PERFORMANCE EVALUATION OF SQL AND NOSQL DATABASE MANAGEMENT SYSTEMS IN A CLUSTER

Christine Niyizamwiyitira and Lars Lundberg

Department of Computer Science and Engineering, Blekinge Institute of Technology, SE-37179 Karlskrona, Sweden.

ABSTRACT

In this study, we evaluate the performance of SQL and NoSQL database management systems namely; Cassandra, CouchDB, MongoDB, PostgreSQL, and RethinkDB. We use a cluster of four nodes to run the database systems, with external load generators.The evaluation is conducted using data from Telenor Sverige, a telecommunication company that operates in Sweden. The experiments are conducted using three datasets of different sizes.The write throughput and latency as well as the read throughput and latency are evaluated for four queries; namely distance query, k-nearest neighbour query, range query, and region query. For write operations Cassandra has the highest throughput when multiple nodes are used, whereas PostgreSQL has the lowest latency and the highest throughput for a single node. For read operations MongoDB has the lowest latency for all queries. However, Cassandra has the highest throughput for reads. The throughput decreasesas the dataset size increases for both write and read, for both sequential as well as random order access. However, this decrease is more significant for random read and write. In this study, we present the experience we had with these different database management systems including setup and configuration complexity.

KEYWORDS

Trajectory queries, cluster computing, SQL database, NoSQL database, Cassandra, CouchDB, MongoDB, PostgreSQL, RethinkDB

1. INTRODUCTION

Immense volumes of data are generated continuously at a very high speed in different domains. Being unstructured and semi structured make these data heterogeneous and complex.However, efficient processing and analysis remain high priorities. The challenges include what technology in terms of software and hardware to use in order to handle these data efficiently. Processing and analysis is needed in different domain such as transportation optimization and different business analytics for telecommunication companies that seek common patterns from their mobile users in order to support business decisions.

The variety of SQL and NoSQL database management systems makes it difficult to pick the most appropriate system for a specific use case. In this paper, five data base systems are evaluated with respect to write and read throughput and latency.Throughput is interesting since telecom data is generated at high pace, and latency is also interesting since the speed of telecom data processing is critical.

DOI : 10.5121/ijdms.2017.9601

Since big data processing requires high performance computing, we use a cluster computing environment in order to take advantage of parallel computing. We consider a case of trajectory data of mobile users.

Trajectory data represents information that describes the location of the user in time and space. A typical application of such data is that a telecommunication company wants to optimize the use of cell antennas and identify different points of interests in order to expand its business. In order to successfully process trajectory data, a proper choice of database system that efficiently respond to different queries is required.

We use trajectory data that are collected from Telenor Sverige (a telecommunication company that operates in Sweden). Mobile users' positionsaretracked every five minutes for an entire week (Monday to Sunday) in a medium sized city. We are interested to know how mobile users move around the city during different hours and days of the week. This will give insights about typical behaviour in certain area at certain time. We expect periodic movement in some areas, e.g., at the location of stores and restaurants during lunch time.

Our data are spatio-temporal where at a given time t a mobile user is located at a position (x,y). The location of a mobile user is a triple (x,y,t) such that user's position is represented as a spatial-temporal point p_i with $p_i=(x_i,y_i,t_i)$.

By optimizing points of interests, different types of queries are proposed. They differ in terms of input and output:

- Distance query: which finds points of interests that are located in equal or less than a distance (or a radius), e.g., one kilometer from a certain position of a mobile user.
- K-Nearest neighbour query: that finds k nearest points of interests from a certain position of a mobile user.
- Range query: that finds points of interests within a space range (the range can be a triangle, polygon, …).
- Region query : that finds the region that a mobile user frequently passes through at certain time throughout the week.

The performance is evaluated on five open source database management systems that are capable to handle big data; Cassandra,CouchDB,MongoDB, PostgreSQL, and RethinkDB. We consider random access requests as well as sequential requests. The hardware is a cluster with four nodes that run the database, with four external load generators for random workloads, and one load generator for sequential workloads. By using this kind of data,an operator knows the locations that are the most, or the least visited during a certain time.Therefore, in order to avoid overloading and underloading at such locations, antenna planning can be updated accordingly. For business expansion, a busy location during lunch time is for instance good for putting up a restaurant.

The rest of the paper is organized as follows; Section 2 defines trajectory data concept, Section 3 summarizes related work, Section 4 gives an overview of database management systems, Section 5 describes the methodology, Section 6 presents the results. Section 7 presents some discussions and analysis, and finally Section 8 draws conclusions.

2. TRAJECTORY DATA[1]

2.1. DEFINITION OF A TRAJECTORY

A trajectory is a function from a temporal domain to a range of spatial values, i.e., it has a start and end time during which a space has been travelled (see Equation 1) [2].

$$[t_{begin} t_{end}] \rightarrow space \quad (1)$$

A complete trajectory is characterized by a list of triples $p = (x, y, t)$, thus a trajectory is defined as a sequence of positions T_{pos}.

$$T_{pos} = \{p_1, p_2, \dots, p_n\} \quad (2)$$

Where $p_i = (x_i, y_i, t_i)$ represents a spatio-temporal point, Figure 1 shows a trajectory.

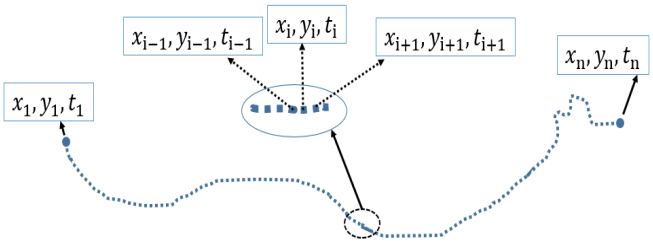

Figure 1. Mobile user's trajectory as a sequence of triples

In this study, the trajectory data space is represented by latitude and longitude;x represents latitude and y represents longitude, and time is represented by t.

2.2. DATA DESCRIPTION

Table 1. Mobile user data description

Mobile User's attributes		Short description
1	User ID	User identification
2	Site ID	Identification number of the site location
3	Weekday	Day of the week that the data has been recorded
4	Time	Clock Time that the data has been recorded
5	Profile ID	The user profile identification such as a salesperson, a store, with a mobile that runs a certain operating system like android or any other
6	Segment ID	Type of client such as Corporate client, Cost Aware, Quality Aware
7	SourceGSM	Used network technology is Global System for Mobile communication (GSM)
8	SourceUMTS	Used network technology is Universal Mobile Telecommunications System (UMTS)
9	SourceLTE	Used network technology is Long Term Evolution (LTE)
10	Easting	User's position with respect to east
11	Northing	User's position with respect to north
12	Latitude decimal	User's location in terms in latitude coordinates
13	Longitude decimal	User's location in terms in longitude coordinates
14	Cell municipality	Municipality of the Cell antenna's location
15	Cell county	County of the Cell antenna's location
16	Cell city	Cell antenna's location in terms of City
17	Cell postcode	Postcode of the Cell antenna's location
18	Cell address	Address of the Cell antenna's location

A location update is generated when a handset is generating traffic either by downloading or uploading data. The data used in this paper are collected every five minutes for an entire week in a medium sized city, i.e., the data is at rest. This data is anonymized in order to comply with the company agreement about undisclosuring users' information.

We have three datasets with different sizes.

1. Dataset0 has 6,483,398 records and 18 attributes,
2. Dataset1 has 12,966,795 records and 18 attributes,
3. Dataset2 has 25,933,590 records and 18 attributes.

Dataset2 has the biggest size, it is four times Dataset0 or two times Dataset1, the dataset size was scaled until the available resources for the experiment, the cluster memory was maximized by the size, thus we stopped at dataset2.

Table 1 shows the 18 attributes in each data record used by Telenor.

2.3. DEFINITION OF TRAJECTORY QUERIES

Trajectories queries on spatio-temporal data are the foundation of many applications such as traffic analysis, mobile user's behaviour, and many others [3] [4]. In the context of location optimization, common trajectory queries that we consider in this study are:Distance query, *k*-nearest neighbour query, Range query, and Region query. We will describe these queries in the subsections below.

Figure 2 visualizes the four query types; C_i is the location of cell i, each C_i is represented by (x_i, y_i) where x_i is latitude and y_i is longitude. A distance query returns a list of cells that are located at a certain distance from a location, e.g., within distance L from the position of C_1. The query returns the list $[C_2, C_3, C_4, C_7]$.

We can find the two cells that are closest to cell C_1, by using a *k*-nearest neighbour query with $k = 2$.

Given a triangular space, a range query returns the cells that belong to that space.

A region query returns the cell that is the most frequently visited at a certain time. e.g., cell C_8at time t_i (see Figure 2).

