Journal of Mobile, Embedded and Distributed Systems, vol. VI, no. 1, 2014
ISSN 2067 – 4074

Big Data Issues: Performance, Scalability, Availability

Laura MATEI
IT&C Security Master Program
The Bucharest University of Economic Studies
ROMANIA
lauramatei13@gmail.com

Abstract: Nowadays, Big Data is probably one of the most discussed topics not only in the area of data analysis, but, I believe, in the whole realm of information technology. The simple typing of the words „big data" on an online search engine like Google will retrieve approximately 1,660,000,000 results. Having such a buzz gathered around this term, I could not help but wonder what this phenomenon means.
The ever greater portion that the combination of Internet, Cloud Computing and mobile devices has been occupying in our lives, lead to an ever increasing amount of data that must be captured, communicated, aggregated, stored, and analyzed. These sets of data that we are generating are called Big Data.

Key-Words: Big Data, NoSQL, database, Hadoop, YCSB, performance test

1. Introduction

A definition for big data is given by Edd Dumbill in his article "Making Sense of Big Data" from the online journal "Big Data", [1]. The Big Data is defined as data that exceeds the processing capacity of conventional database systems. The data is too big, moves too fast, or does not fit the strictures of your database architectures. To gain value from this data, you must choose an alternative way to process it.
In other words, big data is something that exceeds the capabilities of commonly used software tools to capture, curate, manage, and process the data within a tolerable elapsed time. This means that in order to better understand big data, one should first understand the new tools, the modern equipment used to manage very large amounts of data. And because data is stored in databases, understanding big data means understanding the new type of databases, namely NoSQL databases.
The necessity of this paper comes from the fact that although big data has become a force of innovation among companies of all sizes and more and more technologies used to handle big datasets are developed [2-4], few means of comparability exist between these big data platforms. While the performance of traditional database systems is well understood and measured, there is neither a clear definition of the performance of big data systems nor a generally agreed upon metric for comparing these system.
The purpose of this paper is to analyze and compare several NoSQL databases in order to discover the various functionality and trade-offs each of them has and to find out which of them is better fitted for what scenario.
The process of analysis and comparison will go through the presentation and description of the chosen databases, the description of testing tools used to analyze selected products, testing of each database regarding its performance on large amounts of data, high availability (stress testing) and security, and, as a last step, the comparison of the resulted output of the tests performed.

2. Existing Big Data solutions

Alongside with the birth of big data came the necessity to find a database that had the ability to store, aggregate, and combine huge amounts of data and then use the results to perform deep analyses. This is how a new genre of non-relational databases was born, seeking to solve problems that relational databases were a bad fit for. These databases are called NoSQL databases.

2.1 NoSQL – definition and terminology

In his white paper, "NoSQL Databases" [5], Christof Strauch affirms that the term of "NoSQL" first came to life in 1998, when it was used to define a non-relational database that omitted the use of SQL. Few years after that, in 2009 the term was picked up again in a conference for non-relational databases in San Francisco organized by Last.fm developer Jon Oskarsson. Another person who is said to be responsible for making this term popular is blogger and Rackspace employee Eric Evans.

On his website, Martin Fowler [6] describes the term of NoSQL as being wordplay: many advocates of NoSQL say that it does not mean a "no" to SQL, rather it means "Not Only SQL". Although he believes that there is no strong definition of this concept out there, Martin Fowler considers NoSQL databases to be a particular rush of recent databases that share the following characteristics: not using the relational model (or the SQL language), no schema, allowing fields to be added to any record without controls, open source, designed to run on large clusters, based on the needs of 21st century web properties.

2.2 NoSQL databases – classification

According to the authors of the book "Seven Databases in Seven Weeks", Eric Redmond and Jim R. Wilson [7], the five main styles of NoSQL databases are relational, key-value, columnar, document-oriented, and graph.

