Performance Comparison of NoSQL Databases in Pseudo Distributed Mode: Cassandra, MongoDB & Redis

Tiroshan Madushanka
National Institute of Informatics
4 PUBLICATIONS **5** CITATIONS

Laksheen Mendis
University of Moratuwa
2 PUBLICATIONS **5** CITATIONS

Performance Comparison of NoSQL Databases in Pseudo Distributed Mode: Cassandra, MongoDB & Redis

Kumarasinghe C.U., Liyanage K.L.D.U. ,
Madushanka W.A.T. and Mendis R.A.C.L
Department of Computer Science and Engineering, University of Moratuwa, Sri Lanka
Email:{chamathk.10, tiroshan.10, dananji.10 & cresclux.10}@cse.mrt.ac.lk

***Abstract*—Understanding the performance of NoSQL databases is essential for the selection of correct database for a specific application. This paper describe in detail and shows results of experiments done on how each database mentioned in the title performs with reference to different workloads and different concurrency simulations in pseudo distributed mode.**

I. Introduction

A. Problem

NoSQL databases promise faster and efficient performance than the legacy RDBMS. This is most significant in use cases involving Big Data. Selecting the correct database for the application is major issue for application developers as the performance of NoSQL databases varies with its category and implementation.

B. Background

Many system architects and IT managers require comparisons between NoSQL databases in their development or production environments using data and frequent user interactions, which represent the workload they will be using in the new application or system. Another way to understand performance trade-offs is to define individual benchmarks under different workloads. Some of the NoSQL databases have built-in tools for performance analysing. This type of performance analysis would not be useful in a decision making scenario as described above. It is important to understand the strengths and weaknesses of each database compared to others. Therefore the YCSB (Yahoo Cloud Serving Benchmark) is used as the tool in this study.

II. YCSB (Yahoo Cloud Serving Benchmark)

Traditionally, there have been a number of benchmarks for relational databases but these cannot be used to benchmark the NoSQL databases, which are non-relational and comes in different categories (Ex: document store, column store, key value, graph etc).

Therefore, YCSB open source benchmarking framework designed by Yahoo Labs tries to alleviate that problem by providing a generic framework to induce load on a data store. On the other hand, this framework comes with the interfaces to populate and stress many popular data serving systems, such as; Cassandra, GemFire, HBase, HyperTable, DynamoDB, MongoDB, Redis, Voldemort, OrientDB, etc.

There are many other alternative benchmarking tools for each specific database, for example MongoDB has a inbuilt tool named mongoperf which can give results of the speed of read/write operations and another tool named mongo-perf which can be used to benchmark performance of throughput, by the number of threads and how the disk seeks happen while running the tests.

A. Configuration

A summary of the steps followed during the configuration of YCSB, to benchmark the NoSQL databases.

- The GitHub project is available for YCSB and to build it from the source, it is necessary clone the project
- YCSB uses Apache Maven as the build tool and running a clean on the package would build YCSB
- There is a YCSB Shell where it is possible to run commands or another option would be to write a shell script to automate the tests
- It is also important to specify the number of records to be inserted for the test, number of operations on the tests and the number of threads the operation to be ran on. This can be done for both loading and running the tests

B. Workloads

The workloads consist of six different types of workloads to simulate different application contexts

- Workload A:
  Workload for heavy updating. In this workload 50% of the operations are reads and 50% are writes. This workload simulates user session recording and updating in web applications.

- Workload B:
  Workload that has more read operations than writes. In this workload 95% of the operations are reads and the rest 5% are writes. This workload simulates generating reports in an ERP solution which has large volumes of data. In this scenario, most operations would be to

read the data from the tables and to write a summary document to another table.

- Workload C:
  Workload has only read operations. This workload simulates online viewers of books and magazines.

- Workload D:
  Workload inserts new records and the most recently inserted records are the most popular. This workload simulates user status updates, where people want to read the latest.

- Workload E:
  Workload queries short ranges of records, instead of individual records. This workload simulates threaded conversations, where each scan is for the posts in a given thread (assuming that threads are clustered by thread id).

- Workload F:
  Workload where the client will read a record, modify it, and write back the changes. This workload simulates an user database, where user records are read and modified by the user or a situation which records user activity.

III. Database Internals

A. Cassandra

1) Introduction: Cassandra is a open source database management system, which is designed to handle large amounts of data licensed under Apache License. This can be used across many commodity servers providing high availability and no single point of failure. This is a project born at Facebook and built on Amazons Dynamo and Googles Big Table. Cassandra aims to run on top of an infrastructure of hundreds of nodes. At this scale large and small components tend to fail, but Cassandra manages to maintain a persistent state which ensures the reliability and scalability of the software systems that depends on this service. Cassandra addresses the problem of failures by employing a peer-to-peer distributed system across homogeneous nodes where data is distributed among all nodes in the cluster.