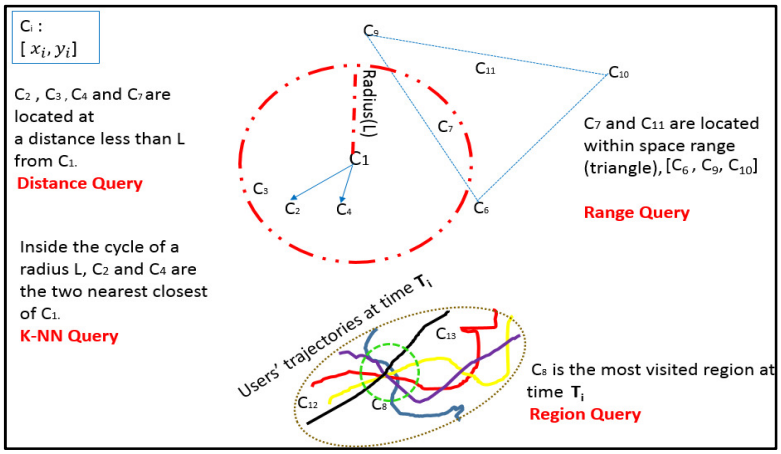

Figure 2. Visualization for Query Types

2.3.1. DISTANCE QUERY

Definition: A distance query returns all the cells in the circle whose distance from a given position is less than a threshold [3], [4]. Figure 3 shows inputs and output of a distance query.

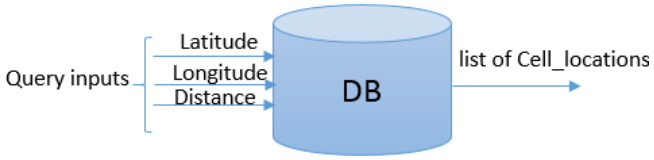

Figure 3. Distance Query

2.3.2. *K*-NEAREST NEIGHBOUR QUERY

Definition: A *k*-Nearest Neighbour (*k*-NN) Query returns from zero up to *k* cells which are the closest to a given position (x, y)[5] [6]; the *k* results are ordered by proximity. *k*-NN is bounded by a distance, and if *k* cells within the given distance from the given position is not indicated, the query behaves like a distance query.

Figure 4 shows the inputs and output of a *k*-NN query.

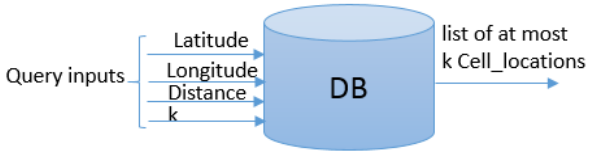

Figure 4.*k*-NNquery

2.3.3. RANGE QUERY

Definition: Range query returns all the cells that are located within a certain space shape (polygon)[3]. In this paper we only consider triangles.Figure 5 shows inputs and output of range query to find cells that are in the triangle. In this paper, the range query retrieve cells that are located inside a triangle. The triangle space is defined by nodes latitude and longitude points.

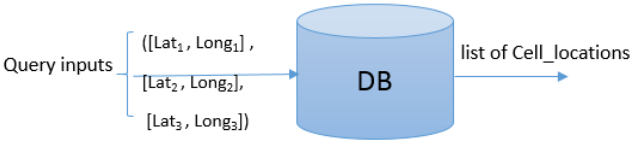

Figure 5. Range query

2.3.4. REGION QUERY

Generally, trajectories of mobile users are independent of each other. However, they contain common behaviour traits such as passing through a region at a certain periods, e.g., passing through the shopping center during lunch time.

Definition: A region query identifies the cell that is the most visited at a given point in time[3].

Figure 6 shows inputs and output of a region query to find the cell that is the most visited at time T_i. In this paper, the region query takes time as input, then at a certain fixed time users are moving around the city, region query picks up the region that most users are located.

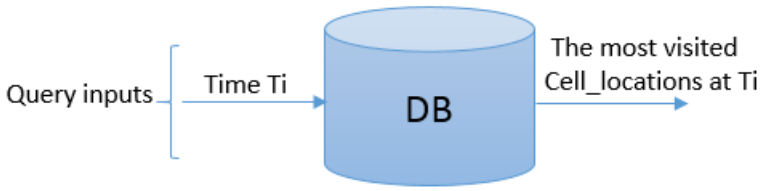

Figure 6. Region query

3. RELATED WORK

In [7], the authors proposed an approach and implementation of spatio-temporal database management systems. This approach treats time-changing geometries, whether they change in discrete or continuous steps. The same approach can be used to tackle spatio-temporal data in other database management systems. We evaluate trajectory queries on existing general purpose database management systems(Cassandra, CouchDB, MongoDB,PostgreSQL, and RethinkDB). In [8], the author describes requirements for database management systems that support location-basedservice for spatio-temporal data. A list of ten representative queries for stationary and moving reference objects is proposed. Some of those queries are related to the queries considered in Section 2.

In [9], Dieter studied trajectory moving point objects, he explained three scenarios, namely constrained movement, unconstraint movement and movement in networks. Different techniques to index and to query in these scenarios define their respective processing performance. The author modelled atrajectory as triple (x, y, t), we use the same model in this study.

In [10], the authors introduced querying moving objects (trajectory) in SECONDO, a DBMS prototyping environment particularly geared for extension by algebra modules. The querying is done using an SQL-like language. In our study, we are querying moving object using SQL and Not Only SQL (NoSQL) querying languages on top of different database management systems. The authors provide a benchmark on range queries and nearest neighbour queries in SECONDO DBMS for moving data object in Berlin. The moving object data were generated using computer simulation based on the map of Berlin [11]. This benchmark could be extended to other queries such as region queries, distance queries, and so on. In our study, we apply these queries on real world trajectory data, i.e., mobile users' trajectory from Telenor Sverige.

In [5], the authors introduced a new type of query, Reverse Nearest Neighbour (RNN) which is the opposite to Nearest Neighbour (NN). RNN can be useful in applications where moving objects agree to provide some kind of service to each other, whenever a service is needed it is requested from the nearest neighbour. An object knows objects that it will serve in the future using RNN. RNN and NN are represented by distance query in our study.

In [12], the authors studied an aggregate query language forGIS and no-spatial data stored in a data warehouse. In [13], the authors studied *k*-nearest neighbour search algorithm for historical moving object trajectories, this*k*-nearest neighbour is one of the queries that is considered in our study.

In [14], the authors presented techniques for indexing and querying moving object trajectories. These data are represented in three dimensions, where two dimensions correspond to space and one dimension corresponds to time. We also represent our data in 3D as(x, y, t), with x, y represent space whereas t represents time.

Query processing on multiprocessors was studied in [15], the authors implemented an emulator; this is a software that uses computing cluster with NVIDIA GPUs or Intel Xeon Phi coprocessors for relational query processing of a parallel DBMS on a cluster of multiprocessors. This study is different from ours in a sense that we evaluate query processing on real physical hardware with existing general purpose database management systems. Query processing using FPGA and GPU on spatial-temporal data was studied in [16]. The authors presented a FPGA and GPU implementation that process complex queries in parallel, the study did not investigate the performance on various existing database systems, adistributed environment was also not considered. In our study we investigate query processing on various database management systems on a cluster. In [17], the authors conducted a survey on mining massive-scale spatio-temporal trajectory data based on parallel computing platforms such as GPU, Map Reduce and FPGA, again existing general purpose database systems were not evaluated. The authors presented a hardware implementation for converting geohash codes to and from longitude/latitude pairs for spatio-temporal data [18], the study shows that longitude and latitude coordinates are the key points for modelling spatio-temporal data.

In our paper, we also use these coordinates for location based querying.The benchmark for NoSQL databases, namely Apache Cassandra, Couchbase, HBase, and MongoDB is presented in [19]. This benchmark was performed on Amazon Web Service (AWS) EC2 instances. They used Yahoo! Cloud Serving Benchmark (YCSB) data. In terms of throughput and horizontal scalability, Cassandra is the best, Hbase is the second, CouchBase is the third and MongoDB is the fourth. In this paper we have not considered CouchBase and HBase, since they are in memory databases, i.e., they used direct memory which is good for processing small data in real-time. Therefore this would be smaller for our workload. We used Cassandra, CouchDB, MongoDB, PostgreSQL, and RethinkDB on real-world workload instead simulated workload.

4. DATABASE MANAGEMENT SYSTEMS OVERVIEW

The presence of unstructured data stimulated the invention of new databases, since Relational Database Management Systems (RDBMS) that uses Structured Query Language (SQL) cannot handle unstructured data efficiently. A new data model, Not Only SQL (NoSQL) was introduced to deal also with unstructured data [20]. The main features of NoSQL follow the CAP theorem (Consistency, Availability, and Partition tolerance). The core idea of CAP is that a distributed system cannot meet these three needs simultaneously (see Figure 7). Depending on the data models, NoSQL can be relational, key value based, column based, and document based. In this study we choose five open source databases that have diverse features of SQL (PostgreSQL) and NoSQL (Cassandra, CouchDB, MongoDB and RethinkDB).

A key value data model means that a value corresponds to a key, in column based systems data arestored by column, each column is an index of the database, queries are applied to columns, whereby each column is treated one by one. A document-based database stores in the JSON or XML format, each document, is indexed and it has a key.

