A Self-Tuning Framework for Cloud Storage Clusters

Siba Mohammad, Eike Schallehn, and Gunter Saake

Institute of Technical and Business Information Systems,
Otto-von-Guericke-University of Magdeburg
Building 29, Universitätsplatz 2, 39106 Magdeburg, Germany
{smohamma,eike,saake}@iti.cs.uni-magdeburg.de

Abstract. The well-known problems of tuning and self-tuning of data management systems are amplified in the context of Cloud environments that promise self management along with properties like elasticity and scalability. The intricate criteria of Cloud storage systems such as their modular, distributed, and multi-layered architecture add to the complexity of the tuning and self-tuning process. In this paper, we provide an architecture for a self-tuning framework for Cloud data storage clusters. The framework consists of components to observe and model certain performance criteria and a decision model to adjust tuning parameters according to specified requirements. As part of its implementation, we provide an overview on benchmarking and performance modeling components along with experimental results.

Keywords: Cloud Storage Clusters, Self-Tuning, Performance Modelling, Regression Analytic, Benchmarking

1 Introduction

Although, conventional database systems are used for Cloud applications where strict consistency and transactional processing are needed, properties of the Cloud environment (multi tenancy, component failure, etc.) and the needs of its application (scalability, availability, and fault tolerance, etc.) resulted in a new breed of data storage systems. These systems were primarily developed for internal use by companies such as Google, Amazon, Facebook, etc. For Cloud-based and big data applications, Cloud storage systems are the storage systems of choice to meet the mentioned requirements.

From the architectural point of view, these systems have a modular, multi-layered architecture. According to application needs, multiple component systems are combined together to provide needed functionalities. As a basic component, a distributed file system (e.g. Google and Hadoop file systems) supports scalable, fault tolerant data storage and access. On top of it, typically lays a structured-data storage system (e.g. Bigtable [4], Cassandra [10]). Systems of this layer structure data in non-relational data models; key-value model being the dominant. They also provide API access and SQL-like query languages

(e.g. CQL). Because of the non-relational nature of the underlying data model, RDBMS-style aggregations and joins were typically not supported. To perform such operations and more complex data analytic, a distributed processing system is used on top of the previous layers; Map Reduce framework being the dominant. Though later versions of Cloud storage systems support joins and aggregations, these operations are internally transformed into Map Reduce jobs (e.g. Hive and Pig)[1]. A more detailed architectural overview and a classification of cloud storage systems can be found in [6].

Though Cloud storage systems were developed to be self-managing regarding many aspects, e.g. dynamically adding or removing resources, there are still numerous decisions to be made to actually fit the requirements of given applications to provide suitable performance. The contributions of this paper are as follows:

- As a precondition for the proposed framework, we relate tasks of (self-)tuning to layers and sub-clusters within a typical Cloud storage architecture.
- We describe the top-level view of our framework applicable to various optimisation goals and parameters describing the application or the configuration.
- Based on measured and/or modelled performance of applications, we describe a decision model suitable for self-tuning of configuration parameters.
- More specifically we address the common problem of adjusting the size of (sub-)clusters in an experimental evaluation.

The rest of the paper is organized as follows. First, we provide a motivational scenario in Sect. 2. In Sect. 3, we give an overview of the framework. After that, we provide more detailed description of benchmarking and modelling components in Sect. 4. Then, we discuss the current experimental results in Sect. 5. After that, we provide an overview of related work in Sect. 6. The paper ends with conclusion and future work in Sect. 7

2 Motivation

The tuning and self-tuning of Cloud storage systems has gained more attention by both industrial and academic research [24, 26, 22, 20, 25]. Because of the typical shared nothing architectures with data partitioning and replication, some performance aspects can be easily addressed for the overall system. Nevertheless, the typical multi-layered distributed architecture of several component systems adds complexity to the tuning tasks. Moreover, if there are several applications with different and possibly changing requirements, using the same data storage cluster, there is little chance to tune for a specific application.