2) Architecture: In Cassandra, data is indexed and written to an in-memory structure called a memtable. A sequentially written commit log in each node captures write activity to ensure data durability. When the memory structure is full data is written to disk in an SSTable data file. Cassandra is a Column family. These databases are designed for storing data in a schema-less way. In a key-value store, all of the data consist of an indexed key and a value. It is built based on the Key-Value model. Therefore when exploring Cassandra, we come across the terms Column family and Keyspace. A keyspace is analogous to a database, while a column family corresponds to a table. Therefore a keyspace contains one or more column families. In Cassandra each column family is stored in a separate file which is sorted in a row. A column family is a container for collection of rows.

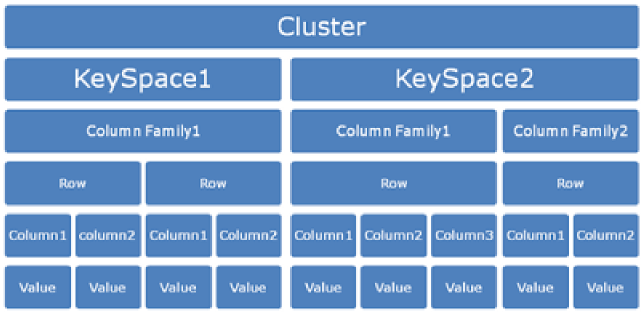

Fig. 1. Data Model of Cassandra

3) Cassandra Internals: Cassandra uses a log-structured merge tree as its storage structure. Cassandra avoids reading before writing. Reading before writing leads to corrupted caches and increases I/O requirements. Therefore to avoid this, the storage engine groups the inserts/updates to be made and then append only the updated parts of a row. Cassandra never re-reads/ re-writes and never overwrites existing data.

Cassandra creates sub-directories for the tables within each keyspace directory. This allows to symlink (symbolically link) a table into a physical driver. This provides the capability to move a table to faster media like SSD's for better performance.

With regard to ACID properties in SQL databases, NoSQL databases also maintain some properties similar to them. As a NoSQL database Cassandra is designed in a way to optimize availability and partition tolerance. Therefore, consistency with regard to Cassandra refers to how up-to-date and synchronize a row of data in on all of its replicas. Cassandra processes data at several stages, on the write path, starting with the immediate logging of a write and ending in compaction. Compaction is the operation of merging multiple SSTables into a single new one.

Indexing in Cassandra: Internally, a Cassandra index is a data partition. Cassandra uses index to pull out the records in question. This partition is the unit of replication in Cassandra. The partitions are distributed by hashing the primary key or the candidate key. Each node indexes its own data. This avoids visiting multiple nodes when retrieving data.

Similar to the other relational databases, Cassandra also has to update its indexes. When a column is updated the index is updated as well. Cassandra removes corresponding obsolete indexes. If a read sees a stale index entry before compaction purges it, the reader thread invalidates it.

B. MongoDB

1) Introduction: MongoDB is a document oriented NoSQL database which uses BSON, language similar to JSON to retrieve and update the database. It is one of the main NoSQL databases which handles large collections of data (big data). MongoDB is written in C++ is an open source project which uses pthreads for concurrency related functionality. MongoDB

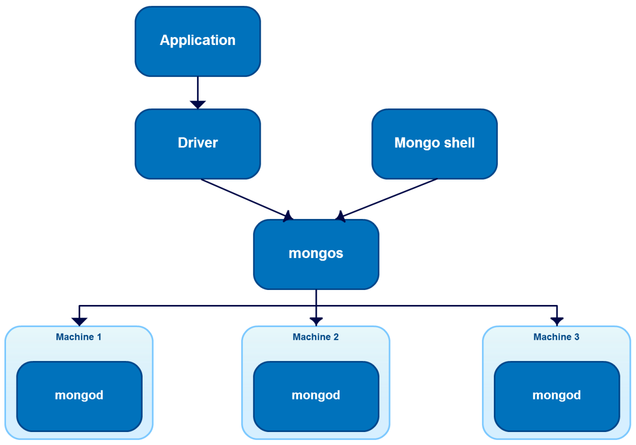

Fig. 2. Architecture of MongoDB

focus is on Data Consistency and Partition Tolerance at the expense of Availability as per the Brewers CAP theorem.

There are many prominent companies that use MongoDB as their back-end database. They are namely, SAP, eBay, SourceForge, Fouresquare and even at the CERN super collider.

2) Architecture: There are three main parts to the MongoDB database,

- MongoDB server commonly known as mongod daemon. It is the primary process which handles data requests, manages data format, and performs background management operations. There can be many mongod daemons running as primary secondary instances.
- MongoDB mongos is the routing service This process routes information and data in the cluster.
- MongoDB shell is the interactive interface. By using JavaScript to command, the developer can examine results of queries and check test cases.

3) MongoDB Internals: In MongoDB, all documents are inserted with a unique hash. This is used when retrieving records while querying. There are several types of other indexes namely,

- Unique Indexes - applied to a field of a document to reject other duplicate documents
- Compound Indexes - applied to several fields of a document to reject other duplicate documents
- Array Indexes - all arrays are indexed by this in a document for performance enhancement
- TTL Indexes - where it is necessary to remove documents after a certain time to leave
- Other Indexes - Geospatial Indexes, Sparse Indexes and Text Search Indexes

For query optimization MongoDB caches previously run query plans as wells as running several other alternative plans and selects the plan on the best response time. Results of alternative plans are also cached in memory for future reference.

