Quantifying the Elasticity of a Database Management System

Christian Tinnefeld, Daniel Taschik, Hasso Plattner

Hasso Plattner Institute
University of Potsdam, Germany
{firstname.lastname}@hpi.uni-potsdam.de

Abstract—**There exist different and well-established approaches for quantifying the performance of a database management system. With the advent of provisioning information technology infrastructure over the Internet, the aspect of elasticity became more important as it defines how well a system adapts to a changing workload. For a database management system there is no commonly agreed approach or model how to quantify its elasticity. In contrast, the cloud storage system (NoSQL) community developed several approaches how to measure elasticity. In this paper we contribute by I) presenting an extensive review of the existing approaches for measuring the elasticity of NoSQL systems, II) compare their parameters and used metrics, III) transfer the lessons learned and introduce a model for quantifying the elasticity of a database management system.**

Keywords-Elasticity, Database Management System, NoSQL

I. Introduction

Minhas et al. say that elasticity is the "ability to grow and shrink processing capacity on demand, with varying load" [1]. Another definition is given by Agrawal et al. which state that elasticity is "the ability to deal with load variations by adding more resources during high load or consolidating the tenants to fewer nodes when the load decreases, all in a live system without service disruption" [2]. A more general definition has been given by the National Institute of Standards and Technology: "Capabilities can be rapidly and elastically provisioned, in some cases automatically, to quickly scale out, and rapidly released to quickly scale in. To the consumer, the capabilities available for provisioning often appear to be unlimited and can be purchased in any quantity at any time" [3]. Figure 1 illustrates frequently used terminology in conjunction with elasticity. After this introduction follows Section 2 which reviews state-of-the-art elasticity benchmark for NoSQL systems and compares the used metrics, how a scale-out is triggered, how data migration is done, how the operational costs are quantified, which query and workload characteristics are applied and how the scale-out is managed. Section 3 then continues with a model that allows to quantify the elasticity of a relational database management system. The model is based on measuring the query processing latency in combination with a breakdown of the utilized hardware resources. Section 4 close with the conclusions and presents the future work.

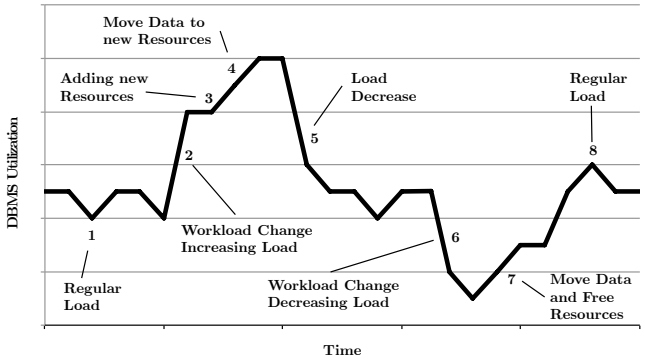

Figure 1: Terminology used in conjunction with elasticity

II. Elasticity Benchmarks for NoSQL Systems

A catalogue of metrics for evaluating cloud services is presented in the work of Li et al. [4]. The metrics presented for elasticity are split up in three groups. (1) Resource acquisition time, (2) resource release time and (3) cost and time effectiveness. The first group describes the time a resource takes to be available for the system from the moment it has been requested until the moment of availability to the system. The second group describes the time to release an unnecessary cloud resource. It can be split into the time to remove the existing deployment

 ISBN: 978-1-61208-334-6

and the time to stop the cloud resource and finally release it. The third group of metrics describes the relation between the costs and the runtime of provisioned cloud resources. Especially the third group has been recognized in our framework considerations by measuring the runtime costs from the moment of provisioning a resource to the moment of de-provisioning.

