Dynamic Workload-Aware Elastic Scale-Out in Cloud Data Stores

SWATI AHIRRAO
Symbiosis Institute of Technology
Symbiosis International University
Near Lupin Research Park, Pune
INDIA
swatia@sitpune.edu.in

Dr. RAJESH INGLE
Pune Institute of Computer Technology
University of Pune
Katraj, Pune
INDIA
ingle@computer.org

Abstract: NoSQL databases store a huge amount of data generated by modern web applications. To improve scalability, a database is partitioned and distributed among the different nodes called as a scale out. However, this scale out feature of the NoSQL database is oblivious to the data access pattern of the web applications, which results in poorly distributed data across all the nodes. Therefore, the cost required for the execution of the query is increased. This paper describes the partition placement strategy, which will place data partitions to the available domains in the Amazon SimpleDB according to the data access pattern of web applications, which leads to an increase in throughput by some percentage. We present the workload-aware elasticity algorithm, which will not only add and remove the domain as per the load, but also places the partitions as per the data access pattern. We have validated the workload-aware elasticity and load distribution algorithm through experimentation over a cloud data store such as Amazon SimpleDB running in the Amazon Cloud. The throughput of the load distribution algorithm is predicted using the regression and the multiple perceptron model.

Key–Words: partition placement, workload-aware elasticity, data partitioning, database scalability, placement strategy .

1 Introduction

Cloud Computing is an emerging trend in providing Infrastructure as a service, Platform as a service, Software as a service, and Database as a service. These services are offered on pay-per usage basis and provide on demand access to the resources. NoSQL databases have become popular because they are scalable in nature. Cloud computing includes the characteristics such as scalability and elasticity, which enables applications to effectively use resources in an on demand fashion. Scalability is achieved using horizontal data partitioning. With this property, NoSQL applications are tuning their performance that is throughput, and response time etc. as per the resources assigned. This is very well suited with the elastic property of a cloud. Elasticity is defined as allocating and deallocating resources as per the needs of the user. Allocation and deallocation of resources as per demand enhances system resources utilization, and also encourages the pay-per-usage model. When the number of clients increase or load increases, web servers and application servers called as stateless systems can be scaled out and scaled in easily as per the demand. On the other hand, the database management systems also called as stateful systems are difficult to scale in and out because of the consistency problem. Therefore, a lot of work is carried out by the researchers to solve this problem and the solution for this is to start with a scalable database system for achieving elasticity. Swati Ahirrao et. al[4] presented the idea of data access pattern and workload-aware partitioning technique. This work is an extension of the idea presented in dynamic workload-aware partitioning[4]. Scaling out and distributing data across a number of partitions does not necessarily effect in a linear increase in the system throughput, because the load distribution is not based on the data access pattern of the web applications. Therefore, distributed transactions occur. In an e-commerce application, when the order is placed by the customer, that order is fulfilled from a warehouses on one partition. But when the warehouse on one partition is out of stock then the order is fulfilled by the warehouse on another partition. In such a way, there is a always a pattern, which warehouse is supplying orders to a particular warehouse. We refer to this pattern as the data access pattern. We have implemented and evaluated this elastic scale-out and scale-in approach in the Amazon SimpleDB, by including the efficient partition redistribution mechanism.
The contributions in this paper are summarized as follows.

- We present the workload-aware placement strategy, which will place the fragments to domains according to the data access pattern.

- We present the workload-aware elasticity algorithm, which will add and remove the domain according to the load. We show the extensive experiments that show the effectiveness of our elasticity algorithm.

- We describe the practical implementation using Amazon SimpleDB running in the Amazon Cloud. We validate the design by evaluating the performance of the system using the TPC-C benchmark.

- We present the detailed analysis of our workload-aware elasticity algorithm using regression and the multiple perceptron model.

The remaining paper is organized as follows. Section 2, presents overview of the related work. Section 3, shows the design of the load distribution scheme. Section 4, discusses about the load distribution algorithm. Section 5, presents the partition placement strategy. Section 6, describes an overview of the workload-aware elasticity framework. Section 7, gives the performance analysis of the algorithm. Section 8, shows an experimental evaluation. Section 9, presents the statistical model and data analysis using regression and the multiple perceptron model. Finally, section 10 concludes the paper.

