

CSD Course
IK2200
KTH, Stockholm

KTH-DB Geo Distributed Key Value Store

Final Report

05 Jan 2015

Authors:

Ahmed Sadek <asadekk@kth.se>

Anargyros Chatzaras <anacha@kth.se>

Georgios Savvidis <gsav@kth.se>

Jawad Munir <jmunir@kth.se>

Tengqingqing Ge <tengge@kth.se>

Mihret Sidelel <sidelel@kth.se>

Table of contents

[Table of contents](#)

[1.Introduction to NoSQL Databases, YCSB, and MiniNet.](#)

[2. Problem statement](#)

[3. System requirements](#)

[4. System topology](#)

[5. Main database functions implemented:](#)

[5.1 Read, write and data replication](#)

[5.2 Inter node communication](#)

[5.2.1 Apache MINA](#)

[5.2.2 Node farm generation](#)

[5.2.3 The nodes interfaces](#)

[5.2.4 The nodes messages](#)

[5.3.The master selection algorithm](#)

[5.4 Latency Aggregation RTT and EWMA](#)

[5.5.The master handover](#)

[6. Results](#)

[7. Conclusions](#)

[8. References](#)

1.Introduction to NoSQL Databases, YCSB, and MiniNet.

This project builds on the idea of NoSQL distributed databases where conventional relational database systems can't provide high level of horizontal scalability, availability with dynamic role allocation. For that reason NoSQL databases came to offer database systems that is more reliable , scalable and less expensive “commodity hardware” that can work with structured and unstructured data types.

The NoSQL databases can be categorized into:

1. Key-values stores, where we distribute the number of keys on a number of nodes allowing scalability and less overhead for popular content.
2. Column family stores, where the record content is segmented into group of columns and the keys points to a group of columns.
3. Document Database, is the next level of key-value allowing nested values associated with each key which work more efficiently for querying documents.
4. Graph Database, instead of tables we use graph models.

Increasing Data Complexity ↓	Type	Examples
	Key-Value Store	 
	Wide Column Store	 
	Document Store	 
	Graph Store	 

source [<http://sqrrl.com/product/nosql/>]

(Yahoo! Cloud Serving Benchmark) YCSB is a tool used for benchmarking different database systems , this is really important in the process of comparison between the different

databases and giving the users an understanding of the capabilities (strong and weak points) of different databases relative to other database solutions. For example some database systems have really fast reading capabilities but slower writing capabilities in comparison to other type, so if the user needs a database system with fast writing speed then YCSB will be a good tool to know which one is faster among the different options. YCSB have already built-in interfaces with different NoSQL database systems and if you need it to communicate with other database system then you need to extend it with a new interface [1].

Mininet is a network emulator based on the software defined networks SDN approach that can be used to deploy different network elements such as Switches, Hosts and Controllers. This enable us to deploy full network (clients and servers) on a single machine with no need for a customized hardware. It uses the openflow protocol for communication between the controller and different switches in the network and uses (ovs-controller) as the default controller, although different controllers can be used [2].

KTH-DB is the database that we should be serving and implementing in this project, it should contain a number of functionalities as defined in section 5 and satisfy certain requirements as explained in section 3.

2. Problem statement

In this project we aim to develop a key-value store database with the name (KTH-DB) that is distributed on a number of nodes (10 nodes) and have the potential to store 10 million records. Our target tasks were:

1. Understand the different tools that will be used in the project YCSB, Mininet, how they work and how to customize them for our needs.
2. Mininet Topology development.
3. Have a basic communication between YCSB client and server nodes.
4. Design the communication mechanisms for the nodes.
5. Design a method for data replication between the master and slave nodes.
6. Dynamic role capabilities of server nodes. Any server node can take role of Master or Slave.
7. Develop a method for master selection and choosing the criterias for this process.