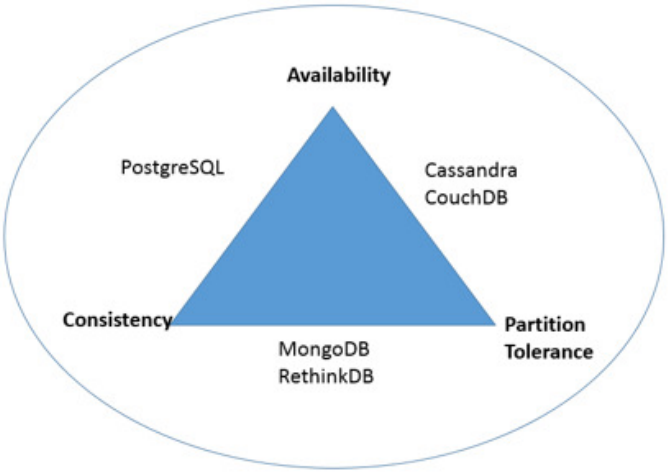

Figure 7. Principles of Distributed Database systems.

4.1. CASSANDRA

Apache Cassandra is an open-source NoSQL column based database. It is written in Java, it is a top level Apache project born at Facebook and built on Amazon's Dynamo and Google's BigTable. It is a distributed database for managing large amounts of structured data across many commodity servers, while providing highly available service with no single point of failure. In CAP, Cassandra has availability and partition tolerance (AP) with eventual (delayed) consistency. Cassandra offers continuous availability, linearly scaling performance, operational simplicity and easy data distribution across multiple data centers and cloud availability zones. Cassandra has a masterless ring architecture[21].The keyspace is similar to database in RDBMS, inside keyspace there are tables which are similar to tables in RDBMS, column and rows are similar to those of RDBMS' tables. The querying language is Cassandra Query Language (CQL) that is very similar to SQL [22]. Cassandra does not natively support spatial indexing but this can be extended via Stratio's Cassandra Lucene index. Stratio's Cassandra Lucene Index is an additional module for Apache Cassandra, it extends its index functionality to provide near real time search for text search, field based sorting, and spatial index.[23].

4.2. COUCHDB

CouchDB is written in Erlang and it stores data as JSON documents. Access documents and query indexes with a web browser, via HTTP. CouchDB indexes, combines, and transforms documents with JavaScript. Itis highly available and partition tolerant, but also eventually consistent,CouchDB supports masterless setup[23].The system does not natively support spatial queries, we add a module GeoCouch for spatial indexon CouchDB[25].

4.3. MONGODB

MongoDB is an open-source NoSQL document database, itis written in C++. MongoDB has a database, inside the database there are collections, like tables in RDBMS. Inside a collection there are documents, these are like a tuple/row in RDBMS, and inside a document there are fields which are like columns in RDBMS [25][26]. MongoDB is consistent and partition tolerant. MongoDB has natively a built in function for spatial queries and it has a sharding (separating very large database into smaller, faster and easily manageable parts called shards across cluster's nodes) feature to support horizontal scalability of the database in master/slave fashion[27].

4.4. POSTGRESQL

PostgreSQL is an open source object RDBMS written in C that has two features according to the CAP theorem; those are availability, i.e., each user can always read and write, and consistency, i.e., all users have the same view of data. PostgreSQL organises data in columns and rows [28, p. 3]. PostgreSQL does not natively support horizontal scalability as well as spatial queries, PostgreSQL is extended by CITUS and PostGIS to support scalability in master/slave fashion and spatial queries indexing respectively[29][30].

4.5. RETHINKDB

RethinkDB is an open source NoSQL database system. RethinkDB is written in C ++, it is horizontally scalable in a master/slave setup, it is mostly designed to facilitate real-time updates for query results to applications[31]. RethinkDB natively support spatial queries using GeoJson.The system uses the ReQL query language that is available for Python, Ruby, and Java.

5. METHODOLOGY

This section presents details about the experimental setup, hardware and software; the measurement procedure is also explained. All the data are represented in the comma separated values (CSV) format.

5.1 EXPERIMENT SETUP

A cluster is made up of 4 nodes, each node is Dell powerEdge R320 with operating system: Ubuntu 14.04.3 LTS x86_64. Each node has 23 GB RAM, disk size of 279.4GB, and a processor (Intel(R) Xeon(R) CPU E5-2420 v2) with12 cores, each core is hyperthreaded into 2 cores, which gives 24 virtual cores.These servers run Java development kit jdk 1.8.0.72.These servers are only running our database management systems, nothing else.Another machine (called load generator) with the same features outside of this database cluster generates the load for sequential writing and reading towards the cluster. This setup is described in Figure 8. We use four load generators for random writing and reading, each of these generators has also the same features mentioned earlier. Figure 9 shows the setup for random load.

Cassandra 3.0.3 is installed at each of the nodes in the cluster in a ring topology with each node has the same role as the other, i.e. master/master fashion. Stratio Lucene is installed and connected to Cassandra at each of the nodes. The replication factor equals the number of nodes, i.e., every node has the same copy of data[32]. The consistency level is Quorum, i.e., return most recent data from a majority of replicas[34].

CouchDB is also installed on each of the cluster nodes in master/master fashion, after that Geocouch is installed and connected to CouchDB[23][24].

MongoDB is installed on the cluster in master/slave fashion, where the master mongos (mongo master server) is installed on one of the nodes, and the three config servers which act like slave servers are installed on the remaining three nodes[35][34].

PostgreSQL is also installed with PostGIS on each of the cluster nodes. After that, CITUS is connected to PostgreSQL in order to support the distribution, and as a result PostgreSQL becomes a distributed database system in master/slave fashion where one of the nodes acts as the master and the others as the slaves [30][35].

RethinkDB is also installed at each of the cluster nodes in a master/slave fashion; one node is a master [31].

Table 2 shows the details of different features of the database management systems that we are evaluating. In Table 2, BASE is Basically Available, Soft state, Eventual consistency, and ACID is Atomicity, Consistency, Isolation, and Durability.

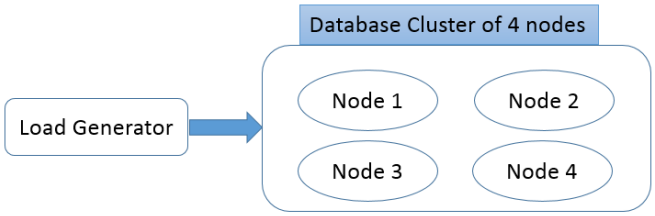

Figure 8. Experiment setup for sequential workload

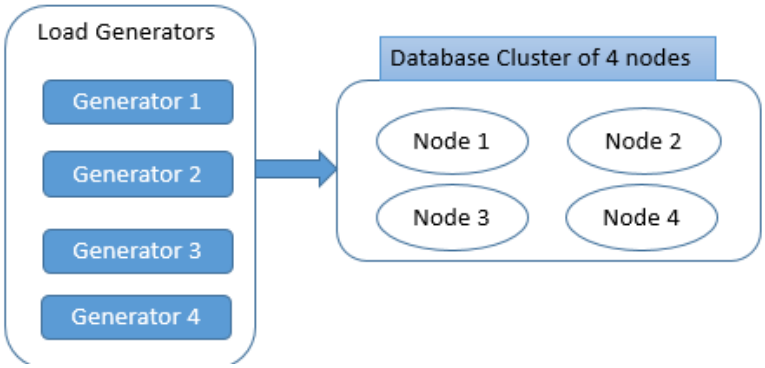

Figure 9. Experiment setup for random workload

Table 2. Database systems features

	Cassandra	CouchDB	MongoDB	PostgreSQL	RethinkDB
Extension module to support Spatial query	Stratio's Cassandra Lucene Index	GeoCouch	Natively support Spatial query	PostGIS	Natively support Spatial query
Extension module to support distributed computing	Natively distributed		Natively distributed	CITUS	Natively distributed
Language	Java	Erlang	C++	C	C++
Query Language	Cassandra Query Language (CQL)	Json	Json	Structure Query Language (SQL)	RethinkDB query Language (ReQL in python)
Partitioning method	Sharding	Sharding	Sharding	Sharding	Sharding
Replication method	Master/Master	Master/Master	Master/slave	Master/slave	Master/slave
Transaction property	BASE	BASE	BASE	ACID	BASE
Version	3.0.3	1.6.1	3.0.9	9.5.3	2.3.1
License	Apache 2.0	Apache 2.0	GNU AGPL v3.0	PostgreSQL	AGPL
Data Model	Column+key	Document+ Key	Document+ Key	Column+ row (relational DBMS)	Document+ Key
First Release	2010	2005	2007	1986	2009

5.2 WRITE PROCEDURE

Two ways of writing are considered; sequential and random order. During sequential writing, the workload is generated by one load generator machine towards the clusterand the given entire dataset is written in sequential order.

For random writing, there are four load generators, each generator contains the same dataset. Each generator makes a write request that writes a quarter of the entire dataset size records in a random order into the database cluster.

For Cassandra, data into CSV format is imported into the Cassandra database cluster. The node that receives the request will get a list of N nodes responsible for replicas of the keys in the write request from ReplicationStrategy. It then sends a RowMutation message to each of the N nodes. Then each node append the write to the commit log and update the in-memory Memtable data structure with the write. All the writes either random or sequential, are written using SSTable loader.

