ASSIGNMENT 2
CLOUD COMPUTING
MONGODB VS COUCHDB

Submitted by:
Rashmi Gulhane
Akansha Gupta

Mongodb

Steps we followed with mongodb

1. Created cluster as specified below:
   Cluster configuration:
   3 config server
   2 masters (query servers)
   2 shard servers
2. CRUD operations using mongo shell
3. CRUD operations using java edition programme
4. Performance evaluation with multiple scenarios
5. Consistency measurement with different scenarios

Tools used for benchmarking

Eclipse - IDE
Java - programming language

Using Multiple masters:

Tried to evaluate mongodb by using multiple query processors for processing common query:

1. Inserted 10k of records by using single master(query processor), it took 11 seconds
2. Used two masters and inserted 5K-5K records from each at the same time then both took 5.6 seconds around.

So with increasing masters query processing time got halved. That means in mongodb multiple threads can operate data at the same time until those are dealing with different documents.

CouchDB

Steps followed for CouchDB

1. 3 Nodes are configured as following:
   CouchDB 1.6 database
   Bigcouch 0.4.2-1
   Number of Shards for each db:2
   Copies of each shards:3

2. CURL Restful API Implementation using JAVA
3. Performance evaluation with multiple scenarios
4. Consistency measurement with different scenarios

Tools used for benchmarking

Eclipse - IDE
Java - programming language
Curl Restful API

Mongodb vs CouchDB

Data sets & Configuration:

OS: Ubuntu 14.04 LTS-64 Bit
CPU: Intel® Core™ i5-4570 CPU @ 3.20GHz
Database File sizes used: 2.4 MB, 12.7 MB, 127MB
No of Documents each of the above file contains: 1K, 10K, 100K, 1000K

Scenarios used for performance measurement:

Used three scenarios for performance measurement:

1. Insertion of Documents

One by one document insertion:

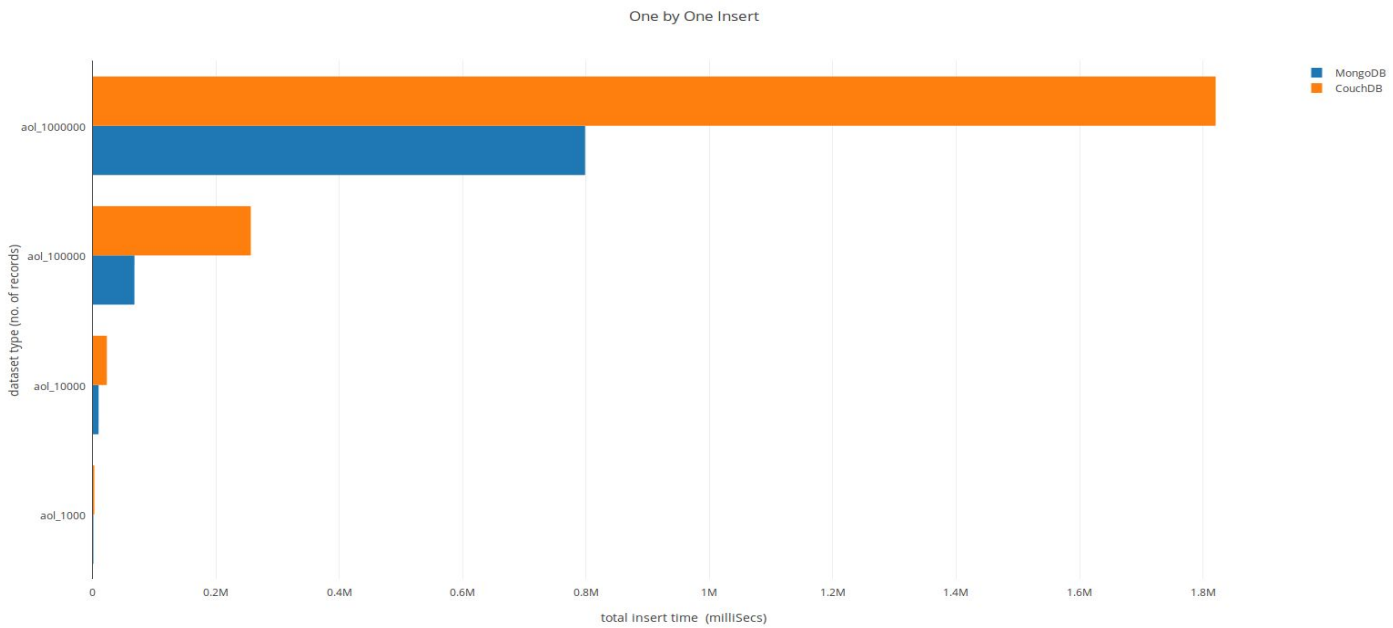
- **Creating records In memory:** Used a java for loop to generate random records to insert and evaluated by varying loop counter range.
- **Reading records from file:** Also tried to read records from a file that contains json documents to insert and evaluated by using different size of files.