2 Related Work

Researchers have carried out a survey of partition placement where data is distributed between a fixed number of nodes. We have surveyed the existing work for improving database scalability and realized that, the existing techniques are either based on partitioning or replication. In this work, our focus is on using partitioning for developing an elastically scalable system. Curino et. al presented the Relational Cloud[7] for fostering scalability. It uses the workload-aware partitioning technique. However, their focus is on improving scalability using partitioning but not on a workload-aware elastic scale out. Sudipto Das et. al proposed ElasTras[12], which uses schema level partitioning for increasing scalability. In schema level partitioning all the related tuples are collocated on single partition. Francisco Cruz et. al presented MET[9] workload-aware elasticity for NoSQL, which will place the partitions to a node as per the YCSB workload access patterns. i.e all the reads, and writes etc. will be placed on different nodes. Marco Serafini et. al implemented Accordion[21], for achieving elastic scalability. It adds and removes servers as per the demands of the users, but does not redistribute the partitions based on data access pattern of web applications. Dimitrios Tsoumakos et. al developed a framework, TIRAMOLA[22], which takes the help of the Markov Decision Process model for decision making. The decision making process includes whether to add a node or remove a node from a cluster. The decision is made by taking into account the parameters such as throughput, response time and the cost of a virtual machine. In TIRAMOLA, the emphasis is on using the Markov Decision Process (MDP) for automatic resizing of the cluster. Evie Kassela et. al present an extended TIRAMOLA[15], which also focuses on automatic resizing of a cluster. But the main emphasis is on the workload-aware approach. It identifies the different workload types and also considers this workload-aware approach for decision making. Athanasios Naskos et. al presented the cloud elasticity using the probabilistic model checking[18], approach for resizing a cluster of virtual machines. In this paper, probabilistic models are used in the decision making process. Previous research does not address the problem of elastic scale out based on the data access pattern of the web application. Our aim in this work, is to place the partitions based on data access pattern by considering the minimum number of domains, so that the resources as well as cost is minimized.

3 Design of Load Distribution Scheme

The database is fragmented using a partitioning key with wid and all the related rows with the same wid are collocated at the same domain. As per the TPC-C benchmark[24], 10% of the transactions are executed in distributed mode. That means in 10% of the cases warehouses do not have stock to fulfil the orders of the customer. In such cases TPC-C randomly selects the supplier warehouse. But in reality it is not random, and the supplier warehouse is a warehouse, which is closer to the warehouse, which is processing that order. In this way, we are analysing the probability of warehouses supplying the orders to the warehouses processing that order. Our goal is to identify the pattern and monitor this behaviour by analyzing the logs of the transactions. Then, these warehouses are redistributed and the warehouses with more coherency are collocated at one domain or partition. Therefore, the partitions are not fixed and are formed dynamically.

4 Load Distribution Algorithm

This workload-aware load distribution algorithm accepts a set of warehouses and a number of domains as input and generates combinations with the optimized load and association. The load distribution is calculated by taking a standard deviation of the workload on the domain from the average load on the domain. Then, these combinations are ranked in ascending order. Then calculate the association for a combination by analyzing the execution of the total number of transactions on that domain, and also by finding the distributed transactions for the same combination. These combinations are ranked based on association. Then, these combinations are ranked in descending order. Both the ranks are calculated and arranged in ascending order. After running this algorithm, we will get combinations in domains with the optimized load and association. We can use these combinations for populating data in workload-aware partitioning.

5 Partition Placement Strategy

The following points are considered while designing this placement strategy.

- Data, which is required for the execution of a transaction should be collocated on a single domain.
- The placement of data to all the domains should be uniform so that the throughput is increased.

To design this partition placement strategy, we have carried out a survey to find the optimized combinations of warehouses. We have calculated the load and association for all possible pairs of combinations of warehouses. We choose the combinations with optimized load and association. These combinations are kept on the domain of Amazon SimpleDB. To find these optimized combinations, we have used the mutation technique in the genetic algorithm. Mutation is a technique used in the genetic algorithm to introduce diversity.