MongoDB does operations on data in place and stores its data on contiguous blocks. By updating data in place it can reduce I/O operations by not requiring movement of data to other locations. It is a known fact that reading from memory is several thousand times faster than reading from Disk, MongoDB uses this to its advantage by keeping frequently accessed indexes in memory.

MongoDB reads and writes to the RAM where the Operating System does the Memory Mapping to the virtual Memory.

C. Redis

1) Introduction: Redis (Remote Dictionary Service) is an open source NoSQL database server which is licensed under BSD-license. Redis is an advanced key-value store and considered as a data structure server since keys can contain many of the foundational data types such as Strings, Lists, Sets, Sorted sets and Hashes. Redis works with an in-memory data set in order to reduce to the overhead in accessing the non-volatile memory and thus supposed to keep the entire key space in memory during operations. When terminated data will be purged from memory. But depending on the use case, data can be persisted either by taking snapshot of the data and dumping it on disk periodically or by maintaining an append-only log of all operations. Redis is implemented using ANSI C and works in POSIX systems (Linux, BSD, OS X and Solaris) without external dependencies.

2) Architecture: Redis architecture is based on BASE approach (Basically Available, Soft-state and Eventually consistent) while dropping the ACID (Atomicity, Consistency, Durability and Isolation) approach in RDBMS. Redis is based on the client/server architecture and consists following components:

- Redis server
- Redis Replica servers (optional)
- Redis Client

3) Redis DB internals: Redis is supporting advanced features like Sorted Sets, and many other complex atomic operations since it is-in memory, and single threaded. In-memory concept is implemented using the concept of Virtual memory which is a very important feature of most modern operating systems. But for efficiency reasons, Redis does not use the Operating System-supplied Virtual Memory facilities and instead implements its own system called Redis Virtual Memory.

The goal of Redis Virtual Memory (VM) is to swap infrequently-accessed data from RAM to disk, without drastically changing the performance characteristics of the database while supporting data sets that are larger than main memory.

The rationale of Redis Virtual Memory is as follows:

- A single page, as managed by the OS, is 4kB.
- The value of a single Redis key may touch many different pages, even if the key is small enough to fit in a single page.

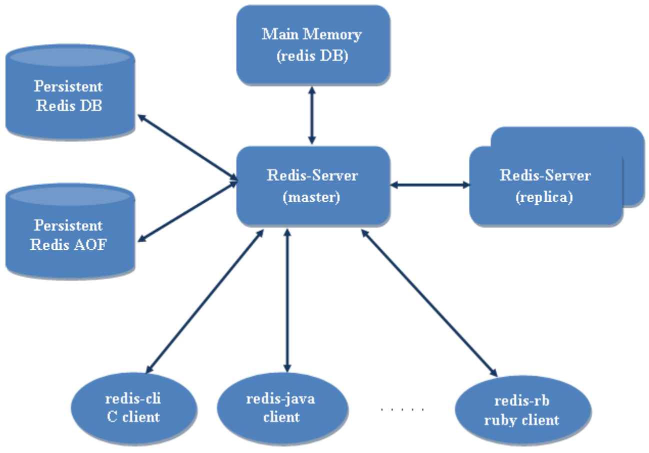

Fig. 3. Architecture of Redis Database

- Redis objects can be an order of magnitude larger in RAM than they are when stored on disk. Therefore, if using the OS' Virtual Memory facilities, the OS would need to perform an order of magnitude more I/O versus a custom Redis Virtual Memory implementation.

In Redis Virtual Memory is out of memory, it transfer values belonging to keys not recently used from memory to disk. When a Redis command will try to access a key that is swapped out, it is loaded back in memory.

In Redis VM implementation every value will be swapped out, but the keys and the top level hash table, will still use RAM, as well as the "page table bitmap", that is a bit array of bits in the Redis memory containing information about used / free pages in the swap file.

4) Virtual Memory Implementation: Redis VM stores the last time that each object was accessed and it maintains a swap file that is divided into pages of configurable size, with the page allocation table stored in memory.

When Redis VM is out of memory and it selects the object with the higher swappability factor as the object that will be swapped to disk.

Also Redis maintains a pool of I/O threads that are solely responsible for loading values from disk into RAM. When a request arrives, the command is read and the list of keys is examined. If any of the keys have been swapped to disk, the client is temporarily suspended while an I/O job is enqueued. After all keys that are needed by a given client are loaded, then the client resumes execution of the command.

IV. System Configuration and Specification

The ultimate objective of this benchmarking process is to compare the performance of different NoSQL databases at various instances in Pseudo distributed mode. If the experiments were carried for different databases (i.e. Cassandra, MongoDB and Redis), on different machines, the results will depend on system specifications. Thus, it is unable to compare.