Weinman [5] suggests a model for calculating elasticity. He states that almost every business tries to match demand and supply. By defining a function $D(t)$ representing a mapping from demand to a resource in time and a second function $R(t)$ representing allocated resources over time, he proposes that a perfect capacity strategy is given, when $D(t) = R(t)$. In this case, the resources for matching an existing demand are available in the right amount. Too little resources do not create loss in revenue neither excess resources create needless costs. He introduces a function describing the financial loss an unmet demand can cause. That costs unnecessarily accrue in situation of excess resources. He also evaluates how monitoring interval and provisioning time for a resource influences the negative impact.

Weinman's paper had a great influence on the work of Islam et al. [6]. After the definition of elasticity in the context of cloud platforms, the authors propose a concept on how a consumer can quantify the elasticity of these platforms. The concept is build on the idea of Weinman and extends its calculation model with a penalty for over- and under-provisioning. Hereby, a differentiation between allocated and charged resources takes place. By normalizing the calculated value, the authors propose a single metric of elasticity for a cloud-based platform. They showcase their approach for an elasticity measurement environment and make use of it for different consumer specific workloads scenarios. The work presented in the above mentioned paper, served as inspiration for the elasticity benchmark framework for relational database management systems in this thesis. The provisioning based calculation model is taken from Islam et al. and has been slightly modified to fit the needs for a relational database management system (DBMS).

One of the most known and famous NoSQL benchmark is the *Yahoo! Cloud Serving Benchmark* (YCSB) [7]. It has been developed at Yahoo to help developers to choose which cloud based data storage might be the best for their workload. It provides a two-tier structure. The first one concentrates on performance whereas the second and more interesting one looks at scalability. Elastic speed-up is measured by monitoring the performance of a system as the number of machines is increased while running a constant workload. A good elastic system must show an improvement in performance. A short disruption in service is accepted while the system is reconfiguring itself. Elasticity itself is not quantified and only the impact of read latency is considered.

Konstantinou et al. [8] conducted a study on the costs and efficiency of adaptive expansion and contraction of NoSQL databases over a cloud platform. The authors took three popular NoSQL representatives (HBase, Cassandra and Riak) and performed experiments with four YCSB workloads. They analyzed how the cloud data storages performed by measuring query throughput, mean query latency as well as CPU and memory consumption during a stress test. Costs for the initialization and reconfiguration of nodes as well as rebalancing of data within the cluster are ascertained in terms of time and data volume. Finally, Konstantinou et al. present a framework for monitoring and automating cluster resize operations. The authors benchmarked only NoSQL systems and did not consider analytical workloads. They did not track financial aspects for operating the platform. Thibault Dory et al. [9] introduce a dimensionless measure for elasticity for cloud databases. In their methodology, Dory defines elasticity as a characterization of how a cluster reacts on node provisioning. Regarding to Dory, elasticity is defined by two properties. The first one is the time a cluster takes to stabilize itself after nodes have been added. The second property is the influence on the cluster's performance. In regard to the first property, they define a cluster as stable when the variations of time needed to fulfill a certain number of requests is equivalent to the variations of a cluster known as being stable. In this case, it is a cluster where no data is being moved from one node to another. The authors suppose that the response time for requests increases after new nodes have been added, and then decrease after a certain amount of time. Dory et al. define the elasticity as a ratio of elastic overhead to the absolute performance of a cluster. This approach of quantifying elasticity is validated against a NoSQL architecture with an OLTP workload using the behavior of a Wikipedia user.

Another cloud-based quality measurement and analysis framework is introduced in the work of Klems et al. [10]. By contributing with an infrastructure, configuration and cluster configuration manager, Klems et al. propose a framework to evaluate the performance, latency and

 ISBN: 978-1-61208-334-6

consistency of the cloud-based data stores Amazon S3, Amazon SimpleDB, DynamoDB and Cassandra using the Yahoo! Cloud Serving Benchmark and YCSB++ benchmark. The authors analyze different scaling strategies and conflicts between contradictory objectives, such as consistency versus high-availability and scalability. In addition, they examine the impact of system changes on performance and availability. Unfortunately, further investigations of elastic scalability in regard to data migration and performance impacts of such, is not presented, though can be expected in future work.