The relational model is probably what most people think of when regarding their database experience, consider the authors. And this comes as no surprise, since relational databases have been for over 40 years, and continue to be, despite their limitations, the most common used database genre. Relational Database Management Systems (RDBMSs) are set-theory-based systems in which tables have two dimensions: rows and columns. The means in which one can interact with an RDBMS is by writing queries in SQL (Structured Query Language). In a RBDMS data values can be of different types: numeric, like integer or float, character strings, dates, the Boolean values true or false and so on. These types can be enforced by the system. An important aspect of RBDMSs, the authors note, is that the tables can morph and join into new, much more complex tables, due to their mathematical basis in relational theory.

Some examples of relational databases include open source databases like MySQL, SQLite, H2, HSQLBD and PostgreSQL, the latter being covered in much more detailed further in this paper.

The key-value (KV) store is the simplest model we covered in the book "Seven Databases in Seven Weeks". The authors explain that a KV store pairs keys to values in the same way that a map or a hash table would in a programming language. Although some key-value implementations permit complex value types such as hashes or lists, the authors note, it is not required for these types to exist in a KV store. Another added bonus for KV implementations is that some of them provide a means of iterating through the keys. As an example of a key-value store, a file system could be considered, if one were to think of the file path as the key and the file contents as the value. An important characteristic of key-value stores is that they can be incredibly performant in many scenarios, because they demand so little resources, but they are usually not very useful when it comes to complex querying and aggregation.

There are many examples of key-value stores out there, the most popular of them being memcached and its relatives, memcachedb and membase, Voldemort, Riak, BerkeleyDB and Redis.

Columnar, or column-oriented databases, as their name suggests, are databases in which data tables are stored as sections of columns of data rather than rows of data, like in the most relational databases. A column-oriented database serializes all of the values of a column together, then the values of the next column, and so on. In [7] it is pointed out that although the difference between column-oriented databases and relational databases may not seem much, the

ISSN 2067 – 4074

impact of the columnar design decision runs deep. In column-oriented databases adding columns is done on a row-by-row basis and is very efficient. A row can either have a different set of columns or none at all, allowing, in this way, tables to remain sparse without them being subjected to a storage cost for null values. Regarding the structure of a columnar database, it is about midway between relational and key-value stores [7].

Some examples of column-oriented databases include Hypertable, HBase and Cassandra. The latter two are covered later in this paper.

Document-oriented databases are named so, because they are made up of a series of self-contained documents. The authors, Eric Redmond and Jim R. Wilson, explain that a document is like a hash containing a unique ID field and values that may be of any type, including more hashes. Documents exhibit a high degree of flexibility because they can contain nested structures and so they are applicable for variable domains. Also, very few restrictions are made on incoming data, as long as it can be expressed as a document. Different document databases have different approaches with regards to indexing, replication, consistency, ad hoc querying and other design decision. The authors of the book point out that it is very important for one to have a good understanding of these design decision before choosing a database.

Two of the most popular document-oriented databases are MongoDB, which its creators claim to be "the leading NoSQL database" and CouchDB. MongoDB is covered later in this paper.

Graph databases are, as the authors of the book "Seven Databases in Seven Weeks" [7] consider, one of the less commonly used database styles. Graph databases are very good at dealing with highly interconnected data. Their structure consists of nodes and the relationships between those nodes. Both nodes and relationships can have properties, like key-value pairs, that store data. The most important aspect of the graph databases' design, describe Eric Redmond and Jim R. Wilson, is that it uses the relationships to traverse through the nodes.

One of the most popular examples of graph databases is nowadays Neo4j [8].

2.3 PostgreSQL

The name of PostgreSQL [9] comes from a series of changes and derivations. The first name of the database was "Interactive Graphics and Retrieval System", or shortly "Ingres" and it was given in the early 1970s when the project was first developed at Berkeley, [7]. After that, in 1980s when an improved version of the database was developed, it became "Postgres", an abbreviated form of "post-Ingres". In 1996 it was renamed to "PostgreSQL" to suggest the database's support for the SQL language and has remained so ever since.

As described by its creators on http://www.postgresql.org [9], PostgreSQL is a powerful, open source object-relational database system that has more than fifteen years of active development. Its architecture has earned PostgreSQL a strong reputation for reliability, data integrity and correctness. It is supported by major operating systems like UNIX (Mac OS X, Solaris, AIX, BSD, HP-UX, SGI IRIX, Tru64), Linux and Windows. It is fully ACID (Atomicity, Consistency, Isolation and Durability) compliant and has full support for joins, foreign keys, triggers, views and stored procedures. It also supports storage of binary large objects, including sounds, pictures and video and has native programming interfaces for C/C++, Java, .Net, Ruby, Python, Perl and others.