Therefore, the experiments were done on a single machine. Below are the specifications:

- Processor : Intel Core i7-3630QM
- Clock Speed : 2.40GHz
- Number of Cores : 4
- Number of Threads : 8
- Instruction Set : 64 bit
- Memory : 8061MB
- Operating System : Ubuntu 14.04.1 LTS
- Kernel : Linux 3.13.0-35-generic
- Java : OpenJDK 7
- YCSB Version : 0.1.4
- Cassandra Version : 2.0.10
- MongoDB Version : 2.7.4
- Redis Version : 2.8.15

V. Experimental Results

This section includes graphs, which shows throughput for the 6 workloads provided by YCSB. For each workload, 3 instances were considered with 1000, 5000 and 10000 records. Also, for each record size, 1,2,4 and 8 threads has been considered.

A. Workload A

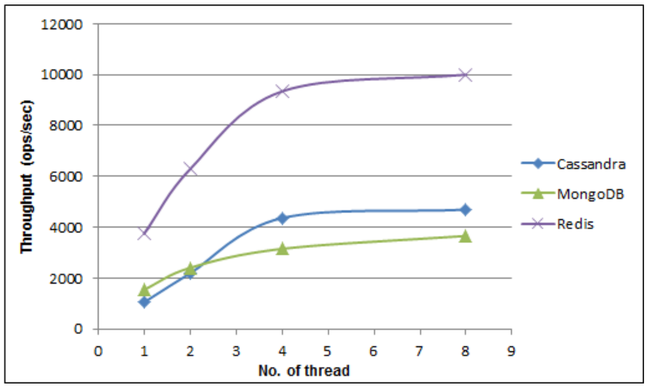

Fig. 4. Throughput at 1000 records and 1000 operations

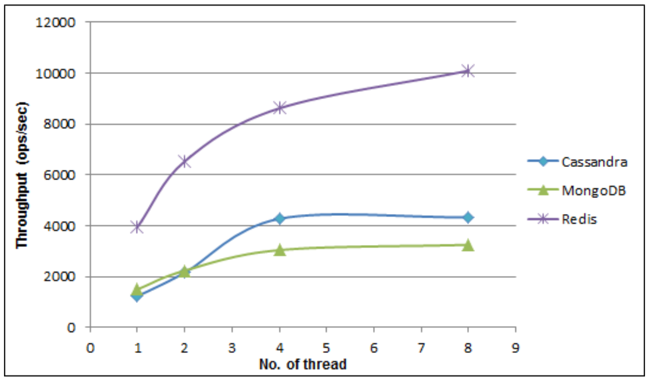

Fig. 5. Throughput at 5000 records and 1000 operations

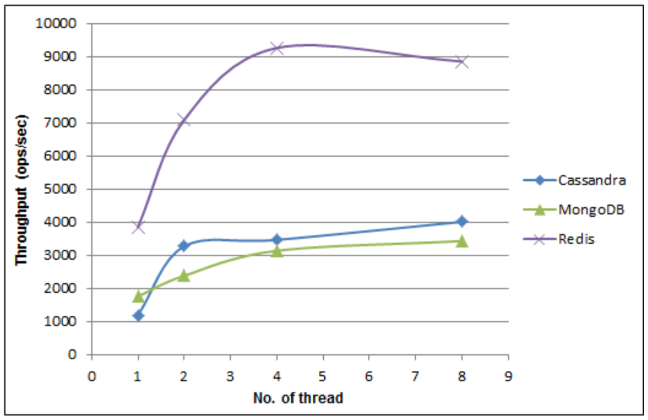

Fig. 6. Throughput at 10000 records and 1000 operations

For the heavy update workload it can be seen that Redis has twice as much throughput as MongoDB and Cassandra databases. At low concurrency (1, 2 threads) MongoDB and Cassandra have similar throughput but Cassandra has slightly better performance when the number of threads and number of records increase.

This may be due to the fact that data replication services provides by MongoDB and Cassandra databases do not exist in pseudo distributed mode for they to use memory mapping on several nodes in order to increase throughput.

It is also notable that speedup gain after 4 threads is not increasing rapidly even though the hardware provides for 8 threads to be run concurrently.