8. Develop a method for master handover.

3. System requirements

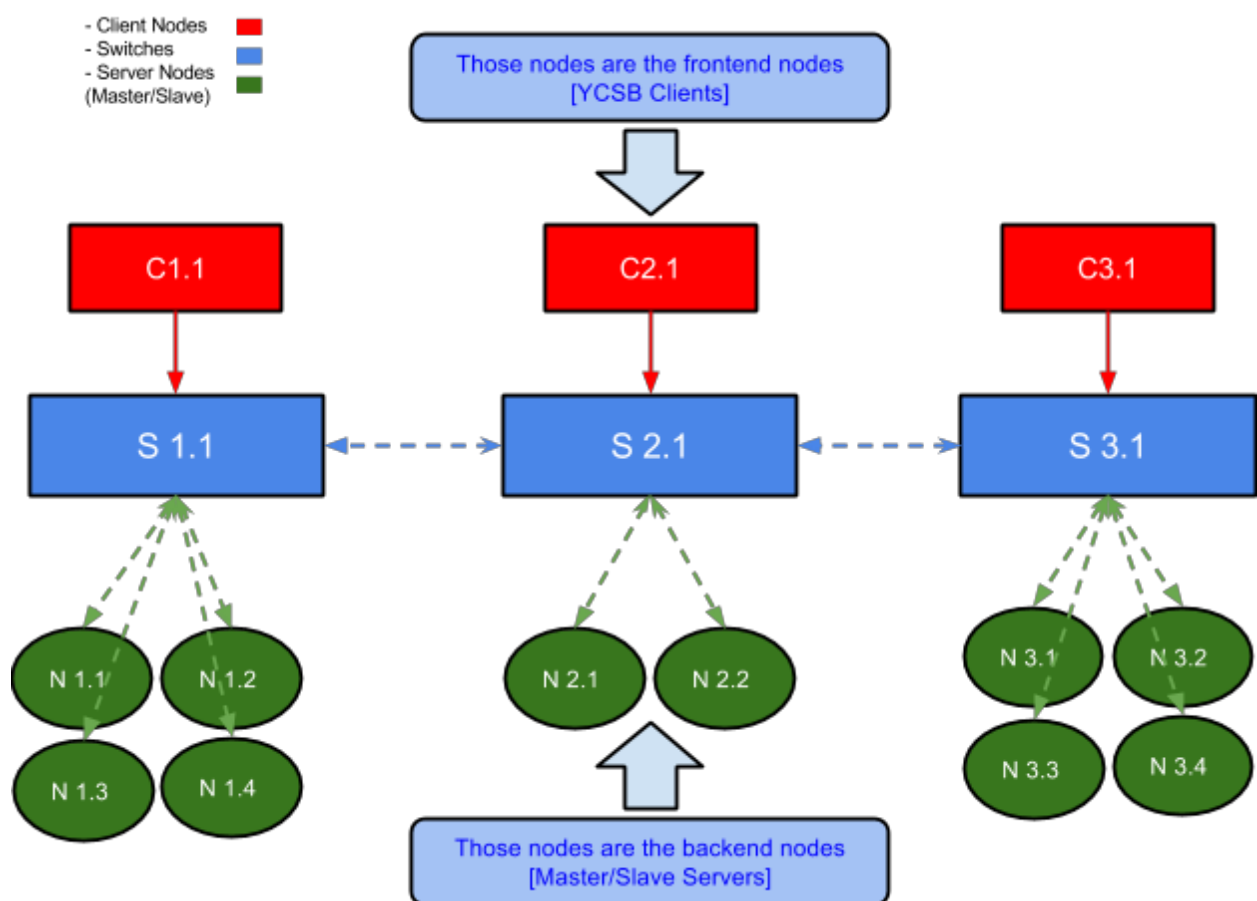
The project aims to accomplish the following set of requirements:

1. The database should be able to store 10 million keys.
2. The YCSB clients collectively should be able to perform 500 write operations per sec and 2000 reads operations per second.

