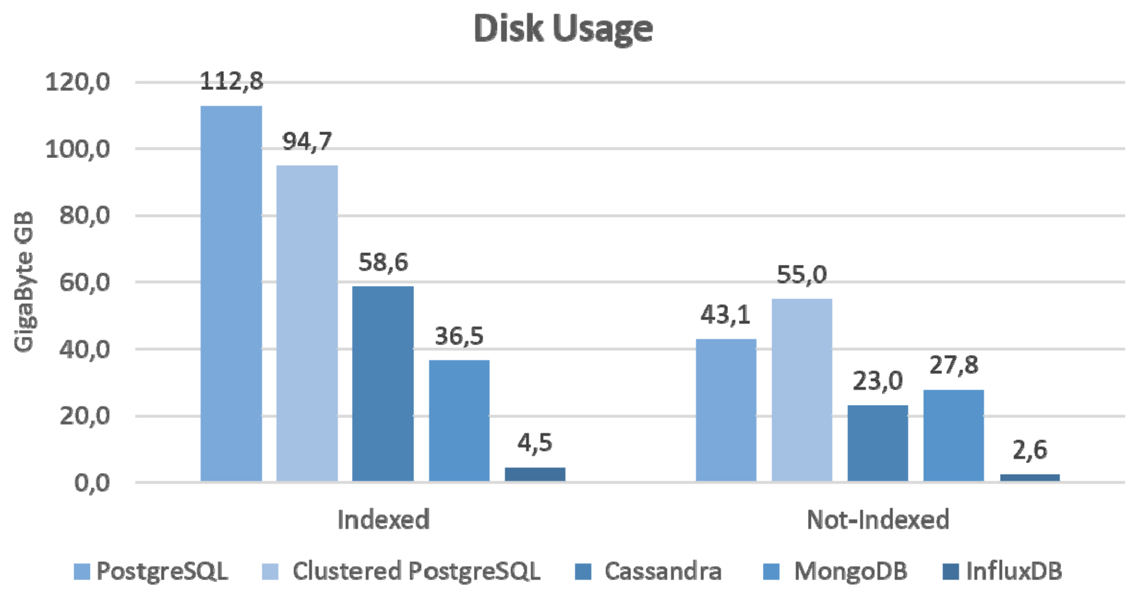


6.3 Disk usage

Figure 8 reported the *Disk Occupancy* measurements from both configurations, *index* and *no-index*, for each system.

As we can see, InfluxDB has the best performances in terms of required storage space. 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, the space needed for 600 millions of points is ≈ 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 especially true for Cassandra, whose indexes require more space than the data itself. Finally, PostgreSQL required the higher amount of space in both versions and configurations.

Figure 8 Disk space need to store 600 M points, in Gigabytes



6.4 Discussion

Although Cassandra is a successful general-purpose NoSQL DBMS, capable to handle great volumes of data, and successfully used for Time Series storage, in our proposed architecture, using a single node configuration, it was always by far outperformed by any other DBMS we investigated, being even not able to deliver results at all, in some tests. MongoDB showed good overall performances. Indeed, in retrieval tests over non-time-based parameters, it performed better than InfluxDB, but during the ingestion phase was outperformed by InfluxDB and PostgreSQL. Then, PostgreSQL was the best DBMS in a many tests in terms of execution time, failing mainly on disk usage, with the higher required disk occupancy among all the considered systems. The use of the CLUSTER command seems to be highly advisable, as it provides remarkable improvements in the retrieval phase, with an acceptable trade-off in terms on ingestion time. Finally, InfluxDB seems to be able to provide more balanced performances, with very impressive results in terms of required storage space. Summarising, given a scenario like the one we investigated, if storage requirements are a key parameter, then InfluxDB is the best choice. Otherwise, PostgreSQL, in combination with the CLUSTER command, is the configuration able to provide the best performances.

7 Conclusions

In the IIoT context, 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. Handling such amount of massive data can pose non-trivial challenges to a System Architect, willing to use the most suited DBMS to store and retrieve them.