B. Workload B

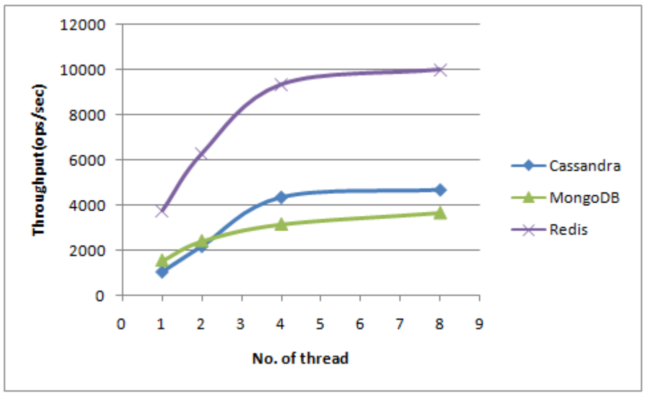

Fig. 7. Throughput at 1000 records and 1000 operations

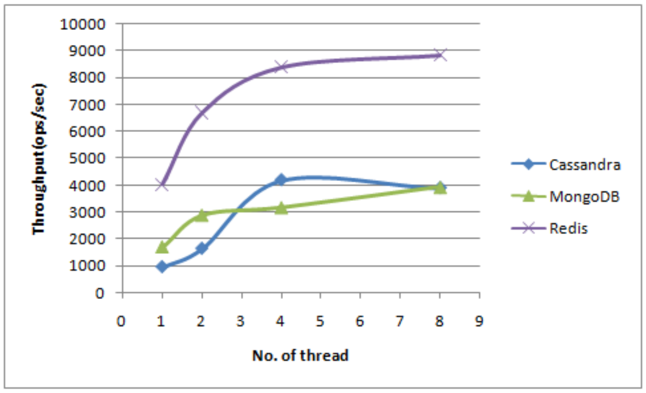

Fig. 8. Throughput at 5000 records and 1000 operations

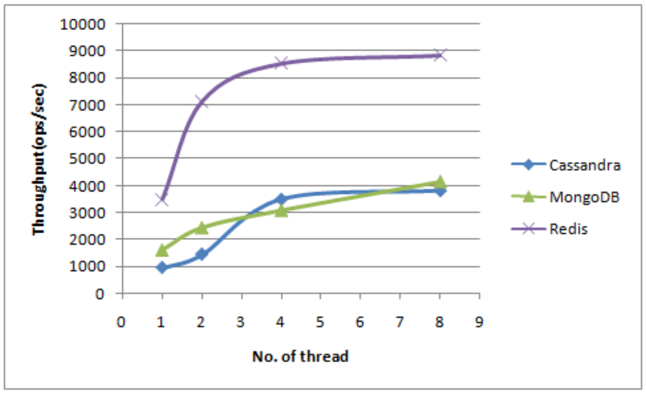

Fig. 9. Throughput at 10000 records and 1000 operations

This most reads workload results are similar to heavy update workload but MongoDB and Cassandra have identical throughput at 8 threads (maximum number of concurrently runnable threads by the hardware). MongoDB and Cassandra exchange places at 2 and 4 threads.

C. Workload C

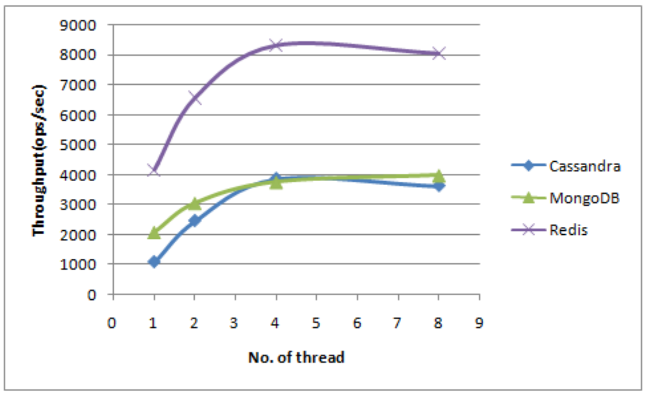

Fig. 10. Throughput at 1000 records and 1000 operations

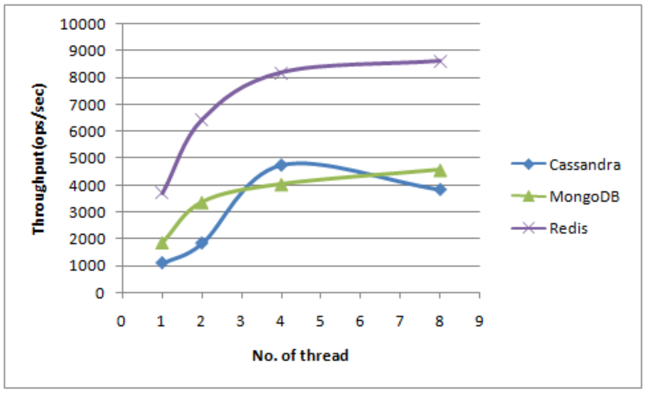

Fig. 11. Throughput at 5000 records and 1000 operations

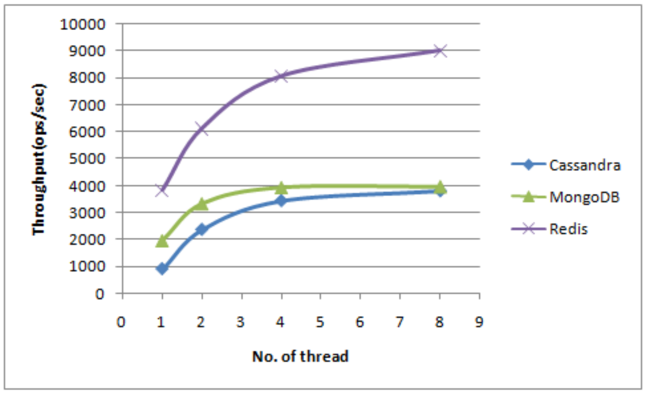

Fig. 12. Throughput at 10000 records and 1000 operations

Cassandra incurs high read latency, which reduces the overall throughput. This can be seen in MongoDB as well. In the graphs for the Workload C, which has a 100% read operations, this can be seen clearly. Whereas the overall throughput of Redis is higher than both MongoDB and Cassandra. With increment of thread count and record count the overall throughput of both Cassandra and MongoDB converges.