Table 1: Notations used for algorithm

Symbol	Description
r	Total number of records.
T	Total number of transactions.
p	Total number of partitions.
w	Total number of warehouses.

6 Overview of Workload-Aware Elasticity Framework

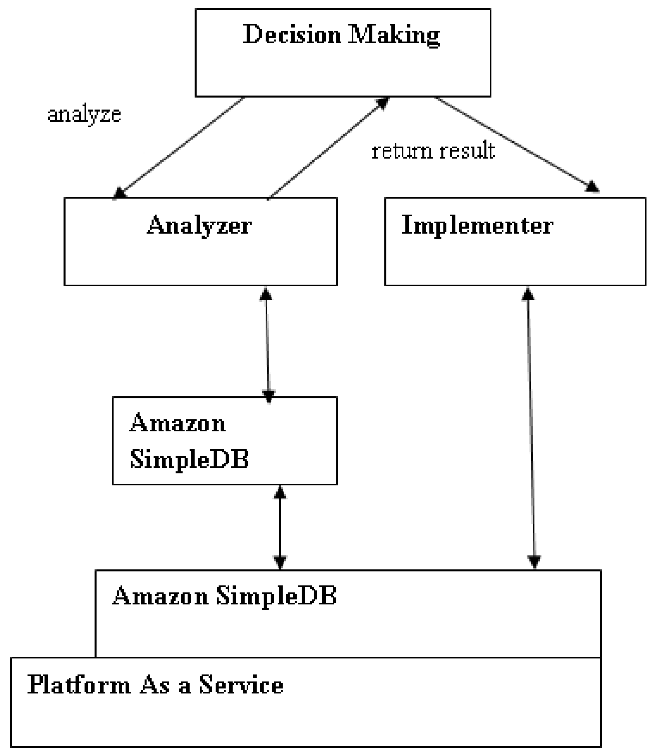

Figure 1: Workload-Aware Elasticity Framework

Figure 1 illustrates the design of the Workload-Aware Elasticity Framework. There are three different modules, analyzer, decision maker, and Implementer.

Analyzer : - It analyzes the load on the Amazon SimpleDB domains running in the Amazon Cloud. It actually collects the number of requests on each domain and calculates the average load on each domain.

Decision Maker :- Based on the average number of requests on a domain, the decision maker decides whether the load on each domain is acceptable or not. If the average load on each of the domains is greater or lesser than the threshold value, the domain is added or removed respectively. The load distribution algorithm is run. Again, the load of each domain is checked so that it falls in the expected range(MinLoad to MaxLoad). Again, if the load on any domain is greater than MaxLoad, the domain is increased and if it is lesser than MinLoad the domain count is decreased. These steps from three to five are repeated until we get a configuration where MinLoad $<$ load on any domain $<$ MaxLoad . The same configuration is returned as the final configuration with a minimum number of the domain.

Implementer :- Implementer which accepts the final configuration from decision maker and implements it.

Algorithm 1: Workload-aware elasticity algorithm

Workload-Aware Elasticity Algorithm

```
Input: 1. Number of Domains.
       2. Set of Warehouse
Output: domains with the optimized load
        balancing and optimized association .
begin
    Averageload = TotalLoad/Numberofdomains;
    if Average load > Max load then
        Numberofdomains = Numberofdomains
        + 1;
        else
            if Average load < Min load then
                Numberofdomains =
                Numberofdomains - 1;
            end
        end
    end
    repeat
        Call loaddistribution();
        if loadonanydomain > Max load then
            Numberofdomains =
            Numberofdomains + 1;
            else
                if loadonanydomain < Min load
                then
                    Numberofdomains =
                    Numberofdomains - 1;
                end
            end
        end
    until Minload < domainload < Maxload;
end
```

7 Performance Analysis of Algorithm

Performance of the workload-distribution algorithm depends majorly on 'r' and 'T'. Thus, the total time complexity can be stated as below.

