Benchmarking Relations
DataBase MySQL against NoSQL DataBase Mongdb

National College of Ireland

Msc in Data Analytics

Project B

21th April 2015

Author(s)	Sapna Gupta
Degree	Msc in Data Analytics
Department	School of computing
Instructor	Michael Bradford

Abstract

Till now relational databases are widely used in all types of big and small industries, due to easy query capabilities and transaction management. As the data is growing rapidly by 4 Vs, volume, variety, velocity, veracity. These databases are not very efficient in storing and retrieving this Big Data. Recently emerged new types of databases called NoSQL databases. These are highly scalable and are more suitable for the usage in the web environment. This paper is more focused on Benchmarking the performance, storage and transaction management of NoSQL Database against Relational Database.

Introduction

A **relational database** is a digital database whose organization is based on the relational model of data, as proposed by E.F. Codd in 1970. This model organizes data into one or more tables (or "relations") of rows and columns, with a unique key for each row. Generally, each entity type defined in a database has its own table, the rows representing instances of that type of entity and the columns representing values attributed to that instance. Because each row in a table has its own unique key, rows in a table can be linked to rows in other tables by storing the unique key of the row to which it should be linked (where such unique key is known as a "foreign key"). Examples of most popular RDBMS are Oracle ,SQL server , Teradata and Mysql

A NoSQL (often interpreted as Not only SQL[1][2]) database provides a mechanism for storage and retrieval of data that is modeled in means other than the tabular relations used in relational databases. Motivations for this approach include simplicity of design, horizontal scaling, and finer control over availability. The data structures used by NoSQL databases (e.g. key-value, graph, or document) differ from those used in relational databases, making some operations faster in NoSQL and others faster in relational databases. The particular suitability of a given NoSQL database depends on the problem it must solve.

NoSQL databases are progressively used in big data and real-time web applications.[3] NoSQL systems are also called "Not only SQL" to emphasize that they may also support SQL-like query languages. Many NoSQL stores compromise consistency (in the sense of the CAP theorem) in favor of availability and partition

tolerance. Barriers to the greater adoption of NoSQL stores include the use of low-level query languages, the lack of standardized interfaces, and huge investments in existing SQL.[4] Most NoSQL stores lack true ACID transactions, although a few recent systems, such as FairCom c-treeACE, GoogleSpanner (though technically a NewSQL database), FoundationDB and OrientDB have made them central to their designs.

Types of NoSQL Databases:

1. Key-values Stores

The main idea here is using a hash table where there is a unique key and a pointer to a particular item of data. The Key/value model is the simplest and easiest to implement. But it is inefficient when you are only interested in querying or updating part of a value, among other disadvantages.

Examples: Tokyo Cabinet/Tyrant, Redis, Voldemort, Oracle BDB, Amazon SimpleDB, Riak

2. Column Family Stores

These were created to store and process very large amounts of data distributed over many machines. There are still keys but they point to multiple columns. The columns are arranged by column family.

Examples: Cassandra, HBase

3. Document Databases

These were inspired by Lotus Notes and are similar to key-value stores. The model is basically versioned documents that are collections of other key-value collections. The semi-structured documents are stored in formats like JSON. Document databases are essentially the next level of Key/value, allowing nested values associated with each key. Document databases support querying more efficiently.

Examples: CouchDB, MongoDb

4. Graph Databases

Instead of tables of rows and columns and the rigid structure of SQL, a flexible graph model is used which, again, can scale across multiple machines. NoSQL databases do not provide a high-level declarative query language like SQL to avoid overtime in processing. Rather, querying these databases is data-model specific. Many of the NoSQL platforms allow for RESTful interfaces to the data, while other offer query APIs.

Examples: Neo4J, InfoGrid, Infinite Graph

For my Benchmarking experiment, I have chosen MySQL from Relational Database category. We have chosen MongoDB from NoSQL Database Category. I have chosen these databases because these are very popular open source and widely used. These are easy to install and readily available.

Key Characteristics of Chosen Data Storage Management Systems

Relational databases are based on a set of principles to optimize performance. Principles used by Relational or NoSQL databases are derived from CAP theorem [11]. According to this theorem, following guarantees can be defined:

 Consistency – all nodes have same data at the same time;

 Availability – all requests have response;

 Partition tolerance – if part of system fails, all system won't collapse.

ACID is a principle based on CAP theorem and used as set of rules for relational database transactions. ACID's guarantees are [17]:

 Atomic – a transaction is completed when all operations are completed, otherwise rollback1 is performed;

 Consistent – a transaction cannot collapse database, otherwise if operation is illegal, rollback is performed; Helvetica

 Isolated – all transactions are independent and cannot affect each other;

 Durable – when commit2 is performed, transactions cannot be undone.