In this paper we presented an empirical analysis we conducted on three NoSQL DBMS and a Relational DBMS, to investigate the achievable performances in terms of ingestion, retrieval and required storage space, for IIoT data, in a Fog-based architecture. In particular, we measured the performances of two widely employed DBMS, namely Apache Cassandra and MongoDB, of a Time Series Management System, i.e., InfluxDB, and of a Relational DBMS, namely PostgreSQL, in handling a data set of about 600 million records (about 60 GB), collected from an instrumented grinding machine. With our data set, MongoDB and PostgreSQL gave the best performances for queries on non-temporal indexed attributes, while Cassandra is outperformed by any other competitor in almost all the tests and turned out to be unstable on a single node configuration. PostgreSQL performed better than NoSQL competitors during the ingestion and retrieval phases, showing performances comparable with InfluxDB in retrieval phase of temporal queries, or even better, when used in combination with the CLUSTER command, showing the worst performances only in terms of disk usage. InfluxDB turned out to be on average the more balanced solution, outperforming competitors under storage aspects, and providing impressive performances on ingestion and time-based queries. In conclusion, given our IIoT use case and architecture, InfluxDB turned out to be the most advisable solution, especially in case of low disk space availability, or queries mostly on temporal data. On the other hand, if high retrieval performances are required on any type of query, and disk space is not a problem, PostgreSQL seems to be a more suitable solution.

After this experimental study, many possible evolutions can be envisioned. We clearly need to run experiments also on a multi-node architecture, rather than on a single database server, to measure the impact of data distribution and parallelisation, which could potentially boost the performances of the NoSQL DBMSs, in particular those of Cassandra. It is worth to add another TSMS to the comparison, in order to better evaluate this kind of storage system. Finally, it would be interesting to consider different queries and data schemas, in order to obtain a complete overview of system capabilities in time-series management.

References

Atzori, L., Iera, A. and Morabito, G. (2010) 'The internet of things: a survey', *Computer Networks*, Vol. 54, No. 15, pp.2787–2805.

Baily, M.N. and Manyika, J. (2013) 'Is manufacturing "cool" again', *Project Syndicate*, Vol. 21.

Bao, Y. et al. (2012) 'Massive sensor data management framework in cloud manufacturing based on Hadoop', *Proceedings of the IEEE 10th International Conference on Industrial Informatics*, IEEE, pp.397–401.

Bhattarai, B.P. et al. (2019) 'Big data analytics in smart grids: state-of-the-art, challenges, opportunities, and future directions', *IET Smart Grid*, Vol. 2, No. 2, pp.141–154.

Bhogal, J. and Choksi, I. (2015) 'Handling big data using NoSQL', *Proceedings of the IEEE 29th International Conference on Advanced Information Networking and Applications Workshops*, IEEE, pp.393–398.

Biswas, A.R. and Giaffreda, R. (2014) 'IoT and cloud convergence: opportunities and challenges', *Proceedings of the IEEE World Forum on Internet of Things (WF-IoT)*, IEEE, pp.375–376.

Bonomi, F. et al. (2012) 'Fog computing and its role in the internet of things', *Proceedings of the 1st Edition of the MCC Workshop on Mobile Cloud Computing*, ACM, pp.13–16.

Brewer, E.A. (2000) 'Towards robust distributed systems', *Proceedings of the 19th Annual ACM Symposium on Principles of Distributed Computing*, Portland, OR.

Brockwell, P.J. and Davis, R.A. (2016) *Introduction to Time Series and Forecasting*, 3rd ed., Springer.

Capgemini Consulting (2015) *Big Data Blackout: are Utilities Powering up their Data Analytics*?

Chebotko, A., Kashlev, A. and Lu, S. (2015) 'A big data modeling methodology for Apache Cassandra', *Proceedings of the IEEE International Congress on Big Data (BigData Congress)*, IEEE, pp.238–245.

Chen, B. et al. (2017) 'Smart factory of industry 4.0: Key technologies, application case, and challenges', *IEEE Access*, Vol. 6, pp.6505–6519.

Chen, M. et al. (2014) *Big Data: Related Technologies, Challenges and Future Prospects*, Springer.

