The Development of a Benchmark Tool for NoSQL Databases

Ion LUNGU, Bogdan George TUDORICA
University of Economic Studies, Bucharest, Romania
Petroleum-Gas University, Ploiesti, Romania
ion.lungu@ie.ase.ro, tudorica_bogdan@yahoo.com

The aim of this article is to describe a proposed benchmark methodology and software application targeted at measuring the performance of both SQL and NoSQL databases. These represent the results obtained during PhD research (being actually a part of a larger application intended for NoSQL database management). A reason for aiming at this particular subject is the complete lack of benchmarking tools for NoSQL databases, except for YCBS [1] and a benchmark tool made specifically to compare Redis to RavenDB. While there are several well-known benchmarking systems for classical relational databases (starting with the canon TPC-C, TPC-E and TPC-H), on the other side of databases world such tools are mostly missing and seriously needed.

Keywords: *NoSQL database, testing, benchmark*

1 Introduction

One of the tools needed by a database administrator (and not only by this category) is a benchmarking tool, a tool which, if used well can give details on the machine performance, on the DBMS performance and (in some cases) on the optimization level (or lack of) of the queries made over that DBMS.

In the last several years we've seen the advent of a new type of databases, the NoSQL ones [2]. The NoSQL databases are, in a certain point of view, the children of the Web 2.0 era (although the concept they are based on is a much older one). To eliminate any confusions, during this paper the term NoSQL is not used as the opposite of the SQL relational database but as a general label for any BASE database system (Basic Availability, Soft state, Eventual consistency).

We should also remark that while in the relational databases faction a certain unification was achieved (while only on the general terms and concepts), in the NoSQL faction almost all solutions are alien to each other, using different structures, concepts and technologies (a taxonomy given in [3] is containing five categories only for the "core" NoSQL systems). As such, any tool aimed at the NoSQL systems faces the difficulty of having to "speak" several "languages". At this moment the only commercial tool capable (to a certain extent) of such a feat is Toad for Cloud Databases (able to interoperate with Amazon SimpleDB, Microsoft Azure Table Services, Microsoft SQL Azure, Apache Hbase, Apache Cassandra, Apache Hadoop HIVE, MongoDB and any ODBC-enabled relational database). Even tools aimed at a single NoSQL are scarce and usually far from functional maturity. As such, not only the benchmarking apps are not available but any other kind of administrative ones are lacking too.

2. Tools used for this project

This project started as and administration software meant only for MongoDB. We chose MongoDB for a multitude of reasons exposed in [4]. Even before starting working

on this administration application we worked with MongoDB for some other applications such as a web page parsing tool written in PHP (see [5]). MongoDB being the database of choice, there are plenty of programming languages usable for developing an application over MongoDB (C, C++, C# & .NET, ColdFusion, Erlang, Factor, Java, Javascript, PHP, Python, Ruby, and Perl). For this case, our selection was Visual C#, for ease of use, nice interoperability with the Microsoft Windows systems and better application performance than say, a PHP or Java software (for this particular reason, C and maybe C++ were the best possible choices but such a decision would have negated the other advantages). We used the 2008 version of the Visual C# environment as that was the most used at the moment we started the research (2010). On the DBMS side, at this moment we are using the 2.0.7-rc0-pre version of MongoDB (although there are some newer versions).

Besides the DBMS and the development environment we are using the MongoDB CSharp driver version 1.4.2.4500-109-g8ac35a5 for connectivity between the MongoDB and Visual C#. Later, during the time we added benchmarking functionality to the application, we used MySQL Connector .Net, version 6.1.6 for connectivity between MySQL and Visual C#, MySQL Community version 5.6.12.1 as a second DBMS and finally MSChart .Net 3.5 add-on and MSChart Visual Studio 2008 add-on for charting.

3. Working methodology

For the benchmarking operations we imagined the following three scenarios (inspired by [6], [7], [8] and [9]):

1. The tested databases are used for OLTP operations. This case presumes the following conditions: the number of read operations is of the same magnitude with the number of write operations; the data from each atomic transaction / row operation has a size in the range of tens of kilobytes. To be more specific, for our application we chosen the following conditions: number of reads = number of writes; each row operations reads or writes a standard record having the following content: three 32-bit integer fields (acting as an id and two other integer fields), 3 float fields, 3 text fields of 100 chars each and 1 small blob field (corresponding to a small document or image file stored in the database). For the blob field we chose to make it of 32,438 bytes. The later size was chosen to make for a total size of the record of 32,768 bytes, permitting fast (even when done mentally) computations of the total transaction data size for various numbers of operations. As a consequence, 32 records mean 1 MB of data, 160 records mean 5 MB of data, 320 records mean 10 MB of data, 1600 records mean 50 MB of data, 3200 records mean 100 MB of data, 16000 records mean 500 MB of data and so on.