$$T = \mathcal{O}(w.p) + \mathcal{O}(r.T) + \mathcal{O}(plogp) \qquad (1)$$

since w, p < r, T

$$T = \mathcal{O}(r.T) \qquad (2)$$

Let D be the number of domains. Performance of the workload-aware elasticity algorithm depends only on d. So, the complexity of the above stated algorithm is

$$T = \mathcal{O}(d) \qquad (3)$$

8 Performance Evaluation

We demonstrate the elasticity and scalability of this system by showing the performance evaluation of prototype implementation on Amazon SimpleDB. We experimentally evaluates the performance of our algorithm, on a machine running in the Amazon Web Services Elastic Compute Cloud (EC2) infrastructure. We evaluate the performance using the TPC-C benchmark.

8.1 Experimental setup

We perform evaluations on a scalable database layer, with the Amazon SimpleDB running in the Amazon Cloud. Table 2 shows experimental setting. Amazon EC2 offers different types of virtual machine instances. We perform experimental evaluation with one medium instance (with 30GB memory, 26(8 core * 3.25 unit) as compute units, 160GB (2*80GB SSD), 64 bit platform, M3 General Purpose Double Extra Large. M3 General Purpose Double Extra Large costs $ 1.064 per hour at the time of our experiments. We use multithreaded requests for simulating the transactional load and number of users.

8.2 TPC-C benchmark

It is an update intensive workload. There are a total of nine tables and five different types of transactions. Figure 2 shows TPC-C schema. These nine tables in the TPC-C schema are mapped to a domain in simpleDB. TPC-C database was populated with 15 warehouses. These nine tables are horizontally partitioned using the load distribution algorithm. In our experimental setting we have 3 warehouses per domain. We have a total number of 5 domains.

Table 2: Experimental Setting

Number of Machines	Environ ment	Description
1 (Master)	CPU	M3 General Purpose Double Extra Large 26(8 core * 3.25unit)
	Memory	30GB DDR2
	Hard Disk	160GB(2*80)
All	OS	Windows 8
	.NET Framework	4.0
	NO SQL Database	Amazon SimpleDB

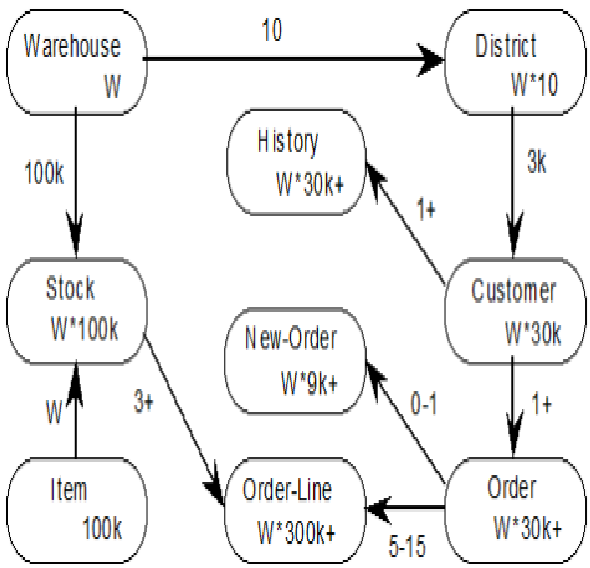

Figure 2: TPC-C Schema

8.3 Mapping of TPC-C to Cloud

TPC-C was designed for web applications with a relational database as a backend. Therefore, to achieve performance in NoSQL data stores, we need to map these relational databases to the data model of Amazon SimpleDB. We have outlined the data model of Amazon SimpleDB from TPC-C. The TPC-C normalized scheme contains nine tables (warehouse, district, item, stock, customer, orderline, and order). To map the normalized data model to Amazon SimpleDB, we combine the nine tables into one domain of Amazon SimpleDB. Figure 3 shows mapping of TPC-C schema to Amazon SimpleDB.