It is noticeable that in order to have robust and correct database those guarantees are important. But when the amount of data is large, ACID may be hard to attain. That why, NoSQL focuses on BASE principle [17, 20]:

 Basically Avaliable – all data is distributed, even when there is a failure the system continues to work;

 Soft state – there is no consistency guarantee;

 Eventually consistent – system guarantees that even when data is not consistent, eventually it will be.

The relational database has been the foundation of enterprise applications for decades, and since MySQL's release in 1995 it has been a popular and inexpensive option. Yet with the explosion in the volume and variety of data in recent years, non-relational database technologies like MongoDB have emerged to address the requirements of new applications. MongoDB is used for new applications as well as to augment or replace existing relational infrastructure.

What is MySQL?

MySQL is a popular open-source relational database management system (RDBMS) that is developed, distributed and supported by Oracle Corporation. Like other relational systems, MySQL stores data in tables and uses structured query language (SQL) for database access. In MySQL, you pre-define your database schema based on your requirements and set up rules to govern the relationships between fields in your tables. In MySQL, related information may be stored in separate tables, but associated through the use of joins. In this way, data duplication is minimized.

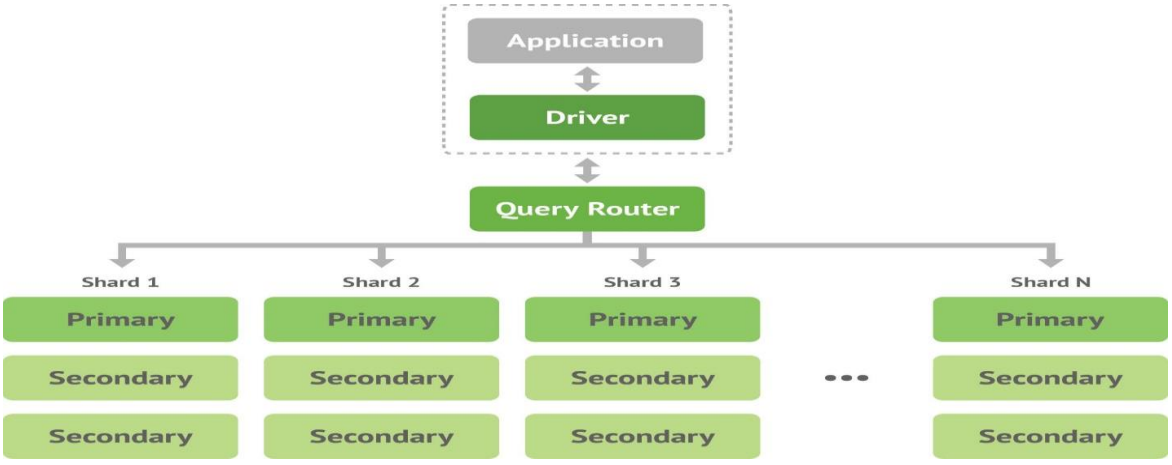
What is MongoDB?

MongoDB is an open-source database developed by MongoDB, Inc. MongoDB stores data in JSON-like documents that can vary in structure. Related information is stored together for fast query access through the MongoDB query language. MongoDB uses dynamic schemas, meaning that you can create records without first defining the structure, such as the fields or the types of their values. You can change the structure of records (which we call documents) simply by adding new fields or deleting existing ones. This data model give you the ability to represent hierarchical relationships, to store arrays, and other more complex structures easily. Documents in a collection need not have an identical set of fields and demoralization of data is common. MongoDB was also designed with high availability and scalability in mind, and includes out-of-the-box replication and auto-sharding.

MYSQL	MONGODB
INSERT INTO users(user_id, age, status) VALUES ("bcd001", 45, "A")	db.users.insert({ user_id: "bcd001", age: 45, status: "A" })
SELECT * FROM users	db.users.find()
UPDATE users SET status = "C" WHERE age > 25	db.users.update({ age: { $gt: 25 } }, { $set: { status: "C" } }, { multi: true })