Based on this assumption our approach applies the concept of creating dedicated sub-clusters for single applications/workloads or groups of workloads with similar requirements as shown Fig. 1. Here, application requirements can be mapped to different tuning knobs to achieve applications optimisation goals. In

[1] https://cwiki.apache.org/confluence/display/Hive/LanguageManual+Joins

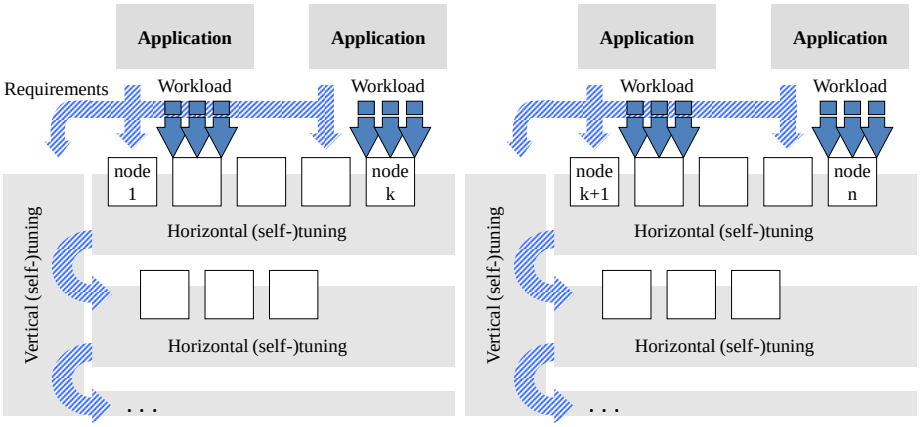


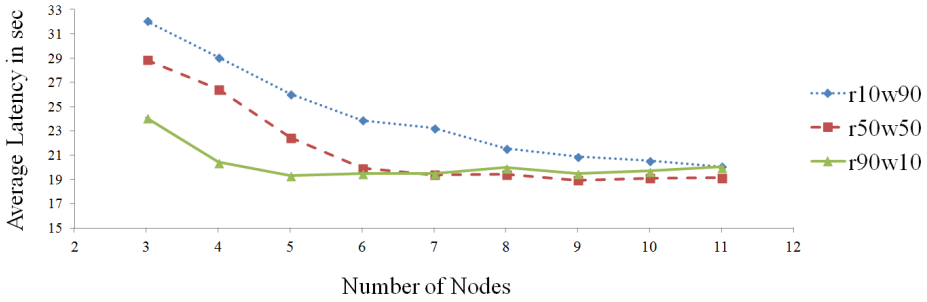
Fig. 1: (Self-)Tuning for Cloud Storage Systems

the multi-layered, modular architecture, these requirements can now be handled on two dimensions. The first one, we call horizontal (self-)tuning, which takes place within layer. The horizontal (self-)tuning includes aspects such as partitioning, load balancing, replication, update strategies, and automatic scaling, etc. Problems on this dimension are better supported because of the homogeneous processes of a single component type within one layer. The second one, we call vertical (self-)tuning, which is carried out across layers. Vertical (self-)tuning includes the mapping of application requirements expressed as optimization goals, service levels, etc. to specific tuning knobs on each level of the storage architecture. For the remainder of the paper, we focus on aspects of horizontal self-tuning.

For illustration purposes, consider the example shown in Fig. 2, which is based on data gathered from experiments described, in more details, in Sect. 4. As shown in Fig. 2a, there are three different workloads, being read-heavy (r90w10), evenly mixed (r50w50), and write-heavy (r10w90) showing very different performance characteristics measured for different cluster sizes (overall latency for entire workload, average over 5 independent runs). Decisions that can be made based on this data include:

- finding the best cluster size for a single workload, e.g. indicated by a global mimimum within resource restrictions, or
- creating an optimal setup of sub-clusters for all workloads.

Based on our overall approach, the latter is of great importance, and the results of optimal sub-cluster configurations given different node constraints are shown in Fig. 2b. For reasons of simplicity, the sum of the overall latency was used as an optimisation goal, though different aggregation functions are conceivable. The optimisation of this given problem can easily be done by brute force algorithms, because it is linear and discrete in the number of nodes and only exponential in the number of workloads. More sophisticated approaches may be required for



