The Journal of Supercomputing (2019) 75:8207–8230
https://doi.org/10.1007/s11227-019-02984-6

Benchmarking large-scale data management for Internet of Things

Abdeltawab Hendawi[1,2] · Jayant Gupta[3] · Jiayi Liu[6] · Ankur Teredesai[7] · Naveen Ramakrishnan[8] · Mohak Shah[6] · Shaker El-Sappagh[4,5] · Kyung-Sup Kwak[4] · Mohamed Ali[7]

Published online: 10 September 2019

Abstract

In the current era of the Internet of Things (IoT), massive number of sensors are used in our daily lives. Sensors are everywhere around us. They exist in our homes, work places, streets, cars, and even ourselves. Examples include home appliances, wearable devices, and medical sensors. These sensors generate huge amount of dynamic, heterogeneous, and unstructured data that need special handling beyond the capabilities of conventional relational databases. Thus, identification of suitable data management platform to store and query this data is necessary. Despite of its popularity and efficiency in processing various types of big data, there is no single-guided study of how NoSQL data stores will behave with the Internet of Things (IoT) datasets. IoT data have its own characteristics that make it special. IoT data come from various sensors, with a wide range of formats, high velocity, and require high throughput processing with low latency. NoSQL data stores are commonly used to provide flexibility and availability for big data handling. However, there is a lack of comprehensive studies about which NoSQL data store performs the best from the two scalability aspects (scale-up and scale-out) in a distributed and parallel processing environment. This paper benchmarks the commonly used NoSQL data stores (*MongoDB*, *Cassandra*, and *HBase*), and compares their performance with real industrial IoT dataset. In addition, we focus on comparing the throughput, latency, and run time of the evaluated NoSQL data stores.

Keywords Benchmarking · NoSQL · Distributed data management · Parallel data management · Internet of things (IoT) · MongoDB · Cassandra · HBase

✉ Kyung-Sup Kwak
kskwak@inha.ac.kr

Extended author information available on the last page of the article

1 Introduction

Since the 1970s, relational databases (RDBs) have been widely used. They are considered as a mature technology for storing structured data and managing their relationships. RDBs have reached an unmatched level of reliability and stability through decades of development. According to a recent market analysis study [3], the RDBs market share is estimated at $35.9 billion in 2015. However, toady's data can scale to terabytes per day. In the same time, they must be available to millions of users worldwide with low latency requirements. Traditional RDBs systems were not designed to easily handle these situations. As a result, the same previous study asserted that the main RDBs vendors have lost between 1.5 and 5.6% of their shares within the last five years. A constant demand for alternatives to the RDBs is raising. Recently, the movement toward the NoSQL has gained attention. The market share of these non-relational database systems has been increased. It reached more than half a billion dollars in 2015, and is expected to exceed $3.5 billion in 2019 [6]. NoSQL databases are usually open-source, low-cost, unstructured (i.e., schema-less), elastic, and horizontally scalable. Their interfaces are simple, which allow developers to start using them quickly. In addition, many NoSQL databases are designed to be distributed by using the *sharding* and *replication* mechanisms. As a result, these novel data storage systems are able to cope with big data. Some studies proposed algorithms, techniques, and guidelines for migrating from RDBs to NoSQL databases [29]. These studies discuss the SQL query translation mechanisms, the effects of denormalization, the join algorithms, the role of (secondary) indexes, and others.

We live in the age of big data. Everything around us is captured digitally and linked to a data source. About 2.7 zettabytes of data are stored in today's digital world [36], and this number is growing exponentially. The most of these data are in unstructured formats, and used mostly for analytical not transactional purposes. These unstructured data span many formats including videos, voice, images, websites and blogs, customer transactions, healthcare records, news, sensors in the world of Internet of Things (IoT), social media posts, etc. To give an example of how big the volume of unstructured data generated in our daily life is, Facebook users post about 100 terabytes a day, Twitter received an average of 175 million tweets a day in 2012 that became 500 million tweets a day in 2016, YouTube users upload 48 hours of videos every minute, while Skype counts more than 2.8 thousand calls per second [28, 36, 46]. The four V's of big data (i.e., volume, variety, velocity, and veracity) are not suitable for SQL databases. Rodriguez-Mazahua et al. [4] provided a comprehensive survey about the big data literature including its tools, major trends, areas of application, and major challenges. In these situations, NoSQL data stores are more appropriate. They have the ability to deal with native forms such as documents, graphs, time series, and multimedia files. A quick look in the funding that NoSQL databases attracted in the last few years and the expected revenue in the near future is it can be intuitively concluded that NoSQL systems will gain a bigger slice in the data management marketplace. Two functional requirements are considered the main reasons for

this noticeable change in data management paradigm: *first*, the growing need for data flexibility and availability over data consistency and *second*, the emphasis on *horizontal scaling* over *vertical scaling* of the hardware resources [25]. In horizontally scaled system, new nodes can be added and configured as the need arises, whereas in vertically scaled system, the existing resources are replaced with better configurations.

There are dozens of NoSQL database systems. It is difficult to keep track of where they excel, where they fail, or even where they differ. In this paper, we conduct an exhaustive performance analysis of the main NoSQL systems in a distributed environment. Our focus lies on studying their popularity, maturity, and applicability to data management and analysis in various areas. Since these NoSQL systems cover the most relevant techniques and design decisions in the space of scalable data management, we compare Cassandra [30] (v2.2.3), HBase [5] (v1.1.2), and MongoDB [35] (v3.2), in a distributed and parallel computing environments. Cassandra and HBase are top two wide-column data stores, and MongoDB is the top document store in terms of industrial popularity [44]. Since our motive comes from the industrial perspective, the used datasets are industrial IoT; this study focuses on the top commonly used NoSQL system in industry.

Based on the experimental results, the architectural features of these systems that affect their operational performance are critically analyzed.

There are some works that compare NoSQL databases to determine their virtues and weaknesses. Catell [11] has compared the concurrency control methods, data storage location (i.e., main memory or disk), the used replication mechanism, and transaction support of some databases. Other studies [49] list the most effective NoSQL databases and list their characteristics. However, this type of analysis does not include a full comparison between databases because it only exposes some of the database advantages and disadvantages. Other studies analyze NoSQL databases based on a given dataset. For instance, Sakr et al. [45] performed a thorough analysis of databases, includes NoSQL databases, suited for cloud computing environments. However, most of these works did not focus on database benchmarking [14]. YCSB [13, 26] (Yahoo Cloud Serving Benchmark) is a popular benchmarking framework for NoSQL databases. It gives an extensible framework to query databases to benchmark various access patterns. These types of data have a different nature of IoT data, and their results cannot be generalized for the IoT environment. To the best of our knowledge, there are no research studies that provide a critical analysis of NoSQL databases based on IoT benchmark. Thus, this paper is the first to benchmark distributed NoSQL data stores based on IoT datasets.

Recently, with the significant advances in computer and communication technologies, IoT big data become the most common type of big data [40]. It can be said that IoT and big data are interdependent technologies and should be developed jointly. Our focus on real IoT sensor data is attributed to the fact that there are about 15 billion devices attached to the IoT [32]. In addition, many market analysis reports predict that the number of IoT devices will increase up to 50 to 75 billion devices in 2020 [15, 27]. This trend is expected to bring about $19 trillion to the global economy in the next 20 years [42]. In addition to this economic impact, it generates really big data as a cause of using large-scale sensors that produce data with

immense *Volume* and *Variety* arriving at a truly rapid *Velocity* [54]. The IoT devices are projected to generate around 600 zettabytes (ZB) of data volumes per year by 2020 [34]. In addition, the IoT workloads along with data management and analysis workloads by 2020 will represent 22% of the total running workloads by that time. At this point, this benchmark conducts evaluation of various NoSQL platforms under IoT big data, where the results allow us to understand the performance behavior of NoSQL systems under various conditions. In this study, we perform two types of scalability analysis over the studied data store, *horizontal scale* and *vertical scale*. Here, our study utilizes a real dataset collected from large-scale IoT sensors. This dataset is collected and provided by a main industrial vendor in the IoT sector. It is stored in columnar text files. The whole dataset is about 100 GB distributed over 550 files. Each file is organized in 106 columns representing the temporal data values for one appliance. The data cover the period from February 2014 to July 2014 with resolution in seconds.