For CouchDB, the CSV data format is imported from the load generator into CouchDB using a couch import script that is written in Node.js. After writing, each row in the source file becomes a document. In CouchDB, every node in the cluster participates in the importing and writing. Writing are done using http_bulk.

For MongoDB, CSV data format is imported using mongo import script towards mongos (master). Each row becomes a document, thereafter mongos shards the data across the cluster.
For PostgreSQL, CSV data format is imported and written on the master, which shards it across the cluster nodes.

For RethinkDB, CSV data format is imported and written to the database through the master node which shards it across the cluster.

When the writing is completed, the message that indicates how many records are written during how long time, will appear on the load generator machine (s).
Each record (row) in all datasets have the same size, i.e., each row is 7.94 KB in CSV format.

5.3 READING PROCEDURE

Reading is also conducted in two ways; sequential and random. In the sequential procedure, the read request is generated from one load generator, each of the queries, distance, *k*-nearest neighbour, range, and region is sent 10 times to each of the database systems(Cassandra, CouchDB, MongoDB, PostgreSQL, and RethinkDB).

For random read, read request, i.e., queries, are generated from 4 load generators, and the responses are gathered at the load generators.

5.4 MEASUREMENT PROCEDURE

In this study, we measure the write and read latency,as well as the throughput. The latency measured in these experiments shows how long each individual write or read request takes to be processed. It does not include network latency between the load generator and the database cluster. Instead, it is measured from the database perspective, i.e., the time that is required to process a single request.

The write latency is the number of milliseconds required to fulfill a single write request. The time period starts when one of the cluster nodes receives write request from the load generator, and ends when the nodes complete a write request.

The read latency is the number of milliseconds required to fulfill a read request. The time period starts when one of the cluster nodes receives read request from the load generator, and ends when the node completes a read request.All the results are the average of ten runs. Measurements are conducted on three datasets of different size, namely dataset0, dataset1, and dataset2 as defined in Section 2. Read latency is measured with respect to the four different queries as defined also in Section 2.

For the write latency, the workload is 100% write only, and for read latency, the workload is 100% read.

We measure the throughput as operations per second, *Throughput = number of completed operations / time to complete those operations.* The latency is measured on the database cluster servers, whereas the throughput is measured on load generators.

6. RESULTS

All datasets are populated into Cassandra and PostgreSQL, and CouchDB without any transformation. Whereas, in MongoDB and RethinkDB, coordinates attributes (latitude and longitude) were combined into an array location attribute in order to be able to use spatial function in MongoDB and RethinkDB. Results for dataset0 and dataset1 are presented as tables in the appendix. Whereas results from dataset2 are plotted in this section.

Results in figures 10 and 11show the write throughput and latency for dataset2 for both sequence and random. Figures 12-19 show the read throughput and the latency for distance, k-NN, range, and region queries in dataset2. The throughput is measured as operations per second whereas the latency is presented in milliseconds.

During the write workload, MongoDB, PostgreSQL, and RethinkDB use one master node only therefore, their latency and throughput are consistent throughout all nodes. However, Cassandra and CouchDB scale their throughput as the number of nodes increases. CouchDB has a slightly inconsistent latency as the number of nodes increases, i.e., the latency goes up and down a little as the number of nodes increases.

The dataset size impacts the throughput negatively, especially for random write since the search grows as the dataset size increases. Therefore, the throughput decreases and the latency increases (see Appendix and figures 10(b)and 11(b)).

During read, Cassandra has in most cases the highest throughput, PostgreSQL has the second highest throughput, and MongoDB has the third whereas CouchDB has the lowest. The read throughput scales up as the number of nodes increases for all the database systems, see figures 12,14,16, and 18. Generally, MongoDB has the lowest read latency for higher number of nodes, see figures 13,15,17, and 19.

For one node, the read latency for Cassandra, MongoDB, and PostgreSQL are similar, and they are significantly lower than CouchDB and RethinkDB.

Generally, Cassandra has the highest read throughput. MongoDB and PostgreSQL have almost similar read throughput following Cassandra.RethinkDB has the fourth highest throughput, whereas CouchDB has the lowest throughput. This is shown in figures 12,14,16,and 18 for different queries. Database systems installed into master/slave fashion exhibit immediate writing consistency, e.g., MongoDB, PostgreSQL, and RethinkDB. Whereas those installed into master/master fashion present eventual consistency, e.g., Cassanda and CouchDB.

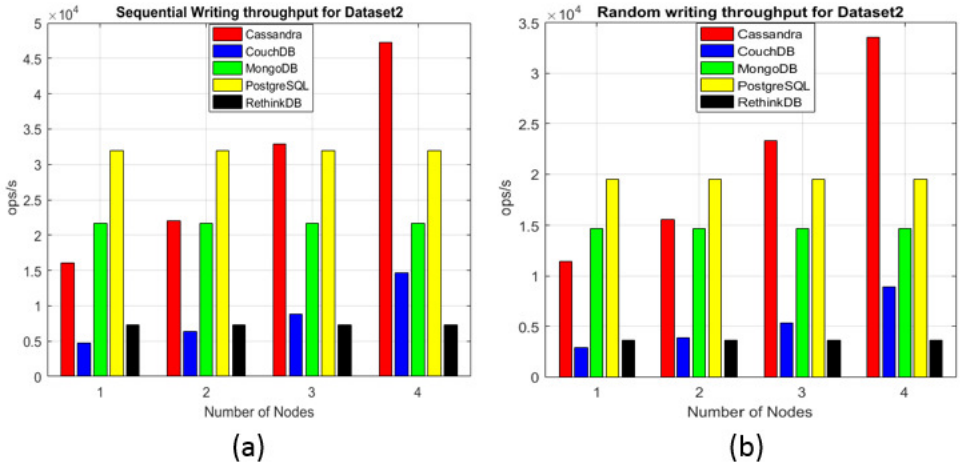

Figure 10. (a) Sequential write throughput for dataset2, **(b)** Random write throughput for dataset2

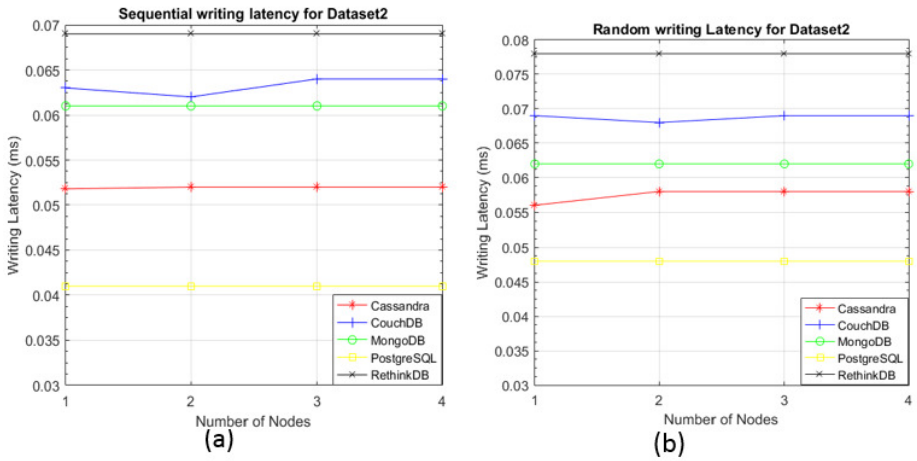

Figure 11. (a) Sequential write latency for dataset2, (b) Random write latency for dataset2

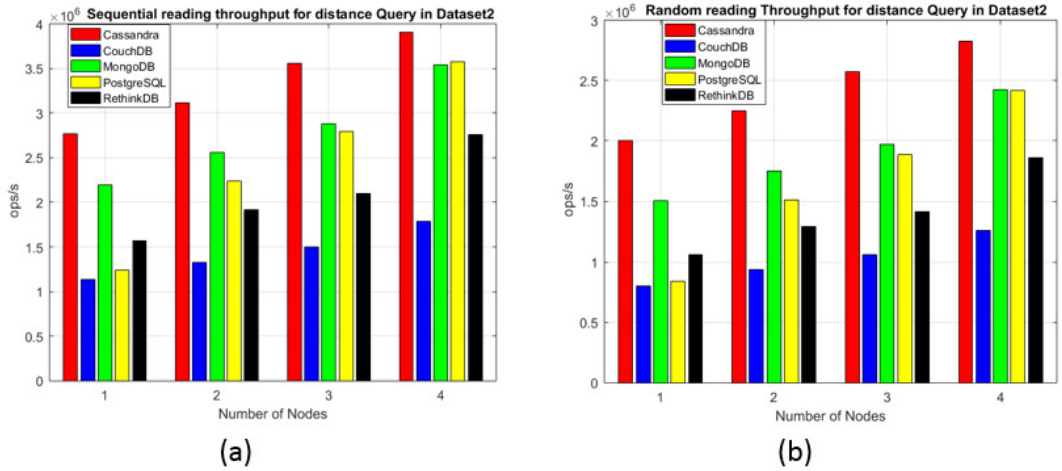

Figure 12. (a) Sequential read throughput for distance query in dataset2, **(b)**Random read throughput for distance query in dataset2

