Measuring Elasticity for Cloud Databases

Thibault Dory, Boris Mejías
Peter Van Roy
ICTEAM Institute
Univ. catholique de Louvain
dory.thibault@gmail.com, peter.vanroy@uclouvain.be,
boris.mejias@uclouvain.be

Nam-Luc Tran
Euranova R&D
Mont-Saint-Guibert, Belgium
namluc.tran@euranova.eu

***Abstract*—The rise of the Internet and the multiplication of data sources have multiplied the number of "Bigdata" storage problems. These data sets are not only very big but also tend to grow very fast, sometimes in a short period. Distributed databases that work well for such data sets need to be not only scalable but also elastic to ensure a fast response to growth in demand of computing power or storage. The goal of this article is to present measurement results that characterize the elasticity of three databases. We have chosen Cassandra, HBase, and mongoDB as three representative popular horizontally scalable NoSQL databases that are in production use. We have made measurements under realistic loads up to 48 nodes, using the Wikipedia database to create our dataset and using the Rackspace cloud infrastructure. We define precisely our methodology and we introduce a new dimensionless measure for elasticity to allow uniform comparisons of different databases at different scales. Our results show clearly that the technical choices taken by the databases have a strong impact on the way they react when new nodes are added to the clusters.**

***Keywords*-Cloud computing; key/value store; elasticity; NoSQL; Cassandra; mongoDB; HBase; Wikipedia.**

I. Introduction

Nowadays there are a lot of problems that require databases capable of storing huge quantities of unstructured data. The datasets are so big that they must be stored on several servers and, as new data are gathered and new users appear, it must be possible to extend the available storage and computing power. This can only be done by adding more resources into the cluster, e.g., adding servers. This addition is likely to have an impact on performance and therefore the goal of this paper is to present the definitions, methodology and results that are the outcome of our study of elasticity for a few chosen distributed databases. We also have defined a new dimensionless number to characterize the elasticity that ease the comparison between databases. The results are analyzed to explain the reason of some unexpected behaviors, but some stay unexplained for now.

This paper summarizes the results of a master's thesis [1]. We present first the detailed methodology and definitions, followed by the databases chosen, the measurement conditions, and the benchmark implementation. Finally, we present and analyze the measurement results.

II. State of the art

The Yahoo! Cloud Servicing Benchmark [2] is the most well known benchmarking framework for NoSQL databases. It was created by Yahoo!. It currently supports many different databases and it can be extended to use various kinds of workloads. The benchmark used for the measurements presented here could have been implemented on top of YCSB as a new workload but it has not been for various reasons. The first reason is simplicity: it seemed easier to implement its functionalities directly instead of extending the big and far more complex YCSB where it would not have been so easy to control all the parameters. The second reason is that we wanted to explore the best methodology for measuring elasticity without being tied to the assumptions of an existing tool.

III. Methodology

A. Definitions

1) Performance: The performance is characterized by the time needed to complete a given number of requests with a given level of parallelization. The chosen levels of parallelization and number of requests used during the measurements are explained in the step by step methodology. In all the measurements of this article, we perform requests in batches called *request sets*. This allows us to decrease variability and improve accuracy in measurement time.

2) Elasticity: The elasticity is a characterization of how a cluster reacts when new nodes are added or removed under load. It is defined by two properties. First, the time needed for the cluster to stabilize and second the impact on performance. To measure the time for stabilization, it is mandatory to characterize the stability of a cluster, and therefore a measure of the variation in performance is needed. The system can be defined as stable when the variations between request set times are equivalent to the variations between request set times for a system known to be stable. That is, a system in which there are no data being moved across the nodes and when all the nodes are up and serving requests. These variations are characterized by the *delta time*, which is the absolute value of the difference

Copyright (c) IARIA, 2011. ISBN: 978-1-61208-153-3

in time needed to complete a request set and the time needed to complete the previous request set. Concretely, for a given database, data set, request set, and infrastructure, the variability is characterized by the median value of the delta times and the system is said to be stable if the last X sets have a delta time smaller than the previously observed value. In this article we fix the value of X to 5, which gives satisfactory results for the measurements done.