Cooper, B.F. et al. (2010) 'Benchmarking cloud serving systems with YCSB', *Proceedings of the 1st ACM Symposium on Cloud Computing*, ACM, pp.143–154.

Das, S. (1994) *Time Series Analysis*, Princeton University Press, Princeton, NJ.

Dastjerdi, A.V. and Buyya, R. (2016) 'Fog computing: helping the internet of things realize its potential', *Computer*, Vol. 49, No. 8, pp.112–116.

Davoudian, A., Chen, L. and Liu, M. (2018) 'A survey on NoSQL stores', *ACM Computing Surveys (CSUR)*, Vol. 51, No. 2, pp.1–43.

Di Martino, S. et al. (2019) 'Industrial internet of things: persistence for time series with NoSQL databases', *Proceedings of the IEEE 28th International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE'19)*, IEEE, pp.340–345.

Esling, P. and Agon, C. (2012) 'Time-series data mining', *ACM Computing Surveys*, Vol. 45, pp.1–34.

Fox, A. and Brewer, E.A. (1999) 'Harvest, yield, and scalable tolerant systems', *Proceedings of the 17th Workshop on Hot Topics in Operating Systems*, IEEE, pp.174–178.

GE (2012) *The Rise of Industrial Big Data*.

Gessert, F. et al. (2017) 'NoSQL database systems: a survey and decision guidance', *Computer Science-Research and Development*, Vol. 32, Nos. 3/4, pp.353–365.

Gubbi, J. et al. (2013) 'Internet of hings (IoT): a vision, architectural elements, and future directions', *Future Generation Computer Systems*, Vol. 29, No. 7, pp.1645–1660.

Hendawi, A. et al. (2019) 'Benchmarking large-scale data management for internet of things', *The Journal of Supercomputing*, Vol. 75, No. 12, pp.8207–8230.

Hermann, M., Pentek, T. and Otto, B. (2016) 'Design principles for Industries 4.0 scenarios', *Proceedings of the 49th Hawaii International Conference on System Sciences (HICSS'16)*, IEEE, pp.3928–3937.

Jensen, S.K., Pedersen, T.B. and Thomsen, C. (2017) 'Time series management systems: s survey', *IEEE Transactions on Knowledge and Data Engineering*, Vol. 29, No. 11, pp.2581–2600.

John, A., Sugumaran, M and Rajesh, R.S. (2016) 'Indexing and query processing techniques in spatio-temporal data', *ICTACT Journal on Soft Computing*, Vol. 6, No. 3, pp.1198–1217.

Kagermann, H. et al. (2013) *Recommendations for implementing the strategic initiative INDUSTRIE 4.0: Securing the future of German manufacturing industry*; final report of the Industrie 4.0 Working Group, Forschungsunion.

Kang, Y-S. et al. (2016) 'MongoDBbasedrepository design for IoT-generated RFID/sensor big data', *IEEE Sensors Journal*, Vol. 16, No. 2, pp.485–497.

Kaur, K. and Rani, R. (2015) 'Managing data in healthcare information systems: many models, one solution', *Computer*, Vol. 48, No. 3, pp.52–59.

Kudo, T. (2019) 'Fog computing with original data reference function', *International Journal of Grid and Utility Computing*, Vol. 10, No. 5, pp.574–582.

Lasi, H. et al. (2014) 'Industry 4.0', *Business and Information Systems Engineering*, Vol. 6, No. 4, pp.239–242.

Lavin, A. and Klabjan, D. (2015) 'Clustering time-series energy data from smart meters', *Energy Efficiency*, Vol. 8, No. 4, pp.681–689.

Leavitt, N. (2010) 'Will NoSQL databases live up to their promise?', *Computer*, Vol. 43, No. 2, pp.12–14.

Lee, J., Bagheri, B. and Kao, H-A. (2015) 'A cyber-physical systems architecture for industry 4.0-based manufacturing systems', *Manufacturing Letters*, Vol. 3, pp.18–23.