The Cloud Service Measurement Index Consortium proposed a framework [11] to define and measure certain quality of service aspects of cloud providers. Its service measurement index is intended to help customers to rank and compare cloud service providers based on customers requirements like accountability, agility, costs, performance, assurance, security and usability. Elasticity, defined as how much a cloud service is able to scale during peak time, is seen as a subcomponent of agility. As part of a case study, Garg et al. present a relative service-ranking vector for elasticity. This approach takes the time a system needs to expand or contract into account. An impact on how data migration might influence the performance is not considered. The remainder of this section gives a detailed breakdown of the different parameters. Table I on the next page summarizes which parameters are considered in the previously mentioned related work.

A. Metrics

Each of the three benchmarks uses different metrics to express elasticity. Islam's [6] approach is based on a financial penalty model. When a system runs in an undesired state, being either under- or over-provisioned a fine will be charged. Undesired in this context means that either there are not enough resources to handle the load or that there are too many, unnecessary resources present which could be de-provisioned for the cause of cost savings. The penalty amount is calculated from the time the system is in an undesired state, representing the responsiveness of the system to scale and change the state into a desired one. The smaller the penalty the more elastic is the system.

The Yahoo! Cloud Serving Benchmark [7] uses quite a basic metric for elasticity. It takes response latency to requests as a measure to express elastic speedup. The authors conducted a benchmark examining the elastic speedup of three cloud-based data stores. For each data store they started with a small cluster offering a load feasible for a three-times bigger system. Then, they added nodes to the cluster until it was stabilized and able to serve the load. The implications on latency have been recorded and taken as a degree of elastic speedup. Dory et al. [9] use a dimensionless metric for elasticity whereas Konstantinous et al. [8] use throughput, latency as well as CPU utilization as their elasticity measure. In our opinion, taking just the latency is not sufficient enough to measure elasticity. I argue that there are more metrics influencing the elasticity of a DBMS than just the responsiveness to queries. Taking fixed budgets for a cloud system or energy consumption of the hardware into account, it is conceivable that there are more dimensions expressing elasticity. I am in favor of Islam's approach, taking the costs per time as a measurement. Because pay-as-you-go is one of the advantages of cloud computing over established server based systems, taking the runtime costs of a system into account creates a more meaningful metric for elasticity.

B. Scaling Trigger

The various benchmarks and frameworks trigger a scaling operation differently. Whereas YCSB and Dory et al. use static scaling, meaning it is triggered either manually or at a fixed moment during the benchmark, Konstantinous and Islam take CPU utilization of the system as scaling trigger. Konstantinous et al. perform a scale-out as soon as one node has more than 40% CPU utilization. A scale-in, on the other hand, is done as soon as the average of all nodes are less than 15% utilized. Islam triggers scaling operations in the event of an undesired provisioning state (under- or over-provisioning).

I emphasize the fact that a framework should be able to trigger scaling operations automatically. In real-world scenarios, DBMS operators use automated tools for scaling instead of issuing manual scaling instructions.

C. Data Migration

Migrating data from an existing cluster to an added node has an impact on elasticity, because transferring data between nodes or throughout an entire cluster takes time and slows down regular operations. Except for Islam et al., all other frameworks consider the time needed for the migration operation. Konstantinous et al. also observe the amount of the moved data. This enables a better differentiation between different architectural