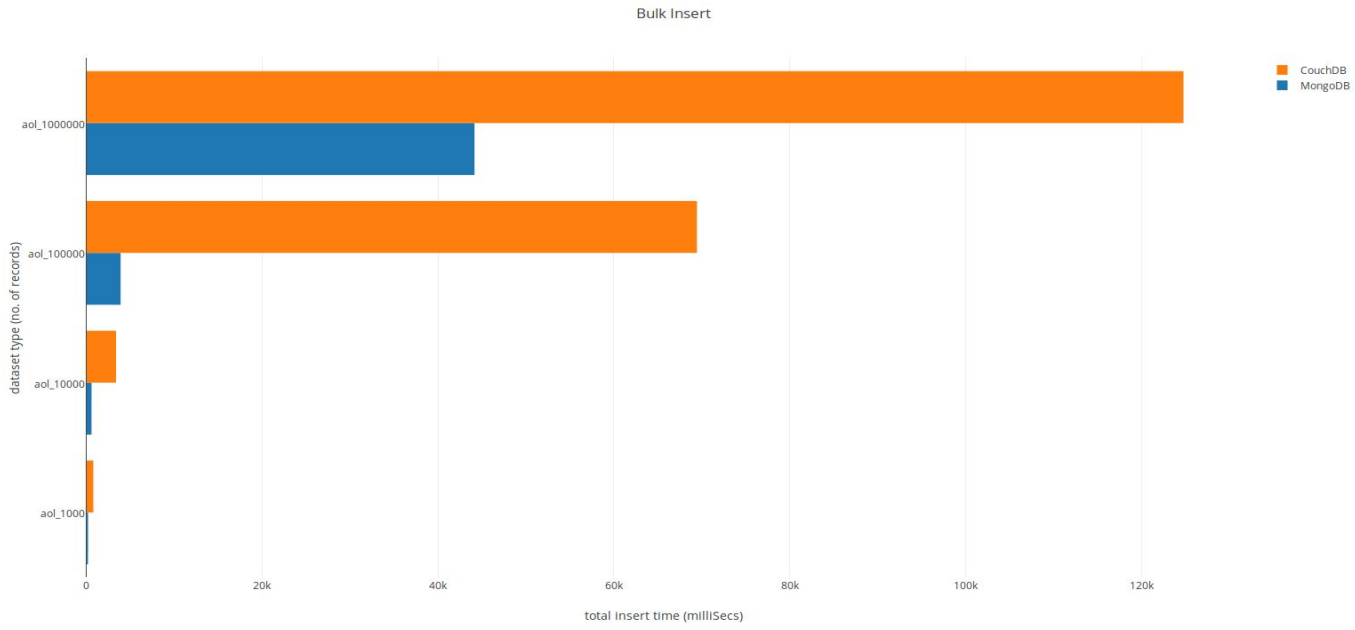
Bulk insertion:

1. **Creating records In memory:** Created a list of random generated records by for loop then inserted into db by single insert statement.

2. **Reading records from file:** Read all json documents from file, created a list then inserted by single insert statement.