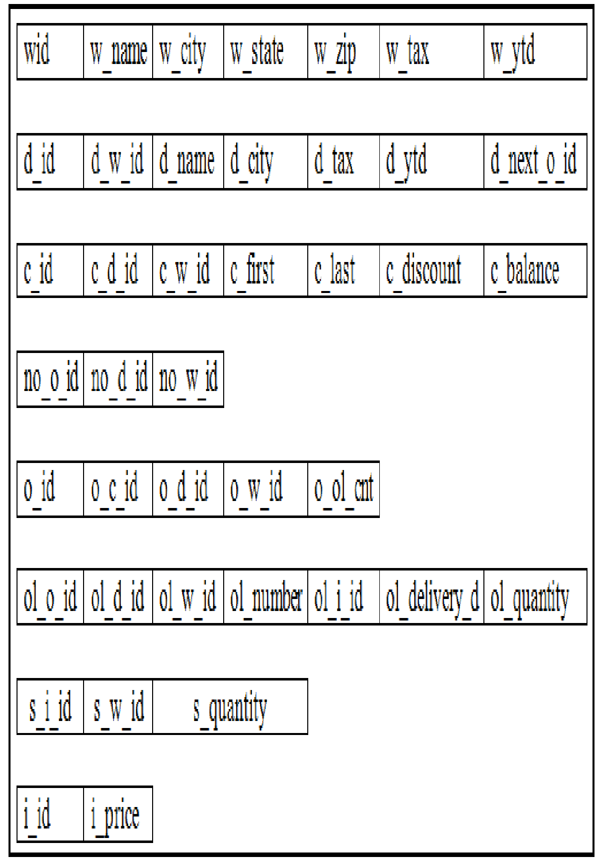

Figure 3: Mapping of TPC-C Schema to Cloud (Amazon SimpleDB Domain)

8.4 Elasticity Evaluation

In this section, we are varying the concurrent users on medium instance running in the Amazon Cloud. The Amazon SimpleDB database is populated with 15 warehouses. We are varying the users from 50 to 450 in steps of 50. The aim of carrying out this experiment is to validate the scalability and elasticity with a varying number of concurrent users. Our workload-aware elasticity algorithm uses three different types of load distribution algorithm. We have identified metrics as throughput for performance evaluation. We define throughput as number of transactions processed per second. On the x-axis we have taken concurrent number of users and on the y-axis we have taken throughput. After carrying out the experiments, we have analyzed the results and observed that our workload-aware load distribution algorithm and the elasticity algorithm gives a higher throughput with a minimum number of resources. We are comparing our load distribution algorithm with two different types of partitioning techniques i) schema level ii) graph partitioning. Figure 4 shows the throughput for schema level, graph, and workload-aware partitioning. Blue line shows the throughput of graph partitioning. Red line shows the throughput of schema level partitiong and black line shows the throughput for workload-aware approach. From figure 4, we can observe, workload-aware partitioning gives higher throughput as compared to schema level and graph partitioning. In schema level partitioning, partitions are formed statically. Once the partition is formed, it is constant. Therefore distributed transactions occurs. Distributed transactions hampers scalability. Graph partitioning uses workload-aware approach. But here workload is already identified in advance, and partitions are formed statically. Once the partitions are formed, they do not change. Therefore distributed transactions occur, which hampers the scalability. In our workload-aware partitioning approach, partitions are formed after analyzing the transaction logs. So the partitions are changing dynamically. Therefore less number of distributed transactions occurs in comparison with schema level and graph partitioning which in turn increases throughput. Figure 5 shows response time for schema level, graph, and workload-aware partitioning. From figure 5, we can observe the response time for workload-aware partitioning is lesser than schema level and graph partitioning.

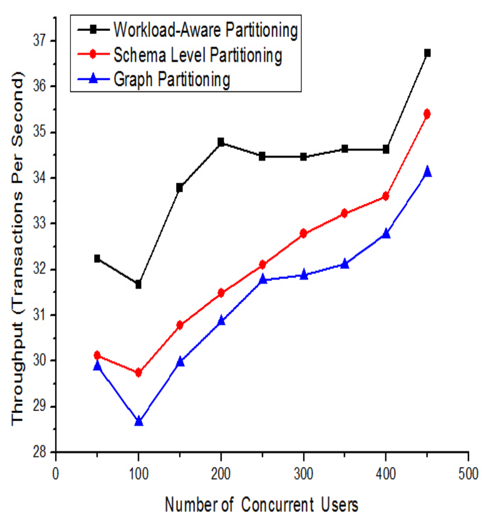

Figure 4: System throughput for varying number of concurrent clients for workload-aware, schema level, and graph partitioning.