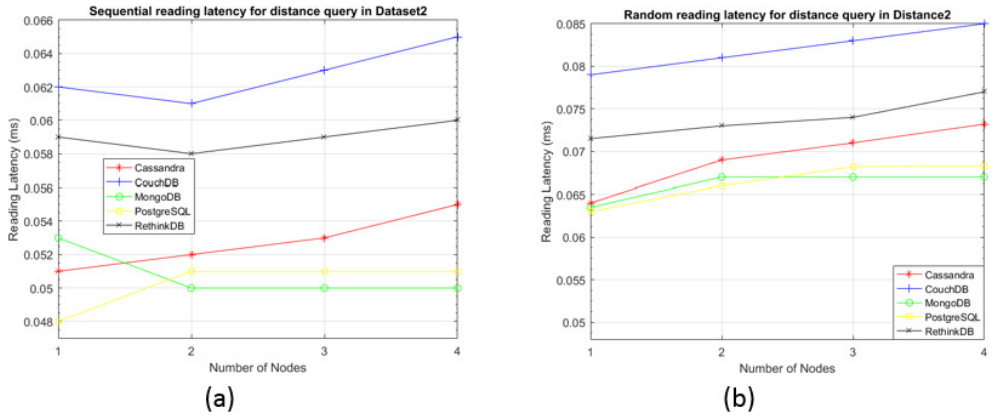

Figure 13. (a) Sequential read latency for distance query in dataset2, (b)Random read latency for distance query in dataset2

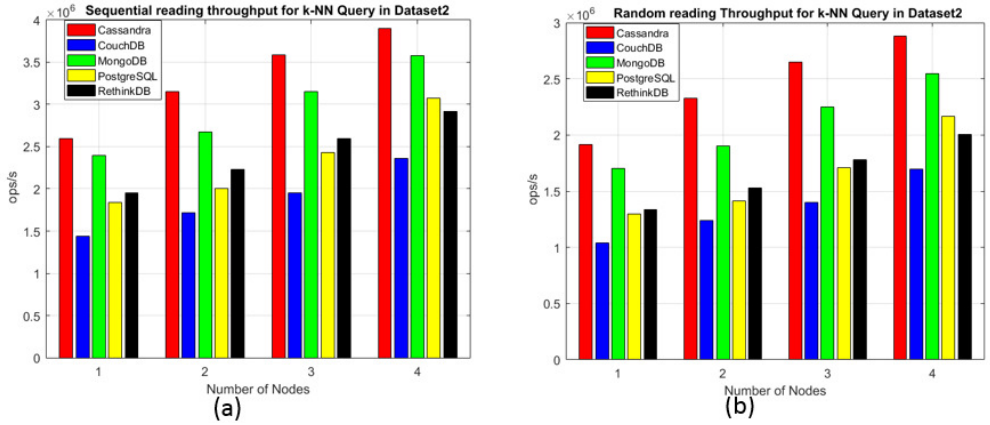

Figure 14. (a) Sequential read throughput for *k*-NN query in dataset2, **(b)**Random read throughput for *k*-NN query in dataset2

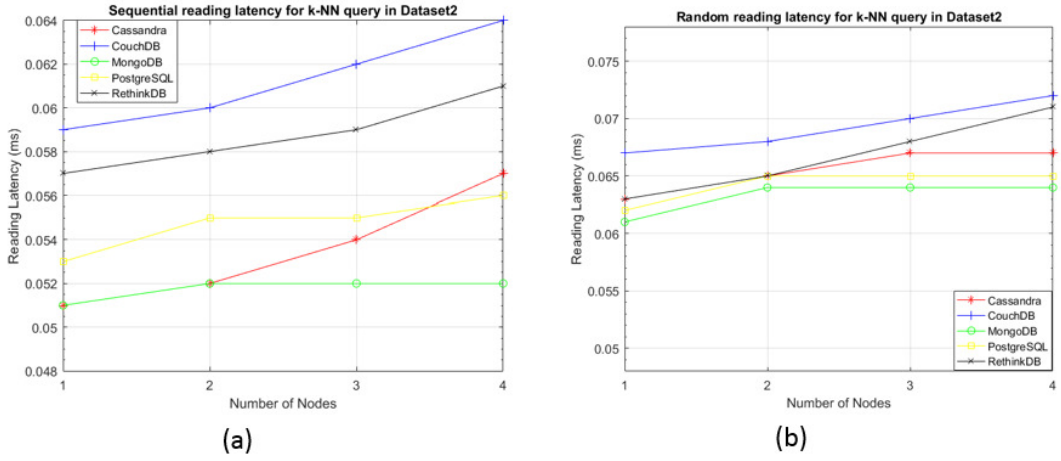

Figure 15. (a) Sequential read latency for *k*-NN query in dataset2, **(b)** Random read latency for *k*-NN query in dataset2

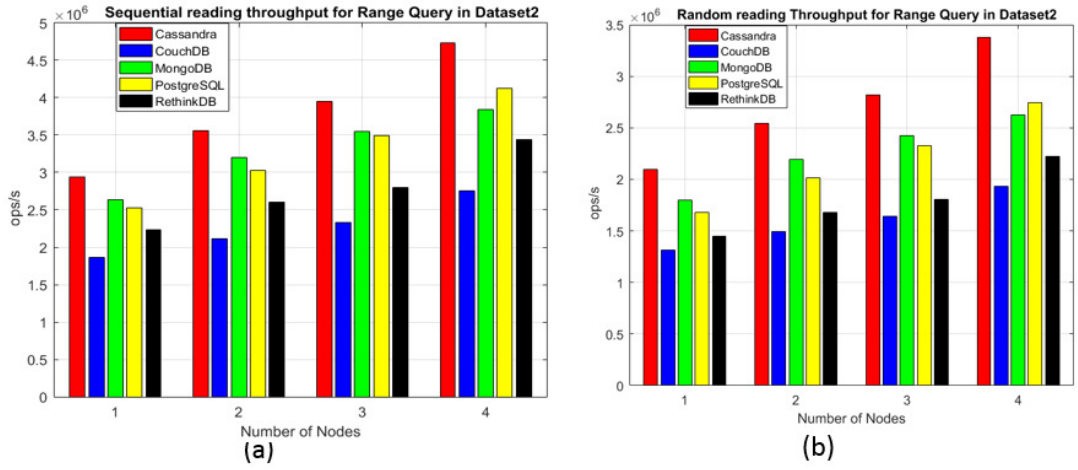

Figure 16.(a) Sequential read throughput for Range query in dataset2, **(b)**Random read throughput for Range query in dataset2

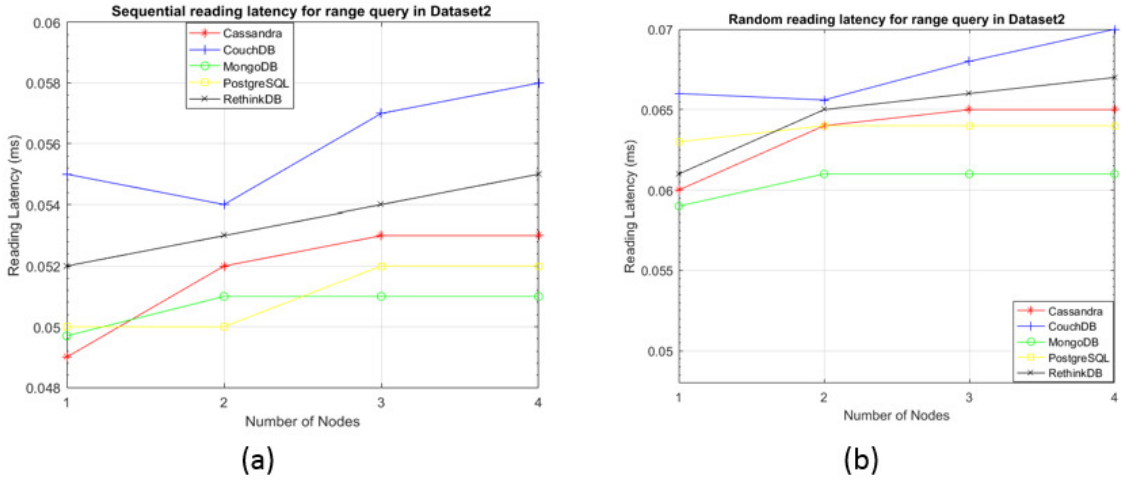

Figure 17. (a) Sequential read latency for range query in dataset2, **(b)** Random read latency for range query in dataset2