Another important aspect about PostgreSQL is that it is highly scalable not only in the sheer quantity of data it can handle, but also in the number of users it can accommodate. Its creators claim that there are active PostgreSQL systems in production environments that manage in excess of 4 terabytes of data.

PostgreSQL also has built-in Unicode support, sequences, table inheritance, and subselects, point in time recovery, table spaces, asynchronous replication, nested transactions (savepoints), online/hot backups, a sophisticated query

planner/optimizer, and write ahead logging for fault tolerance.
All of these characteristics have contributed to the rise of PostgreSQL and made it one of the most appreciated NoSQL databases. But most of its popularity comes from PostgreSQL's most important feature: it's adherence with the SQL standard. Below are described a series of examples of how SQL works in PostgreSQL.
Creating a table consists of giving it a name and a list of columns with types and constraint information. The constraint information is optional. Each table should also declare a unique identifier for one of its columns, called Primary Key. In the following example the tabled is called countries and the SQL code looks like this [7]:

```
CREATE TABLE countries (
country_code char(2) PRIMARY KEY,
country_name text UNIQUE
);
```

Figure 1. Create Table example

In the above code example the table countries receives two constraints: Primary Key for *country_code* column and Unique for *country_name* column. This means that we are not allowed to have duplicate rows for none of the columns. We can populate the previously created table by inserting a few rows using the following SQL command:

```
INSERT INTO countries
(country_code, country_name)
VALUES ('us','United States'),
('mx','Mexico'),
('au','Australia'),
('gb','United Kingdom'),
('de','Germany'),
('ll','Loompaland');
```

Figure 2. Insert example

After this we can validate that the proper rows were inserted by reading them using the following SQL statement:

```
SELECT * FROM countries;
country_code | country_name
--------------+---------------
us | United States
mx | Mexico
au | Australia
gb | United Kingdom
de | Germany
ll | Loompaland
(6 rows)
```

Figure 3. Select example

An example of how to remove a record from the database is as follows:

```
DELETE FROM countries WHERE
country_code = 'll';
```

Figure 4. Delete example

2.4 HBase

HBase [10] is a column-oriented database that is highly scalable and uses strictly consistent reads and writes [7]. It was modeled based on BigTable, a high performance, propriatary database that was developed by Google and described in in the 2006 white paper "Bigtable: A Distributed Storage System for Structured Data", [11]. Nowadays, HBase has become a top-level Apache project, a distributed, scalable, big data store that provides Bigtable-like capabilities on top of Hadoop and HDFS.
An important aspect is that nowadays HBase is actively used and developed by a number of high-profile companies for their big data problems such as Facebook, eBay, Meetup, Ning, Yahoo! and many others. Some of the database's key features are [10]:

- strongly consistent reads/writes: HBase is not an "eventually consistent" DataStore. This makes it very suitable for tasks such as high-speed counter aggregation;
- automatic sharding: HBase tables are distributed on the cluster via regions, and regions are automatically split and re-distributed as your data grows;
- automatic RegionServer failover;
- Hadoop/HDFS Integration: HBase supports HDFS out of the box as its distributed file system.
- MapReduce: HBase supports massively parallelized processing via

ISSN 2067 – 4074

MapReduce for using HBase as both source and sink;

- Java Client API: HBase supports an easy to use Java API for programmatic access;
- Thrift/REST API: HBase also supports Thrift and REST for non-Java front-ends;
- Block Cache and Bloom Filters: HBase supports a Block Cache and Bloom Filters for high volume query optimization;
- Operational Management: HBase provides build-in web-pages for operational insight as well as JMX metrics.

Its developers also point out that HBase is really more a "Data Store" than a "Data Base" and that is not suitable for any type of big data problem one might have. HBase lacks many of the features one could find in an RDBMS, such as typed columns, secondary indexes, triggers, and advanced query languages, etc. and that is why it is better suited when dealing with hundreds of millions or billions of rows. It is also recommended as best practice to not use anything less than 5 DataNodes, due to things such as HDFS blocks replication which has a default of 3, plus a NameNode.