2. The tested databases are used for Web 2.0 operations. This case presumes the following conditions: the number of read operations is two to three orders of magnitude higher than the number of write operations (e.g. for YouTube, as per the latest statistics, the read to write size ratio is somewhere around 1389:1); the data from each atomic transaction / row operation has a size in the range of Megabytes (e.g. for YouTube is quite large, the average atomic transaction size is, depending on resolution and quality, of 20-150 Megabytes, but not all Web 2.0 services are data intensive). For our application we chosen the following conditions: number of reads = 500 * number of writes; each row operations reads or writes a standard record having the following content: one 32-bit integer fields (acting as an id), 5 text fields of 500 chars each and 1 large blob field (corresponding to media content stored in the database). For

the blob field we gave it the size of 5,241,346 bytes. Again the blob size was chosen to make for a round size of the record (5,242,880 bytes = 5 MB), permitting fast computations of the total transaction data size for various numbers of operations.

3. The tested databases are used for OLAP operations. This case presumes the following conditions: the number of read operations is one to two orders of magnitude higher than the number of write operations; the data from each atomic transaction / row operation has a size in the range of fractions kilobytes. For our application we chosen the following conditions: number of reads = 100 * number of writes; each row operations reads or writes a standard record having the following content: ten 32-bit integer fields, ten float fields and 7 text fields of 132 chars each (again for the sake of a round record size – 1024 bytes = 1 kilobyte, permitting fast computations of the total transaction data size for various numbers of operations).

3. Preparations before testing

As the operating system we worked our application over is Microsoft Windows XP, there are a few measures to take to compensate for the multi-core, multi-tasking, multi-threading, time-sharing character of such a system.
First we took care to make the maximum amount of computing resources available for the application while preventing (as much as possible) other applications to interfere with the testing:

```
using System.Diagnostics;
using System.Threading;
…
Process.GetCurrentProcess().Proce
ssorAffinity = new IntPtr(2);
//make the process use the second
core or processor which is
usually less loaded than the
first
Process.GetCurrentProcess().Priority
Class = ProcessPriorityClass.High;
//raise the priority of the process
Thread.CurrentThread.Priority =
ThreadPriority.Highest;
//raise the priority of the thread
```

Second, before starting the test, we took care to “warm up” the CPU cache and pipelines:

```
stopwatch.Reset();
stopwatch.Start();
while (stopwatch.ElapsedMilliseconds
< 1500)
//A period of 1500 ms for CPU cache
and pipelines stabilization with a
randomly chosen operation in it.
      {
       i = (i + 1) % 10;
       }
stopwatch.Stop();
```

4. Data generation

In the design stage we hypothesized that the content of the transaction data may have some influence over the transaction time so we decided not to use any pre-stored data but to generate it randomly instead at every benchmark run, in quantities and structures depending on the type and size of the run.
Two distinct random generation methods were used for numbers and respectively for strings.
For various format of numbers we used the Random class. To make sure that no data sequence is repeated between two runs, we took care to seed the random number generator with a different value (given by a small trick – we used the Guid class as a seed generator):

```
Random rndNum = new
Random(int.Parse(Guid.NewGuid().ToSt
ring().Substring(0, 8),
System.Globalization.NumberStyles.He
xNumber));
```

For integer field content we used directly the Random generator such us:

```
id = rndNum.Next(0, 4000000);
```

For float field content we used divisions of random values such us:

```
val3 = (float)rndNum.Next(-
2000000, 2000000) /
(float)rndNum.Next(0, 4000000);
```

For BLOB fields, we randomly generated ASCII codes which were later converted to chars / bytes. We also took the precaution to only generate chars from a small portion of the ASCII table to avoid a possible later invalidation of the queries containing the data caused by the apparition of a special character:

```
for (j = 0; j < 32438; j++)
      blob[j] =
char.ConvertFromUtf32
      (rndNum.Next(97, 122))[0];
```

On the other hand, for strings we used a different approach (again based on a programming trick) which seemed to be a bit faster that the char by char direct generation:

```
Txt1 = "";
for (j = 0; j < 10; j++)
      {
      string piece =
Path.GetRandomFileName();
      piece = piece.Substring(0,
10);
      txt1 = txt1 + piece;
      }
```

Finally, it is worth to be mentioned the fact that the data generation is highly time consuming (as we will see at the end of the fifth section of this paper) and as such, we took the measure to clock it separately than the rest of the test.