 ISBN: 978-1-61208-334-6

TABLE I: Overview of Elasticity Benchmarks and Frameworks for NoSQL Systems

	Properties	Islam et al.	YCSB	Dory et al.	Konstantinous et al.
Metrics	Throughput	-	-	-	x
	Latency	-	x	-	x
	CPU Usage	-	-	-	x
	Monetary	x	-	-	-
	Dimensionless	-	-	x	-
Scaling Trigger	Static	-	x	x	-
	Latency-based	x	-	-	-
	CPU Utilization	x	-	-	x
Consideration of Data Migration	Time	-	x	x	x
	Amount	-	-	-	x
Operating Costs	Taken into Account	x	-	x	-
Query Characteristic	OLTP	x	x	x	x
	OLAP	-	-	-	-
Workload Characteristics	Sinus Shaped	x	x	-	-
	Plateau Shaped	x	x	-	-
	Exponential Shaped	x	x	-	-
	Linear	x	x	x	x
	Random	x	x	-	-
	Zipfian	-	x	-	x
Scale Management (Monitoring, Cloud/ Cluster Management, Rebalancing)	Provide Toolset	-	-	-	x
	External Tools (e.g. Amazon Autoscale)	x	-	-	-
	Manual	-	x	x	-
Applicability Difficulty	Simple	x	x	-	x
	Difficult	-	-	x	

characteristics of databases. I comply that the time for adding a new node and getting it ready to serve should be reflected in an elasticity benchmark. In addition, it is of interest to track the time for data migration and the time for provisioning computing resources, booting a node, starting the DBMS instance and registering it with the existing cluster.

D. Operational Costs

Elasticity is a time-critical property. The faster a system is able to adapt, the more it is considered elastic and this has consequences on operational costs for the system, e.g. if computing resources are provisioned and not used, the landscape generates more costs than necessary to serve the load. Islam et al. and Dory et al. pay attention to the operational costs in their elasticity determination. As already mentioned in the metrics part of this section, pay-as-you-go is the reason why operational costs have to be regarded and must be considered in an elasticity benchmark.

E. Query & Workload Characteristics

All four presented frameworks execute only transactional queries in their benchmarks. The applied workload patterns are different. Islam et al. as well as YCSB have a huge variety of different workload patterns available. Dory et al. and Konstantinous et al. use a linear workload and Konstantinous makes use of an additional zipfian workload. Nevertheless, all frameworks lack a real-world workload scenario. YCSB allows customizing and implementing industry-related scenarios. The benchmark must reflect a workload pattern, which simulates the intended area of deployment of the DBMS. This can be a simple sinus-like workload, but in an enterprise environment it is conceivable to run complex analytical workloads.

F. Scale Management

To measure elasticity properly, it is necessary for the framework to provide tools for monitoring load, to add and remove nodes, to redistribution of data and to manage the cluster. The work of Konstantinous et al. is the only framework that provides a conventional

 ISBN: 978-1-61208-334-6

toolset. Islam et al. are using tools from a third-party provider, in this case, Amazon CloudWatch and Auto Scaling. It can only be assumed that monitoring and provisioning is handled manually in the work of Dory et al. and YCSB, because scaling is not automated at all. In our opinion, a proper framework must provide these tools to supplement missing features in the DBMS. Only without manual intervention, realistic measurements can be conducted.

III. Quantifying the Elasticity of a Database Management System

After extensively reviewing approaches for quantifying the elasticity of a NoSQL system, we present an elasticity calculation model for relational DBMS which is inspired by the work of Islam et al. [6]. To determine elasticity, Islam et al. offer a way for consumers to measure elasticity for cloud platforms by defining financial penalties for over- and under-provisioning. Under-provisioning means that the system has less computing resources available than actually necessary to fulfill all requests (demand) against the system in a desired time. Over-provisioning, on the other hand, describes the state of a system in which there are excess resources available than really needed to fulfill the demand against the system. These excess resources lead to higher runtime costs that are avoidable. The financial reflection model of Islam et al. is taken and got adapted to fit the needs for an elasticity benchmark for relational DBMS. To build a calculation model for elasticity, a few assumptions need to be stated. Elasticity enables very cost-efficient operation. Therefore, costs play a key role in defining a metric for elasticity. The costs for a utilized node to run for one hour are defined as 100 cents. The price is derived from the Amazon AWS EC2 pricing list [12] for an *Amazon M3 Double Extra Large Instance*. The hardware sizing of this instance type is powerful enough to run a relational DBMS and therefore the price can be valued as reasonable. At Amazons AWS EC2, a resource is allocated for a minimum time period of one hour. To avoid complexity in situations where a resource is allocated and therefore charged but not available, because it is already de-provisioned or not yet booted, the chargeable time period is reduced to one second. Consequently, the calculation of chargeable supply as used by Islam et al. is discarded. The moment the resource is requested, it is charged until the second it is de-provisioned.