We make the hypothesis that just after the bootstrap of the new nodes, the execution time will first increase and then decrease after an elapse of time. This is illustrated graphically in Figure 1 by the shape of the curve. In case the time needed for stabilization is very short, the average value and therefore the shape of the curve could be nearly unaffected by overhead related to elasticity, but at least the standard deviation will increase due to the additional work needed to move data to new nodes. It is important to take this standard deviation into account because highly variable latency is not acceptable. To characterize the elasticity in this article, we will take both the execution time and the standard deviation into account.

To characterize the elasticity with a single dimensionless number, we therefore propose the following formula:

$$Elasticity = \frac{A + B}{(Rt1 + Rt2)^2 * F} \quad (1)$$

Here A and B are the surface areas shown in Figure 1, where A is related to the execution time increase and B is related to the standard deviation, $Rt1$ is the average response time of one request for a given load before the bootstrapping of the new nodes, $Rt2$ is the average response time once the cluster has stabilized with the same load applied. Finally, F is the factor to suppress the dependency to the number of requests per node in the cluster. It is given by

$$F = \frac{Number\ of\ requests}{cluster\ size} \quad (2)$$

In all the measurements of this article, we assume that $N = M$, that is, we double the number of nodes.

The triangular area defined by the edges (Rt1,Rt2), (Bootstrap,Stable), and (Rt1,Stable) is not counted because even for perfect elasticity this triangle will exist as a performance ramp from level $Rt1$ to $Rt2$. The area $A + B$ is then purely due to elasticity and has a dimension of time squared. The value $Rt1 + Rt2$ are both inversely proportional to the average performance and have a dimension of time. The elasticity is therefore the ratio of the elastic overhead $A + B$ to the absolute performance $(Rt1 + Rt2)^2 * F$ and is a dimensionless number. The division by F removes the scaling factor of the size of the request set (e.g., the 10000 mentioned above).

B. Step by step methodology

Figure 3 illustrates the step by step methodology used during the tests. It is based on the following parameters : N the number of nodes, R the size of a request set and r the percentage of read requests. In practice, the methodology is defined by the following steps:

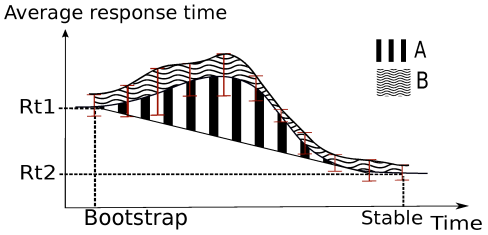

Figure 1. Surface areas used for the characterization of the elasticity

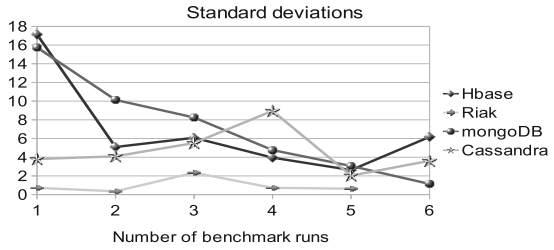

Figure 2. Observed standard deviations for 10000 requests with 80% reads

1) Start up with a cluster of $N = 6$ nodes and insert all the Wikipedia articles.
2) Start the elasticity test by performing request sets that each contain $R = 10000$ requests with $r = 80\%$ read requests and as many threads as there are nodes in the cluster when the elasticity test begins. The time for performing each request set is measured. (Therefore the initial request sets execute on 6 threads each serving about 1667 (≈10000/6) requests.) This measurement is repeated until the cluster is stable, i.e., we do enough measurements to be representative of the normal behavior of the cluster under the given load. We then compute the median of the delta times for the stable cluster. This gives the variability for a stable cluster.
3) Bootstrap new nodes to double the number of nodes in the cluster and continue until the cluster is stable again. During this operation, the time measurements continue. We assume the cluster is stable when the last 5 request sets have delta times less than the one measured for the stable cluster.
4) Double the data set size by inserting the Wikipedia articles as many times as needed but with unique IDs for each insert.
5) To continue the test for the next transition, jump to step (2) with a doubled number of requests and a doubled number of threads.