4. System topology

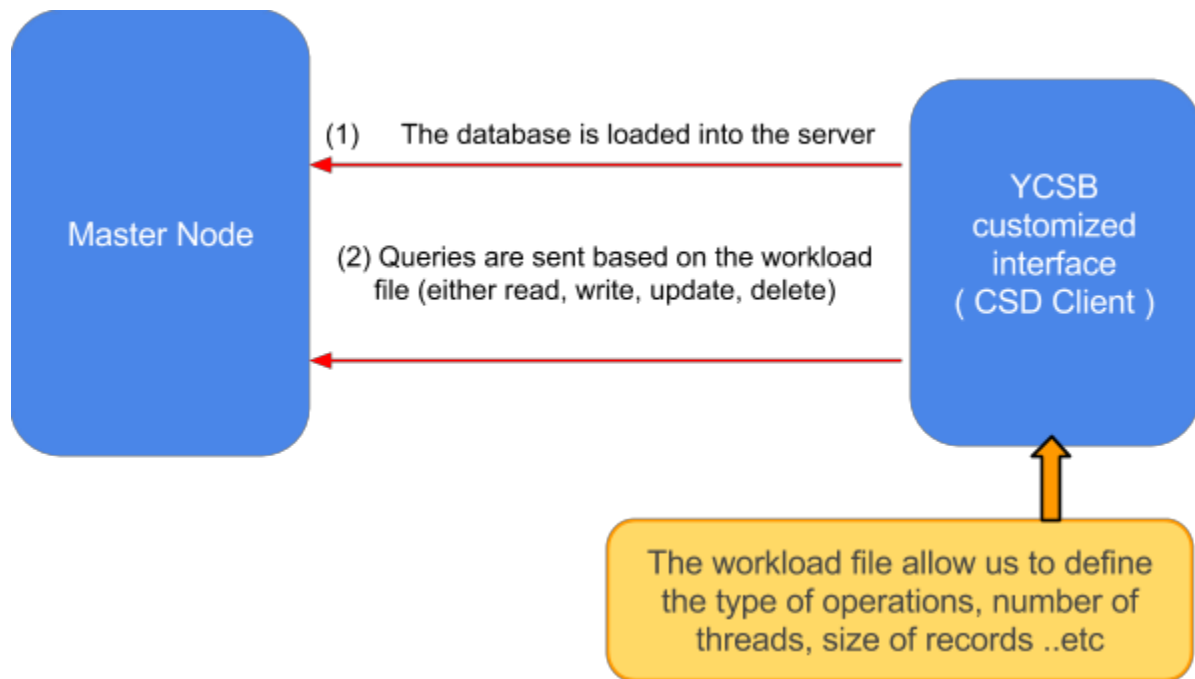
Our system consisted mainly of two sides; the client side (front end) and the server side (backend). All the nodes will be running on the Mininet framework as a switched network.

Using Mininet, we implemented the following following network topology:



“Figure 2. Network topology over Mininet”

There are three clients, running the yahoo bench marking tool to benchmark the KTH-DB database system.



“Figure 3. YCSB Usage Scenario”

On the serving side, there are 10 serving nodes. Nodes can take dynamic role: either as a Master (it can read and write data to the database) or as a slave (it can read only data from the database).

Our technical requirements are the following:

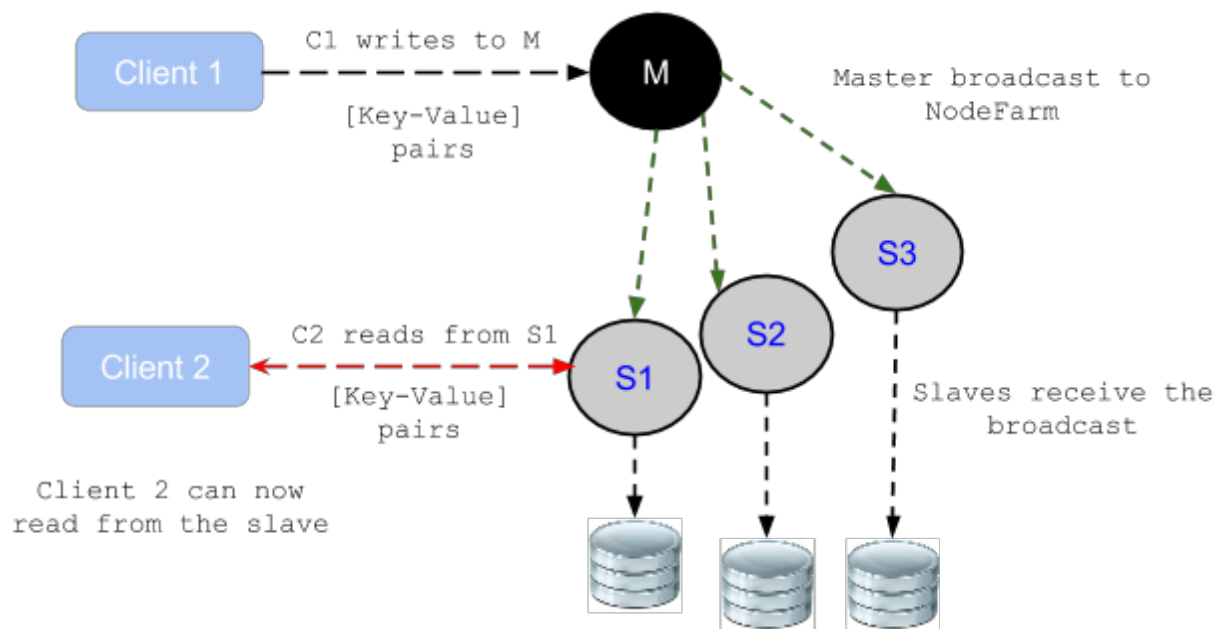
- 1- Storing 10 million key-Value pairs.
- 2- Achieving 2000 read/sec and 500 write/sec.

5. Main database functions implemented:

Our system consists of a number of functions that work together to deliver the requirements specified in the previous section.

5.1 Read, write and data replication

- 1- The Master node is the only node allowed to write and read from the database file. This way we can have a data consistency, while the slave node can only read.
- 2- For the slave node to read data that it didn't write it needs to receive a copy of what the master node is writing.
- 3- The master node broadcast the key-value data record to the slave nodes once it receive them and then each slave write this received key-value to itself.



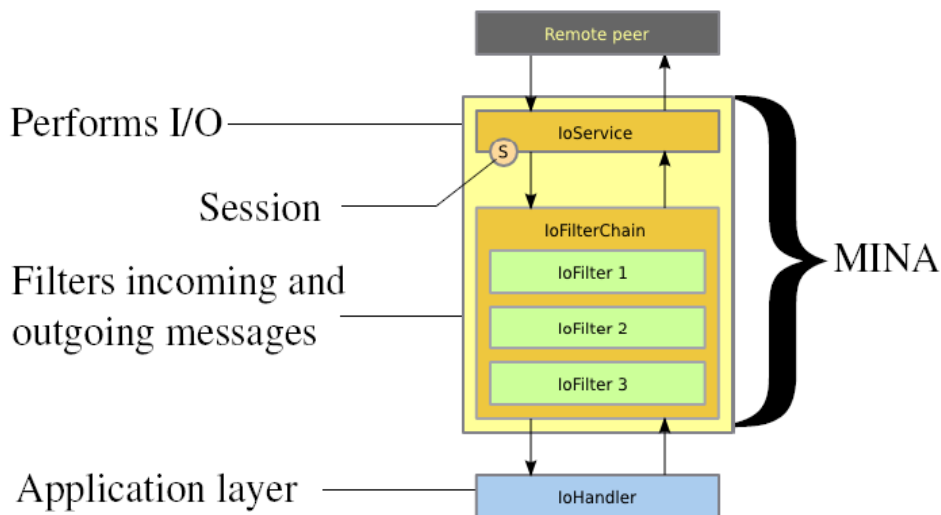
“Figure 4. Data replication diagram”

5.2 Inter node communication

The inter node communication is where we handle communication between all the nodes, this is achieved by using interfaces on each nodes and messages exchanged between the nodes and connections opened between the nodes by Apache Mina.