The performance analysis is based on a set of metrics including *latency, throughput, run time,* and *processing rate*. When comparing performance of different NoSQL systems, it should be noted that there is no single "winner takes all" among the databases. It is always possible for one NoSQL database to outperform another and yet fall behind its competitor when the rules of engagement change. As a result, it depends on the use case and deployment conditions. Our results assert this fact. The evaluation results using the real 100 GB IoT data provide the following key findings. *Cassandra* gives the best performance with highly distributed setups. *MongoDB* behaves better in a non-distributed context. It always defeats *Cassandra* and competes with *HBase* in a centralized context. *HBase* outperforms both *Cassandra* and *MongoDB* in a low distributed setup. In terms of execution robustness, both *HBase* and *MongoDB* are exception-free during run time. That is not the case for *Cassandra*, which happens to give two kinds of timeout exceptions (*ReadTimeoutException* and *WriteTimeoutException*). Specifically without IoT dataset, *Cassandra* can achieve up to three orders of magnitude better processing rate than the other two data stores. However, with a small-size cluster, *HBase* occasionally defeats *Cassandra* and *MongoDB*. Detailed analysis and findings are given in Sect. 5.

The main contributions of this paper are listed as follows:

- To the best our knowledge, this is the first study to evaluate the most popular NoSQL data stores using real IoT datasets, with its nature of being sparse and unstructured.
- This study considers the distributed context rather than a standalone node installation of the NoSQL systems.
- Our experiments evaluate several measurements including latency, throughput, run time and cover the scalability behavior of the studied NoSQL systems.
- These evaluations are conducted using various types of workloads that range from simple operations to aggregate queries to give a comprehensive study.

The rest of this paper is organized as follows. Section 2 describes the related work. Section 3 describes the architectural characteristics of Cassandra, HBase, and MongoDB. Section 4 states the experimental setup for the benchmark. This is followed

by Sect. 5 that describes benchmarking with an industrial dataset describing the dataset, running workloads, results, discussion, and scalability (scale-out) analysis of different data stores. Finally, Sect. 6 concludes the paper.

2 Related work

With the rise of NoSQL databases in industry, research in the field has risen as well. The majority of early papers dealt with comparisons between individual databases such as MongoDB or Cassandra, or concerned themselves with comparisons of the trade-offs that NoSQL and relational database designs afforded the user, as can be seen in Yi and Manoharan's work on NoSQL and SQL database comparisons [31]. Additionally, many of these early papers served as introductions into NoSQL and its new model of data transactions and consistency [47]. When Abramova compared MongoDB and Cassandra [2], a large portion of the article was devoted to a general survey about what NoSQL databases were and what type of NoSQL databases currently existed. Researchers have also evaluated NoSQL databases in the context of big data due to its efficiency. A recent general survey about NoSQL systems is introduced in [17]. Comparison research studies of SQL and NoSQL databases have shown that many NoSQL engines outperform SQL databases on a wide variety of database operations, such as deletes, reads, and writes [31]. In [33], the major NoSQL is evaluated and compared using workloads represented by Yahoo! Cloud Serving Benchmark, while the study in [51] focuses on the study of data loading/unloading capabilities when major NoSQL systems are used along with Spark. The real feature was tested and compared on three NoSQL systems (Couchbase, MongoDB, RethinkDB) in [39]. The work in [21] evaluates the response time on relational versus non-relational (NoSQL) database models when they used to handle e-government data. Most of the work in this domain study and compare several NoSQL systems; however, the authors in [16] focus just on the column-based Apache HBase.

Performance evaluation is an extremely important research field that allows us to quantify whether or not progress has been made. Even though large-scale and massive data management exponentially grew due to the birth of Web 2.0, few performance evaluations were conducted in this field at the time. The definition of big data is to realize large and complex data manipulation. Big data face the hurdles of large-scale data, diverse data format, non-relational structures, and data streaming. Therefore, unlike traditional data techniques, the current big data application faces more challenges than ever before. There are four main challenges encountered when sustainability, efficiency, and profitability are considered in big data applications: scalability of computing and data storage architecture and algorithms; querying and processing technologies (including data organization and system management); planning techniques and tools; and fault tolerance [7]. Thus, the characteristics of big data performance evaluation are different from those of traditional performance evaluation. With similar motivation, Cattell [11] contrast NoSQL systems on their data model, consistency mechanisms, storage mechanisms, durability guarantees, availability, query support and other dimensions, where he emphasized on

the trade-off between some of the dimensions (e.g., consistency v/s availability) to achieve the necessary requirements.

In [8], the authors present a relatively traditional database performance evaluation and argues three important factors that dependently affect performance. They are multi-programming level, query mix, and degree of data sharing. To illustrate the relationship between three factors, the authors developed the methodology to scientifically test it. In the experiment of query mix selection, the authors chose a resource utilization approach. CPU utilization and disk utilization were used in measuring the efficiency. Each metric has low and high levels. The authors created four combinations to control effects of each variable. Even though three factors discussed above are critical to the performance of big data application, the paper primarily focuses on the multi-user environment. Big data platforms not only include the multi-user environment but also the distributed system and other analytics tools.

Dede et al. [18] compared the performance between HDFS and MongoDB for Hadoop by using simple queries. In addition, analysis of connecting Hadoop and MongoDB to avoid the problem of data locality was studied. The novel parts of the paper are to consider scalability tests and fault tolerance in terms of performance evaluation. For example, since Hadoop usually stores data into the computing node, it is highly possible that the failures of nodes affect access to data. Also, the efficiency of distributed system and cloud computing affects performance of big data applications. Rank join queries are used extensively in modern analytics tasks. Thus, Ntarmos et. al. [37] evaluated join queries in NoSQL databases. They implemented MapReduce algorithms for index building and statistical structures, and also provided the capability for online updates. They performed scalability analysis of the queries on Hadoop and HBase installed on Amazon EC2 and "in house" clusters.

There have been various research efforts on benchmarking distributed computing frameworks for various domains. Among them, BigDataBench [53] and Bigbench [24] are benchmark suites that evaluate structured, unstructured and semi-structured data on various workloads (e.g., Sort, Index, PageRank, and K-means) using different distributed technologies. This work did a comprehensive performance analysis by varying the data size from 32 GB to 1 TB reported using MIPS (million Instruction Per Second) and Cache MPKI (Miss Per Kilo Instruction) metrics. Their analysis focused on identifying performance trends of the workloads. With the growth of Internet of Things (IoT), data from sensors are prevalent. Thus, Van et al. in their work [52] compared Cassandra and MongoDB for the storage of sensor data. The experiments were performed in both physical and virtual environments to understand the impact of the environmental changes. However, for their experiments, they used minimal homogeneous sensor data.

With the increased usage of NoSQL data stores in the industry, there have been comparative studies on industrial use cases. Tilmann et al. [43] presented a comprehensive performance evaluation of six data stores (HBase, Cassandra, Voldemort, Redis, VoltDB, MySQL) in the context of application performance monitoring. They evaluated these systems with data and workloads that can be found in application performance monitoring, as well as, online advertisement, power monitoring, and many other use cases. Endpoint Corporation did a broad comparison between the then–current top NoSQL databases [20], going over various benchmarks that

they judged to be indicative of modern applications, as well as a short section detailing various problems encountered when setting up each database for testing. These include insert workloads, read–write–read workloads, etc. This evaluation focuses more on the basic operations of each database, and focuses less on complex queries that a user might run.