To calculate the elasticity for an elastic RDBMS, it is necessary to sum up the number of used nodes. A resource can be charged from the moment of provisioning of a node to the moment a service on that node is serving, because booting time of a prepared image with all required services pre-installed can be done in a constant time. Hence, for easier consideration it is assumed that the chargeable time begins at the moment the framework detects an under-provisioned state and ends at the moment the framework de-provisions the node. The already mentioned penalties for over- and under-provisioning need to be specified as well. An under-provisioning penalty of 10 cents for every second in an under-provisioned state is specified. This reflects six times the costs of an additional node. As Weinman et al. [5] emphasize, the benefit of using resources should clearly outweigh the costs of it. The state of over-provisioning is not penalized because too many provisioned resources create avoidable costs, which are thereby treated as a penalty. So only the pure costs of the provisioned resources are taken into account. In summary, the following numbers need to be ascertained to calculate elasticity: maximum allowed latency, number of used DBMS nodes, runtime of utilized RDBMS nodes, time while being under-provisioned.

The maximum allowed latency is configured by the benchmark executor and describes in this case the longest acceptable response time for a benchmark suite run. A benchmark suite run is defined as a set of queries, which get executed by a benchmark client. The benchmark client repeats a benchmark suite run over and over until the framework controller is stopping the client. The amount and runtime of server nodes can be retrieved from the cluster management controller that is responsible for provisioning of RDBMS server nodes. The time while the RDBMS is under-provisioned can be gathered by taking the latency until a benchmarking suite run has finished and subtract the maximum allowed latency for a benchmark suite run as demonstrated in Formula 1.

$$f(t) = \text{Benchmark\_runtime}_t - \text{upper\_threshold}_t \quad (1)$$

To get the time when the benchmark suite runtime was above the upper limit, the function $f_{\text{cutoff}}(t)$ needs to be ascertained. Therefore, only values above the limit as represented in Formula 2 are taken into account.

$$f_{\text{cutoff}}(t) = \begin{cases} f(t) & \text{if } f(t) > 0 \\ 0 \end{cases} \quad (2)$$

 ISBN: 978-1-61208-334-6

Now, that only the amount of time, when the runtime of the benchmark suite has taken longer than the upper bound per node has been determined, the accumulation over all recorded instances can be done as seen in Formula 3

$$P(t_0, t_{\text{end}}) = \sum_{0}^{\#instances} \int_{t_0}^{t_{end}} f_{\text{cutoff}}(t)\mathrm{d}t \tag{3}$$

The next step is to ascertain the penalty sum for all instances. The sum will be multiplied with the defined penalty amount. This results in an overall penalty as seen in Formula 4.

$$P = P(t_0, t_{\text{end}}) \times p_{\text{under-provisioned}} \tag{4}$$

Finally, the penalty and the runtime costs will be aggregated. To calculate the runtime costs, the number of running nodes for a certain time frame needs to be multiplied by the costs of it. Formula 5 shows the calculation of it.

$$C_{\text{nodes}} = (\sum_{n=1}^{\#_{nodes}} runtime(n)) \times c_{node} \tag{5}$$

Formula 6 shows the calculation of the final elasticity by adding up the penalty and the runtime costs for the provisioned resources C_{nodes} and dividing it by the runtime of the experiment. This results in a comparable value in cents per seconds.

$$E = \frac{P + \#\text{nodes}_{\text{charged}} \times c_{\text{nodes}}}{t_{end} - t_0} \tag{6}$$

The apparent significance of time-to-serve for an elastic relational DBMS does not need to be measured explicitly. It is implicitly provided by the runtime of the benchmarking suite. The longer it takes for a node to be ready to serve, the longer the cluster stays in an under-provisioned state. This results in a much higher penalty than for systems with a very low time-to-serve value.