5.2.1 Apache MINA

Apache MINA is a network application framework which helps users develop high performance and high scalability network applications easily. It provides an abstract event-driven asynchronous API over various transports such as TCP/IP and UDP/IP via Java NIO [3]. So instead of writing low level socket communication, we use the application after customizing it to our needs. MINA works as an abstraction layer between the application (on client or server) and the network layer.



source [<https://mina.apache.org/mina-project/userguide/ch2-basics/application-architecture.html>]

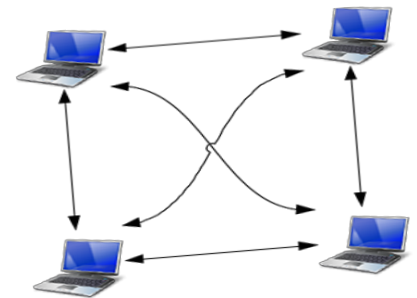
Figure 5. MINA framework layer architecture

In general, to create a MINA based Application, we need to do the following :

1. Create an I/O service - Choose from already available Services (*Acceptor) or create your own
2. Create a Filter Chain - Choose from already existing Filters or create a custom Filter for transforming request/response
3. Create an I/O Handler - Write business logic, on handling different messages

5.2.2 Node farm generation

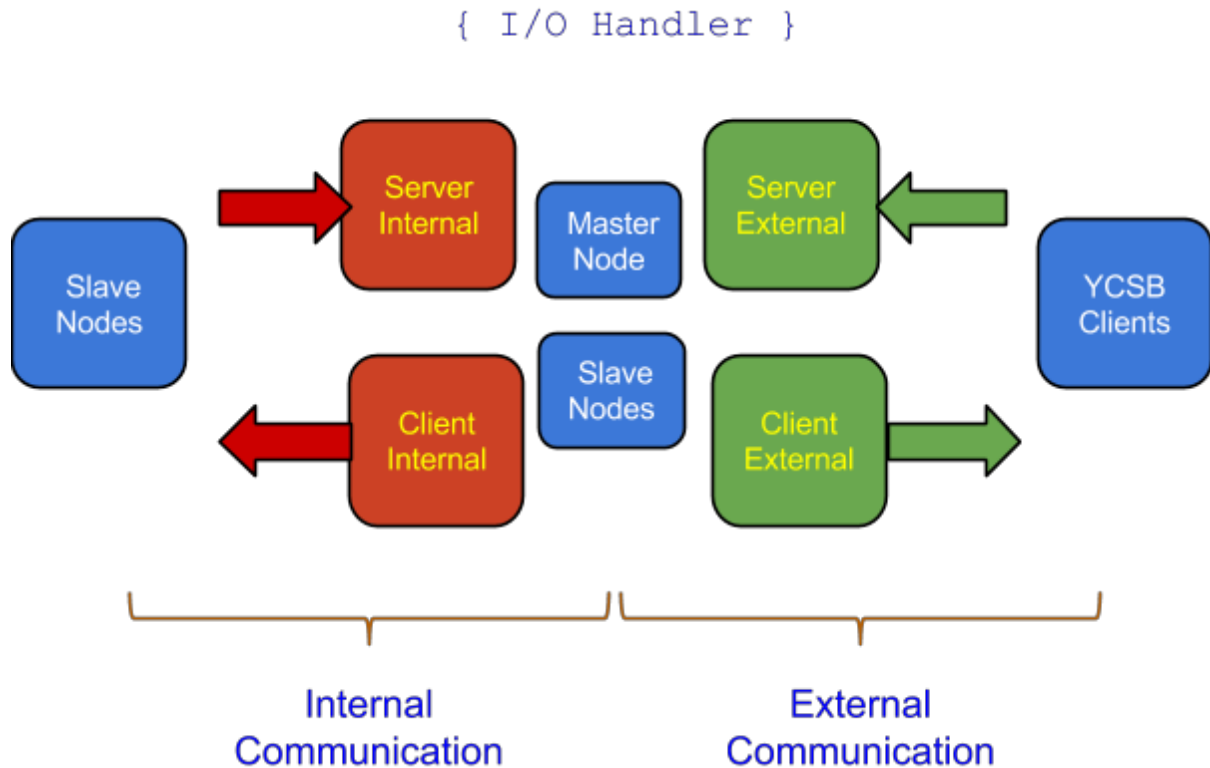
When each server node starts, it creates a mesh connection with each other node in the cluster, this allow the nodes to have continuous communication as the roles change with no overhead time in initiating and closing the connections.