There is an inherent trade-off between latency and throughput. For any hardware, when the load increases, the latency of individual requests increases because there is more contention for disk, CPU, network, and other resources. Cooper et al. [13] designed a framework, YCSB, to compare data stores in terms of latency and performance. In particular, their experiments were designed to find the distributed technology that achieved the desired throughput while preserving an acceptable latency with the least amount of hardware resources. Building on this work, Dey et al. [19] proposed YCSB+T framework to evaluate operational cost of NoSQL systems (e.g., Wiredtiger). In particular, they combined atomic database operations to create single transaction operation. Hendawi et al. employed YCSB framework to evaluate popular NoSQL systems from the scalability perspective [26].

There have even been efforts toward the development of some SQL-based plugins for NoSQL engines that allow the user to perform standard SQL queries, such as Impala, Presto, and SlamData [47]. Another benchmark suite is big data benchmark [50], which is based on the workloads and queries from Pavlo et al. [38]. It measured the response time on queries such as scans, aggregations, and joins for five distributed technologies (Hive, Tez, Shark, Impala, and Redshift). Their input dataset consisted of unstructured HTML documents and two SQL tables. On the other hand, the original work by Pavlo et al. compared MapReduce(MR)-based approach and equivalent SQL statements executed by two database systems hosted on a distributed infrastructure. From the experiments in this work, it is concluded that both parallel database systems displayed significant performance advantage over Hadoop MR.

In addition to the quantitative comparison introduced in this study, the following table gives a qualitative comparison (adapted from [23]) of the studied NoSQL System.

Our study here is distinguishable from all previous work by focusing on benchmarking the three most popular NoSQL data stores (*Cassandra*, *Hbase*, and *MongoDB*) using real industrial IoT dataset. This study is comprehensive within a distributed context, not a single node test. Our experimental evaluation runs various workloads that include atomic operations and aggregate queries rather than simple queries as others do. In addition, our study here focuses on comparing the throughput, latency, run time, and scalability, not just the response time as exists in most of the related work.

3 NoSQL systems

Fundamentally, a centralized database system is required to provide consistency, and availability. When it comes to a distributed database system, the CAP, (*Consistency*, *Availability*, *Partition tolerance*), theorem defined by Fox and Brewer [9, 22] states

that a distributed system can only meet two out of the three distinct needs simultaneously. These needs differ for databases handling of structured versus unstructured data. Application of the CAP theorem on SQL databases and NoSQL databases results in two different principles [10]. First, *ACID* that stands for *Atomicity, Consistency, Isolation and Durability* applies to systems that focus on very strong consistency and availability of data. ACID is a core feature of SQL databases. Second, BASE stands for *Basically Available, Soft state and Eventual consistency*. This property set applies to the systems that focus on partition tolerance and availability more than consistency. BASE is the feature of NoSQL databases. The existing NoSQL databases could be classified into four main types, *key-value, document, wide-column, and graph databases* [1]. Storage of large-scale distributed data is favored in the first three, whereas on the other side, storage of information with complex relationships, like social network data and semantic topologies, is favorable in graph databases (Table 1).

This paper evaluates the three popular NoSQL databases, *Cassandra*, *MongoDB*, and *HBase*. In the following few paragraphs, we give a brief background about each of them.

- *Cassandra* Apache Cassandra was developed at Facebook as a combination of Amazon's DynamoDB [48] and Google's Bigtable [12]. Cassandra is a wide-column data store where the columns are grouped into column families and referenced by using the syntax, column-family/column-qualifier. Cassandra is a distributed system where each node acts as both master and slave. The system performs all the critical functions in a decentralized manner. The nodes communicate with each other using a peer-to-peer communication protocol in which nodes periodically exchange state information about themselves and about other nodes they know about. This protocol is known as Gossip protocol [30]. Cassandra uses a three-step process to ensure data durability while writing the data. First, the data are written to a commit log, then to an in-memory data structure, and finally, when the data structure is full, to a disk. Cassandra ensures fault tolerance by providing custom data replication capabilities. To ensure physical redundancy, data can be replicated among physical data center racks to avoid single point of failure. To handle NoSQL data, Cassandra's dynamic schema design allows flexible data storage. Addition of new nodes to the cluster by design (peer-to-peer architecture) provides a linear increment in performance.
- *MongoDB* MongoDB was developed by the company 10gen, now known as MongoDB Inc. In MongoDB, each node is referred to as a shard that controls a subset of data. Auto-sharding is a key feature in MongoDB that facilitates scaling horizontally by splitting data across multiple nodes. Assignment of data between shards is rebalanced automatically. With sharding, you could add more nodes to support data and meet the demands of read and write operations. MongoDB is a document database where records are stored as documents in Binary JSON (BSON) syntax. A collection in MongoDB is equivalent to a table/relation in SQL. MongoDB uses simple query language which provides capabilities for writing rich document-based queries with support for search by field, range-based query, and regular expressions.

Table 1 A qualitative comparison of the studied NoSQL systems

Dimension	MongoDB	IIBase	Cassandra
Model	Document	Wide-column	Wide-column
CAP	CP	CP	AP
Read performance	Random: high (memory-mapped) Scans with good sharding key: high Scans with random sharding key: low (Scatter-and-gather)	Random: medium Sans: high	Random: medium Sans: only through compound indexing
Disk latency per get (Id)	~ One disk seek	~ Several disk seeks	~ Several disk seeks
Write performance	Depends on sharding keys, no sequential I/O	High, danger of hotspots, append-only I/O	High, append-only I/O
Network latency	Configurable: nearest slave, master ("Read preference")	Designated region server	Configurable: R replicas contacted
Durability	Configurable: none, journaled, replicated ("Write concern")	WAL, built-in row-level versioning	WAL, W replicas written
Replication	Asynchronous master–slave	Filesystem level (HDFS)	Consistent hashing (virtual nodes and equal-sized partitions)
Sharding	Hash- or range-based on attribute(s)	Range-based (row-key)	Consistent hashing
Consistency	Master reads/writes: linearizable slave reads: eventual	Linearizable	Configurable: N, R, W never linearizable last-writer-wins no isolation for multi-column writes
Atomicity	Single document	Single row, or explicit locks	Single column (multi-column updates may cause dirty write anaomaly)
Conditional updates	Yes, mastered	Yes, mastered	Yes, paxos coordinated
Interface	Binary TCP, many client APIs	Thrift	Thrift or TCP/CQL

Table 1 (continued)

Dimension	MongoDB	IIBase	Cassandra
Model	Document	Wide-column	Wide-column
CAP	CP	CP	AP
Special data types	JSON object, array, set and counter operations	Counter	Counter
Querys	Query by example (filter, sort, project), range queries, MapReduce, aggregation	Get by ID (row-key), scans over row-key ranges, project CFs/columns)	Get by ID (partition key) and optionally filter and sort over clustering key
Indexing	Hash and B-tree indexes	None	Compound index (partition and clustering keys), secondary index (flash-based on column)

– *HBase* HBase was initially developed by Powerset and is now an Apache open-source project. HBase architecture is composed of a typical master–slave type of architecture and uses ZooKeeper [20] as a distributed coordination service to maintain the server state. The master server (comparable to the name node in HDFS) manages and monitors HBase cluster operations plus load balancing. The region servers are data nodes with HBase tables split across several region servers. Region servers contain regions which in turn contain stores. The stores are organized into memstores (data gets stored in memory first) and filestores (actual disk storage).