Liu, R. and Yuan, J. (2019) 'Benchmark time series database with IoTDB-Benchmark for IoT scenarios', *arXiv preprint arXiv:1901.08304.*

Liu, X. and Nielsen, P.S. (2016) 'A hybrid ICT-solution for smart meter data analytics', *Energy*, Vol. 115, pp.1710–1722.

Madsen, H. (2007) *Time Series Analysis*, Chapman & Hall/CRC.

Marjani, M. et al. (2017) 'Big IoT data analytics: architecture, opportunities, and open research challenges', *IEEE Access*, Vol. 5, pp.5247–5261.

Mell, P. and Grance, T. et al. (2011) *The NIST Definition of Cloud Computing*, US Department of Commerce.

Mourtzis, D., Vlachou, E. and Milas, N. (2016) 'Industrial big data as a result of IoT adoption in manufacturing', *Procedia Cirp*, Vol. 55, pp.290–295.

Naqvi, S.N.Z., Yfantidou, S. and Zim´anyi, E. (2017) *Time Series Databases and InfluxDB*, Studienarbeit, Universit´e Libre de Bruxelles.

Pereira, D.A., de Morais, W.O. and de Freitas, E.P. (2018) 'NoSQL real-time database performance comparison', *International Journal of Parallel, Emergent and Distributed Systems*, Vol. 33, No. 2, pp.144–156.

PostgreSQL (n.d.) *PostgreSQL the Most Advanced Open-Source Object Relational Database*.

Ramesh, D., Sinha, A. and Singh, S. (2016) 'Data modelling for discrete time series data using Cassandra and MongoDB', *Proceedings of the 3rd International Conference on Recent Advances in Information Technology (RAIT'16)*, IEEE, pp.598–601.

Ren, L. et al. (2011) 'Resource virtualization in cloud manufacturing', *Computer Integrated Manufacturing Systems*, Vol. 17, No. 3, pp.511–518.

Simmhan, Y. (2018) 'Big data and fog computing', *Encyclopedia of Big Data Technologies*, pp.1–10.

Stonebraker, M. (2010) 'SQL databases v. NoSQL databases', *Communications of the ACM*, Vol. 53, No. 4, pp.10–11.

Syafrudin, M. et al. (2018) 'Performance analysis of IoT-based sensor, big data processing, and machine learning model for real-time monitoring system in automotive manufacturing', *Sensors*, Vol. 18, No. 9, pp.1–24.

TPC (n.d.) *TPC*, Available online at: http://www.tpc.org/

Van der Veen, J.S., Van der Waaij, B. and Meijer, R.J. (2012) 'Sensor data storage performance: SQL or NoSQL, physical or virtual', *Proceedings of the IEEE 5th International Conference on Cloud Computing*, IEEE, pp.431–438.

Vermesan, O. and Friess, P. (2013) *Internet of Things: Converging Technologies for Smart Environments and Integrated Ecosystems*, River Publishers.

Wang, S. et al. (2016) 'Implementing smart factory of industrie 4.0: an outlook', *International Journal of Distributed Sensor Networks*, Vol. 12, pp.1–10.

Yi, S. et al. (2015) 'Fog computing: platform and applications', *Proceedings of the 3rd IEEE Workshop on Hot Topics in Web Systems and Technologies (HotWeb'15)*, IEEE, pp.73–78.

Yin, S. and Kaynak, O. (2015) 'Big data for modern industry: challenges and trends [point of view]', *Proceedings of the IEEE*, Vol. 103, No. 2, pp.143–146.

Zhou, K., Liu, T. and Zhou, L. (2015) 'Industry 4.0: towards future industrial opportunities and challenges', *Proceedings of the 12th International Conference on Fuzzy Systems and Knowledge Discovery (FSKD'15)*, IEEE, pp.2147–2152.

Notes

1 The German group who defined the concept of Industry 4.0, in 2011, as key point of the German government high-tech strategy

2 https://www.mongodb.com/

3 http://cassandra.apache.org/

4 https://www.influxdata.com/

5 https://docs.datastax.com/en/dse/5.1/cql/cql/cql using/useSASIIndexConcept.html