5.2.3 The nodes interfaces

For easy and modular communications between the serving nodes, we created 4 I/O handlers, for input/output handling for internal communication and external communication. [Frontend- Backend] and two interfaces [Backend-Backend] Figure [6].

- 1- Server **External** interface - to handle external communication from the YCSB clients.
- 2- Client **External** interface - I/O handler for YCSB clients.
- 3- Server **Internal** interface - to handle communication from the other nodes in the node farm.
- 4- Client **Internal** interface - to initiate communication to the other nodes in the internal node farm.



“Figure 6. System interfaces”

5.2.4 The nodes messages

There are 5 types of messages exchanged in the system for different activities.

The message types are the following:

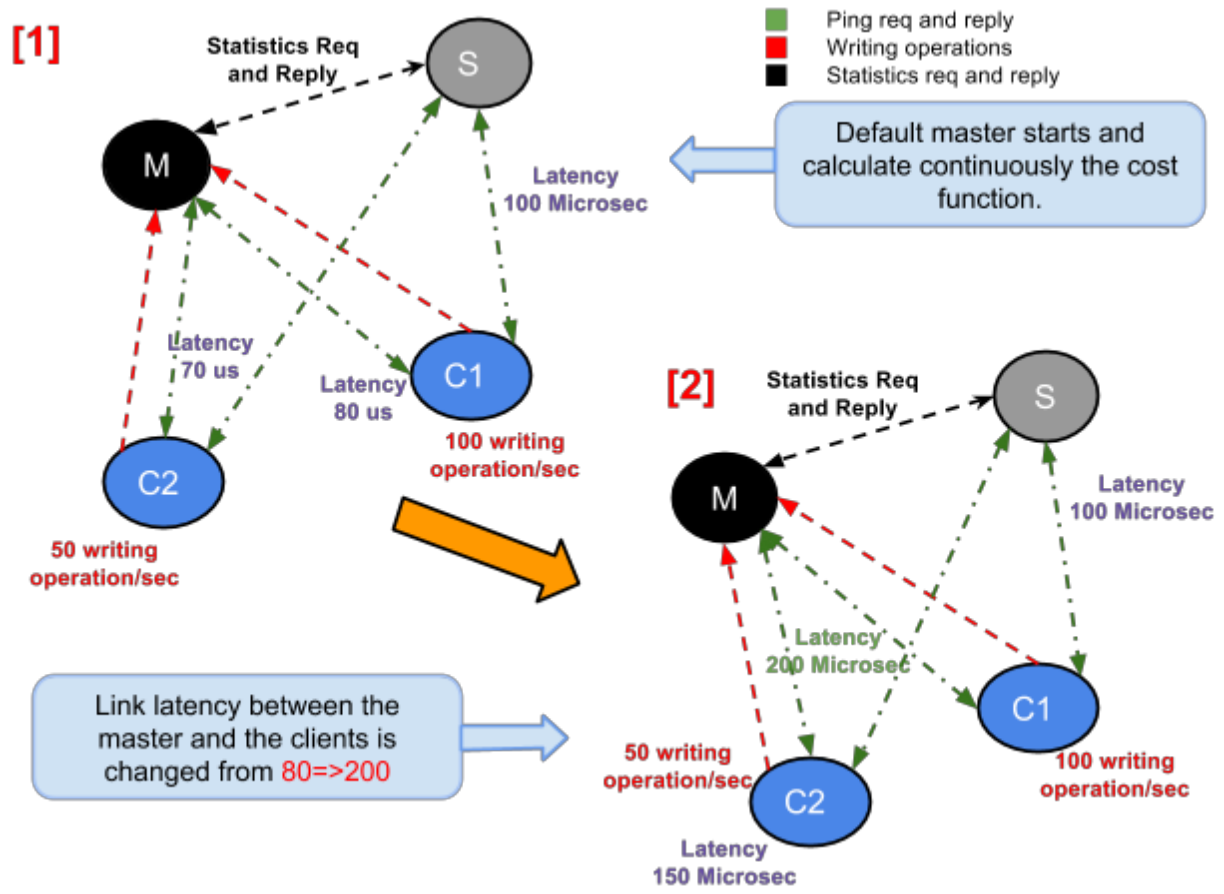
1. Master Moved, which is broadcasted by the master in case of master handover.
2. Statistics Request, which is sent from master to slave asking for latency measurements.
3. Statistics Results, which is sent from slave to master as a response to the Statistics Request
4. Operation Read, sent from YCSB client to Master/Slave to read data.
5. Operation Write, Sent from YCSB client to Master to write data.