Database Architectures

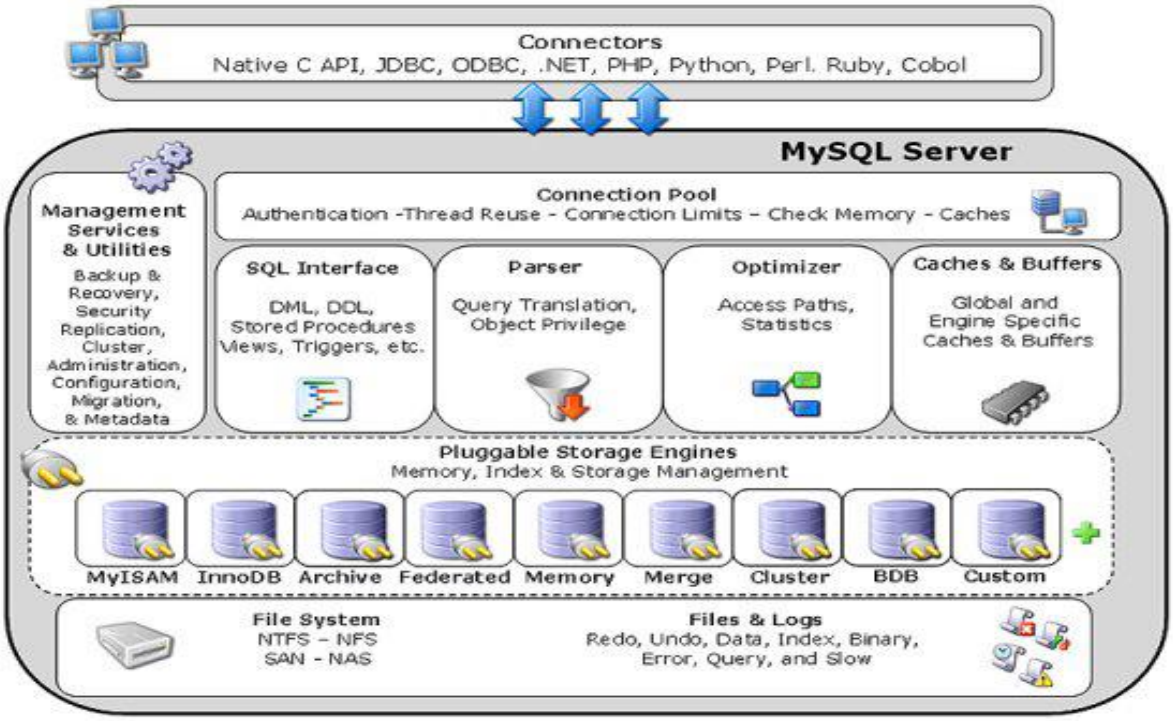
MongoDB uses auto sharding for horizontal scalability and replica sets for high availability. Clients connect to the MongoDB process (mongos in a sharded environment), optionally authenticate themselves if security is turned on, and perform inserts, queries, and updates of the documents in the database. The mongos routes the query to the appropriate shards (mongod) to fulfill the particular command.



The MySQL server architecture model and API. Thus, although there are different capabilities across different storage engines, the application is shielded from these differences.

The pluggable storage engine architecture provides a standard set of management and support services that are common among all underlying storage engines. The storage engines themselves are the components of the database server that actually perform actions on the underlying data that is maintained at the physical server level.

isolates the application programmer and DBA from all of the low-level implementation details at the storage level, providing a consistent and easy application



This efficient and modular architecture provides huge benefits for those wishing to specifically target a particular application need—such as data warehousing, transaction processing, or high availability situations—while enjoying the advantage of utilizing a set of interfaces and services that are independent of any one storage engine.

	MYSQL	MONGODB
Rich Data Model	No	Yes
Dynamic Schema	No	Yes
Typed Data	Yes	Yes
Data Locality	No	Yes
Field Updates	Yes	Yes
Easy for Programmers	No	Yes
Complex Transactions	Yes	No
Auditing	Yes	Yes
Auto-Sharding	No	Yes

Test Plan

For this benchmarking experiment, I am using YCSB an open source and a very popular Benchmarking tool readily available.

The YCSB – Yahoo! Cloud Serving Benchmark is one of the most used benchmarks to test NoSQL databases [10]. YCSB has a client that consists of two parts: workload generator and the set of scenarios. Those scenarios, known as workloads, are combinations of read, write and update operations performed on randomly chosen records. The predefined workloads are:

 Workload A: Update heavy workload. This workload has a mix of 50/50 reads and updates.

 Workload B: Read mostly workload. This workload has a 95/5 reads/update mix.

 Workload C: Read only. This workload is 100% read.

 Workload D: Read latest workload. In this workload, new records are inserted, and the most recently inserted records are the most popular.

 Workload E: Short ranges. In this workload, short ranges of records are queried, instead of individual records.

 Workload F: Read-modify-write. In this workload, the client will read a record, modify it, and write back the changes.