HBase tables consists of rows, keys, column families, columns, and values. Rows are identified by keys, column families have names, and in a column family there can be multiple columns, each one of them with an identifier. The combination of row key and column name (including both family and qualifier) creates an address for locating data values.

Although it doesn't have support for SQL, HBase comes with a JRuby-based command-line program, called HBase Shell, which enables one to interact with HBase, add and remove tables, alter table schema, add or delete data and other things. One can fire up HBase Shell by running the following command in the terminal (first the HBase system has to be installed and running):

```
${HBASE_HOME}/bin/hbase shell
```

Once the shell is up and running there several easy commands that can be used to create and alter a table, insert, update, delete and select data. In the following lines some of these commands are presented. The examples were taken from the book "Seven Databases in Seven Weeks".

Here is an example of how to create a table called *test* with a single column family called text:

```
Hbase> create 'test, 'text'
```

Figure 5. Create command example in HBase

To add data to an HBase table, one can use the *put* command. The following command inserts a new row into the *test* table with the key 'Home', adding 'Welcome to HBase!' to the column called 'text:'.

```
hbase> put 'test', 'Home', 'text:',
       'Welcome to HBase!'
```

Figure 6. Put command example in HBase

One can query data from an HBase table using get command:

```
hbase> get 'test', 'Home', 'text:'
```

Figure 7. Get HBase

The output of this command will look something like:

```
COLUMN CELL Text:
timestamp=1295774833456
value=Welcome to HBase!
1 row(s) in 0.0570 seconds
```

Figure 8. Get Output HBase

In order to alter a table in HBase, one has first to put it offline, meaning to disable it, This can be done using the following command:

```
hbase> disable 'test'
```

Figure 9 Disable Table HBase

The one can modify column family characteristics using the alter command:

```
hbase> alter 'test',
{ NAME => 'text', VERSIONS =>
hbase* org.apache.hadoop.
hbase.HConstants::ALL_VERSIONS }
```

Figure 10. Alter HBase

In the above command HBase is instructed to alter the text column family's Versions attribute.
After the table has been altered, one can put the table back online, meaning to enable it. In order to do this the following command can be used:

```
hbase> enable 'test'
```

Figure 11. Enable HBase

2.5 MongoDB

As we find out from [7], the name of the data base comes from "Hu(mongo)us" in order to suggest that MongoDB is an extremely large database. In the authors' opinion, MongoDB is the rising star in the NoSQL world. It was publicly released in 2009 and it was designed as a scalable database with performance and easy data access as core design goals.
MongoDB is a document-oriented database, as Eric Redmond and Jim R. Wilson describe in their book, that allows data to persist in a nested state, and, most importantly, it can query that nested data in an ad hoc manner. It enforces no schema, meaning that documents can optionally contain fields or types that no other document in the collection contains. It is a JSON document database, though, technically, data is stored in a binary form of JSON known as BSON.
The two authors [7] believe that the most important feature of MongoDB is the fact that it manages to blend the powerful query ability of a relational database and the distributed nature of other data stores like Riak or Hbase. A Mongo document can be likened to a relational table row without a schema, whose values can nest to an arbitrary depth.
Further authors [7] note that MongoDB is an excellent choice for web projects that require large-scale data storage, but have very little budget to acquire big hardware. Because of its lack of structure schema MongoDB can change and grow along with the data model. The MongoDB is recommended for someone who is at the beginning with a web application but dreams of enormity or for someone who has already grown large and needs to scale servers horizontally.
An important fact that one should also know about MongoDB is that, although it is incredibly flexible it should never be regarded as a toy. In fact there are some huge production MongoDB deployments out there such as the CERN Institute in Geneva (the European Organization for Nuclear Research) where it is used to collect data from the LHD (Large Hadron Collider).
Also, MongoDB is described by the company behind it, 10gen on their website as being the leading NoSQL database.
Although it does not offer support for SQL, MongoDB offers some very easy to use and intuitive commands to interact with the database. Below are some examples of such commands.
Before beginning to interact with database, one should create a folder in which the MongoDB data will be stored. This is a requirement of the system in order to prevent typos.
In order to create a new database, in this case called *student* on can run the following command in the terminal:

```
$ mongo student
```