5. The benchmarking

The benchmarking consists of cycles of "record" write operations followed by cycles of "record" read operations (the concept of record has actually no meaning in the NoSQL world; the closest concepts are the ones of document or the one of key-value pair; see [3], [4], [10] and [11]). The number of cycles and the content of the "records" depend on the type of the intended benchmark (see section 3). The connections to the DBMS are made in the usual ways.
Note: at this moment the benchmark application is capable of working only over MongoDB and MySQL but we intend for future developments to add Oracle database and MS SQL capabilities on the relational DBMS side and Redis and CouchDB on the NoSQL side.
The basic write operations are looking like the following:

- For MongoDB (repeated for every "field"):

```
var element =
BsonElement.Create("id",
BsonString.Create(id.ToString()));
document.Add(element);
```

- For MySQL (one transaction for the entire record):

```
string mysql_query = "INSERT INTO
oltpbenchmark_table (id, val1, val2,
val3, val4, val5, val6, den1, den2,
den3) VALUES(" + id.ToString() + ",
" + val1.ToString() + ", " +
val2.ToString() + ", " +
val3.ToString() + ", " +
val4.ToString() + ", " +
val5.ToString() + ", \"" + blob_s +
"\", \"" + txt1 + "\", \"" + txt2 +
"\", \"" + txt3 + "\")";
MySqlCommand mysql_cmd = new
MySqlCommand(mysql_query,
mysql_connection);
mysql_cmd.ExecuteNonQuery();
```

The basic read operations are looking like the following:

- For MongoDB:

```
foreach (var document in cursor)
      {
      id=document.GetElement(1).Valu
e.ToInt32();
      …
```

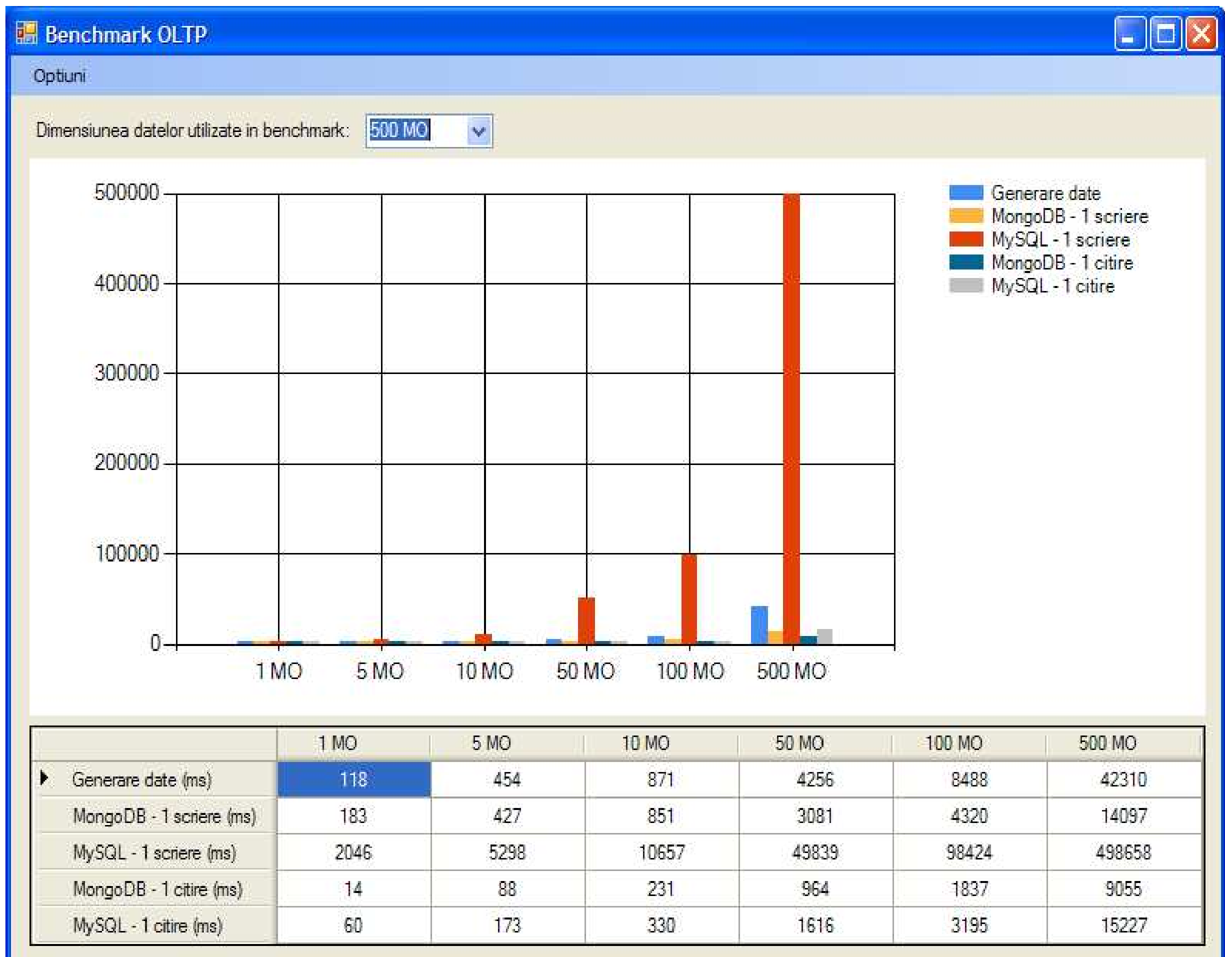
- For MySQL:

```
string mysql_query2 = "SELECT *
FROM oltpbenchmark_table";
MySqlCommand mysql_cmd2 = new
MySqlCommand(mysql_query2,
mysql_connection);
MySqlDataReader mysql_dataReader
= mysql_cmd2.ExecuteReader();
while (mysql_dataReader.Read())
      {
       id =
mysql_dataReader.GetInt32(0);
      …
```