Copyright (c) IARIA, 2011. ISBN: 978-1-61208-153-3

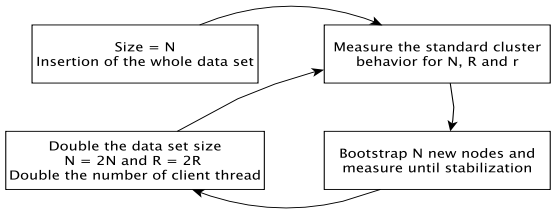

Figure 3. Step by step methodology

C. Justification of the methodology

One approach to characterize the variability is to use the standard deviation of request set times and a statistical test to compare the standard deviations. However, our experience shows that the standard deviation is too sensitive to normal cluster operations like compaction and disk pre-allocations. Figure 2 shows that the standard deviation can vary more than a factor of 4 on a stable cluster made of six 4GB Rackspace instances. This is why we use the delta time characterization instead. Because it is based only on the average values, it tends to smooth these transient variations. The median of all the observed delta times is used instead of the average to be less sensitive to the magnitude of the fluctuations.

Remark that we still use the standard deviation as part of the characterization of the elasticity. This characterization captures all the important information about the elasticity (time needed to stabilize, loss of performance, and variability) with the two surface areas (A and B) and normalizes it into a dimensionless number that can be used for comparisons.

Finally, the number of observations needed to have an idea of the normal behavior of a database cluster cannot be fixed in advance. Experience shows that, from one system to another, high variability in performance can arise at different moments. This variability is mainly due to the writes of big files on the disk, like compactions, disk flushes, and disk pre-allocations, all of which can happen at very different moments due to the randomness of the requests and the technical choices made by each database. The variability has a measurable result that will be discussed in the result section. In practice, the observations were stopped when the performance and standard deviation got back to the level observed before the compactions or disk pre-allocations happened.

D. Properties of the methodology

All the parameters are updated linearly in respect to the number of nodes that are bootstrapped in the elasticity test, but all those parameters are not updated at the same time during the methodology. However, the measurements obey several invariants, which are given in italics below.

The size of the request sets is always increased at the same time as the number of client threads, which implies that on the client side, *the number of requests done by each client thread is independent of cluster size*. On the database nodes, there are two different situations. When the elasticity test begins and during the entire first phase of the test, as many threads as there are nodes in the cluster are started, and therefore, *the amount of work done by each node in the cluster is independent of cluster size*.

The second phase starts when new nodes are bootstrapped and lasts as long as the cluster needs time to stabilize. During this time, the amount of work done by the nodes already present in the cluster should decrease progressively as newly bootstrapped nodes start to serve part of the data set. In a perfect system, all the nodes in the enlarged cluster should eventually do an amount of work that has decreased linearly regarding to the number of nodes added in the cluster. It is important to note that the eventual increase in performance that would appear at this point is not a measure of the scalability as defined earlier. This is due to the fact that, at this point, neither the data set nor the number of client threads has been increased linearly regarding to the number of nodes added. The goal of the elasticity test is only to measure the impact of adding new nodes to a cluster that serves a constant load.

Once the elasticity test ends, the size of the data set inserted into the database is increased linearly according to the number of nodes just added. As a consequence, during the next round of the elasticity test the amount of data served by each node has not changed. Therefore, *once the number of threads is increased at the beginning of the next elasticity test, the total amount of work (number of requests served and data set size) per database node does not change*.

IV. Databases chosen

The three databases selected for this study are Cassandra [3] 0.7.2, HBase [4] 0.90.0 and mongoDB [5] 1.8.0 because they are popular representatives of the current NoSQL world. All three databases are horizontally scalable, do not have fixed table schemas, and can provide high performance on very big data sets. All three databases are mature products that are in production use by many organizations [6] [7] [8]. Moreover, they have chosen different theoretical approaches to the distributed model, which leads to interesting comparisons.

All three databases are parameterized with a common replication factor of 3 and strong consistency for all requests in order to ensure a comparable environment on both the application and server side.

V. Measurement conditions

This section describes the budget allocated, the infrastructure and the data set used as well as the benchmark implementation.