Figure 12. Create a database in MongoDB

MongoDB's equivalent to tables are collections. The following code creates/inserts a *towns* collection:

```
> db.towns.insert({
name: "New York",
population: 22200000,
last_census: ISODate("2009-07-31"),
famous_for:
[ "statue of liberty", "food" ],
mayor : {
name : "Michael Bloomberg",
party : "I"
}
})
```

Figure 13. Insert Mongo

With the show collections command, one can verify if a collection exists.

```
> show collections
```

Figure 14 Show Mongo DB

ISSN 2067 – 4074

The output of this command will be:

```
                                    sytem
.indexes
                                    towns
```

Figure 15. Show Output Mongo

The contents of a collection can be listed using the find() command:

```
> db.towns.find()
```

The output of the command is:

```
{
"_id" : ObjectId("4d0ad975bb30773266f39f
"name" : "New York",
"population": 22200000,
"last_census": "Fri Jul
 31 2009 00:00:00 GMT-0700 (PDT)",
"famous_for" : [ "statue of liberty", "f
"mayor" : { "name" : "Michael Bloomberg"
 "party" : "I" }
}
```

Figure 16. Find Output Mongo

What is interesting about MongoDB is that its native tongue is JavaScript. Commands are actually just JavaScript functions. For example the following command will list available functions related to the given object:

```
> db.help()
> db.towns.help()
```

If one wants to inspect the source code for a function, one can call it without parameters:

```
> db.towns.insert.
```

3. Testing tools - Yahoo! Cloud Serving Benchmark

Nowadays more and more platforms for big data and NoSQL databases are developed by both large and small companies, and yet there are very little means to test these big data solutions to see which one best fits a certain scenario. This paper seeks to solve this issue, testing several NoSQL databases and then comparing the results to find out which one of these databases are best to be used in different conditions. In order to do that, this paper will cover the output of three types of tests: performance tests, high availability tests (or stress tests) and security tests.

In this section we give an overview of testing tools used to perform the three types of tests mentioned above.

With the many new serving databases available including HBase, MongoDB, Redis and many more, it can be difficult for one to decide which system is right for his application, partially because the features differ between systems, and partially because there is not an easy way to compare the performance of one system versus another.

The goal of the Yahoo! Cloud Serving Benchmark (YCSB) [11] project is to develop a framework and common set of workloads for evaluating the performance of different "key-value" and "cloud" serving stores.

The developers of the YCSB platform designed it to be an open-source benchmark tool that is suitable for different classes of applications. And this is the reason why the key feature of YCSB platform is its extendibility. It supports easy definition of new workloads and also makes it easy to benchmark other systems.

The authors of the paper "Benchmarking Cloud Serving Systems with YCSB" [12] believe that the biggest difference between the wide varieties of big data solutions comes from the different data models that these applications use. They give as example the column-oriented BigTable that is used in Cassandra and HBase in opposition with the document model used in CouchDB or the simple hash-table model of Voldemort. But the authors also note that not the data model is the problem when it comes to testing big data applications. The data models can easily be documented and qualitatively compared. The problem consists in comparing the performance of these different types of systems. The authors of the paper further explain that some systems decided to optimize the writing performance using sequential Input/Output, while other systems have been optimized to do quicker reads. The authors also mention that replication,