D. Workload D

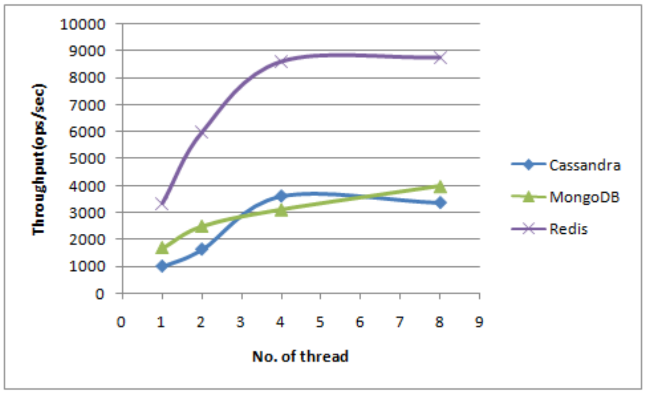

Fig. 13. Throughput at 1000 records and 1000 operations

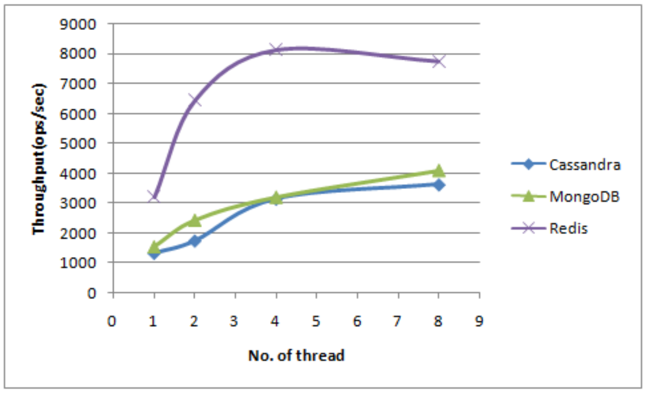

Fig. 14. Throughput at 5000 records and 1000 operations

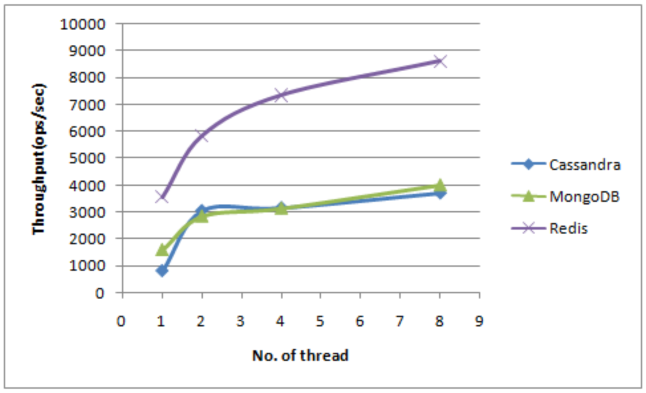

Fig. 15. Throughput at 10000 records and 1000 operations

It can be seen that neither database uses caching mechanisms to optimise for latest data written to be retrieved quickly as the throughput hardly change from the Workload A in each of the databases.