Copyright (c) IARIA, 2011. ISBN: 978-1-61208-153-3

A. Budget and infrastructure

We first explain our decisions regarding budget and infrastructure, since they affect the whole measurement process. The budget allocated for all the tests of this article is 800 euros. We choose to use the 4 GB cloud instances from Rackspace. This allowed us to perform measurements at full load for up to 48 nodes with all three databases.

Using cloud instances instead of dedicated servers has consequences on performance. Indeed several instances are sharing the same physical computer and therefore using cloud instances adds variability, depending on the resources usage of the other instances, to the measurements.

Finally, the data set per node has been chosen large enough to be sure that the subset of the data stored on each node could not fit entirely in RAM. It is important to remind the reader that the databases studied here are made to handle "Bigdata" problems where typically it would cost too much to fit all the dataset into memory. Therefore, with a focus on "Bigdata", it is natural to consider databases that cannot fit into memory.

B. Data set

The data set is made of the first 10 million articles of the English version of Wikipedia. They can be downloaded as a single archive provided [9] by Wikimedia itself. The dump was downloaded on March 7, 2011 and it takes 28 GB of disk space.

C. Benchmark implementation

The benchmark is written in Java and the code source is available as a GitHub repository under a GPL license [10]. The benchmark framework is used to automate the parts of the methodology that concerns the insertion of articles as well as applying the load and computing the results.

To approximate the behavior of Wikipedia users, the requests are fully random. Meaning that for each request, a uniform distribution (the Java class java.util.Random, initialized without seed) is used to generate a integer in the range of the IDs of the inserted documents. Then, after the article has been received by the client, a second integer is generated using a uniform distribution to decide if the client thread should update this article or not. Update simply consist in appending the string "1" at the end of the article.

VI. RESULTS

Figures 4 to 11 give graphs showing the elastic behavior of all databases at all transition sizes. These graphs represent the measured average time in seconds needed to complete a request set versus the total execution time in minutes. Standard deviations are indicated using symmetric (red) error bars, but it is clear that this does not imply improved performance during stabilization (downward swing)! The first part of each graph shows the normal behavior of the cluster under load. The first arrow indicates when the new nodes are bootstrapped and the second arrow indicates when all the nodes report that they have finished their data transfers. The graphs also show the standard deviations and the two thin (red) lines show the acceptable margins for the delta time that are computed from the first part of the graph.

Table I
STABILIZATION TIME (IN MINUTES, LOWER IS BETTER)

Database	Cluster size variation	Data tr. time	Add. time	Total time
Cassandra	6 to 12 nodes	113	28	141
HBase	6 to 12 nodes	3.3	9	12.3
mongoDB	6 to 12 nodes	172	11	183
Cassandra	12 to 24 nodes	175	26	201
HBase	12 to 24 nodes	3.2	14	17.2
mongoDB	12 to 24 nodes	330	22	352
Cassandra	24 to 48 nodes	86	2	88
HBase	24 to 48 nodes	8	37	45

Table II
ELASTICITY (LOWER IS BETTER)

Database	Cluster old and new size	Score
Cassandra	6 to 12 nodes	1735.
HBase	6 to 12 nodes	646.
mongoDB	6 to 12 nodes	4626.
Cassandra	12 to 24 nodes	1044.
HBase	12 to 24 nodes	70.
mongoDB	12 to 24 nodes	4009.
Cassandra	24 to 48 nodes	3757.
HBase	24 to 48 nodes	73.

Table I shows the stabilization times (in minutes), which consists of the times for all the nodes to finish their data transfers as well as the additional times needed for the whole cluster to achieve stabilization once all the data transfers are done. The time needed to finish all the data transfers is measured using tools provided by the databases to monitor data transfers across the cluster. The additional time to achieve stabilization is the time when the cluster reaches a stable level minus the time when the cluster reported that all the data transfers were done.

Table II shows the dimensionless elasticity scores according to the definition in Section III-A. In practice, the curves have been approximated by cubic splines interpolating the given point and those splines have been integrated using a recursive adaptive Simpson quadrature. The lower the elasticity score, the better the elasticity.