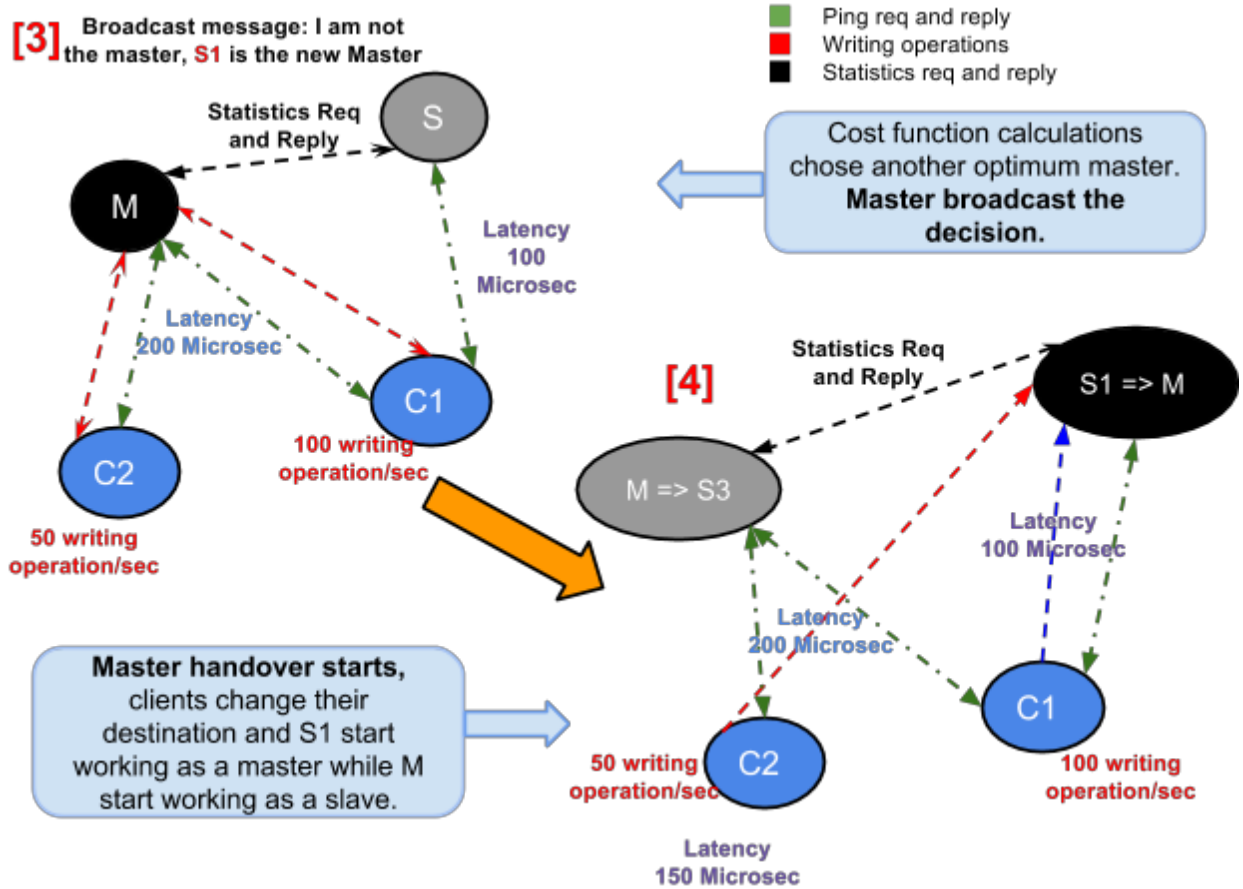
5.3.The master selection algorithm

The master selection algorithm aims to change which node is performing the role of master between the serving nodes Figure [7]. The aim of the exchange is to select the master node that provides the optimum conditions for the clients who are performing write operations. The optimum conditions for the clients is achieved by having a master node as close as possible (min latency) to the most active clients (active client is a client who is doing write operations).