E. Workload E

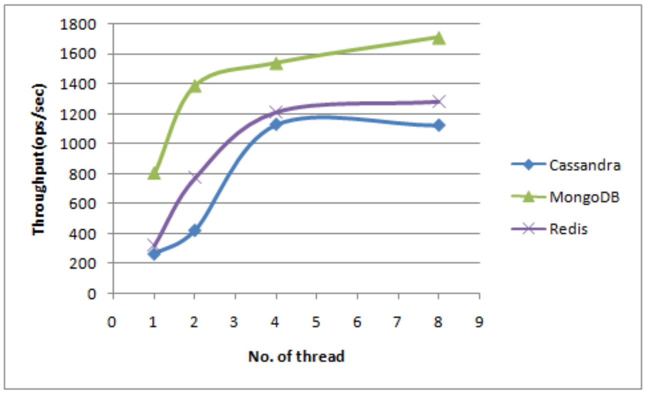

Fig. 16. Throughput at 1000 records and 1000 operations

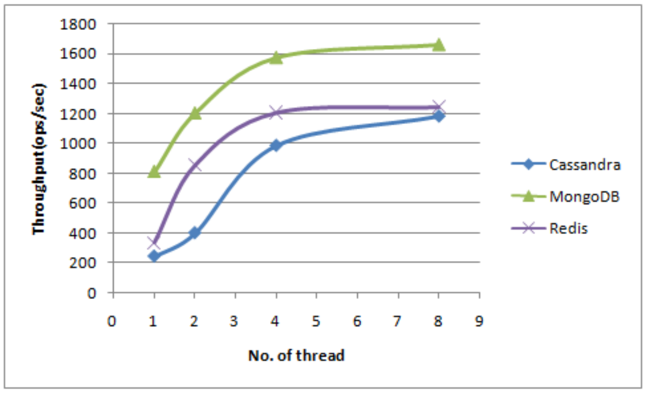

Fig. 17. Throughput at 5000 records and 1000 operations

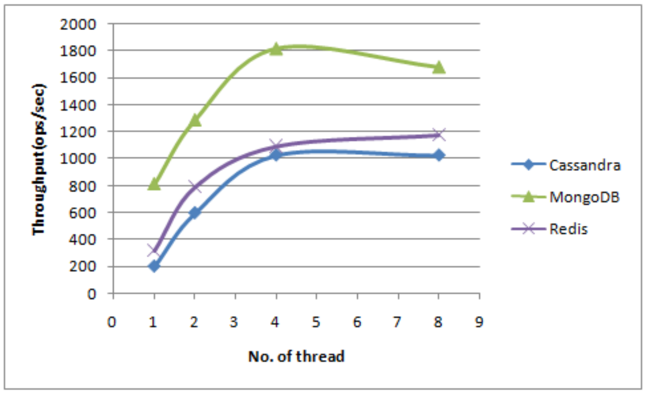

Fig. 18. Throughput at 10000 records and 1000 operations

For short range queries it can be seen that MongoDB outperforms Cassandra and surprisingly Redis as well. Cassandra and Redis have similar throughputs but Redis have slightly better performance but this seem to degrade with the number of records in the database.

F. Workload F

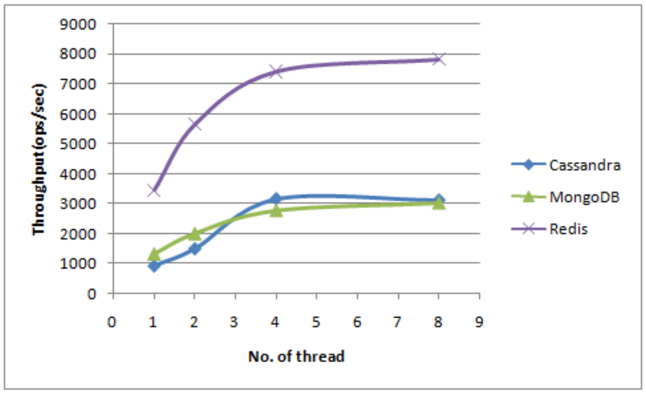

Fig. 19. Throughput at 1000 records and 1000 operations

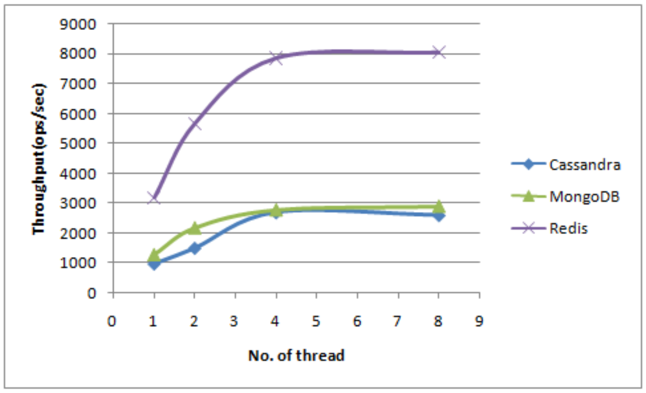

Fig. 20. Throughput at 5000 records and 1000 operations

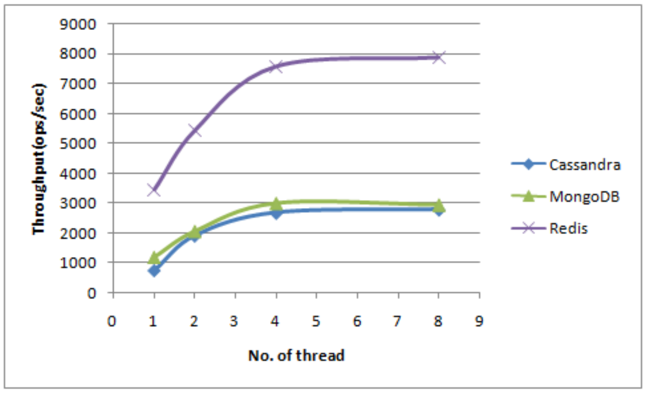

Fig. 21. Throughput at 10000 records and 1000 operations

In the read, modify write workload Redis has almost 4 times the throughput of MongoDB and Cassandra. This might be due to the fact that Redis does its memory operations in memory before it writes to the disk where as MongoDB writes data in disk.

G. Throughput and Latency with respect to the number of Threads

An experiment was carried out to identify the behaviour of the 3 databases for 1000000 records with 1-50 threads at 50 instances. Below are summarized graphs for read latency, update latency and throughput.