A. Analysis of the results

Analysis of the measurement results is made more difficult by the variability of the cluster performance under load before new nodes are bootstrapped. Those variabilities are very clear for Cassandra on Figure 4 and 5, for HBase on Figure 8 and for mongoDB on Figure 11. These big variabilities in performance have different origins but all of

Copyright (c) IARIA, 2011. ISBN: 978-1-61208-153-3

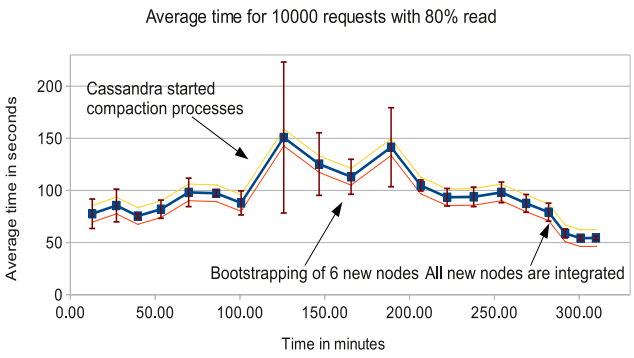

Figure 4. Elasticity under load Cassandra (6→12 n.)

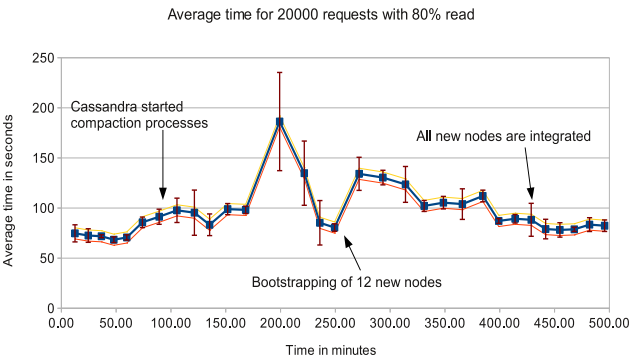

Figure 5. Elasticity under load Cassandra (12→24)

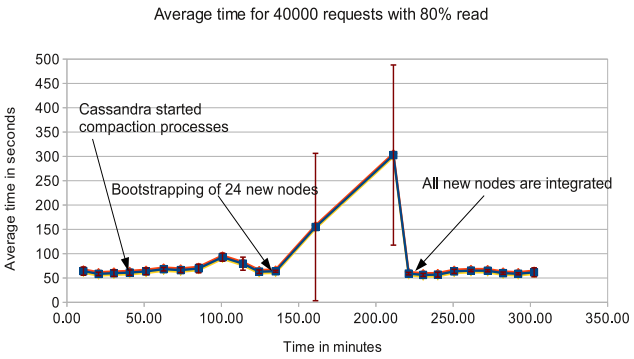

Figure 6. Elasticity under load Cassandra (24→48)

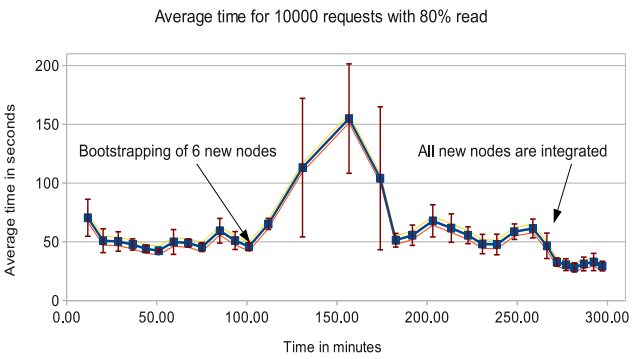

Figure 7. Elasticity under load mongoDB (6→12 n.)