 Workload G: Update mostly workload. This workload has a 5/95 reads/updates mix.

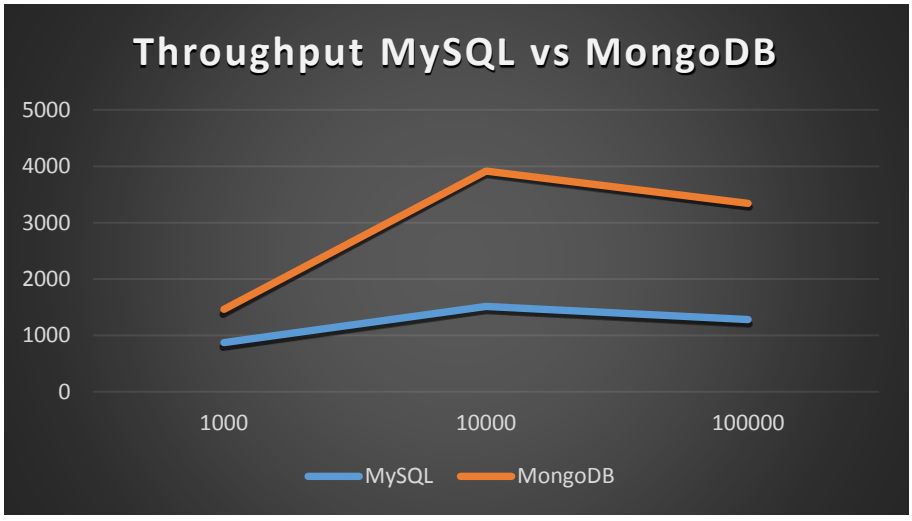
 Workload H: Update only. This workload is 100% update.

Evaluation and Result

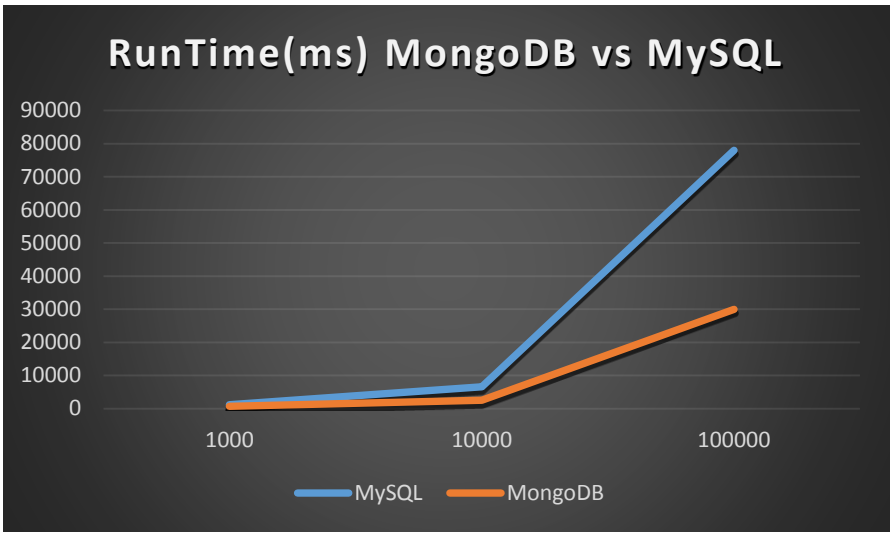
In this section we will describe the experiments while using different workloads and data volumes. Tests were running using Ubuntu Server 12.04 32bit Virtual Machine on VMware Player. As experimental setup it is important to notice that VM has available 2GB RAM and Host was single-node Core 2 Quad 2.40 GHz with 4GB RAM and Windows 7 Operating System. The tested versions of NoSQL databases are MongoDB version 2.4 and Relational Database MySQL 5.7. We are only using workload A, B, C, D and F.

Workload A :

```
# Yahoo! Cloud System Benchmark
# Workload A: Update heavy workload
#   Application example: Session store recording recent actions
#
#   Read/update ratio: 50/50
#   Default data size: 1 KB records (10 fields, 100 bytes each, plus key)
#   Request distribution: zipfian

recordcount=1000
operationcount=1000
workload=com.yahoo.ycsb.workloads.CoreWorkload

readallfields=true

readproportion=0.5
updateproportion=0.5
scanproportion=0
insertproportion=0

requestdistribution=zipfian
```