  HBase is also a wide-column data store. Furthermore, HBase favors random access and low latency schemas and opts for consistency over availability as opposed to Cassandra. Also, as HBase is built on top of the distributed file system, it supports sparse storage. In addition, HBase has a native query syntax that differs from existing SQL syntax. Thus, to run SQL-based queries in HBase, additional tools like Apache Phoenix are required. Apache Phoenix (v4.5.2) is an open-source tool built by Salesforce and is now an Apache open-source project. Phoenix adds a query engine above HBase that is comprised of a parser, compiler, optimizer, execution engine, and metadata repository stores. Phoenix is comparable to a JDBC driver that implements all standard and optional metadata interfaces in JDBC. Phoenix is extremely lightweight which requires no server installations and is implemented as a JAVA library.

It is important to note here that graph model-based data management systems such as Noe4j are not suitable to our dataset. Neo4j and similar systems are more appropriate for data from social network graphs, road network graphs, semantic web graphs, etc., where the data are stored in the form of a graph of nodes and edges. So, we do not include graph-based model in this study.

4 Benchmarking setup

4.1 Metrics for measurement

In our experiments, we use latency and throughput-based metrics to measure the experimental results. Latency metric represents the time interval between the function call and the response to the call. Throughput metric represents the rate of successful functional execution. The client provides the run time report for each workload having average latency and average throughput measures over all the operations that are executed in the workload.

4.2 Infrastructure

The experiments are conducted on an eight-node cluster connected via Gigabit Ethernet Cable. Each node of the cluster has an Intel(R) Core(TM)2 Quad CPU Q6600@2.40 GHz processors and 6 GB of RAM. We admit that the number of

machines used in our experiments is small. However, this is the available dedicated cluster in our institution, and we have no other cluster resources. We also tried to rent some cloud services but we could not afford renting an isolated and dedicated big cluster.

4.3 Scalability assessment

We varied the cluster nodes from 1 to 8 to measure scale-up performance of the data stores. For our experiments, the increase in the number of nodes is accompanied by proportional increase in data. A single node is associated with 5 GB of raw data. Thus, our experimental range of the data is from 5 GB (5 million records) to 40 GB (40 million records). It is usually sufficient to provide two peers for each node in a cluster of three or more nodes. We follow similar strategy in our experiments. Furthermore, peer nodes add a slight network overhead that is minimal compared to other data transfer overhead. It is important to state that we do not control the index creation deletion. Rather, we keep it to the nature of each NoSQL system to manage the indexes freely in its own way. We need also to state that we do not have experiments dedicated for errors test, rather, we just comment on the error occurred during the execution of our queries.

4.4 Configuration decisions: Cassandra

One of the key aspects of Cassandra is setting up nodes to communicate system information across nodes. Such nodes are usually termed as peers and are required for each node. Initial set of experiments provides the first glimpse at the failure count in the data stores. In the experiments, some execution errors are primarily found in Cassandra, no such errors are observed in HBase and MongoDB. Moreover, no error is observed in the read–modify–write step of the experiments. Twenty-four errors are observed in insertion step, 40 errors are observed in read step, 267 errors are observed in update step, and 1082 errors are observed in scan step. Note that the numbers are accumulated over all the multi-node experiments.

The errors occurred because the Cassandra coordinator on certain occasions assumes that there are enough replicas alive to process a query with the requested consistency level, but for some reasons, one or several nodes are too slow to answer within the predefined timeout. This could be due to a temporary overloading of these nodes, even that they have just failed, or have been turned off. In general, there are two types of timeout exception associated with Cassandra [41].

– *ReadTimeoutException* [Cassandra timeout during read query at consistency ONE (one response is required but only 0 replica responded)] If the query is a read, a ReadTimeoutException will be thrown. During reads, Cassandra does not request data from every replica to minimize internal network traffic. Instead, some replicas are only asked for a checksum of the data. A read timeout may occur if enough replicas have responded to fulfill the consistency level and if only checksum responses have been received. In Java, DataRetrieved() function

allows to check whether you are in this situation or not. Note that by default, the Cassandra driver will automatically retry the query if only checksum responses have been received, making this cause of read timeout not so likely to reach the code.

– *WriteTimeoutException* (Cassandra timeout during write query at consistency ONE: One replica is required but only 0 acknowledged the write) In a similar situation as above, if the query is a write, then a WriteTimeoutException will be thrown. If the driver differentiates between read and write timeout here, this is because a read is obviously a non-mutating operation, whereas a write is likely to be. If a write timeouts at the coordinator level, there is no way to know whether the mutation has been applied or not on the non-answering replica.

All the FAILED operations are executed after a sleep time of 1 second. Further, the sleep time of thread allowed the CPU to complete other parallel tasks and possibly provide it higher chance to recover and complete the step successfully. In most of these experiments, no FAILED operations are observed or the number of "errors" reduced to less than 10. Furthermore, we found no significant difference in the performance due to the operation retrial.

4.5 Configuration decisions: MongoDB

The benchmark process involves bulk deletion of the data. Therefore, an optimized way to remove all the data is important. MongoDB provides two ways to perform deletion. The first is to use *db.collection.remove()* function to remove documents from the data. This method preserves the indexes, which results in an increased insertion throughput. The second is to use *db.collection.drop()* function to remove documents from the data which almost instantly removes the data. However, insertion takes longer due to creation of new indexes. During our experiments, we found the use of *drop()* functionality to be optimal. This choice is further supported by the official MongoDB site [35] suggesting that the use of drop() method to drop the entire collection and the indexes, and then recreate the collection and rebuild the indexes is optimal strategy.

Moreover, indexing based on shard key serves the purpose of distributing the data across different data nodes. Shard keys have different property compared to regular keys (primary or secondary). Usually, the keys group data chunks containing similar key values (values residing in a particular range). Therefore, a good shard key should not be monotonically increasing. Thus, we use the hashing utility of MongoDB that hashes the shard key resulting in uniform distribution of data across the shards.

4.6 Configuration decisions: HBase

HBase follows a master–slave architecture; thus, all the requests are sent to a single master process. The master subsequently distributes the requests to the slave processes across the cluster. Based on this architecture, we want to answer two

questions, first, whether hosting a master and slave together might prove to affect the system. Second, whether master–slave systems prove to be a heavy bottleneck to improve system throughput.

We conducted additional experiments to observe the difference in performance when master and region servers are separated. Our results demonstrated that putting the master with region server together provides slightly better performance than putting them separately. This might be due to added network cost incurred by separating master and region server. We also conducted experiments using backup master and evaluated the performance. The observed results are similar to that of single master. The reason for this is that backup master always maintains a state similar to that of master. Thus, additional operations are needed to be performed resulting in lower HBase performance.

4.7 Datasets and workloads

The industrial dataset represents real use cases necessary to build the system for industry. In this section, we describe the set of experiments conducted to understand the scalability and performance of the evaluated data stores.

One of the key properties of our workload is the immutability of data. Therefore, all the queries are read based and involve the complete scan of tables. The results highlight the relative performance to calculate different functions for each of the data store. All the benchmarking results are measured using the query processing rate and total run time measures. Moreover, we discuss the dataset and details on running the workloads.

This batch of experiments leverages a real industrial IoT sensor dataset stored in columnar text format as csv files. The uniqueness of our IoT datasets comes from the fact that this dataset is highly unstructured and sparse, and it is predominately used for evaluating aggregate queries. In a preprocessing step, we decided to convert our dataset into CSV to avoid mixing the conversion burden while measuring the query actual cost for each NoSQL system.