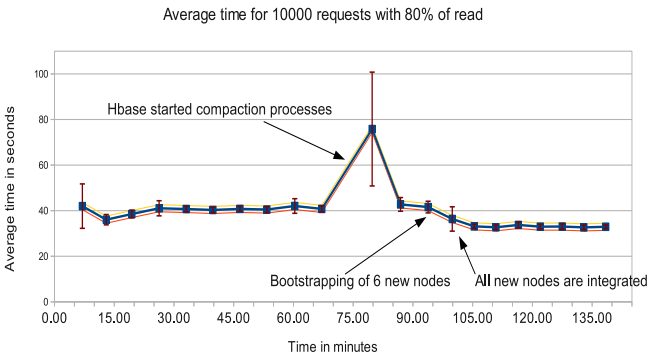

Figure 8. Elasticity under load HBase (6→12 nodes)

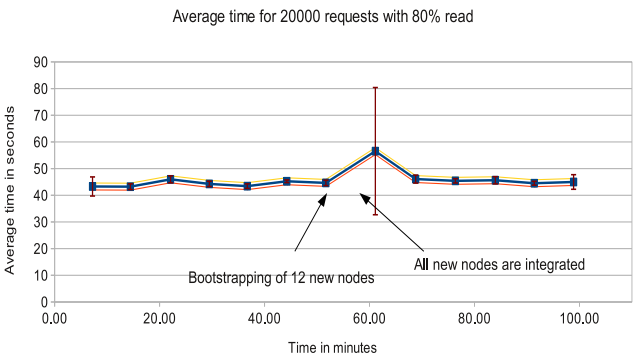

Figure 9. Elasticity under load HBase (12→24 n.)

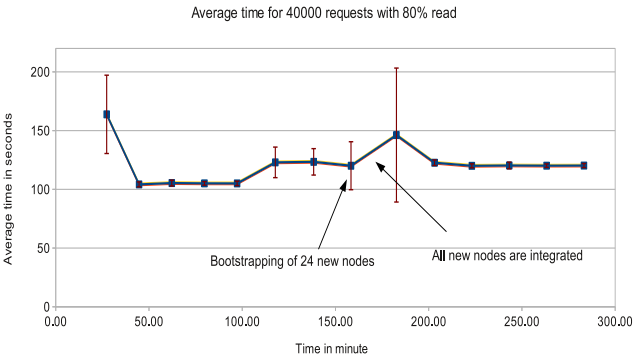

Figure 10. Elasticity under load HBase (24→48 n.)

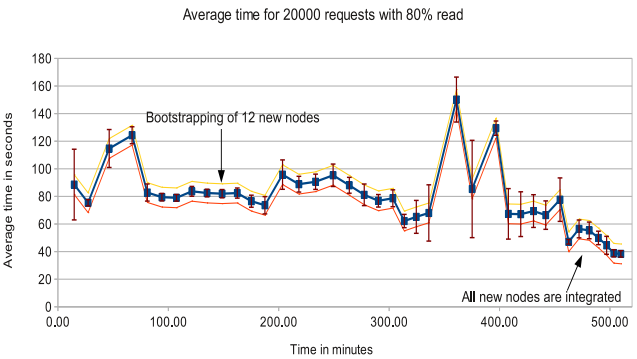

Figure 11. Elasticity under load mongoDB (12→24)

them have the same immediate cause: the writing of at least one big file on the disk. First, for Cassandra and HBase, big writes are triggered when compactions or disk flushes occur. The flushes occur when the *memtable* is full, and compactions follow after a few flushes [3] [11]. A load constantly updating data will, sooner or later, trigger compactions and disk flushes. Second, for mongoDB: big writes are only triggered when disk pre-allocations occur. mongoDB uses the mmap function provided by the operating system instead of implementing the caching layer itself, meaning that it is the OS itself that decides when to flush. mongoDB pre-allocates big files when it needs new storage space instead of increasing the size of existing files. In practice, mongoDB allocates space on disk each time it needs to store a lot of new data, like during chunks movements or big inserts.

Note that compaction is part of normal database operation that is needed both when handling client requests and when handling bootstrapped nodes during elastic growth. So we make no effort to remove the compaction cost from our measurement of elasticity. It is important to note that the only requests that will be slowed down by the writing of big files are the ones sent to nodes currently writing those big files. Therefore, when the number of nodes increases, the probability to send requests to a node currently doing a lot of I/O decreases. Indeed, looking at Figure 6 for Cassandra and Figure 10 for HBase, we observe the overall performance is more stable for bigger clusters.