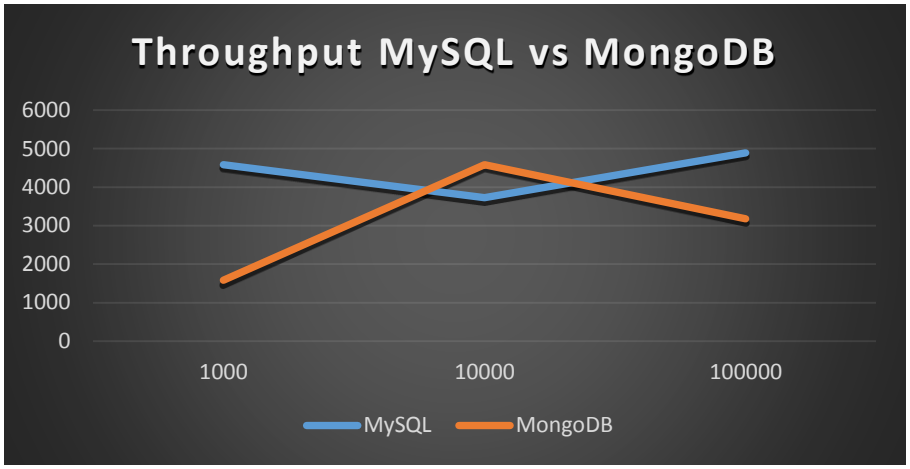
	MySQL	MongoDB
1000	876.4242	1464.129
10000	1512.173	3916.96
100000	1282.084	3341.688



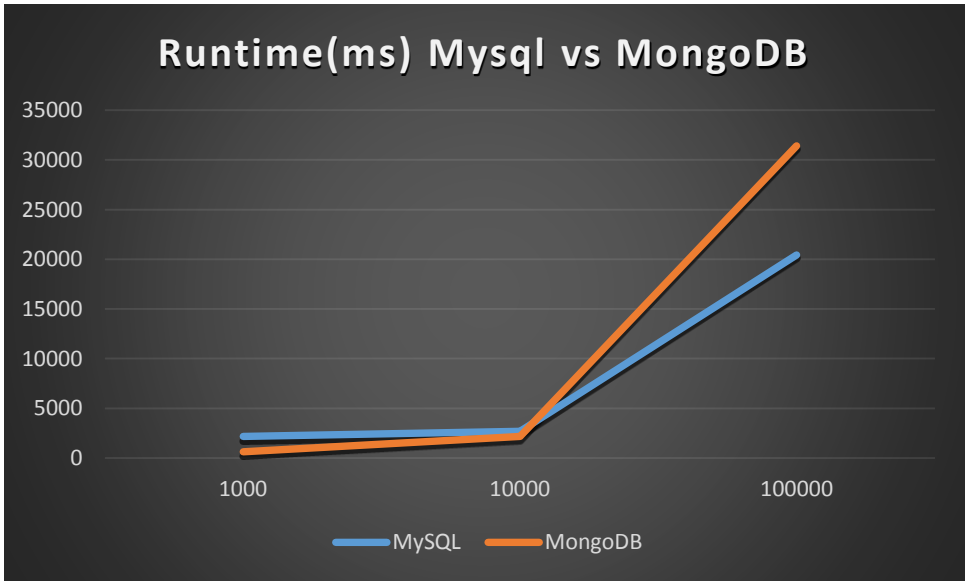
	MySQL	MongoDB
1000	1141	683
10000	6613	2553
100000	77998	29925

Workload B:

```
# Yahoo! Cloud System Benchmark
# Workload B: Read mostly workload
#   Application example: photo tagging; add a tag is an update, but most operations are to read tags
#
#   Read/update ratio: 95/5
#   Default data size: 1 KB records (10 fields, 100 bytes each, plus key)
#   Request distribution: zipfian

recordcount=1000
operationcount=1000
workload=com.yahoo.ycsb.workloads.CoreWorkload

readallfields=true

readproportion=0.95
updateproportion=0.05
scanproportion=0
insertproportion=0

requestdistribution=zipfian
```