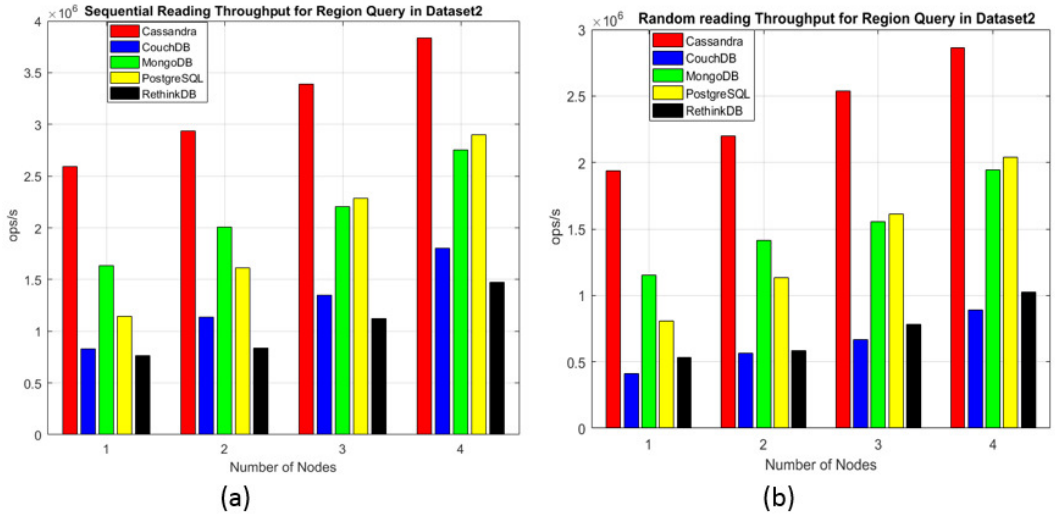

Figure 18. (a) Sequential read throughput for region query in dataset2, **(a)**Random read throughput for region query in dataset2

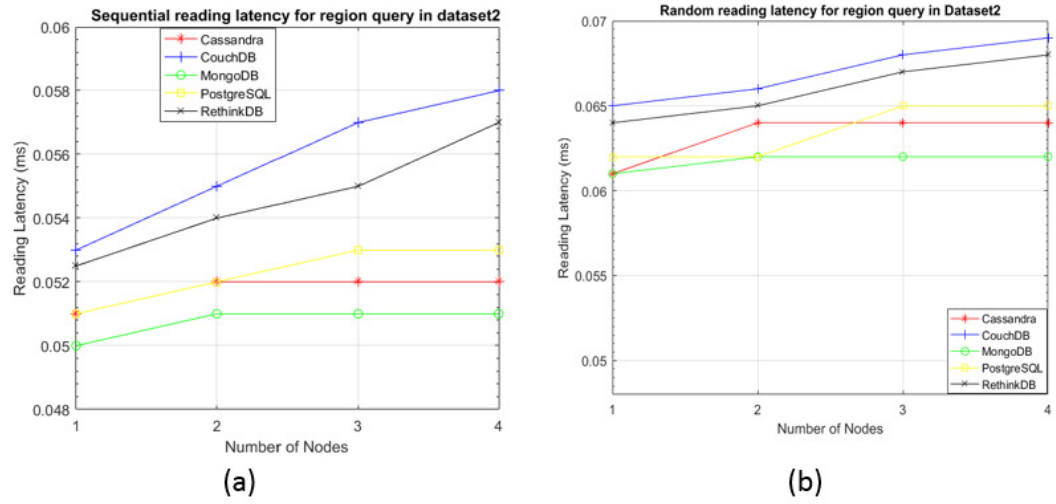

Figure 19. (a) Sequential read latency for region query in dataset2, **(b)**Random read latency for region query in dataset2

7. DISCUSSION AND ANALYSIS

In terms of scalability, Cassandra outperforms the other database systems throughout our experiments. Cassandra shows lower write latency for one node and it slightly increases for two nodes, then it stays stable for more nodes. Cassandra does not presents the best write and read latency, but it has the highest throughput, this shows that Cassandra has more parallelism.

PostgreSQL presents the lowest write latency, followed by MongoDB, which is followed by Cassandra, CouchDB, and RethinkDB which has the highest write latency. In general, Cassandra has the highest write throughput as the number of nodes increases whereas RethinkDB has the lowest throughput.

CouchDB is scalable, however, the slowest in reading and its reading throughput is similarly affected. We observe that Cassandra and CouchDB present similar speed up, However, Cassandra has higher throughput than CouchDB.The reason is that CouchDB serves mainly as backend to serve the web whereby retrieving a lot of records at the same time may become very slow.Usually, there will be cashing functions and closest region hosting that will support CouchDB when it is backing the web. The writing and reading throughput of CouchDB is not as high as expected; this is caused by fetching data over HTTP protocol which is essentially a high latency protocol. Therefore, CouchDB scalability does not exploit the parallelism by achieving higher throughput as expected.MongoDB, PostgreSQL, and RethinkDB are installed in master slave fashion, thus they are not scalable for write since only the master node writes.

PostgreSQL presents the best write latency and the highest throughput. From one node until three nodes PostgreSQL outperforms other database systems. This is due to delayed commit that happens at the end of whole dataset write workload, thus speeding up the writing. Introducing an explicit commit after each record (row) insert could slow down PostgreSQL significantly. Delayed commit is usually the default in PostgreSQL, it may cause data loss in case of database crash, and therefore it should not be used for very sensitive data like bank transactions.

MongoDB has in most cases the lowest read latency because it has a spatial function that quickly process spatial queries. MongoDB is fast for reads because it shards data across nodes, when a query is launched, only the concerned nodes will respond to the query. This avoid going over the whole dataset. The same principle is also applied in PostgreSQL with the help of CITUS, the extension that horizontally scales PostgreSQL across commodity servers using sharding.

Since RethinkDB is designed for real-time applications such as game live score and online multiplayer games, during which writing must be acknowledged by the server and subsequently are available to the client. Such data are in most cases relatively small in order to be processed at low latency, due to big size datasets that are used in our experiments, RethinkDB suffers from high latency and low throughput. However, the performance is better for read than write.
As expected, sequential processing has higher throughput and a little shorter latency than random processing. Increase of the dataset size causes the throughput to decrease significantly, this is a result of the overhead that becomes higher as the dataset's size increases. The latency is also affected by the increase of the datasetin such a way that the latency becomes a bit higher. However, this increase is not significant. This is intuitively true since the throughput is measured from the load generator, and latency is measured from the database servers.

The decrease of throughput with the increase of the dataset's size is even more noticeable for random writing and reading. Random writing and reading has lower throughput comparing to sequential writing and reading for dataset0, dataset1, and dataset2 respectively. This is caused by the overhead due to the random search order across the entire dataset, thus the search becomes exhaustive for both writing and reading. We saw a throughput decrease of almost 10% from sequential to random processing.

In order to get better results, we have used new hardware and we have used workload of different sizes. We considered also random and sequential processing for both write and read. Compared to the benchmarking of Cassandra, Couchbase, HBase, and MongoDB [19], we have similar trends where Cassandra outperforms MongoDB. Moreover, we include the comparison for five database systems, Cassandra, CouchDB, MongoDB, PostgreSQL and RethinkDB. Our results can serve as benchmark for other studies.

8. CONCLUSIONS

In this paper, we evaluate the write and read throughput as well as the latency of five SQL and NoSQL database management systems namely; Cassandra, CouchDB, MongoDB, PostgreSQL, and RethinkDB. The evaluation is conducted on a cluster using telecommunication data collected from Telenor Sverige. We did measurements on three datasets of different sizes; dataset0, dataset1, and dataset2.We measured the write throughput and latency of each of the datasets, and the readthroughput and latency for four queries, namely distance, *k*-nearest neighbour, range, and region queries. Both writing and reading are experimented in sequential, and in random on a database cluster system of four nodes.

For read queries, all database management systems are scalable as the number of nodes increases.However, only Cassandra and CouchDB show scalability for data writing. It is observed that as the dataset's size increases the throughput decreases and the latency increases.

The write throughput results show that on a single server, PostgreSQL performs better than others whereas Cassandra exhibits the highest throughout for higher number of nodes. PostgreSQL also presents the lowest latency for all writes. The reading results for all four queries show that Cassandra has the highest throughput even though it does have the lowest latency. This is a result of more parallelism in Cassandra. CouchDB has the lowest read throughput and highest latency though it is scalable, i.e., as the number of nodes increase the throughput increases as well.

In our experiments, we observed that in some cases PostgreSQL when featured by CITRUS, shows lower reading latency and horizontal scalability features than MongoDB, CouchDB, and RethinkDB. If the data to be processed requires the flexibility of traditional relational database (SQL), PostgreSQL would be preferred, if scalability matters, one would choose Cassandra. MongoDB, CouchDB, and RethinkDB would favour data that are transferred over the web, since they are document oriented that are easy to interpret for the web.

During our experiments, we experienced installation challenges of different database management systems. In terms of installation, Cassandra was straight forward except that spatial query extension that was challenging to be incorporated into the system. CouchDB was the most challenging, it took more time than the other database systems, especially installing it in a distributed fashion on many nodes. MongoDB was also straight forward with sharding that was a bit challenging. PostgreSQL was straight forward.However incorporating the distributing platform CITUS was challenging. RethinkDB was the easiest to install. By ranking these database systems according to easiness of installation, RethinkDB is the first, MongoDB is the second, Cassandra is the third, PostgreSQL is the fourth, and CouchDB is the fifth.

As far as mobile users' data analytics is concerned, since the processing and analysis is not done on the fly as the data come in, immediate consistency is not a big issue. Hence Cassandra would suits to process it, because it has high throughput and a relatively low latency with eventual consistency and availability across the cluster.