(a) Measured latency characteristics for different workloads

(b) Optimal allocation of nodes to workloads

Number of Nodes	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
Nodes for r10w90	3	3	4	3	4	5	5	6	6	8	8	9	9	10	11
Nodes for r50w50	3	3	3	5	5	5	6	6	6	6	6	6	7	7	7
Nodes for r90w10	3	4	4	4	4	4	4	4	5	4	5	5	5	5	5
Latency in sec	85	81.4	78.3	74.9	71.9	68.9	66.3	64.2	63.1	61.8	60.8	60.1	59.6	59.2	58.7

Fig. 2: Optimal allocation of nodes for three workloads 10% read and 90% write (r10w90), 50% read and 50% write (r50w50), and 90% read and 10% write (r90w10) for different cluster sizes

non-discrete cases and those involving more complex parameter combinations. Furthermore, the general framework presented in this paper will have to deal with the fact, that measurements gathered from monitoring the system or test runs are incomplete within the huge space of possible parameter combinations. To predict the performance, a model of it needs to be derived from the available data.

3 A Framework for Tuning Cloud Data Storage Cluster

In this section, we discuss our approach of addressing the aforementioned problem scenario. After formalizing the problem, we illustrate different components of our infrastructure and their functionalities.

3.1 Problem Statement and Solution Approach

For our framework, we define the general optimization approach as follows: the optimisation goal *opt* is to find a cluster configuration c out of a set of possible configurations CC that minimizes (assuming a standard form of the problem) the costs for all workloads w of a set of workloads WL that need to be supported by the overall cluster.

$$opt = \underset{c \in CC}{\text{minimize}} \quad \Gamma_{w \in WL} cost(c, w)$$

Here Γ represents some aggregation function suitable to the given cost components considered, e.g. sum for energy consumption or average or maximum for response time. Constraints can be defined on the cluster configuration as discussed below:

Cluster Configurations in CC. These independent variables are controlled variables and represent the actual knobs that can be used to achieve the optimisation goal. Typical configuration aspects are for instance the cluster size, hardware being used, replication factor and other database parameters, etc. Formally, c can be described as an n-tuple that holds relevant parameters as components, e.g. $c_1 = \{cn = 10, rf = 3\}$ for a cluster of 10 nodes and a replication factor of 3.

Workload characteristics in WL. These independent variables describe the application, but are not controlled by the systems administrators or developers, i.e. though they may be highly dynamic, they can not be changed deliberately to achieve an optimisation goal. These include for instance workload characteristics, access frequencies, user numbers, data volume and schema, etc., which, again, can be modelled as an n-tuple, e.g. $wl_1 = \{r = 90, w = 10, sf = 10, nc = 5\}$ for a workload having 90% read operations performing on a 10GB database of 5 column families.

Optimisation Goal *opt*. The dependent variables used in prediction models for system optimisation are typically those, for which an optimal value should be achieved. For Cloud storage, these may include variables such as throughput, latency, energy consumption, resource utilisation, consistency, etc. The optimisation task, for which the model is being used, may be multi- objective, requiring specific techniques not discussed in this paper.

Not all of the possible parameters describing a workload or a cluster configuration may be relevant or desirable to consider in a given application scenario. Furthermore, there might be strong correlations between some of the independent variables, which can be used to simplify the models creation and application. While we discuss techniques to create a performance model in this paper, here it is not our intention to investigate the complex space of variables and their dependencies in its entirety, but rather focus on – in our opinion – a most relevant subspace to discuss the modelling and prediction techniques, namely finding the optimal size of sub-clusters for a given set of workloads. To achieve this, we express the relation between performance metrics of a workload w with cluster configuration c as a cost function where N is the total number of nodes in the infrastructure:

$$opt = \sum_{w \in WL} cost(w, n_w)) \rightarrow min$$

subject to

$$\sum_{w \in WL} n_w \leq N$$

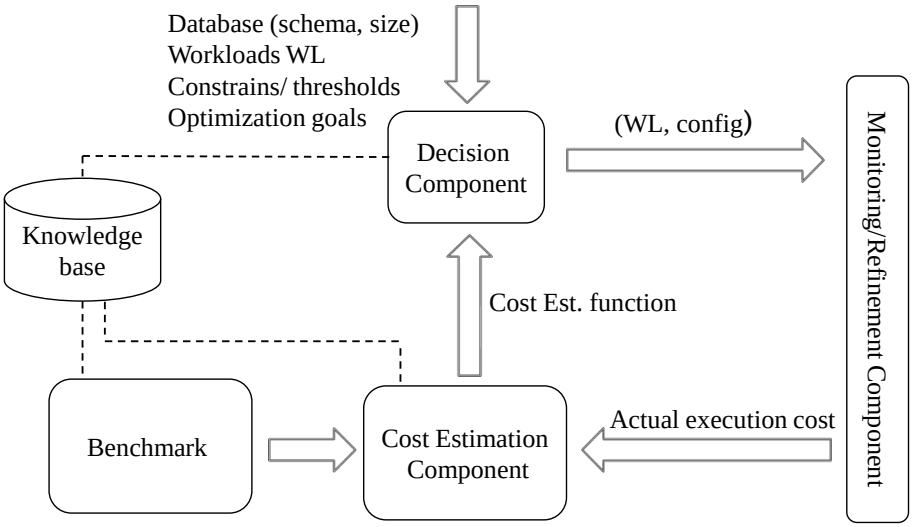


Fig. 3: Self-Tuning Framework for Cloud Data Management Systems

3.2 Framework Architecture

As we illustrate in the Fig 3, our framework is composed from the following components:

Benchmark. The purpose of this component is to generate the training data needed to model the performance of the data storage cluster for a certain workload with different cluster sizes or possibly different configurations.

Cost Estimation Component. This component uses statistical-based data-driven modeling to build performance models as mathematical functions. These functions are derived by regression techniques done on statistical data gathered from the benchmarking phase.

Decision Component. Tuning knobs are expressed as different independent variables during the modeling process. Based on conditions derived from workload thresholds, this component performs a filtering process on the value-space of the independent variables. Then it solves the optimization problem of the cost models, based on the optimisation goals of different workloads, to find preferable values of the tuning knobs.

Monitoring/Refinement Component This components is responsible for adding measurement to the knowledge-base and initiating a re-modeling process if

$$|Actual - Prediction| \geq threshold$$

Knowledge-base Stores information for reuse by the framework. Information includes workloads description (i.e. schema and data access pattern) and cost models.

4 Benchmarking and Performance Modeling

In this section, we provide our current theoretical and empirical results in implementing the tuning framework. As first steps in this direction, we developed a benchmark and a cost estimation component. Our approach for predicting the performance of a Cloud database cluster is to benchmark the cluster based on several runs of workloads. Then, build a cost(performance) model using regression analytic techniques. We provide more details in the following subsections.

4.1 Benchmarking Cloud Storage Clusters

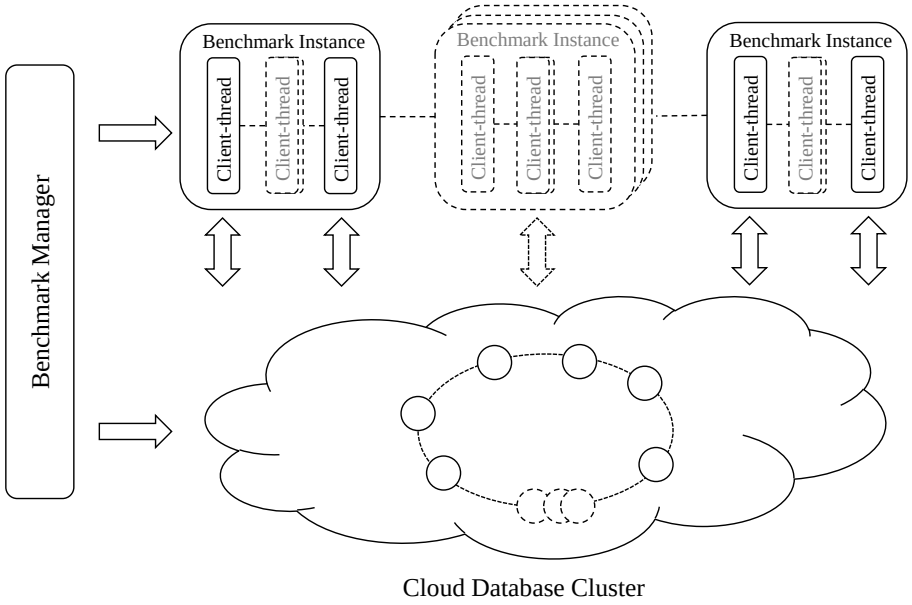


Fig. 4: Benchmark Architecture

The purpose of our benchmark is to generate data to model the performance of a Cloud storage cluster for a certain workload with different cluster sizes. Many benchmarks [4, 7, 1, 2] exist for comparing the performance of several database systems for certain workloads or even checking the performance of one database systems with different workloads to identify bottlenecks. Our implementation supports the typical requirements of a benchmark such as allowing workload configuration: read/write ratio, data size, throughput, etc. and it automates the testing process for an increasing number of database system cluster size. We provide the architecture of our benchmark in Fig. 4. The essential component of the benchmark is the benchmark manager which is responsible for starting