	MongoDB		CouchDB	
	Document Insertion (in millisec)	Document Insertion in Bulk (in millisec)	Document Insertion (in millisec)	Document Insertion in Bulk (in millisec)
1K	1312	149	2320	805
10K	9928	601	23292	3389
100K	68054	3905	256668	69449
1000K	798552	44151	1820575	124786

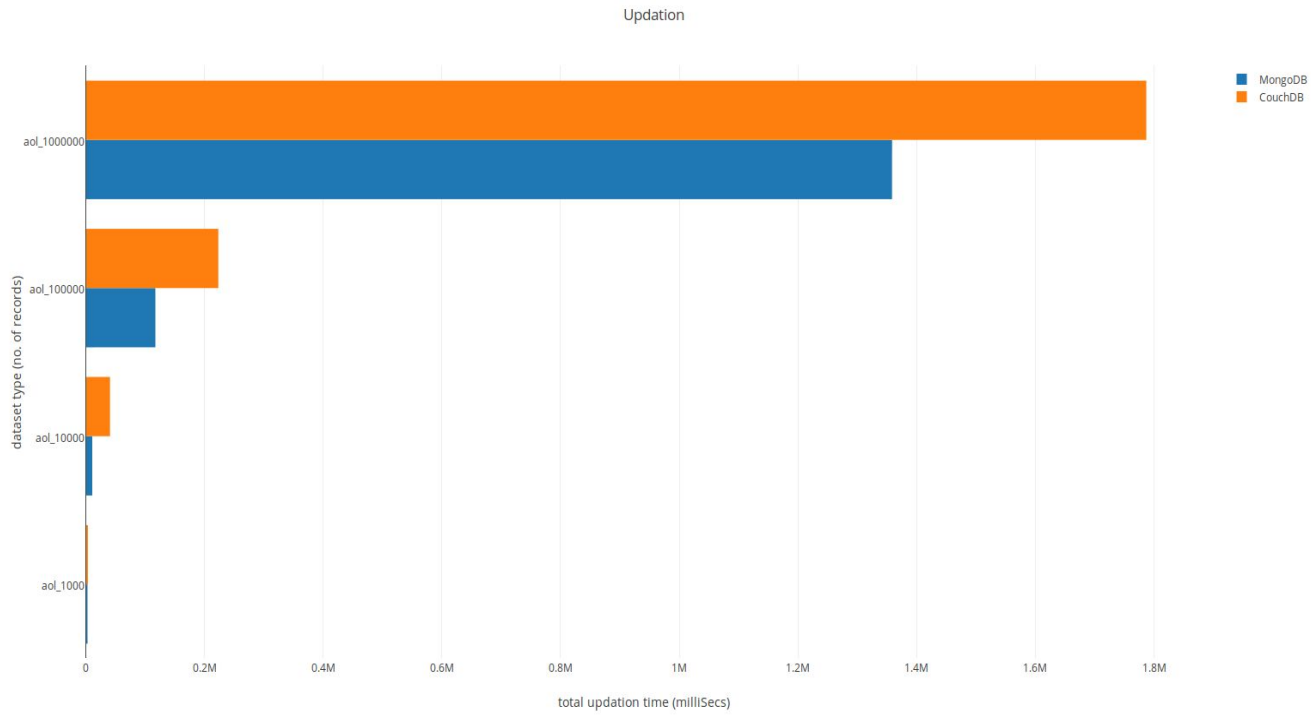




2. Updation of Documents

The following readings are noted down while updating each document in the database.

No of Documents	MongoDB (in millisec)	CouchDB (in millisec)
1K	1577	3560
10K	10853	40728
100K	117261	223256
1000K	1358505	1786911



Scenarios used for consistency:

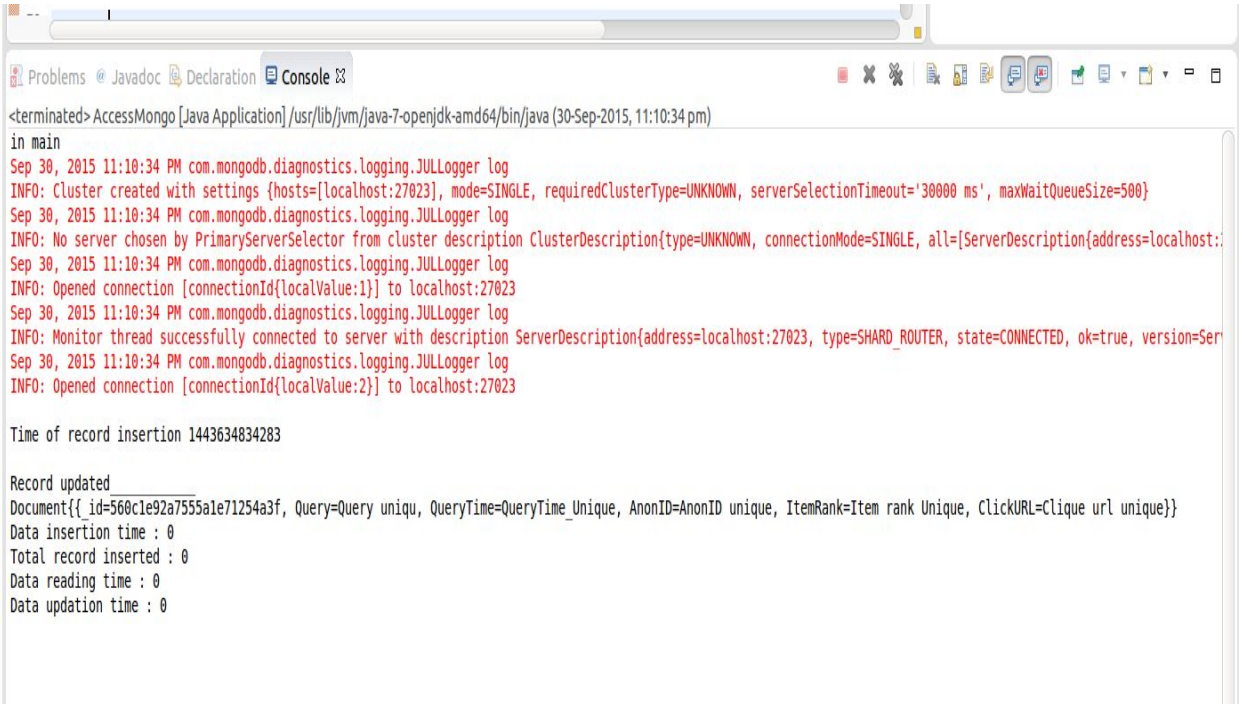
Updated one record from a query processor. Another query processor was set to check that record using a loop. Printed the Timestamp for each of the above operations

Consistency Check for MongoDB:

We used two query processor nodes for consistency measurement. Below are scenarios:

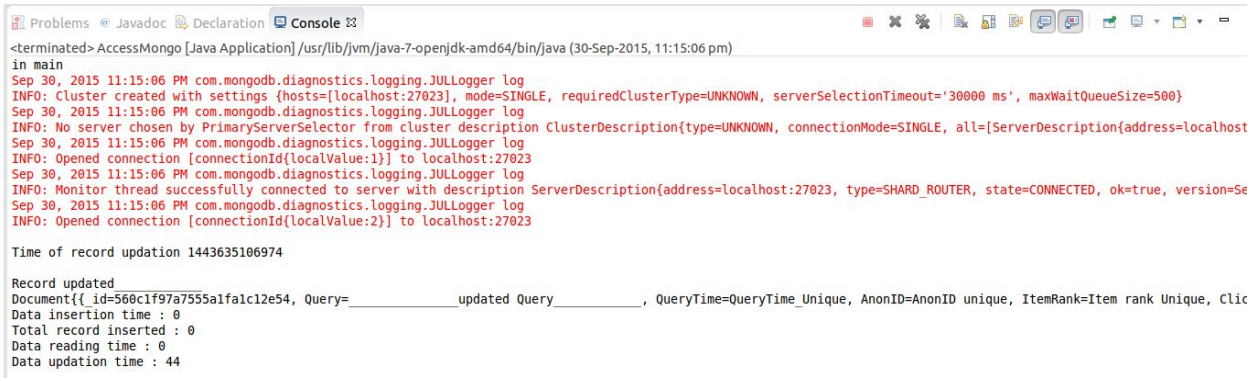
1. Updated one record from a query processor. Another query processor was set to check that record using a loop. It reported for updated record by time difference of **30 milliseconds.**