At the corresponding moments during the operations, several Stopwatch class objects are started, stopped and reset in accordance with their purposes (clocking the times for data generation, the write operations for MongoDB, the write operations for MySQL, the read operations for MongoDB, the read operations for MySQL). We chose to use the Stopwatch class for clocking the operations because it gives for a pretty accurate measurement of time.

Finally the results (given in milliseconds) are stored in a dataGridView and represented on a Chart for ease of lecture and interpretation.

The product of an OLTP benchmark run can be seen in Fig.1.



	1 MO	5 MO	10 MO	50 MO	100 MO	500 MO
Generare date (ms)	118	454	871	4256	8488	42310
MongoDB - 1 scriere (ms)	183	427	851	3081	4320	14097
MySQL - 1 scriere (ms)	2046	5298	10657	49839	98424	498658
MongoDB - 1 citire (ms)	14	88	231	964	1837	9055
MySQL - 1 citire (ms)	60	173	330	1616	3195	15227

Fig. 1. The results of an OLTP benchmark run based on a 500 MB data chunk, with clocking at 1 MB, 5 MB, 10 MB, 50 MB, 100 MB and 500 MB

In the Fig.1, the timings are given for data generation (first row of timings), MongoDB write operations (the second row), MySQL write operations (the third row), MongoDB read operations (the fourth row) and MySQL read operations (the fifth row). The conclusions of a single run of the test are the following:

- The MySQL write operations require much higher times than all other types of operations (going as far as 20 times bigger) because they are the only ones which involve direct disk operations. All the other operations are more or less memory based (the data generation is made in memory, MongoDB is based on a RAM cache technology, also the MySQL reads are cached).
- Even when taking into consideration only the read operations timings, MongoDB performance is better than the one of MySQL (which is to be expected, given the fact that all major NoSQL products are lighter, less complex and, as a consequence, they are supposed to be faster than their relational counterparts; see [1] and [3]).
- The data generation consumes actually 2 to 5 times more time than the actual read or write operations (except for the MySQL writes). At the present moment we are considering this an issue and searching for an alternative approach.
- The read operations are consistently faster than the write operations for both DBMS products, which is again to be expected.

6. Conclusions

This paper presented an approach for a obtaining a benchmarking tool aimed at measuring the performance of various DBMS, be they relational or NoSQL.

The used working methodology is far from perfect as it doesn't take into account the expected statistical fluctuations. From this point of view, a complete approach would consist of a large enough number of runs, with the extreme results disregarded and the other results taken into account on average.