the benchmark instances. It also acts as data storage system cluster controller, performing operations of:

- Preparing and starting the database cluster for certain number of nodes.
- Creating the database schema, generating and loading data before the actual workload, if needed.
- Rebooting database and operating system to flush the file system caches, main memory and CPU caches in the case of a cold run.

The benchmark instances are responsible for starting the workload and collecting measurements of performance. We designed the benchmark to allow specifying the following workload characteristics:

- Database schema (table, number of columns), record size and replication factor.
- Data access specifications: read/write ratio, number of rows to be read or written, throughput (number of concurrent access).

As specified by the workload setting and based on the replication factor and the maximum number of nodes intended for the data storage cluster, several phases of the benchmark are performed. Each phase is defined by the number of nodes in the cluster (cluster size). The cluster size varies between the data replication factor and the maximum number of nodes available. In each phase, multiple remote benchmark-instances are started by the benchmark-manager using SSH (Secure Socket Shell). Each benchmark-instance starts multiple client-threads depending on the number of the CPU cores and the memory size of the host machine. After the workload ends, statistical data describing the performance are retrieved from all client-generator machines and combined in one output file. Within one phase, the benchmark automatically repeats the experiment a number of times defined by the user and the average measurement is used for the modeling process. After the experiment is done for the current number of nodes, the benchmark starts again for next number of nodes.

4.2 Performance Modelling

This step includes analyzing the collected data to discover the underlying model. There are several machine learning techniques used for modeling. These include clustering, tree-based, genetic evolutionary algorithms and neural networks [9]. Regression analytic techniques are considered one of the simplest techniques for predictive modeling. Their process relies on statistical and regression analysis to find a formula or mathematical model to represent the relationship between a dependent variable being the measured performance aspect (e.g. latency) and one or more application-specific requirement such as workload criteria (e.g. read/write ratio) or system configuration aspects (e.g. database cluster size).

As stated by Mark Kotanchek et al. [8], there is an infinite number of predictive models that fit a finite data set. Our goal is to find a model that fits the

data and has a relatively small error. To achieve this, we use different regression analysis techniques and measure the error rate. The dependent variable in our implementation is the response time or latency and the independent variables are the number of nodes in the database cluster (cluster size) and the read/write ratio.

5 Experiment and Evaluation

Table 1: Software and Hardware Configuration

OS Ubuntu: 13.04 kernel		Version: Linux 3.8.0-35-generic-pae	
CPU: Intel(R) Xeon(R) E5-2650 0 2.00GHz		2 Cores Cache size: 20480 KB	
Disk:	90.18 GB 7200RPM	Memory:	8 GB
Network	100 MBits	Java Version	1.7.0_25
Cassandra Version 1.2.13		Virtual machines for cluster deployment: 11	
Replication factor: 3		Virtual machines for generating workload: 3	

For our experiment, we choose Cassandra [10]. Our approach is database agnostic, and Cassandra was chosen only as an example of Cloud storage systems. Cassandra was designed for internal use by Facebook and was later adopted by Apache. Large clusters of Cassandra are being used by systems like Netflix, Spotify, and eBay[2], etc. Cassandra provides scalable structured data storage, supporting tune-able consistency, column family data model and a SQL-like query language called CQL. We deploy Cassandra on a network of virtual machines in our labs. Configuration of the testbed for this experiment is illustrated in Table 1. For the deployment of the benchmark, we dedicate another set of virtual machines in the same network with the same configuration. With the goal of modeling the performance of the database cluster with different cluster sizes and different workloads, the workloads we tested vary in the read/write percentage. Other workload criteria such as the schema, consistency level, row size, and goal throughput (concurrent accesses) is kept the same. Each workload operates on one column family. Each read or write operation touches one row. Write operations insert randomly generated strings. Read operations are select point queries; the whole row is retrieved. The percentage of read/write operations in the tested workloads are: 0, 10, 30, 50, 70, 90, and 100. Each workload was tested with Cassandra cluster of sizes that vary between 3 (replication factor) and 11 (the maximum number of virtual machines dedicated for the database in our infrastructure). Each experiment is repeated 5 times and the average value is used for the modelling process.