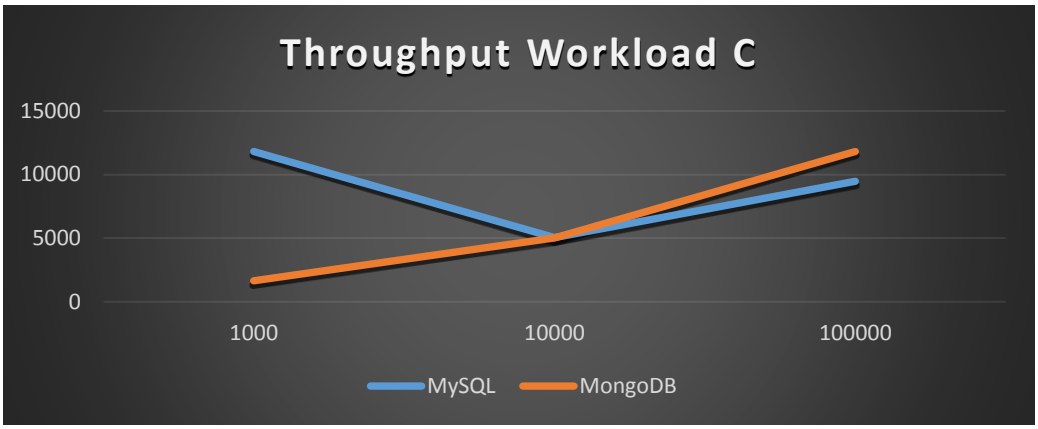
	MySQL	MongoDB
1000	4587.156	1579.779
10000	3727.171	4587.156
100000	4894.044	3181.572



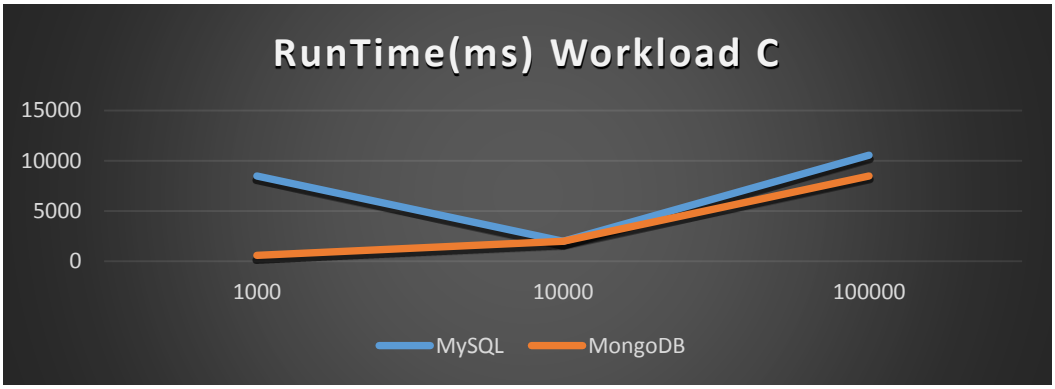
	MySQL	MongoDB
1000	2180	633
10000	2683	2180
100000	20433	31431

Workload C:

```
# Yahoo! Cloud System Benchmark
# Workload C: Read only
#   Application example: user profile cache, where profiles are constructed elsewhere (e.g., Hadoop)
#
#   Read/update ratio: 100/0
#   Default data size: 1 KB records (10 fields, 100 bytes each, plus key)
#   Request distribution: zipfian

recordcount=1000
operationcount=1000
workload=com.yahoo.ycsb.workloads.CoreWorkload

readallfields=true

readproportion=1
updateproportion=0
scanproportion=0
insertproportion=0

requestdistribution=zipfian
```



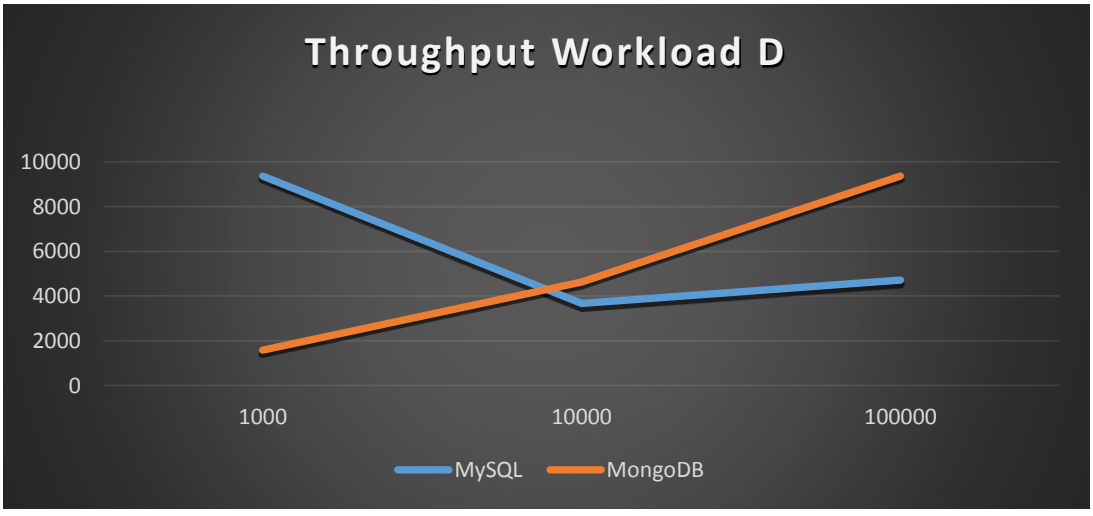
	MySQL	MongoDB
1000	11807.77	1658.375
10000	5055.612	5017.561
100000	9479.572	11807.77