On this infrastructure, the technical choice taken by mongoDB to make small but frequent disk flushes leads to less variability in performance than Cassandra. One could wonder what is the cause of the variability observed at the beginning of the chart on Figure 11 for mongoDB as no new nodes were bootstrapped at this time. This is caused by the fact that during the insertion, some nodes stored more chunks than the other and only started to distribute them across the cluster during the start of the test.

The variability of HBase performance is quite different from Cassandra even if their technical choices are close. By default the *memtable*'s size of Cassandra is 64MB and HBase is 256MB, leading to more frequent flushes and compactions for Cassandra but on the other hand, the compactions are also made on smaller files for Cassandra. The effect of compactions is only visible on Figure 8 and not on Figure 9 nor on Figure 10. This could be because the number of nodes is bigger and the effect of the compaction impacted a smaller number of requests.

Finally, there are no results for mongoDB going from 24 to 48 nodes. This is due to several problems encountered with mongoDB during the insertion of the articles. Starting with a cluster of size 12, mongod processes started to crash because of segmentation faults that caused data corruption, even with the journaling enabled. This problem was temporarily fixed by increasing the maximum number of files that can be opened by the mongod processes. But for 24 nodes, the segmentation faults were back with another problem. Eight threads were used to insert the articles, each of them making its requests to a different mongos router process, but all the writes were done on the same replica set. The elected master of this replica set was moving the chunks to other replica sets but not as fast as it was creating them, leading to a disk full on the master and at this point all the inserts stopped instead of starting to write chunks on other replica sets.

Elasticity

For the analysis of the elasticity results, we first explain some technical choices of the databases. The databases can be divided in two groups depending on the kind of work that the databases have to do when new nodes are added.

In the first group, which contains Cassandra and mongoDB, the databases have to move the data stored on the old nodes to the new nodes that just have been bootstrapped. In the case of a perfectly balanced cluster, that means moving half of the data stored on each of the old nodes to the new ones.

In the second group, which in this article contains only HBase, the database (HBase itself) and the storage layer (Hadoop Distributed File System [12]) have been separated to be handled by two distinct entities in the cluster. At the HBase level, each *region server* is responsible for a list of *regions* meaning that it has to record the updates and writes into *memtables* and it also acts as a cache for the data stored in the HDFS level. When new nodes running both a *region server* and a *datanode* are bootstrapped, the new *region servers* will start to serve a fair share of the available regions but the existing data will not be moved to the new *datanode*. Therefore there will be not big data transfer on new node bootstrapping.

The fact that HBase does not have to move all the data appears very clearly on the charts. HBase only needs a few minutes to stabilize while Cassandra and mongoDB take hours. The technical choices taken by HBase are a big advantage in terms of elasticity for this methodology. In Figures 9 and 10, HBase moves new regions to the region servers quickly, but the new region servers still need to load data, this is why the peaks happen *after* the new nodes are integrated.

For Cassandra, the impact of bootstrapping new nodes is less than the variability induced by normal operations for clusters smaller than 24 nodes, after that the impact is much more important than the usual variability of the cluster's normal operations. Note that the performance of Cassandra only improves when all the nodes are fully integrated because new Cassandra nodes only start serving requests when they have downloaded all the data they should store. The time needed for the cluster to stabilize increased by 54% between the tests of 6 to 12 nodes and 12 to 24 nodes, while it decreased by an impressive 50% between the

Copyright (c) IARIA, 2011. ISBN: 978-1-61208-153-3

tests of 12 to 24 nodes and 24 to 48 nodes. The nonlinear increase is due to the fact that new nodes know which are the old nodes that should send them data thanks to the nodes' *Tokens*, leading to simultaneous data transfers between nodes across the cluster. On the other hand, the 50% decrease is still to be explained.

With mongoDB, the variability in performance added by the bootstrap of new nodes is much bigger than the usual variability of the cluster. Unlike Cassandra, newly bootstrapped mongoDB nodes start serving data as soon as complete chunks have been transferred. Therefore newly bootstrapped nodes that serve the few chunks already received will pre-allocate files to make room for the next chunks received leading to a lot of requests potentially served by nodes writing big files to disk and therefore degrading the performance. The time needed for the cluster to stabilize increased by 92% between the tests of 6 to 12 nodes and 12 to 24 nodes. This almost linear increase is due to the fact that there is only one process cluster wide, the balancer, that moves the chunks one by one.