[2] Usecase higlights for Cassandra are found on http://planetcassandra.org and http://www.datastax.com/customers

The result from our experiment is illustrated by the surface in Fig. 5 which represents how the cluster behaves (its latency in ms) with different workloads (characterized by their read/write percentage) and different cluster sizes. We

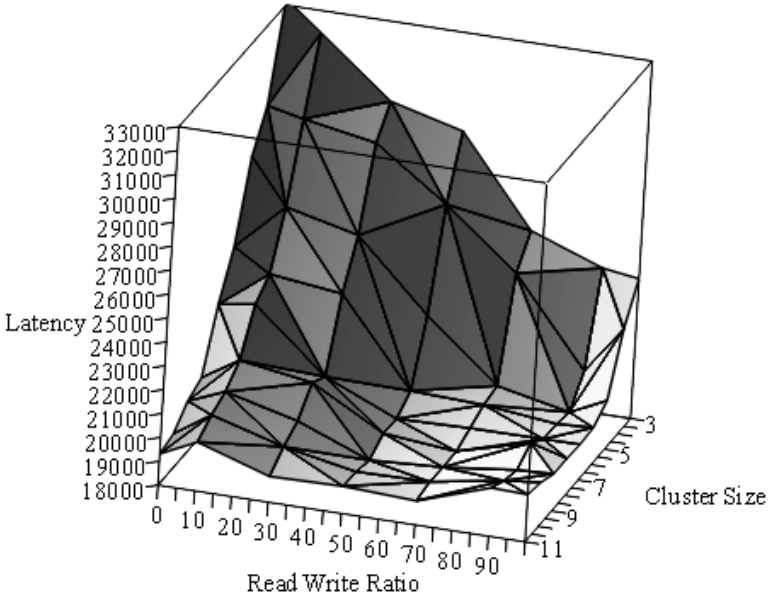


Fig. 5: Benchmarking Results

test several regression analytic techniques: simple linear regression, polynomial regression with several degrees, and exponential regression. As a result from the regression process using the cubic regression gives the best residual standard deviation among the tested techniques, with a slight difference from the quadratic regression. Fig. 6 illustrates the surface representing the resulted cubic model versus the points representing the input measurements.

To validate the result from the regression process, we test the model prediction power against new workloads and calculate the mean absolute error percentage. The cubic model gives high prediction accuracy of 96.4%.

The result from our experiment and evaluation shows that the cubic model has the best residual standard deviation and characterizes the performance with high prediction accuracy. Such a model (even with the low number of independent parameters) can be beneficial to avoid allocating resources to the database cluster that will obtain insignificant benefit from them. Our benchmark allows specifying several workloads parameters, which allows extending the model. However, more experiments must be done to generate the statistical (training data) that is required for creating extended models.