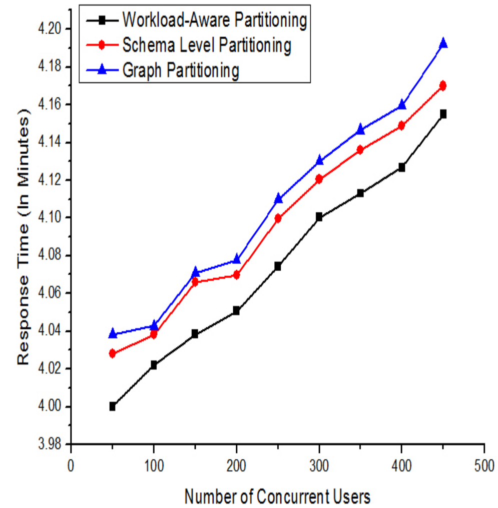

Figure 5: Response Time for varying number of concurrent clients for workload-aware, schema level, and graph partitioning.

9 Statistical Model and Data Analysis

9.1 Regression

In this section, we present the exhaustive analysis of our elasticity algorithm which uses our workload-aware load distribution algorithm. The aim of using regression is to predict the output of the response variable. Our response variable is throughput. We have identified a dependent variable as throughput and an independent variable as the number of users. With regression we are modelling the relationship between the number of users and throughput. Dependent and independent variables were entered and the coefficient of determination (R2) was found through F test. The coefficient of determination R^2 obtained is 0.79 and adjusted R^2 is 0.76. Table 3 shows the value of R^2. The coefficients of predictors gives the following equation of the form for throughput. Table 4 shows the equation for throughput.

$$Throughput = 2.885 * x + 7639 \tag{4}$$

9.2 Multiple Perceptron Model

The experimental models and correlations developed by classical methods are complex, less accurate, and difficult to predict the nonlinear relationship between the dependent and independent variables. The artificial neural network is used to carry out nonlinear sta-

Table 3: Analysis of workload-aware elasticity using regression.

Model	R	R Square	Adjusted R Square
1	.889^a	.790	.760

Table 4: Throughput of workload-aware elasticity using regression.

Model	Unstandardized Coefficients		Standardized Coefficients	t	Sig.
	B	Std. Error	Beta		
Constant Users	7639.639 2.885	158.222 .562	.889	48.284 5.130	.000 .001

tistical modeling. It includes many advantages such as ability to implicitly detect nonlinear relationships between the dependent and independent variables, accuracy, and efficiency rather than the conventional statistical technique. The input data is partitioned into 70% of data set as a training data set and 30% of testing data set. We have chosen the number of hidden layers of 1 neuron. The supervised learning paradigm, in, which a network is trained for a particular set of inputs to produce the desired outputs.

Table 5: Analysis of workload-aware elasticity using multiple perceptron model.

Training	Sum of Squares Error	1.698
	Relative Error	0.485
	Training Time	0:00:00.00
	Stopping Rule Used	1 consecutive step(s) with no decrease in error
Testing	Relative Error	.^{b}
	Sum of Squares Error	0.020

As seen from the model summary of the multilayer perceptron model, the value of sum of squares error is 0.020 and the mean square error is 0.00285. The value of R^2 is 98 percent and adjusted R^2 is 97 percent. After analyzing these statistics from multiple linear regression and the artificial neural network, we observed that the artificial neural network gives less error and more accuracy.