The complete dataset had 550 files amounting to a size of 100 GB. The files are divided such that each file includes a data of a short range of time. The main advantage of having many small files over one big file is to make the data loading and distribution faster. Each file consisted of data elements such as date, time, sensor measurements, and error flags for a single appliance with a total of 106 columns for each row. The data are sparse as the flag value is present only when an error is raised or a system value is changed. The data have a resolution in seconds, where the data are distributed across February 2014 to July 2014.

The workloads (queries) for this dataset can be divided into distributed query statements that retrieve information from two or more nodes. Further, these statements can be seen as two different types, first, parallelizing the query statements where queries can be performed independently across the nodes and second, the query statements where the query requires results from one node to be able to compute results on the other node. The priority may depend on the distribution of data across nodes.

5 Results and discussion

We divide the analysis of results into two subsections.

First, the data stores are compared based on their performance on running the queries. The results allow us to gain insights for aggregate queries. Our charts show change in the number of nodes on *x*-axis and processing rate on *y*-axis, keeping the data size constant. The results allow us to understand the performance change that occurs on the addition of new resources for the same load (or scaling out). It is important to note that data size is same for different data stores and cluster sizes.

Second, the scalability of different data store is assessment. Our charts show the change in the number of records on *x*-axis and run time on *y*-axis keeping the data store constant. The results allow us to analyze the relative gain in the performance for each data store. Further, the comparative charts help us to understand boundaries of different cluster sizes.

5.1 Scalability with number of nodes (horizontal scalability)

Workload 1 "Find the Number of Appliances"

A simple query that lists the number of appliances in the data store. Each appliance is stored as a separate table. This essentially meant that all the collections or tables are needed to be listed. This query can be parallelized appending each table, (appliance data), to a final list of appliances.

Figure 1a shows the processing rate for running workload 1 on different data stores. The chart shows that Cassandra outperforms significantly. Further, the results show that Cassandra performs around 100 and 4850 times better than HBase and MongoDB, respectively. The high difference on performance can be attributed to a centralized index in Cassandra and the absence of one in HBase and MongoDB. Figure 1b shows the processing rate for HBase and MongoDB. The chart shows that HBase performs around 47 times better than MongoDB. It is because scanning the document data store (MongoDB) is costly compared to column data store (HBase).

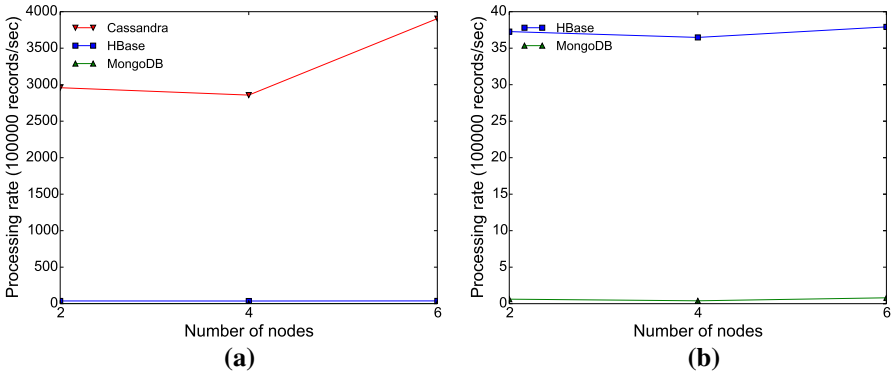

Fig. 1 Processing rate for running workload 1 **a** comparative chart for all the data stores, **b** comparative chart for HBase and MongoDB

Workload 2 "Find the Start Date, End Date, and the Number of Days of Data for an Appliance"

This query analyzed the performance of *unique* function on the date field for each of the appliance. The query can be parallelized as the data for each appliance is stored in a separated table and sorted by date.

Figure 2a shows the processing rate for the second query. The chart shows that the rate for HBase is higher than Cassandra for 2 nodes and the trend changes for 4 and 6 nodes. The results show that Cassandra perform 2 and 41 times better than HBase and MongoDB respectively.

Workload 3 "Filter-Based Queries"

When the locking flag goes from 0 to 1, it means that there is an error. A display code shows the specific error. The query is designed to find, if any of the appliances faced errors. If there are any errors, the specific start and end of the error with display code are shown. The query is the conditional *select* query. The filter required knowledge of error state (flag values) occurred in earlier rows. Therefore, depending on the distribution of the table, the query may have to wait for results from different nodes.

Figure 3a shows the processing rate for running workload 3 on different data stores. The chart shows that Cassandra performs 24 and 36 times better than HBase and MongoDB, respectively. Further, Fig. 3b shows that both HBase and MongoDB follow similar trend with slight improvement over the number of nodes.

Workload 4. "Find Mean, Min, Max, Count and Standard Deviation for a Given Column"

The queries are required to compute standard mathematical functions (such as, min, max, etc) for a particular column. Moreover, the operations could be performed in parallel depending on the distribution of the column. All the functions would come under distributed SQL statements. We rely on internal implementation of these functions for each of the data store.

Figure 4a, b shows the processing rate for running workload 4. The charts show that Cassandra performs 6.5 and 10.5 times compared to HBase and

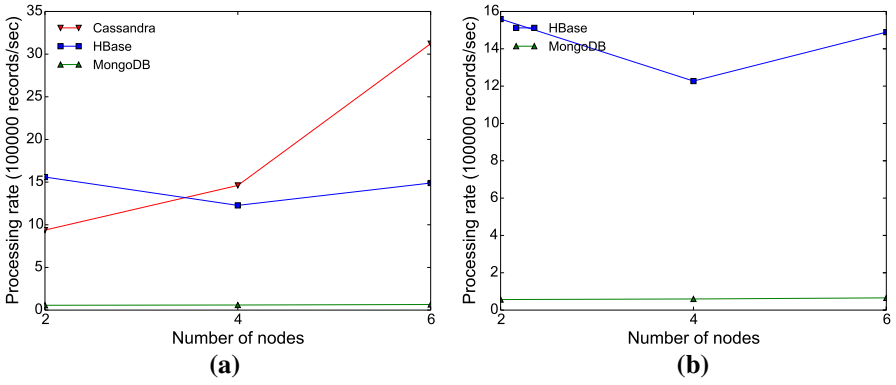

Fig. 2 Processing rate for running workload 2 **a** comparative chart for all the data stores, **b** comparative chart for HBase and MongoDB

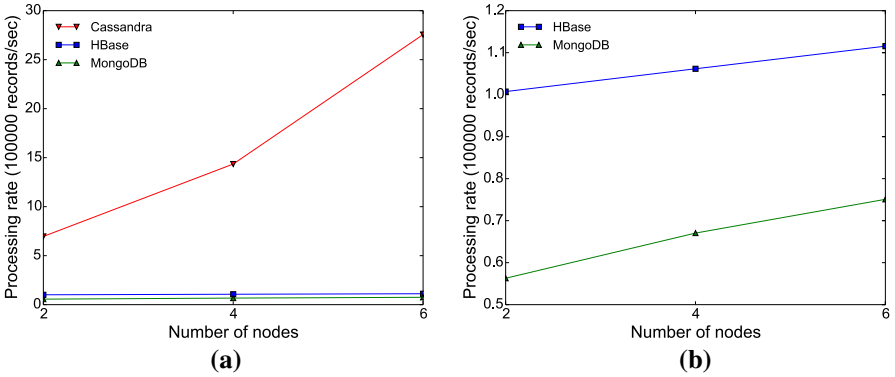

Fig. 3 Processing rate for running workload 3 **a** comparative chart for all the data stores, **b** comparative chart for HBase and MongoDB