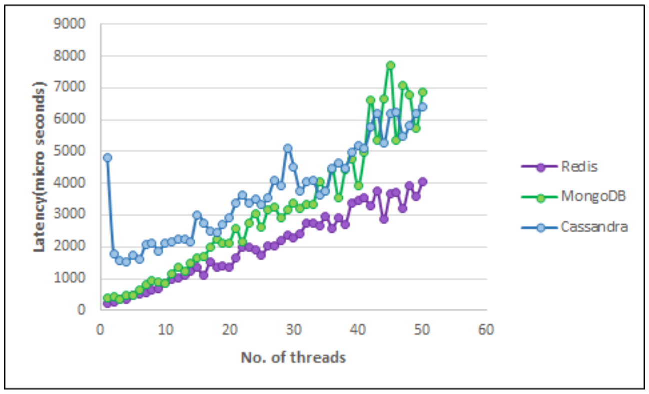

Fig. 22. Read latency for 100000 records

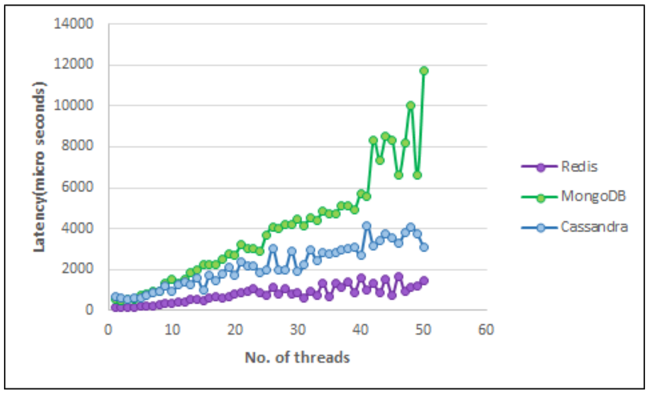

Fig. 23. Update latency for 100000 records

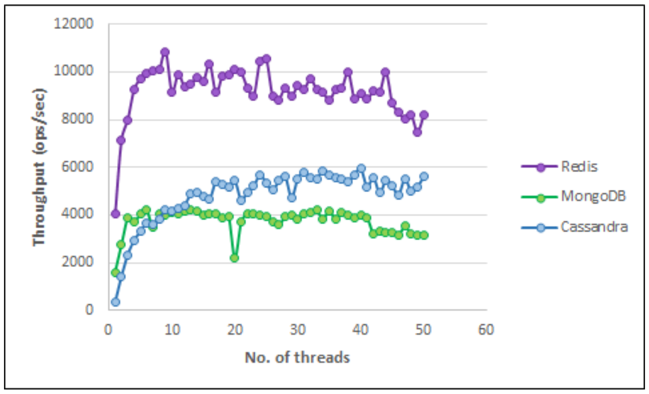

Fig. 24. Throughput for 100000 records

It can be seen that only in Cassandra that the throughput increase with the number of threads. In Redis, throughput decreases with the number of threads and in MongoDB the throughput remains close to unchanged until about 40 threads after which it steps down.
Since Cassandra optimizes writes in contrast to read operations, it completes almost 100% of the update operations immediately, therefore YCSB configures Cassandra to provide weak consistency by default. Cassandra is more suited to distributed systems, whereas the experiment was carried in a pseudo distributed mode. Therefore the read latency of Cassandra is higher than MongoDB. In distributed mode Cassandra is able to read locally without incurring cross-datacenter latency, which will show a lower read latency than MongoDB.

VI. Conclusion

In summary it can be concluded that in pseudo distributed mode Redis has better performance for applications which doesn't require range queries and developers can choose Redis over Cassandra and MongoDB for applications that have such requirements. Further analysis can be done on the performance of these databases for large number (millions) of records and operations to benchmark the databases further. YCSB can be used as the tool for this purpose.

VII. References

[1] Cooper, B.F., Silberstein, A., Tam, E., Ramakrishnan, R., and Sears, R. Benchmarking cloud serving systems with ycsb. In *Proceedings of the 1st ACM symposium on Cloud computing* (New York,NY,USA,2010), SoCC '10,ACM,pp.143-154.

[2] Core Workloads. RetrievedSeptemberJuly , 2014 [ONLINE] Available: https://github.com/brianfrankcooper/YCSB/wiki/Core-Workloads

[3] NoSQL Apache Cassandra Documentation — Planet Cassandra. 2014. [ONLINE] Available: http://planetCassandra.org/documentation/.

[4] mongod. RetrievedSeptember, 2014 [ONLINE] Available: http://docs.mongodb.org/manual/reference/program/mongod/bin.mongod

[5] Indexes and MongoDB. MongoDB Documentation Project, [ONLINE] Available: https://docs.mongodb.org/master/MongoDB-indexes-guide.pdf

[6] SCons: A software construction tool. RetrievedSeptember, 2014 [ONLINE] Available: http://www.scons.org/

[7] Paksula, M., Introduction to store data in Redis, a persistent and fast key-value database ,In *Proceedings of AMICT 2010-2011 Advances in Methods of Information and Communication Technology* (Helsinki, Finland,2011),AMICT,pp.39-47.

[8] Running a Workload. RetrievedAugust, 2014 [ONLINE] Available: https://github.com/brianfrankcooper/YCSB/wiki/Running-a-Workload

[9] mongodb/mongo GitHub. 2014. mongodb/mongo GitHub. [ONLINE] Available: https://github.com/mongodb/mongo.