data partitioning, consistency and other features alike have an impact on the performance of a system.
The creators of the YCSB framework point out that it is very hard to make an apples-to-apples comparison because of the high number of different systems based on different types of workloads. Usually, in order to test the performance of a given system people have to manually download, install and evaluate multiple platforms. The creators of YCSB believe that this process is time consuming and expensive.
In other words, the development of the YCSB framework came as a necessity to create a standard benchmark and a benchmarking tool that could assist one in the analysis of different cloud systems and other similar classes of applications.
In the way it is described in [13], the Yahoo! Cloud Serving Benchmark platform was created to evaluate two types of benchmarking tiers: performance and scalability. The center of attention for the performance tier is the latency of requests when the database is under load. The authors of the paper describe that the latency in a serving system is very important, since there usually is a user at the end that waits for an answer. However the authors point out that on any given hardware setup the latency of individual requests increases proportionally with the load of the system. This means that the designers of an application must decide on an appropriate hardware configuration that would be able to preserve an acceptable latency.
In order to conduct the performance benchmark tier, the developers of YCSB created a workload generator that serves two purposes. Firstly, it defines the data set and loads it into the database and secondly, it executes operation against the dataset while it measures the performance.
The scaling tier focuses on the performance impact when more machines are added to the system. Two metrics were used by the authors of the paper "Benchmarking Cloud Serving Systems with YCSB" [13] to measure this tier: scale up and elastic speedup.
The scale up metric seeks to answer the question: „How does the data perform as the number of machines increases?". The main focus is to find out whether the performance (for example latency) of the database remains the same when the number of servers, amount of data and offered throughput scale proportionally.
The elastic speedup tries to find out what happens with the performance of a system when a workload is running on a given number of servers and then one or more servers are added. If the system offers good elasticity, its performance should improve once the servers added. And the system should also have in this case a very short or even non-existent period of discontinuance, while it reconfigures itself to use the new server or servers.
The developers of the YCSB created a Core Package for their framework that consists in a series of workloads which are destined to evaluate different aspects of a system's performance. Each workload is designed to execute a series of read/write operations, data sizes, request distributions, and so on. The results of these operations can then be used to analyze the system's performance at one particular point. The YCSB Core Package offers only a small series of evaluation tools and if someone wishes to perform a much more profound analysis of a system, he can easily create his own set of workload parameters or even write new Java code. The YCSB application has been designed to be easily extensible and developers are also offered the opportunity to create new sets of workloads for different styles of databases that are not covered in the core package.
When developing the core package, the programmers had to examine which types of workloads made a system to perform well or poorly. For example, some databases may be highly optimized for reads and perform poorly on writes. Other databases may perform well on inserts, but they are no optimized on updates. The workloads chosen by the developers of the YCSB for the core package explore the above mentioned tradeoffs directly.
The developers of YCSB created a workload client that can generate the

ISSN 2067 – 4074

workloads automatically. The purpose of this client is to make many random choices when generating load, such as: which operation to perform (insert, update, read or scan), which record to read or write, how many records to scan, and so on. All these decisions are governed by random distributions. YCSB has several built-in distributions: uniform, zipfian, latest, multinomial.
Items can also be specified. For example, we might assign a probability of 0.95 to the Read operation, a probability of 0.05 to the Update operation, and a probability of 0 to Scan and Insert. The result would be a read-heavy workload.
As mentioned in the paragraphs above the architecture of the YCSB platform was created in such a way that it can be easily extensible. Its developers even mention in [13] that one of their primary goals when creating the application was to make it extensible. It was one of their motivations to make it easy for developers to benchmark the increasing variety of cloud serving or other similar systems. An important extensible module from the YCSB project is the Database Interface Layer, which is responsible with the translation of simple requests, such as reads, from the client threads into calls against the database.
In order to be able to benchmark a new type of database, all one has to do is to add a new class to the Database Backend of the YCSB application and in this class to implement the methods that correspond to the specific CRUD, Create, Read, Update and Delete, operations of that database. The methods that have to be implemented are: read(), insert(), update(), delete(), scan().
Below is an example of loading the workload A with YCSB on HBase:

```
hduser@master:/YCSB/ycsb-0.1.4$
bin/ycsb load hbase -P
workloads/workloada -p
columnfamily=f1 -p
recordcount=1000000 -p
threadcount=4 -s | tee
-a load.dat
```

Figure 17. Workload on a HBase database

And the output is:

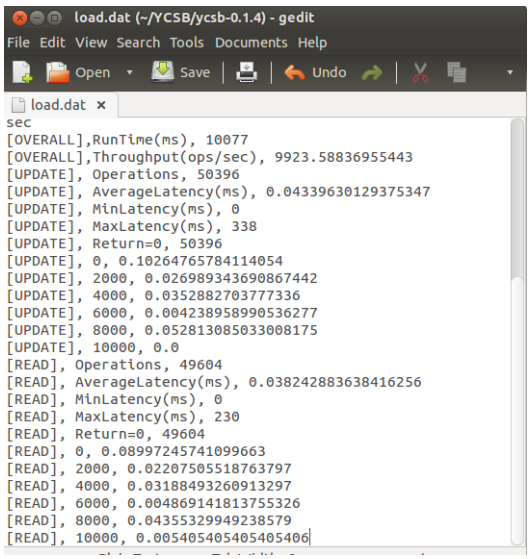

Figure 18 Workload output

4. Conclusion

Nowadays an open-source version of the YCSB platform is available for download from https://github.com/brianfrankcooper/YCSB. It offers support for many of the databases that this paper proposes to analyze such as: HBase, Cassandra [14], MongoDB, Redis. Given also the fact that the framework is very easy to customize in order to perform workloads on new types of databases, it could be a good testing tool for one of the problem debated in this paper: performance test.
The YCSB was not designed to run stress tests (high availability tests) on systems but it offers some support in the sense that one can start a workload with the YCSB and then kill the server while the workload is running and then observe the errors and performance impact. However its developers point out in [13] that there are many different types of failures that can occur on one system besides the crash of a server and that a proper availability benchmark should cause or simulate a variety of faults and examine their impact. High availability testing has been noted in the paper as future work. This being mentioned, the stress testing that can be performed using the YCSB current version are enough to cover the purpose of this thesis, so the YCSB

framework could also be used to do availability tests.

Acknowledgement

Parts of this research have been published in the Proceedings of the 7th International Conference on Security for Information Technology and Communications, SECITC 2013.

References

[1] Edd Dumbill, Making Sense of Big Data, *Big Data*. March 2013, 1(1): 1-2. doi:10.1089/big.2012.1503

[2] Dragos Iulian COGEAN, Marin FOTACHE, Valerica GREAVU-SERBAN, *NoSQL in Higher Education. A Case Study*, International Conference on Informatics in Economy, Proceedings of the 12th International Conference on INFORMATICS in ECONOMY (IE 2013), Education, Research & Business Technologies, Bucharest, Romania, April 25-28, 2013, pp. 352-360

[3] Catalin Boja, Adrina Pocovnicu, Lorena Batagan, Distributed Parallel Architecture for "Big Data", *Informatica Economica Journal*, Vol. 16 No. 2, 2012, pp. 116 – 127, ISSN 1453-1305

[4] James Manyika, Michael Chui, Brad Brown, Jacques Bughin, Richard Dobbs, Charles Roxburgh,Angela Hung Byers, *Big data: The next frontier for innovation, competition, and productivity*, 2012

[5] Christof Strauch, *NoSQL Databases*, [online] http://www.christof-strauch.de/nosqldbs.pdf

[6] Martin Fowler, NoSQL Databases, [online] http://martinfowler.com

[7] Eric Redmond and Jim R. Wilson, *Seven Databases in Seven Weeks*, The Pragmatic Bookshelf, 2012

[8] Wiki, Neo4j, [online] http://en.wikipedia.org/wiki/Neo4j

[9] PostgreSQL, [online] http://www.postgresql.org

[10] Apache Software Foundation, The Apache HBase™ Reference Guide, [online] http://hbase.apache.org/book/

[11] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, Robert E. Gruber, Bigtable: A Distributed Storage System for Structured Data, Proceedings of the OSDI '06 Conference, pp. 205-218

[12] Yahoo, Yahoo Cloud Serving Benchmark, [online] http://labs.yahoo.com/news/yahoo-cloud-serving-benchmark/

[13] Brian F. Cooper, Adam Silberstein, Erwin Tam, Raghu Ramakrishnan, Russell Sears, Benchmarking Cloud Serving Systems with YCSB, , *ACM Symposium on Cloud Computing*, 2012, [online] http://ipij.aei.polsl.pl/django-media/lecture_file/ycsb.pdf

[14] Cassandra Wiki, [online] http://wiki.apache.org/cassandra/