10 Conclusion and Further Work

We have presented the load distribution strategy, which will distribute the combinations to the domains. We also presented the workload-aware elasticity algorithm, which will add and remove domains as per the requirements of the user so that resources are utilized. We provide the implementation of load distribution and the elasticity algorithm on Amazon SimpleDB running in the Amazon Cloud. These two algorithms are evaluated using TPC-C benchmarks. We have also portrayed the detailed analysis of our algorithm using different statistical methods such as regression and neural networks. We have analyzed the results and observed that that these two algorithms give us a higher throughput with a minimum number of resources. The advantage of our load distribution and the workload-aware elasticity algorithm over the existing strategies is that the load is distributed as per the data access pattern of web applications so that the numbers of distributed transactions are minimized with the minimum number of resources. But on the other hand, the disadvantage of using these techniques are that the combinations are formed dynamically based on analysis so the migration of data is an overhead. We are planning further to work on incorporating the Markov Decision Process for decision making.

References:

[1] Agrawal, Divyakant and El Abbadi, Amr and Antony, Shyam and Das Sudipto(2010). ‘Data management challenges in cloud computing infrastructures’. Databases in Networked Information Systems, Springer, pp. 1–10.

[2] Aguilera, Marcos K and Merchant, Arif and Shah, Mehul and Veitch, Alistair and Karamanolis, Christos(2007). ‘Sinfonia: a new paradigm for building scalable distributed systems’. ACM SIGOPS Operating Systems Review, vol.41, num. 6, pp. 159–174.

[3] Ahirrao, Swati and Ingle, Rajesh. ‘Scalable transactions in Cloud Data Stores’. Advance Computing Conference (IACC), 2013 IEEE 3rd International, pp. 116–119.

[4] Ahirrao, Swati and Ingle, Rajesh. ‘Dynamic workload-aware partitioning in OLTP cloud data stores’. Journal of Theoretical and Applied Information Technology, volume 60(1), pp. 89–94.

[5] Baker, Jason and Bond, Chris and Corbett, James C and Furman, JJ and Khorlin, Andrey and Larson, James and Léon, Jean-Michel and Li, Yawei and Lloyd, Alexander and Yushprakh, Vadim (2011). ‘Megastore: Providing Scalable, Highly Available Storage for Interactive Services’. CIDR, volume 11, pp. 223-234.

[6] Bernstein, Philip A and Cseri, Istvan and Dani, Nishant and Ellis, Nigel and Kalhan, Ajay and Kakivaya, Gopal and Lomet, David B and

Manne, Ramesh and Novik, Lev and Talius, Tomas (2011). 'Adapting microsoft SQL server for cloud computing'. Data Engineering(ICDE), 2011 IEEE 27th International Conference on, pp. 1255–1263.

[7] Curino, Carlo and Jones, Evan PC and Popa, Raluca Ada and Malviya, Nirmesh and Wu, Eugene and Madden, Sam and Balakrishnan, Hari and Zeldovich, Nickolai(2010). 'Relational cloud: A database-as-a-service for the cloud'.

[8] Curino, Carlo and Jones, Evan and Zhang, Yang and Madden, Sam (2010). 'Schism: a workload-driven approach to database replication and partitioning'. Proceedings of the VLDB Endowment, volume 3, pp. 48–57.

[9] Cruz, Francisco and Maia, Francisco and Matos, Miguel and Oliveira, Rui and Paulo, Joao and Pereira, José and Vilaça, Ricardo (2013). 'MeT: workload aware elasticity for NoSQL'. Proceedings of the 8th ACM European Conference on Computer Systems, pp. 183–196.

[10] Chang, Fay and Dean, Jeffrey and Ghemawat, Sanjay and Hsieh, Wilson C and Wallach, Deborah A and Burrows, Mike and Chandra, Tushar and Fikes, Andrew and Gruber, Robert E (2008). 'Bigtable: A distributed storage system for structured data'. ACM Transactions on Computer Systems, volume 26.

[11] Das Sudipto and Agrawal, Divyakant and El Abbadi, Amr. 'G-store: a scalable data store for transactional multi key access in the cloud'. Proceedings of the 1st ACM symposium on Cloud computing, pp. 163–174.