The elasticity scores give an accurate idea of the elasticity performance of the databases, disadvantaging databases for the height of the peak and the time needed before stabilization. Note that, for HBase, the decreasing score is due to relatively smaller peaks as the cluster grows and the last one can also be explained by the fact that the performance is less, so the elasticity is relatively better with respect to this worse performance. Globally, the elasticity score also shows the advantage of HBase for clusters of all sizes.

VII. Conclusions and future work

The main conclusion of our measurements is that the technical choices taken by each database have a strong impact on the way each of them reacts to the addition of new nodes. For this definition of elasticity, HBase is a clear winner. This is due to its technical choices and architecture leading to much less data transfer on node addition.

This article gives measurement results only for systems that scale up, and not for systems that scale down. We decided for this limitation because we wanted to explore in detail what happens when a system scales up, and experience has borne out that these measurements are sufficiently surprising and technically difficult to carry out. We expect that future work measuring systems that scale down will give a fresh set of surprises.

We plan to continue expanding the cluster sizes to see if the current trends will last or if some other bottleneck will appear at some point. For example it would be interesting to see if it is possible to reach any bottleneck with systems, like HBase and mongoDB, using a centralized approach to store the localization information in the cluster.

We also intend to solve the problems encountered for mongoDB to measure its performance optimally. Then it would also be interesting to do the same tests but with different values for the parameters like the read-only percentage or using a different statistical distribution. We plan to extend our coverage of the measurement space and continue to refine our new elasticity measure. Finally, we intend to measure the performance of other databases like Riak and distributed caches like infinispan and ehcache.

VIII. Acknowledgements

We would like to thank Euranova for the idea of studying the elasticity of distributed databases and for their support that helped us improve this article [13]. Special thanks go to the director of Euranova R&D, Sabri Skhiri, for his insightful comments. We also thank Ivan Frain and Samuel Richard for their constructive comments.

References

[1] T. Dory, "Study and Comparison of Elastic Cloud Databases: Myth or Reality?" pldc.info.ucl.ac.be, Programming Languages and Distributed Computing (PLDC) Research Group, Université catholique de Louvain, Tech. Rep., Aug. 2011.

[2] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears, "Benchmarking cloud serving systems with YCSB," in *SoCC*, 2010, pp. 143–154.

[3] A. Lakshman and P. Malik, "Cassandra: a decentralized structured storage system," *SIGOPS Oper. Syst. Rev.*, vol. 44, pp. 35–40, April 2010. [Online]. Available: doi.acm.org/10.1145/1773912.1773922

[4] Apache HBase, "Frontpage," hbase.apache.org, Jun. 2011.

[5] mongoDB, "Frontpage," www.mongodb.org, Jun. 2011.

[6] Apache Cassandra, "Frontpage," cassandra.apache.org, Jun. 2011.

[7] HBase Wiki, "PoweredBy," wiki.apache.org/hadoop/Hbase/PoweredBy, Jun. 2011.

[8] mongoDB, "Production Deployments," www.mongodb.org/display/DOCS/Production+Deployments, Jun. 2011.

[9] Wikipedia, "Latest dump of Wikipedia English," download.wikimedia.org/enwiki/latest/enwiki-latest-pages-articles.xml.bz2, Jun. 2011.

[10] GitHub, "Wikipedia-noSQL-Benchmark," https://github.com/toflames/Wikipedia-noSQL-Benchmark/, Jun. 2011.

[11] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber, "Bigtable: A distributed storage system for structured data," *ACM Trans. Comput. Syst.*, vol. 26, no. 2, 2008.

[12] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The Hadoop distributed file system," in *Proceedings of the 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*, ser. MSST '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 1–10. [Online]. Available: dx.doi.org/10.1109/MSST.2010.5496972

[13] Euranova, "Frontpage," euranova.eu, Jun. 2011.

Copyright (c) IARIA, 2011. ISBN: 978-1-61208-153-3