Insert command from one query processor:

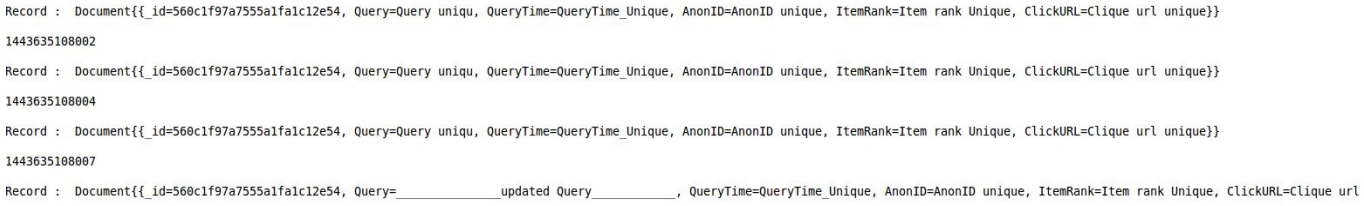


On insert from another query processor

Update command from one query processor:



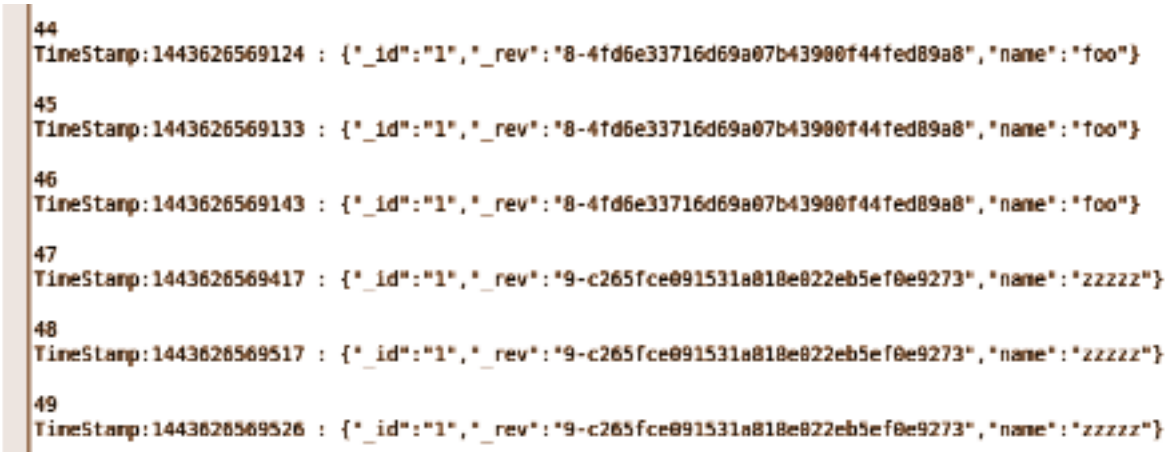
Update receive on another query processor



Consistency Check for CouchDB:

On Document Updater side:

On Document Reader Side:



Timestamp Difference Obtained from update and read operation on two different Nodes:
Timestamp Difference= 1443626569417-1443626568634
= 783 milliseconds

Conclusion:

MongoDB	CouchDB
Master/slave configuration (auto failover with replica sets)	Master-Master Configuration
Use when required high Consistency	Use when required high Availability
Better update-in-place than CouchDB	Update with conflict detection
Sharding built-in	Only CouchDB 2.0 Developer preview supports Sharding. Stable version is not available
Use when data changes too much	Use when data changes occasionally

Final Conclusion: MongoDB is faster out of the box; CouchDB is safer out of the box.

Note: High quality images are given in separate folder attached with submission.