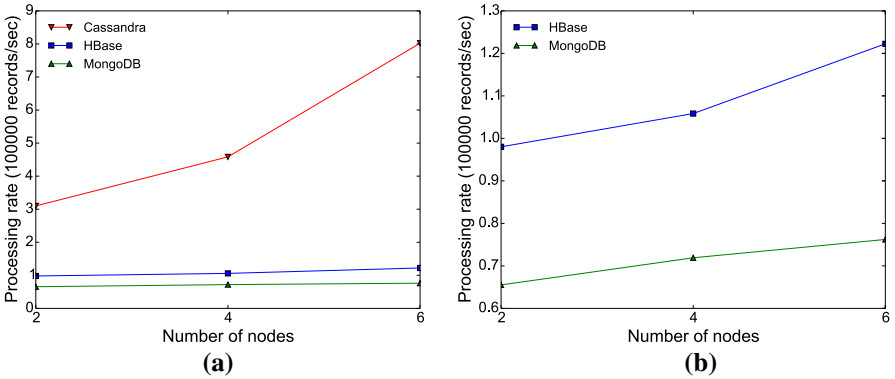

Fig. 4 Processing rate for running workload 4 **a** comparative chart for all the data stores, **b** comparative chart for HBase and MongoDB

MongoDB, respectively. The results are very similar to that of workload 3 results. This is primarily because both the queries require an iteration through all the rows in the database.

5.2 Scalability with number of records (vertical scalability)

In this section, we discuss the run time results for each data store, where we run similar workloads on different cluster sizes. The results allow us to analyze the behavior of different data stores independently. Further, Cassandra-based experiments run up to 600 million records (550 files), whereas HBase and MongoDB experiments run up to 280 million records (300 files).

5.2.1 Cassandra

Figure 5a–d shows the run time on *x*-axis and number of records on *y*-axis for running workload 1, workload 2, workload 3, and workload 4, respectively, on Cassandra. Workload 1 results are quicker and have the order of 1–2 s. The efficiency is because Cassandra creates an index of all the tables for each node. For workload 2, workload 3, and workload 4, the charts show that the difference in run time across different cluster sizes increases as the number of records increases with lower run time for larger clusters. This clearly shows that query load is efficiently distributed across the nodes in Cassandra, which results in better run time performance for larger clusters.

5.2.2 HBase

Figure 6a–d shows the run time on *x*-axis and number of records on *y*-axis for running workload 1, workload 2, workload 3 and workload 4 respectively on HBase. Workload 1 results are of the order of 1 minute. This occurs due to heavier index structure in HBase. Workload 2 results are of the order of 100 s whereas, workload 3

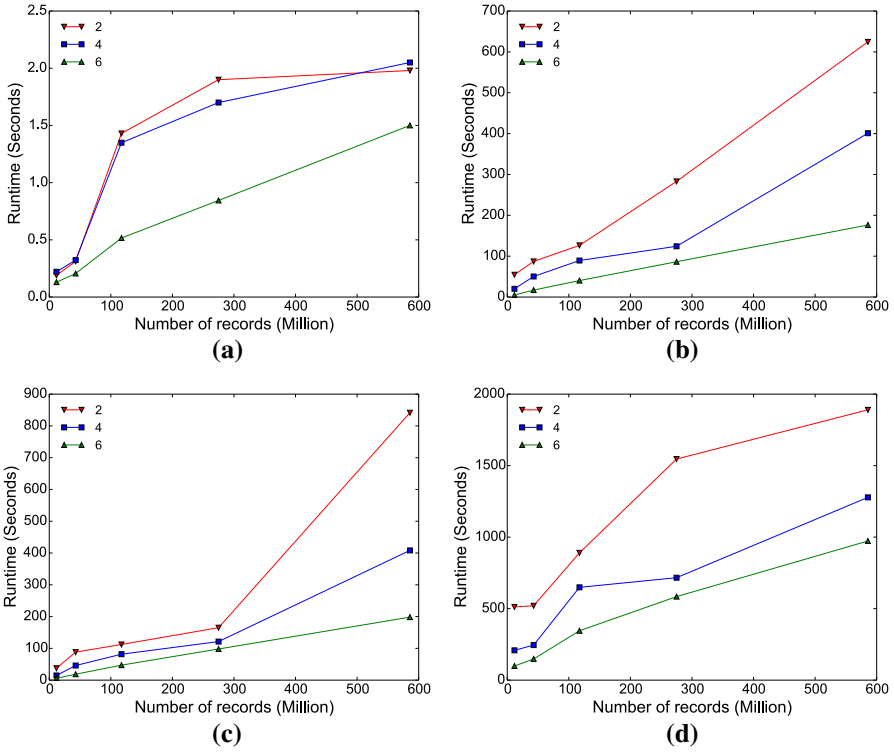

Fig. 5 Run time of the queries on Cassandra for different cluster sizes with variable data load: **a** workload 1, **b** workload 2, **c** workload 3, **d** workload 4

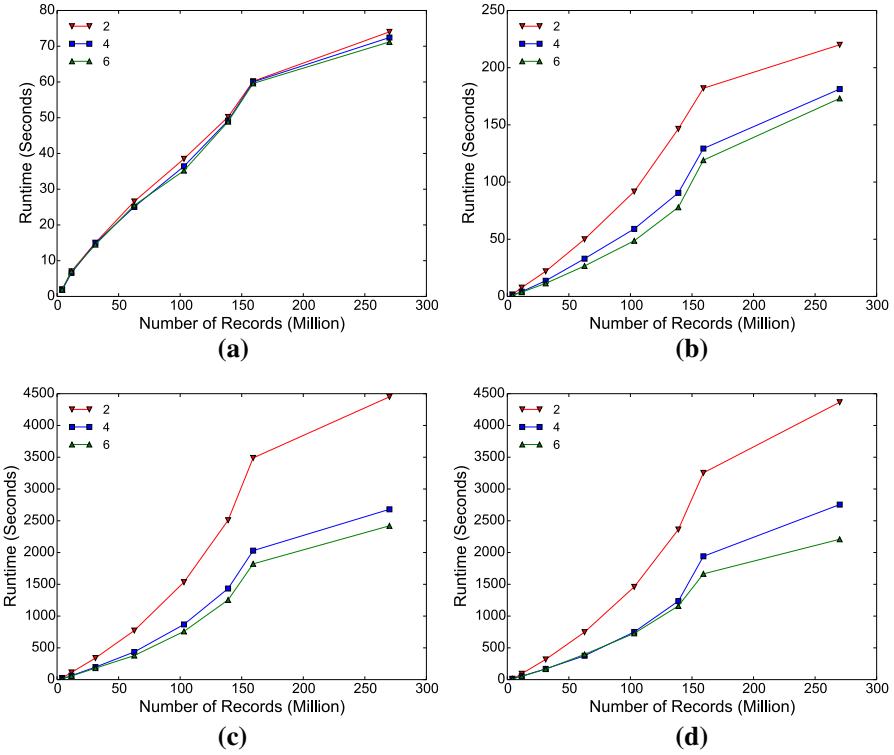

Fig. 6 Run time of the queries on HBase for different cluster sizes with variable data load: **a** workload 1, **b** workload 2, **c** workload 3, **d** workload 4

and workload 4 results have the order of 1000 s. The results occur due to parallelization of *unique* queries and sequential execution of select and function based queries.

5.2.3 MongoDB

Figure 7a–d shows the run time on x-axis and number of records on y-axis for running workload 1, workload 2, workload 3, and workload 4, respectively, on MongoDB. All the query results are of the order of 1000 s. Workload 1 is expensive because MongoDB does not have a centralized index of the table resulting in complete scan of data. Further, no significant difference can be seen across different cluster sizes. This suggests that the centralized architecture proves to be a bottleneck in adequate scaling.