APPENDIX

I. Writing Throughput (Operations Per Second) And Latency In Milliseconds

1. DATASET0

A. SEQUENTIAL WRITING

Table 3. Throughput(Th) and its standard deviation (Stdv)

Nodes	Cassandra		CouchDB		MongoDB		PostgreSQL		RethinkDB	
	Th.	Stdv	Th.	Stdv	Th.	Stdv	Th.	Stdv	Th.	Stdv
1node	18222.02	0.55	5178.01	0.55	24437.98	0.72	39848.78	0.39	14714.92	0.41
2nodes	26006.40	0.54	6538.31	0.49	24437.98	0.72	39848.78	0.39	14714.92	0.41
3nodes	35899.21	0.42	9113.57	0.9	24437.98	0.72	39848.78	0.39	14714.92	0.41
4nodes	48311.46	0.51	15709.71	0.80	24437.98	0.72	39848.78	0.39	14714.92	0.41

Table 4. Latency (Lat.) and its standard deviation (Stdv)

Nodes	Cassandra		CouchDB		MongoDB		PostgreSQL		RethinkDB	
	Lat.	Stdv	Lat.	Stdv	Lat.	Stdv	Lat.	Stdv	Lat.	Stdv
1node	0.036	0.000045	0.044	0.000055	0.039	0.000052	0.033	0.000039	0.049	0.000041
2nodes	0.037	0.000051	0.043	0.000049	0.039	0.000052	0.033	0.000039	0.049	0.000041
3nodes	0.037	0.000044	0.045	0.00002	0.039	0.000052	0.033	0.000039	0.049	0.000041

B. RANDOM WRITING

Table 5. Throughput (Th) and its standard deviation (Stdv)

Nodes	Cassandra		CouchDB		MongoDB		PostgreSQL		RethinkDB	
	Th.	Stdv	Th.	Stdv	Th.	Stdv	Th.	Stdv	Th.	Stdv
1node	15488.72	0.75	3883.51	0.65	20039.14	0.71	29886.59	0.81	9417.55	0.72
2nodes	22105.44	0.61	4903.73	0.65	20039.14	0.71	29886.59	0.81	9417.55	0.81
3nodes	30514.33	0.72	6835.18	0.62	20039.14	0.71	29886.59	0.81	9417.55	0.81
4nodes	41064.74	0.59	11782.28	0.58	20039.14	0.71	29886.59	0.81	9417.55	0.81

Table 6. Latency (Lat.) and its standard deviation (Stdv)

Nodes	Cassandra		CouchDB		MongoDB		PostgreSQL		RethinkDB	
	Lat.	Stdv	Lat.	Stdv	Lat.	Stdv	Lat.	Stdv	Lat.	Stdv
1node	0.0383	0.000056	0.048	0.000066	0.043	0.000058	0.032	0.000052	0.054	0.000061
2nodes	0.0385	0.000062	0.046	0.000053	0.043	0.000058	0.032	0.000052	0.054	0.000061
3nodes	0.0392	0.000057	0.050	0.000059	0.043	0.000058	0.032	0.000052	0.054	0.000061
4nodes	0.0399	0.000060	0.0499	0.000064	0.043	0.000058	0.032	0.000052	0.054	0.000061

1. DATASET1

A. SEQUENTIAL WRITING

Table 7 Throughput (Th) and its standard deviation (Stdv)

Nodes	Cassandra		CouchDB		MongoDB		PostgreSQL		RethinkDB	
	Th.	Stdv	Th.	Stdv	Th.	Stdv	Th.	Stdv	Th.	Stdv
1node	16976.68	0.65	4799.31	0.75	22395.15	0.72	34522.88	0.83	10033.11	0.63
2nodes	23414.21	0.68	6426.20	0.68	22395.15	0.72	34522.88	0.83	10051.77	0.63
3nodes	35255.01	0.73	8919.24	0.61	22395.15	0.72	34522.88	0.83	10033.11	0.63
4nodes	47707.11	0.69	15081.17	0.53	22395.15	0.72	34522.88	0.83	10033.11	0.63

Table 8. Latency (Lat.) and its standard deviation (Stdv)

Nodes	Cassandra		CouchDB		MongoDB		PostgreSQL		RethinkDB	
	Lat.	Stdv	Lat.	Stdv	Lat.	Stdv	Lat.	Stdv	Lat.	Stdv
1node	0.046	0.000049	0.053	0.000059	0.049	0.000072	0.039	0.000078	0.057	0.000082
2nodes	0.046	0.000064	0.052	0.000061	0.049	0.000072	0.039	0.000078	0.057	0.000082
3nodes	0.047	0.000061	0.054	0.000065	0.049	0.000072	0.039	0.000078	0.057	0.000082
4nodes	0.047	0.000050	0.054	0.000062	0.049	0.000072	0.039	0.000078	0.057	0.000082

B. RANDOM WRITING

Table 9. Throughput (Th) and its standard deviation (Stdv)

Nodes	Cassandra		CouchDB		MongoDB		PostgreSQL		RethinkDB	
	Th.	Stdv	Th.	Stdv	Th.	Stdv	Th.	Stdv	Th.	Stdv
1node	13241.81	0.76	3263.53	0.83	16796.36	0.81	23820.78	0.64	5518.21	0.71
2nodes	18263.09	0.73	4369.81	0.82	16796.36	0.81	23820.78	0.64	5528.47	0.71
3nodes	27498.91	0.62	6065.08	0.85	16796.36	0.81	23820.78	0.64	5518.21	0.71
4nodes	37211.55	0.69	10255.19	0.73	16796.36	0.81	23820.78	0.64	5518.21	0.71

Table 10. Latency (Lat.) and its standard deviation (Stdv)

Nodes	Cassandra		CouchDB		MongoDB		PostgreSQL		RethinkDB	
	Lat.	Stdv	Lat.	Stdv	Lat.	Stdv	Lat.	Stdv	Lat.	Stdv
1node	0.0463	0.000051	0.054	0.000061	0.0483	0.000065	0.0455	0.000075	0.0573	0.000063
2nodes	0.0467	0.000054	0.053	0.000055	0.0483	0.000065	0.0455	0.000075	0.0573	0.000063
3nodes	0.0467	0.000063	0.055	0.000071	0.0483	0.000065	0.0455	0.000075	0.0573	0.000063
4nodes	0.0467	0.000058	0.055	0.000062	0.0483	0.000065	0.0455	0.000075	0.0573	0.000063

II. Reading Throughput (Operations Per Second) And Latency In Milliseconds

1. DATASET0

A. DISTANCE QUERY

A. SEQUENTIAL READING

Table 11. Throughput (Th) and its standard deviation (Stdv)

Nodes	Cassandra		CouchDB		MongoDB		PostgreSQL		RethinkDB	
	Th.	Stdv	Th.	Stdv	Th.	Stdv	Th.	Stdv	Th.	Stdv
1node	3791460.81	0.76	1581316.58	0.67	3087332.38	0.65	1800943.88	0.86	2315499.28	0.68
2nodes	4265393.42	0.73	1852399.42	0.82	3601887.77	0.76	3241699	0.85	2818868.69	0.74
3nodes	4874735.33	0.62	2091418.70	0.81	4052123.75	0.89	4052123.75	0.73	3087332.38	0.66
4nodes	5358180.16	0.69	2493614.61	0.75	4987229.23	0.68	5186718.4	0.91	4052123.75	0.83

Table 12. Latency (Lat.) and its standard deviation (Stdv)

Nodes	Cassandra		CouchDB		MongoDB		PostgreSQL		RethinkDB	
	Lat.	Stdv	Lat.	Stdv	Lat.	Stdv	Lat.	Stdv	Lat.	Stdv
1node	0.032	0.000042	0.042	0.000051	0.033	0.000062	0.031	0.000074	0.041	0.000062
2nodes	0.033	0.000063	0.041	0.000062	0.031	0.000054	0.034	0.000065	0.039	0.000074
3nodes	0.035	0.000061	0.043	0.000058	0.031	0.000045	0.034	0.000056	0.041	0.000062
4nodes	0.036	0.000068	0.045	0.000079	0.031	0.000063	0.034	0.000058	0.042	0.000067

B. RANDOM READING

Table 13. Throughput (Th) and its standard deviation (Stdv)

Nodes	Cassandra		CouchDB		MongoDB		PostgreSQL		RethinkDB	
	Th.	Stdv	Th.	Stdv	Th.	Stdv	Th.	Stdv	Th.	Stdv
1node	3033168.65	0.67	1233426.93	0.79	2346372.60	0.76	1332698.47	0.76	1713469.47	0.73
2nodes	3412314.73	0.89	1444871.55	0.75	2737434.71	0.83	2398857.26	0.83	2085962.83	0.82
3nodes	3899788.27	0.74	1631306.59	0.72	3079614.05	0.71	2998571.57	0.82	2284625.96	0.59
4nodes	4286544.13	0.79	1945019.4	0.86	3790294.21	0.59	3838171.61	0.82	2998571.57	0.81

Table 14. Latency (Lat.) and its standard deviation (Stdv)

Nodes	Cassandra		CouchDB		MongoDB		PostgreSQL		RethinkDB	
	Lat.	Stdv	Lat.	Stdv	Lat.	Stdv	Lat.	Stdv	Lat.	Stdv
1node	0.043	0.000053	0.058	0.000068	0.043	0.000072	0.042	0.000072	0.0515	0.000078
2nodes	0.048	0.000056	0.060	0.000068	0.045	0.000054	0.047	0.000073	0.052	0.000076
3nodes	0.048	0.000052	0.061	0.000061	0.045	0.000049	0.0475	0.000046	0.054	0.000064
4nodes	0.0482	0.000059	0.064	0.000062	0.045	0.000060	0.0475	0.000069	0.056	0.000072

ACKNOWLEDGEMENTS

This work is part of the research project "Scalable resource-efficient systems for big data analytics" funded by the Knowledge Foundation (grant: 20140032) in Sweden. We also thank Telenor Sverige for providing the data.