	MySQL	MongoDB
1000	8469	603
10000	1978	1993
100000	10549	8469

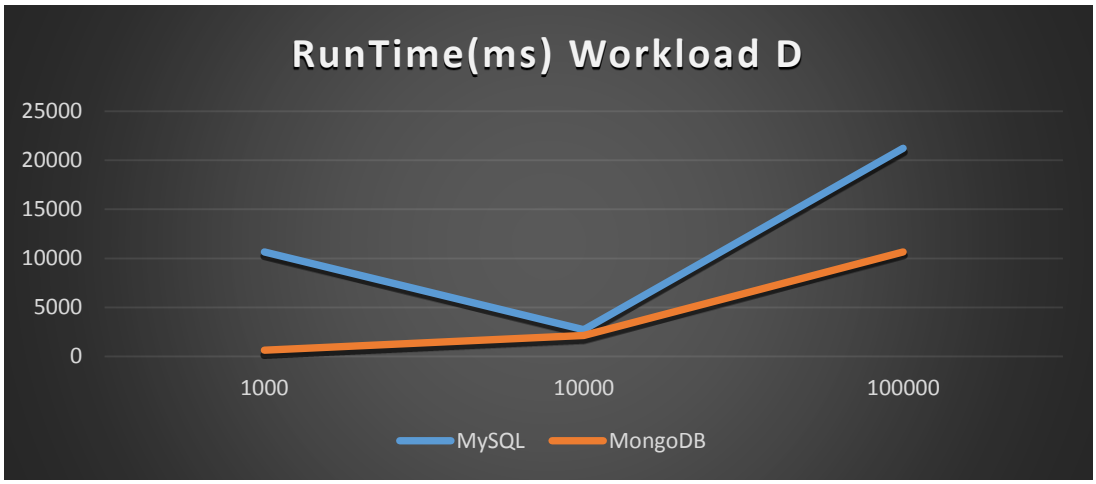
Workload D:

```
# Yahoo! Cloud System Benchmark
# Workload D: Read latest workload
#   Application example: user status updates; people want to read the latest
#
#   Read/update/insert ratio: 95/0/5
#   Default data size: 1 KB records (10 fields, 100 bytes each, plus key)
#   Request distribution: latest

# The insert order for this is hashed, not ordered. The "latest" items may be
# scattered around the keyspace if they are keyed by userid.timestamp. A workload
# which orders items purely by time, and demands the latest, is very different than
# workload here (which we believe is more typical of how people build systems.)

recordcount=1000
operationcount=1000
workload=com.yahoo.ycsb.workloads.CoreWorkload

readallfields=true

readproportion=0.95
updateproportion=0
scanproportion=0
insertproportion=0.05

requestdistribution=latest
```



	MySQL	MongoDB

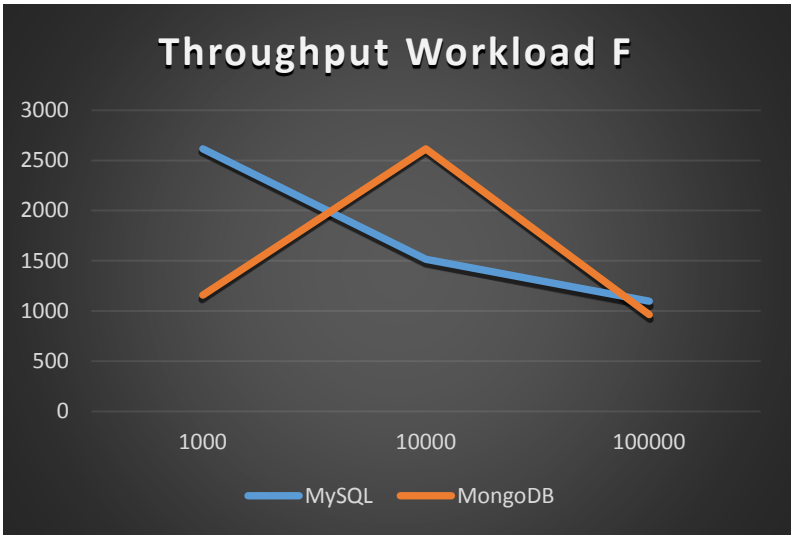
1000	9371.193	1574.803
10000	3676.471	4627.487
100000	4710.094	9371.193