[12] Das Sudipto and Agarwal, Shashank and Agrawal, Divyakant and El Abbadi, Amr and Bunch, Chris and Chohan, Navraj and Krintz, Chandra and Chohan, Jovan and Kupferman, Jonathan and Lakhina, Puneet and others (2010). 'Elastras: An elastic, scalable, and self managing transactional database for the cloud'. Technical Report 2010-04, CS, UCSB.

[13] Das, Sudipto and Agrawal, Divyakant and El Abbadi, Amr (2009). 'Elastras: An elastic transactional data store in the cloud'. USENIX HotCloud, volume 2.

[14] DeCandia, Giuseppe and Hastorun, Deniz and Jampani, Madan and Kakulapati, Gunavardhan and Lakshman, Avinash and Pilchin, Alex and Sivasubramanian, Swaminathan and Vosshall, Peter and Vogels, Werner (2012). 'Dynamo: amazon's highly available key-value store'. ACM SIGOPS Operating Systems Review, volume 41, pp. 205–220.

[15] Kassela, Evie and Boumpouka, Christina and Konstantinou, Ioannis and Koziris, Nectarios. 'Automated workload-aware elasticity of NoSQL clusters in the cloud'. Big Data, 2014 IEEE International Conference on, pp. 195–200.

[16] Levandoski, Justin J and Lomet, David B and Mokbel, Mohamed F and Zhao, Kevin (2011). 'Deuteronomy: Transaction Support for Cloud Data', CIDR, volume 11, pp. 123–133.

[17] Minhas, Umar Farooq and Liu, Rui and Aboulnaga, Ashraf and Salem, Kenneth and Ng, Jonathan and Robertson, Sean. 'Elastic scale-out for partition-based database systems' Data Engineering Workshops (ICDEW), 2012 IEEE 28th International Conference on, pp. 281–288.

[18] Naskos, Athanasios and Stachtiari, Emmanouela and Gounaris, Anastasios and Katsaros, Panagiotis and Tsoumakos, Dimitrios and Konstantinou, Ioannis and Sioutas, Spyros. 'Cloud elasticity using probabilistic model checking', arXiv preprint arXiv:1405.4699.

[19] Rafiq, Taha (2013). 'Elasca: Workload-Aware Elastic Scalability for Partition Based Database Systems'

[20] Silberstein, Adam and Chen, Jianjun and Lomax, David and McMillan, Brad and Mortazavi, Masood and Narayan, PPS and Ramakrishnan, Raghu and Sears, Russell (2012). 'Pnuts in flight: Web-scale data serving at yahoo'. Internet Computing, IEEE, volume 16, pp. 13–23.

[21] Serafini, M., Mansour, E., Aboulnaga, A., Salem, K., Rafiq, Taha, Minhas, U. F. (2014). 'Accordion: Elastic scalability for database systems supporting distributed transactions'. Proceedings of the VLDB Endowment, volume 7.

[22] Tsoumakos, Dimitrios and Konstantinou, Ioannis and Boumpouka, Christina and Sioutas, Spyros and Koziris, Nectarios. 'Automated, elastic resource provisioning for nosql clusters using tiramola'. Cluster, Cloud and Grid Computing (CCGrid), 2013 13th IEEE/ACM International Symposium on, IEEE, 2013, pp. 34–41.

[23] Wei, Zhou and Pierre, Guillaume and Chi, Chi-Hung(2012) 'CloudTPS: Scalable transactions for web applications in the cloud'. Services Computing, IEEE Transactions on, volume 5, pp. 525–539.

[24] TPC-C Benchmark Standard specification revision 5.11, February 2010 http://www.tpc.org/tpcc/ (Accessed October 2014)

[25] Leutenegger, Scott T and Dias, Daniel. 'A modeling study of the TPC-C benchmark', Volume 22, 1993, ACM (Accessed October 2014)

[26] IBM SPSS version 20: Reference Guide for IBM SPSS Statistics, http://www.ibm.com/software/analytics/spss/ (Accessed October 2014)

[27] Amazon SimpleDB Documentation, 2010 http://aws.amazon.com/simpledb/ (Accessed October 2014)

[28] P. J. Sadalage,'NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence', Addison-Wesley, 2012 (Accessed October 2014)