In these set of experiments, most of the queries involved scanning of the tables. The results show similar trends across all the queries, where Cassandra performs better than HBase and MongoDB. We believe this is because of the architectural distinction between the three data stores. Cassandra is a decentralized peer-to-peer wide-column data store, HBase is a master–slave wide-column data store, and MongoDB is a centralized document data store. The decentralized nature of Cassandra

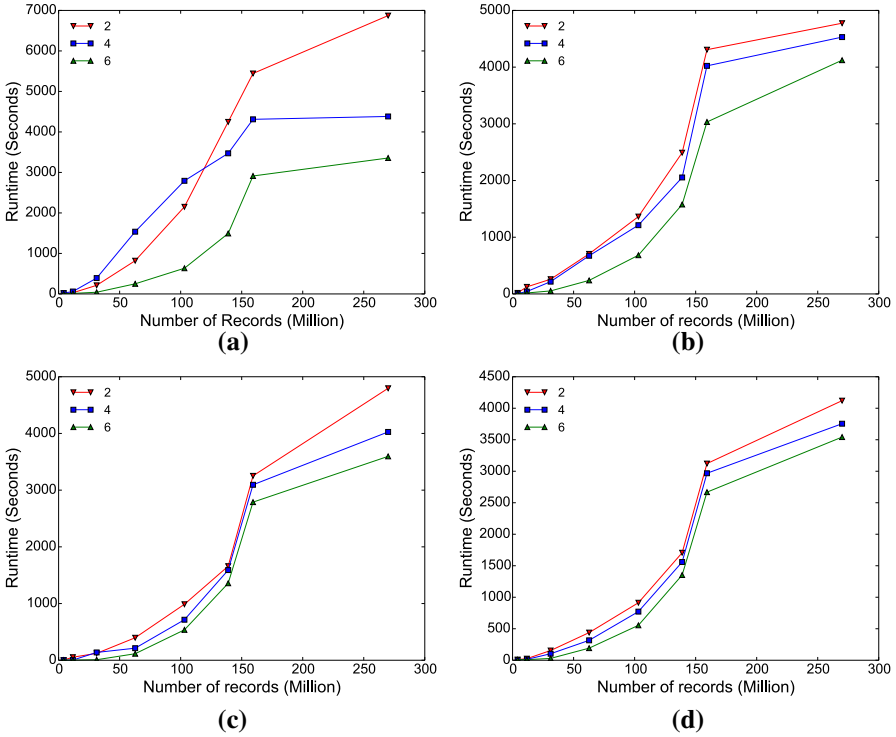

Fig. 7 Run time of the queries on MongoDB for different cluster size with variable data load: **a** workload 1, **b** workload 2, **c** workload 3, **d** workload 4

allows higher throughput and lower latency across different operations. However, we believe that with appropriate indexing structure, query performance can be significantly improved for HBase and MongoDB. For this work, we tried to observe the performance without custom optimizations for the evaluated NoSQL data stores.

6 Conclusion

This paper provides an intensive and comprehensive performance evaluation for the three widely used NoSQL systems (Cassandra, MongoDB, and HBase). The evaluation metrics include throughput, latency, and run time, and span two main scalability directions, vertical and horizontal. In this study, we test these NoSQL systems in a distributed environment using real IoT datasets. Interestingly, the experimental results show differences in the performance attitude of the studied systems. As a general finding, in most cases Cassandra would perform better than HBase and MongoDB in highly distributed environment, while HBase could be the best with few nodes cluster.

Acknowledgements This work was supported by National Research Foundation of Korea-Grant funded by the Korean Government (Ministry of Science and ICT)-NRF-2017R1A2B2012337).

References

1. Abramova V, Bernardino J (2013) NoSQL databases: Mongodb vs cassandra. In: Proceedings of the International C* Conference on Computer Science and Software Engineering, ACM, pp. 14–22
2. Abramova V, Bernardino J, Furtado P (2014) Which NoSQL database? A performance overview. Open J Databases (OJDB) 1(2):17–24
3. Adrian M (2016) DBMS 2015 numbers paint a picture of slow but steady change. https://blogs.gartner.com/merv-adrian/2016/04/12/dbms-2015-numbers-paint-a-picture-of-slow-but-steady-change/. Accessed Apr 2016
4. Anagnostopoulos I, Zeadally S, Exposito E (2016) Handling big data: research challenges and future directions. J Supercomput 72(4):1494–1516
5. Apache. https://hbase.apache.org/. Accessed July 2018
6. Aslett M (2015) NoSQL by the numbers. http://www.odbms.org/2015/07/nosql-by-the-numbers/. Accessed July 2015
7. Barbierato E, Gribaudo M, Iacono M (2014) Performance evaluation of NoSQL big-data applications using multi-formalism models. Future Gener Comput Syst 37:345–353
8. Boral H, DeWitt DJ (1984) A methodology for database system performance evaluation, vol 14. ACM, New York
9. Brewer E (2010) A certain freedom: thoughts on the cap theorem. In: Proceedings of the 29th ACM SIGACT-SIGOPS symposium on Principles of distributed computing, ACM, pp. 335–335
10. Brewer EA (2000) Towards robust distributed systems. In: PODC, vol 7
11. Cattell R (2011) Scalable SQL and NoSQL data stores. ACM Sigmod Record 39(4):12–27
12. Chang F, Dean J, Ghemawat S, Hsieh WC, Wallach DA, Burrows M, Chandra T, Fikes A, Gruber RE (2008) Bigtable: a distributed storage system for structured data. ACM Trans Comput Syst (TOCS) 26(2):4
13. Cooper BF, Silberstein A, Tam E, Ramakrishnan R, Sears R (2010) Benchmarking cloud serving systems with YCSB. In: Proceedings of the 1st ACM symposium on cloud computing, ACM, pp. 143–154
14. Corbellini A, Mateos C, Zunino A, Godoy D, Schiaffino S (2017) Persisting big-data: The NoSQL landscape. Inf Syst 63:1–23
15. Danova T (2013) Morgan Stanley: 75 billion devices will be connected to the Internet of Things by 2020. http://www.businessinsider.com/75-billion-devices-will-be-connectedto-the-internet-by-2020-2013-10. Accessed Oct 2013
16. Das N, Paul S, Sarkar BB, Chakrabarti S (2019) NoSQL overview and performance testing of HBase over multiple nodes with MYSQL. In: Abraham A, Dutta P, Mandal J, Bhattacharya A, Dutta S (eds) Emerging technologies in data mining and information security. Springer, Singapore, pp 269–279
17. Davoudian A, Chen L, Liu M (2018) A survey on NoSQL stores. ACM Comput Surv (CSUR) 51(2):40
18. Dede E, Govindaraju M, Gunter D, Canon RS, Ramakrishnan L (2013) Performance evaluation of a MongoDB and Hadoop platform for scientific data analysis. In: Proceedings of the 4th ACM workshop on scientific cloud computing. ACM, pp 13–20
19. Dey A, Fekete A, Nambiar R, Rohm U (2014) YCSB+ T: benchmarking web-scale transactional databases. In: 2014 IEEE 30th International Conference on Data Engineering Workshops (ICDEW). IEEE, pp 223–230
20. E. Corporation (2015) BenchMarking Top NoSQL databases. https://www.datastax.com/wp-content/themes/datastax-2014-08/files/NoSQL_Benchmarks_EndPoint.pdf. Accessed May 2015
21. Flores A, Ramírez S, Toasa R, Vargas J, Urvina-Barrionuevo R, Lavin JM (2018) Performance evaluation of NoSQL and SQL queries in response time for the e-government. In: 2018 International Conference on eDemocracy & eGovernment (ICEDEG). IEEE, pp 257–262
22. Fox A, Brewer EA (1999) Harvest, yield, and scalable tolerant systems. In: Proceedings of the seventh workshop on hot topics in operating systems, 1999. IEEE, pp 174–178