For that we calculate a cost function that takes into consideration two parameters:

1. The number of writing operations coming from the clients Figure [1]. Every time master node performs a write operation it keeps track of which client performed the operation so that the throughput of that client is calculated.
2. The latency measurement from all the nodes to the clients. Each slave node pings the different YCSB clients and sends the result (RTT) back to the master.





“Figure 7. Master handover state transition”

5.4 Latency Aggregation RTT and EWMA

For latency calculations we use the following algorithm:

1. The master sends a request (**StatisticsRequest**) to each node in the **NodeFarm** asking for measurements and provide them with updated **YCSBClientList**.
2. Each node in the farm start pingging the clients one by one and pars the RTT time out of the reply.
3. Due to spicks and to allow more smother change in RTT values, we calculate exponential moving average on the RTT values allowing us to consider past sample as well as new samples. for EMA we use an open source library EWMA [4].
4. Each node in the farm send back the results (**StatisticsResults**) to the master.
5. Master input this measurements to the **CostFunction** to calculate the **CostValue** for each node.

The **CostValue** is based on the latency from serving nodes to client nodes and the number of write operations done by each client.

$$Cost = \sum_{i=1}^3 EMA(throughput_i) * EMA(delay_i)$$

In the above equation, the $throughput_i$ is the number of operations per second performed by $client_i$ ($i = 1, 2, 3$); the $delay_i$ is the Round-trip-time from the serving cost to the $client_i$ ($i = 1, 2, 3$).

Then the master selection is the minimum outcome from the cost function where it compares between:

- A- Each slave cost value. B- Current master own cost value.

5.5.The master handover

The master handover starts after the broadcast of a new master (**NewMasterSelected**). It involves the following actions:

1. Old-Master redirecting YCSB clients to new master for writing requests.
2. Old-Master stop master functionality and switch to slave mode.
3. New-Master switch from slave mode to master mode.

6. Results

1. We have done a number of tests with different configurations scenario to demonstrate the functionalities that we implemented. Master Selection happens every 10 second and Statistics collections (Delay measurements and throughput) happens every 10 seconds also.

No	No of Servers	No of clients	No of records	No of operations	No of threads	Type of operations	Ops/sec	Average time (us)	Run Time (ms)
Scenario 1. two clients: one client is loading the database, the second is reading only									
1	10	2	1000	100000	10	read	3579	2263	27,000
Scenario 2. two clients: one client is loading the database, the second is reading only									
2	10	2	10000	10000	10	read	388	11600	25,000
Scenario 3. one client is loading and run the database for reading and writing									
3	10	1	100000	1000	10	50-50	542	2000	184,169
Scenario 4. one client is loading and run the database for reading and writing									
4	10	1	10000000	10000	10	50-50	4500 for load 150 for read and run	10000	2000 sec for load and 8000ms for run

7. Conclusions

1. The performance of our system is affected by the hardware architecture and whether it's running on an operating system or a virtual machine.
2. YCSB can give an answer about which database can satisfy your needs, this needs will be based on the workload file configurations.
3. Our database system stores the key-value data in the memory and then flush it to the disk, which was a design decision to minimize IO time. However, as the number of database records increases, the memory needed is increased which put a pressure on the machine and can cause a memory dump.

8. References

- [1]. <https://github.com/brianfrankcooper/YCSB/wiki>
- [2]. <http://mininet.org/>
- [3]. <https://mina.apache.org/>
- [4]. <https://github.com/dropwizard/metrics/tree/master/metrics-core/src/main/java/com/coda/hale>