REFERENCES

[1] C. Niyizamwiyitira and L. Lundberg, "Performance Evaluation of Trajectory Queries on Multiprocessor and Cluster," in Computer Science & Information Technology, Vienna,Austria, 2016, vol. 6, pp. 145–163.

[2] S. Spaccapietra, C. Parent, M. L. Damiani, J. A. de Macedo, F. Porto, and C. Vangenot, "A conceptual view on trajectories," Data Knowl. Eng., vol. 65, no. 1, pp. 126–146, 2008.

[3] Y. Zheng and X. Zhou, Computing with spatial trajectories. Springer Science & Business Media, 2011.

[4] N. Pelekis and Y. Theodoridis, Mobility data management and exploration. Springer, 2014.

[5] R. Benetis, C. S. Jensen, G. Karčiauskas, and S. Šaltenis, "Nearest neighbor and reverse nearest neighbor queries for moving objects," in Database Engineering and Applications Symposium, 2002. Proceedings. International, 2002, pp. 44–53.

[6] E. Frentzos, K. Gratsias, N. Pelekis, and Y. Theodoridis, "Nearest neighbor search on moving object trajectories," in Advances in Spatial and Temporal Databases, Springer, 2005, pp. 328–345.

[7] M. Erwig, R. H. Gu, M. Schneider, M. Vazirgiannis, and others, "Spatio-temporal data types: An approach to modeling and querying moving objects in databases," GeoInformatica, vol. 3, no. 3, pp. 269–296, 1999.
[8] Y. Theodoridis, "Ten benchmark database queries for location-based services," Comput. J., vol. 46, no. 6, pp. 713–725, 2003.
[9] D. Pfoser, "Indexing the trajectories of moving objects," IEEE Data Eng Bull, vol. 25, no. 2, pp. 3–9, 2002.
[10] V. T. De Almeida, R. H. Güting, and T. Behr, "Querying moving objects in secondo," in null, 2006, p. 47.
[11] C. Düntgen, T. Behr, and R. H. Güting, "BerlinMOD: a benchmark for moving object databases," VLDB J., vol. 18, no. 6, pp. 1335–1368, 2009.
[12] L. I. Gómez, B. Kuijpers, and A. A. Vaisman, "Aggregation languages for moving object and places of interest," in Proceedings of the 2008 ACM symposium on Applied computing, 2008, pp. 857–862.
[13] Y.-J. Gao, C. Li, G.-C. Chen, L. Chen, X.-T. Jiang, and C. Chen, "Efficient k-nearest-neighbor search algorithms for historical moving object trajectories," J. Comput. Sci. Technol., vol. 22, no. 2, pp. 232–244, 2007.
[14] D. Pfoser, C. S. Jensen, Y. Theodoridis, and others, "Novel approaches to the indexing of moving object trajectories," in Proceedings of VLDB, 2000, pp. 395–406.
[15] K. Y. Besedin and P. S. Kostenetskiy, "Simulating of query processing on multiprocessor database systems with modern coprocessors," in Information and Communication Technology, Electronics and Microelectronics (MIPRO), 2014 37th International Convention on, 2014, pp. 1614–1616.
[16] R. Moussalli, I. Absalyamov, M. R. Vieira, W. Najjar, and V. J. Tsotras, "High performance FPGA and GPU complex pattern matching over spatio-temporal streams," GeoInformatica, vol. 19, no. 2, pp. 405–434, Aug. 2014.
[17] P. Huang and B. Yuan, "Mining Massive-Scale Spatiotemporal Trajectories in Parallel: A Survey," in Trends and Applications in Knowledge Discovery and Data Mining, Springer, 2015, pp. 41–52.
[18] R. Moussalli, M. Srivatsa, and S. Asaad, "Fast and Flexible Conversion of Geohash Codes to and from Latitude/Longitude Coordinates," in Field-Programmable Custom Computing Machines (FCCM), 2015 IEEE 23rd Annual International Symposium on, 2015, pp. 179–186.
[19] " no sql benchmark - Google Search." [Online]. Available: https://www.google.rw/search?q=no+sql+benchmark&oq=no+sql+benchmark&aqs=chrome..69i57j0l5.4785j0j7&sourceid=chrome&ie=UTF-8. [Accessed: 22-Nov-2017].
[20] J. Han, E. Haihong, G. Le, and J. Du, "Survey on NoSQL database," in Pervasive computing and applications (ICPCA), 2011 6th international conference on, 2011, pp. 363–366.
[21] "What is Apache Cassandra?," Planet Cassandra, 18-Jun-2015. [Online]. Available: http://www.planetcassandra.org/what-is-apache-cassandra/. [Accessed: 23-Feb-2016].
[22] "CQL." [Online]. Available: http://docs.datastax.com/en//cassandra/2.0/cassandra/cql.html. [Accessed: 23-Feb-2016].
[23] "Stratio/cassandra-lucene-index," GitHub. [Online]. Available: https://github.com/Stratio/cassandra-lucene-index. [Accessed: 23-Mar-2016].
[24] "Apache CouchDB." [Online]. Available: http://couchdb.apache.org/. [Accessed: 16-Aug-2016].
[25] "couchbase/geocouch," GitHub. [Online]. Available: https://github.com/couchbase/geocouch. [Accessed: 16-Aug-2016].
[26] tutorialspoint.com, "MongoDB Overview," www.tutorialspoint.com. [Online]. Available: http://www.tutorialspoint.com/mongodb/mongodb_overview.htm. [Accessed: 23-Feb-2016].
[27] "MongoDB for GIANT Ideas," MongoDB. [Online]. Available: https://www.mongodb.com/. [Accessed: 23-Feb-2016].
[28] "Sharding Introduction — MongoDB Manual 3.2," https://github.com/mongodb/docs/blob/master/source/core/sharding-introduction.txt.[Online]. Available: https://docs.mongodb.org/manual/core/sharding-introduction/. [Accessed: 24-Feb-2016].
[29] J. Worsley and J. D. Drake, Practical PostgreSQL. O'Reilly Media, Inc., 2002.
[30] "Multi-node setup on Ubuntu or Debian — Citus 5.1.0 documentation." [Online]. Available: http://docs.citusdata.com/en/v5.1/installation/production_deb.html. [Accessed: 16-Aug-2016].

[31] "PostGIS — Spatial and Geographic Objects for PostgreSQL." [Online]. Available: http://postgis.net/. [Accessed: 16-Aug-2016].
[32] "RethinkDB: the open-source database for the realtime web." [Online]. Available: https://www.rethinkdb.com/. [Accessed: 16-Aug-2016].
[33] "Stratio/cassandra-lucene-index," GitHub. [Online]. Available: https://github.com/Stratio/cassandra-lucene-index. [Accessed: 30-Mar-2016].
[34] "Consistency & Cassandra," Planet Cassandra, 10-Apr-2013. [Online]. Available: http://www.planetcassandra.org/blog/consistency-cassandra/. [Accessed: 29-Sep-2016].
[35] "How To Create a Sharded Cluster in MongoDB Using an Ubuntu 12.04 VPS," DigitalOcean. [Online]. Available: https://www.digitalocean.com/community/tutorials/how-to-create-a-sharded-cluster-in-mongodb-using-an-ubuntu-12-04-vps. [Accessed: 29-Sep-2016].
[36] "Multi-node setup on Ubuntu or Debian — Citus 5.1.0 documentation." [Online]. Available: http://docs.citusdata.com/en/v5.1/installation/production_deb.html. [Accessed: 16-Aug-2016].

AUTHORS

Christine Niyizamwiyitira is a PhD student in Computer science at Blekinge Institute of Technology (BTH) in Computer Science and Engineering Department. Her research interests includes Real-time systems, cloud computing, high performance computing, Database performance, and Voice based application. Her current Research focuses on Scheduling of real time systems on Virtual Machines (uniprocessor & multiprocessor) and Big data processing.

Lars Lundberg is a professor in Computer Systems Engineering at the Department of Computer Science and Engineering at Blekinge Institute of Technology in Sweden. He has a M.Sc. in Computer Science from Linköping University (1986) and a Ph.D. in Computer Engineering from Lund University (1993). His research interests include parallel and cluster computing, real-time systems and software engineering. Professor Lundberg's current work focuses on performance and availability aspects.