References

[1] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears, *Benchmarking cloud serving systems with YCSB*, Proceedings of the 1st ACM symposium on Cloud computing, ser. SoCC '10. New York, NY, USA: ACM, 2010, pp. 143–154. ISBN: 978-1-4503-0036-0, doi: 10.1145/1807128.1807152

[2] Avrilia Floratou, Nikhil Teletia, David J. DeWitt, Jignesh M. Patel, Donghui Zhang, *Can the elephants handle the NoSQL onslaught?*, Proceedings of the VLDB Endowment, VLDB Endowment Hompage archive, Volume 5 Issue 12, August 2012, Pages 1712-1723

[3] Bogdan Tudorica, Bucur Cristian - A comparison between several NoSQL databases with comments and notes, The proceedings of „2011 - Networking in Education and Research" IEEE International Conference, June 23, 2011 – June 25, 2011, Alexandru Ioan Cuza University from Iasi.

[4] Bogdan Tudorica - Challenges for the NoSQL systems: Directions for Further Research and Development, The International Journal of Sustainable Economies Management (IJSEM), Volume 2: Issue 1 (2013), DOI: 10.4018/IJSEM.2013010106, ISSN: 2160-9659, EISSN: 2160-9667.

[5] Bucur Cristian, Bogdan Tudorica - A Research on Retrieving and Parsing of Multiple Web Pages for Storing Them in Large Databases, The Proceedings of the 19th International Economic Conference - IECS 2012, The Persistence of the Economic Crises: Causes, Implications, Solutions, 15 June, 2012, Sibiu, Romania.

[6] Jim Gray, *Benchmark Handbook: For Database and Transaction Processing Systems*, Morgan Kaufmann Publishers Inc. San Francisco, CA, USA 1992, ISBN:1558601597

[7] Plale, B., Jacobs, C., Jensen, S., Ying Liu, Moad, C., Parab, R., Vaidya, P., *Understanding Grid resource information management through a synthetic database benchmark / workload*, IEEE International Symposium on Cluster Computing and the Grid, 2004 (CCGrid 2004), 19-22 April 2004, Page(s): 277 – 284, Print ISBN: 0-7803-8430-X, INSPEC Accession Number: 8198955, doi: 10.1109/CCGrid.2004.1336578

[8] Rim Moussa, *TPC-H Benchmark Analytics Scenarios and Performances on Hadoop Data Clouds*, Networked Digital Technologies Communications in Computer and Information Science Volume 293, 2012, pages 220-234

[9] Yingjie Shi, Xiaofeng Meng, Jing Zhao, Xiangmei Hu, Bingbing Liu, Haiping Wang, *Benchmarking cloud-based data management systems*, CloudDB '10 Proceedings of the second international workshop on Cloud data management, pages 47-54, ACM New York, NY, USA 2010, ISBN: 978-1-4503-0380-4, doi: 10.1145/1871929.1871938

[10] Ashok Joshi, Sam Haradhvala, Charles Lamb, *Oracle NoSQL Database - Scalable, Transactional Key-value Store*, IMMM 2012, The Second International Conference on Advances in Information Mining and Management, pages: 75-78, IARIA, 2012, ISBN: 978-1-61208-227-1, Venice, Italy, October 21, 2012 - October 26, 2012

[11] Rick Cattell, *Scalable SQL and NoSQL data stores*, ACM SIGMOD Record archive, Volume 39 Issue 4, December 2010, Pages 12-27, ACM New York, NY, USA, doi: 10.1145/1978915.1978919

Ion LUNGU is a Professor at the Economic Informatics Department at the Faculty of Cybernetics, Statistics and Economic Informatics from the Academy of Economic Studies of Bucharest. He has graduated the Faculty of Economic Cybernetics in 1974, holds a PhD diploma in Economics from 1983 and, starting with 1999 is a PhD coordinator in the field of Economic Informatics. He is the author of 22 books in the domain of economic informatics, 57 published articles (among which 2 articles ISI indexed) and 39 scientific papers published in conferences proceedings (among which 5 papers ISI indexed and 15 included in international databases). He participated (as director or as team member) in more than 20 research projects that have been financed from national research programs. He is a CNCSIS expert evaluator and member of the scientific board for the ISI indexed journal Economic Computation and Economic Cybernetics Studies and Research. He is also a member of INFOREC professional association and honorific member of Economic Independence academic association. In 2005 he founded the master program Databases for Business Support (classic and online), who's manager he is. His fields of interest include: Databases, Design of Economic Information Systems, Database Management Systems, Decision Support Systems, Executive Information Systems.

Bogdan George TUDORICA is a teaching assistant in the Modeling, Economic Analysis and Statistics department from the Petroleum-Gas University of Ploiesti, Romania. At this moment he is also a PhD student at the Bucharest University of Economic Studies, Romania. His field of study for the PhD thesis is the management of large data volumes.