IV. Conclusions and Future Work

In this paper, known quantification models and methods from NoSQL systems are evaluated and taken into account to propose a model for quantifying the elasticity of a database management system. Features and characteristics as well as pre-conditions for elasticity measurements are identified, presented and defined. The proposed elasticity calculation model is a provisioning based quantification model. The corner stones of the model are the aspects of maximum allowed latency, number of used DBMS nodes, runtime of utilized RDBMS nodes and the time while the DBMS is being under-provisioned.

The presented elasticity model enables relevant elasticity determination experiments yielding unique comparable values for a relational DBMS under a specific workload. This has to be demonstrated in future work by conducting a case study. Here, different workloads will be executed on different relational DBMSs and the resulting elasticity will me calculated and compared.

References

[1] U. F. Minhas, R. Liu, A. Aboulnaga, K. Salem, J. Ng, and S. Robertson, "Elastic Scale-Out for Partition-Based Database Systems," 2012 IEEE 28th International Conference on Data Engineering Workshops, 2012, pp. 281–288. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6313694

[2] D. Agrawal, A. Abbadi, S. Das, and A. Elmore, "Database scalability, elasticity, and autonomy in the cloud," in Database Systems for Advanced Applications, ser. Lecture Notes in Computer Science, J. Yu, M. Kim, and R. Unland, Eds., vol. 6587. Springer Berlin Heidelberg, 2011, pp. 2–15. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-20149-3_2

[3] P. Mell and T. Grance, "The NIST Definition of Cloud Computing Recommendations of the National Institute of Standards and Technology," National Institute of Standards and Technology NIST, Gaithersburg, MD, vol. 145, 2011.

[4] Z. Li, L. O'Brien, H. Zhang, and R. Cai, "On a Catalogue of Metrics for Evaluating Commercial Cloud Services," 2012 ACM/IEEE 13th International Conference on Grid Computing, 2012, pp. 164–173. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6319167

[5] J. Winman, "Time is Money : The Value of On-Demand," http://www.joeweinman.com, pp. 1–29, 2011.

[6] S. Islam, K. Lee, A. Fekete, and A. Liu, "How a consumer can measure elasticity for cloud platforms," Proceedings of the third joint WOSP/SIPEW international conference on Performance Engineering - ICPE '12, 2012, p. 85. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2188286.2188301

[7] 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 - SoCC '10, 2010, p. 143. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1807128.1807152

 ISBN: 978-1-61208-334-6

[8] I. Konstantinou, E. Angelou, C. Boumpouka, D. Tsoumakos, and N. Koziris, "On the elasticity of nosql databases over cloud management platforms," in Proceedings of the 20th ACM International Conference on Information and Knowledge Management, ser. CIKM '11. New York, NY, USA: ACM, 2011, pp. 2385–2388. [Online]. Available: http://doi.acm.org/10.1145/2063576.2063973

[9] T. Dory, B. Mejias, P. V. Roy, and N. L. Tran, "Measuring elasticity for cloud databases," in CLOUD COMPUTING 2011: Proceedings of the The Second International Conference on Cloud Computing, GRIDs, and Virtualization, I. 978-1-61208-153-3, Ed., 2011, pp. 154–160.

[10] M. Klems, D. Bermbach, and R. Weinert, "A Runtime Quality Measurement Framework for Cloud Database Service Systems," 2012 Eighth International Conference on the Quality of Information and Communications Technology, 2012, pp. 38–46. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6511780

[11] S. K. Garg, S. Versteeg, and R. Buyya, "A framework for ranking of cloud computing services," Future Generation Computer Systems, vol. 29, no. 4, 2013, pp. 1012–1023. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S0167739X12001422

[12] Amazon, "Amazon EC2 Prizing List," 2013. [Online]. Available: http://aws.amazon.com/de/ec2/pricing/

 ISBN: 978-1-61208-334-6