23. Gessert F, Wingerath W, Friedrich S, Ritter N (2017) NoSQL database systems: a survey and decision guidance. Comput Sci Res Dev 32(3–4):353–365
24. Ghazal A, Rabl T, Hu M, Raab F, Poess M, Crolotte A, Jacobsen H-A (2013) Bigbench: towards an industry standard benchmark for big data analytics. In: Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data. ACM, pp 1197–1208
25. Han J, Haihong E, Le G, Du J (2011) Survey on NoSQL database. In: 2011 6th International Conference on Pervasive Computing and Applications (ICPCA). IEEE, pp 363–366
26. Hendawi A, Gupta J, Jiayi L, Teredesai A, Naveen R, Mohak S, Ali M (2018) Distributed NoSQL data stores: performance analysis and a case study. Accepted. In: Big data 2018. IEEE, pp 2–18
27. Hendawi AM, Gupta J, Shi Y, Fattah H, Ali M (2017) The microsoft reactive framework meets the internet of moving things. In: Proceedings of the International Conference on Data Engineering, ICDE, California, USA, 2017. IEEE
28. internetlivestats. Internet live stats. http://www.internetlivestats.com/one-second/#tweets-band. Accessed Oct 2017
29. Kim H-J, Ko E-J, Jeon Y-H, Lee K-H (2018) Techniques and guidelines for effective migration from RDBMS to NoSQL. J Supercomput. https://doi.org/10.1007/s11227-018-2361-2
30. Lakshman A, Malik P (2010) Cassandra: a decentralized structured storage system. ACM SIGOPS Oper Syst Rev 44(2):35–40
31. Li Y, Manoharan S (2013) A performance comparison of SQL and NoSQL databases. In: 2013 IEEE pacific rim conference on communications, computers and signal processing (PACRIM). IEEE, pp 15–19
32. Macaulay J, Buckalew L, Chung G (2015) Internet of things in logistics: a collaborative report by DHL and Cisco on implications and use cases for the logistics industry. Report, DHL Customer Solutions & Innovation, Troisdorf
33. Martins P, Abbasi M, Sá F (2019) A study over NoSQL performance. In: World conference on information systems and technologies. Springer, pp 603–611
34. McKendrick J (2016) With Internet Of Things and big data, 92% of everything we do will be in the cloud. https://www.forbes.com/sites/joemckendrick/2016/11/13/. Accessed Nov 2016
35. Mongo. https://www.mongodb.com/. Accessed July 2018
36. Mulcahy M (2017) Big data statistics & facts for 2017. https://www.waterfordtechnologies.com/big-data-interesting-facts/. Accessed Feb 2017
37. Ntarmos N, Patlakas I, Triantafillou P (2014) Rank join queries in NoSQL databases. Proc VLDB Endow 7(7):493–504
38. Pavlo A, Paulson E, Rasin A, Abadi DJ, DeWitt DJ, Madden S, Stonebraker M (2009) A comparison of approaches to large-scale data analysis. In: Proceedings of the 2009 ACM SIGMOD International Conference on Management of Data. ACM, pp 165–178
39. Pereira DA, Ourique de Morais W, Pignaton de Freitas E (2018) NoSQL real-time database performance comparison. Int J Parallel Emerg Distrib Syst 33(2):144–156
40. Plageras AP, Psannis KE, Stergiou C, Wang H, Gupta BB (2018) Efficient IoT-based sensor big data collection-processing and analysis in smart buildings. Future Gener Comput Syst 82:349–357
41. Planet-Cassandra. http://www.planetcassandra.org/blog/cassandra-error-handling-done-right/. Accessed July 2018
42. Press G (20147) Internet of Things by the numbers: market estimates and forecasts. https://www.forbes.com/sites/gilpress/2014/08/22/internet-of-thingsby-the-numbers-market-estimatesand-forecasts/#285c030b9194. Accessed Aug 2014
43. Rabl T, Gómez-Villamor S, Sadoghi M, Muntés-Mulero V, Jacobsen H-A, Mankovskii S (2012) Solving big data challenges for enterprise application performance management. Proc VLDB Endow 5(12):1724–1735
44. Ranking. http://db-engines.com/en/ranking. Accessed July 2018
45. Sakr S, Liu A, Batista DM, Alomari M (2011) A survey of large scale data management approaches in cloud environments. IEEE Commun Surv Tutor 13(3):311–336
46. Sayce D (2016) Number of tweets per day? https://www.dsayce.com/social-media/tweets-day/. Accessed Nov 2016
47. Silva YN, Almeida I, Queiroz M (2016) Learning SQL: beyond traditional relational databases. In: SIGCSE
48. Sivasubramanian S (2012) Amazon dynamoDB: a seamlessly scalable non-relational database service. In: Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data. ACM, pp 729–730

49. Tauro CJ, Aravindh S, Shreeharsha A (2012) Comparative study of the new generation, agile, scalable, high performance nosql databases. Int J Comput Appl 48(20):1–4
50. UC Berkley. https://amplab.cs.berkeley.edu/benchmark/. Accessed July 2018
51. ul Haque A, Mahmood T, Ikram N (2018) Performance comparison of state of art NoSQL technologies using apache spark. In: Proceedings of SAI Intelligent Systems Conference. Springer, pp 563–576
52. Van der Veen JS, Van der Waaij B, Meijer RJ (2012) Sensor data storage performance: SQL or NoSQL, physical or virtual. In: 2012 IEEE 5th International Conference on Cloud Computing (CLOUD). IEEE, pp 431–438
53. Wang L, Zhan J, Luo C, Zhu Y, Yang Q, He Y, Gao W, Jia Z, Shi Y, Zhang S, et al (2014) Bigdatabench: a big data benchmark suite from internet services. In: 2014 IEEE 20th international symposium on high performance computer architecture (HPCA). IEEE, pp 488–499
54. Zikopoulos P, Eaton C et al (2011) Understanding big data: analytics for enterprise class hadoop and streaming data. McGraw-Hill Osborne Media, New York

Affiliations

Abdeltawab Hendawi[1,2] · Jayant Gupta[3] · Jiayi Liu[6] · Ankur Teredesai[7] · Naveen Ramakrishnan[8] · Mohak Shah[6] · Shaker El-Sappagh[4,5] · Kyung-Sup Kwak[4] · Mohamed Ali[7]

Abdeltawab Hendawi
hendawi@uri.edu; a.hendawi@fci-cu.edu.eg

Jayant Gupta
gupta423@umn.edu

Jiayi Liu
jason.liu@lge.com

Ankur Teredesai
ankurt@uw.edu

Naveen Ramakrishnan
Naveen.Ramakrishnan@us.bosch.com

Mohak Shah
mohak.shah@lge.com

Shaker El-Sappagh
sh.elsappagh@inha.ac.kr; shaker elsapagh@yahoo.com

Mohamed Ali
mhali@uw.edu

[1] Department of Computer Science and Statistics, University of Rhode Island, Kingston, RI, USA

[2] Faculty of Computers and Information, Cairo University, Giza, Egypt

[3] Computer Science and Engineering, University of Minnesota, Twin Cities, Minneapolis, MN, USA

[4] Department of Information and Communication Engineering, Inha University, Incheon, South Korea

[5] Information Systems Department, Faculty of Computers and Informatics, Benha University, Kaliobeya, Egypt

[6] LG Electronics, Seoul, South Korea

[7] Center for Data Science, University of Washington Tacoma, Tacoma, WA, USA

[8] Center for A. I., Robert Bosch LLC, Palo Alto, USA