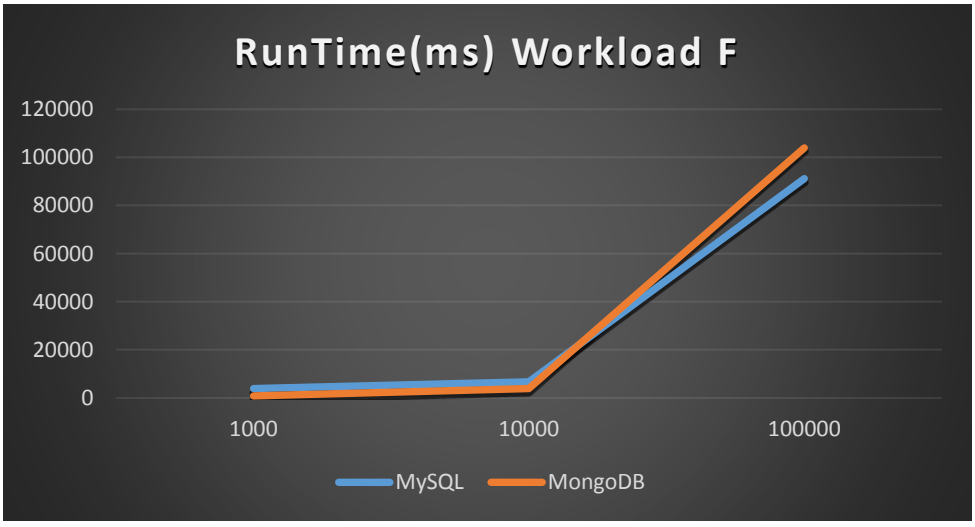
	MySQL	MongoDB
1000	10671	635
10000	2720	2161
100000	21231	10671

Workload F:

```
# Yahoo! Cloud System Benchmark
# Workload F: Read-modify-write workload
#   Application example: user database, where user records are read and modified by the user or to r
#
#   Read/read-modify-write ratio: 50/50
#   Default data size: 1 KB records (10 fields, 100 bytes each, plus key)
#   Request distribution: zipfian

recordcount=1000
operationcount=1000
workload=com.yahoo.ycsb.workloads.CoreWorkload

readallfields=true

readproportion=0.5
updateproportion=0
scanproportion=0
insertproportion=0
readmodifywriteproportion=0.5

requestdistribution=zipfian
```



	MySQL	MongoDB
1000	2614.379	1157.407
10000	1514.693	2614.379
100000	1096.828	962.2602



	MySQL	MongoDB
1000	3825	864
10000	6602	3825
100000	91172	103922

As per the above results except for workload A, it shows MongoDB performed better in all the results than MySQL database.

Storage and retrieval of Structured, Semi-Structured and Unstructured Data

Mysql is traditional database used for decades, is not easy to replace with NoSQL databases. Storage and retrieval of structured data is very easy and efficient, but there is huge challenge as it does not support Unstructured and semi structured data. Data has to be fully structured in order to store or retrieve data from Mysql server. Where as in MongoDB , storing semi-Structured and Unstructured Data is not a big problem , but retrieval and data transformation is not as efficient as in Mysql database. Many industries are using mongodb which stores data in JSON-like documents that can vary in structure.

Conclusion

The development of the Web need databases able to store and process big data effectively, demand for high-performance when reading and writing, so the traditional relational database is facing many new challenges. NoSQL databases have gained popularity in the recent years and have been successful in many production systems. In this paper we analysed and evaluated two of the most popular NoSQL databases: MongoDB and MySQL. In the experiments we test the execution time according to database size and the type of workload. We test six different types of workloads: mix of 50/50 reads and updates; mix of 95/5 reads/updates; read only; read-modify-write cycle; mix of 5/95 reads/updates; and update only. With the increase of data size, MongoDB started to reduce performance, sometimes showing poor results. Differently, MySQL just got faster while working with an increase of data. Also, after running different workloads to analyze read/update performance, it is possible to conclude that when it comes to update operations, MongoDB is faster than MySQL, providing lower execution time independently of database size used in our evaluation. As overall analysis turns out that MongoDB fell short with increase of records used, while MySQL still has a lot to offer. In conclusion MySQL show the best results for almost all scenarios.