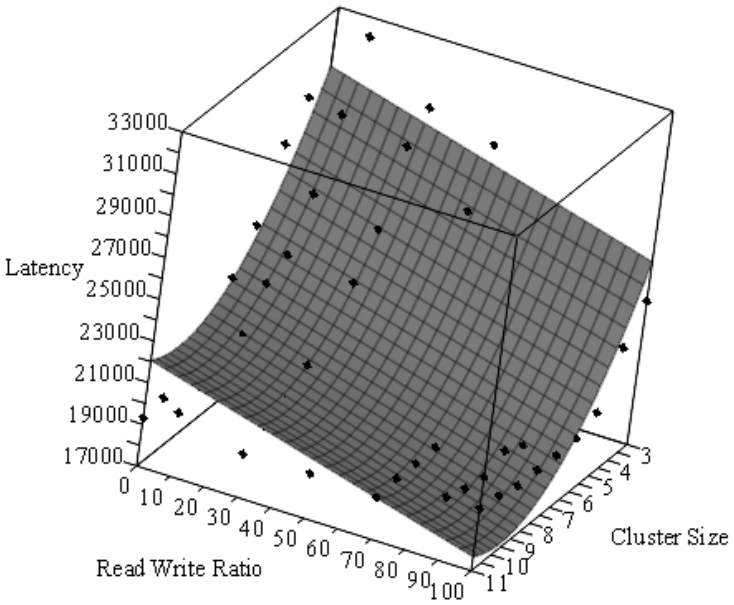


Fig. 6: Regression Analysis Results vs. Input Measurements

6 Related Work

The work described in this paper is based on three areas of research: benchmarking, performance modeling, and (self-)tuning. In the next paragraphs, we provide a short overview of related work of these fields in the context of cloud storage systems.

Related work on benchmarking Cloud storage systems includes general purpose benchmarks [4, 7, 1, 2] which measure latency for different systems and focus on providing details about selecting workloads and benchmark architecture that corresponds to the Cloud environment and its applications. Several studies [12, 3] build on these benchmarks. Another group of benchmarks focus on how a system performance changes with different technical, or platform choices. An example of that can be found in [11] which focuses on analyzing the performance of Cassandra on two platforms using HDD or Flash memory. Another example [12] provides read/write and structured query benchmark which investigates how different implementation techniques of different systems affect the performance. There is also the work of Rabl et al. [5] which provides an overview of the performance impact of different storage architectures. The last group of benchmarks examine specific properties of Cloud storage systems such as replication, consistency, and elasticity. An example of such work can found in [3] which uses different replication strategies and consistency levels and measures their effect on latency and throughput.

Since virtualization is a major technique for cloud environment, a large part of work [14, 18, 17] is dedicated to performance modeling for cloud application in virtualized environments. The work of Noorshams et al. [14] investigates performance modeling for virtualized storage systems. Another example is the work of Kraft et al. [18] in which they present a simple model for predicting the degradation in performance that results from storage devices contention in virtualized environments. Other aspects such as modeling the scalability behavior of network/CPU intensive applications can be found in [17]. Different techniques for building models exists. A part of the research efforts uses machine learning techniques such as [19], which uses Kernel Canonical Correlation Analysis to model the execution time of MapReduce jobs. Another approach can be found in the work of [15] where they present an analytical model of the Spotify storage architecture that allows to estimate the distribution of response time of storage system.

Related work on (self-)tuning for Cloud storage systems falls in two parts. The first one includes tuning database systems for specific workload, optimization goal, or execution environment. Examples of such efforts include work of [26] which aims to reduce energy consumption and thus cooling costs by applying resource aware data placement and migration strategy. A part of work in this category falls under scheduling. Chi et al. perform cost aware scheduling of queries based on service level agreements [22] whereas Polo et al. perform Map Reduce jobs scheduling to maximize resource utilization [20]. The second part of the (self-)tuning efforts for Cloud storage systems is external to the database system and includes tuning the underlying resources to achieve the optimization goals of the database workload. Example of this is the work of [25] which focuses on partitioning the CPU capacity of physical machines among different database appliances. A more general example is the work of Xiong et al. [21] where they perform cost aware resource management. Herodotou et al. developed a self-tuning framework, starfish [24], for big data analytics. An interesting approach incorporates different DBMSs within one system (called DBMS+) and depends on the query optimizer, of the incorporated systems, to perform the tuning process [23]. Then the tuning process selects the appropriate execution plan for each request.

7 Conclusion and Future Work

While Cloud storage systems are good at self-management, e.g. automatically mapping to available resources, there are still many open issues to actually make them self-tuning, i.e. adjust their parameters to application-specific requirements. In this paper, we presented an approach how such self-tuning functionality can be integrated with an according system, by observing, modelling, predicting the performance, and adjusting the configuration depending on a described decision model. The practical evaluation applied Cassandra and focused on the problem of assigning an optimal number of nodes to various workloads running on sub-clusters to achieve the best possible overall latency.

As discussed throughout this paper, many open questions remain. From our point of view, the most important ones are to be related to multi-objective optimisation, which is the standard case for most real-life systems, where some controlled trade-off between related or even contradicting goals has to be found. Furthermore, we currently investigate the effect of heterogeneous environments and their effect on predictability and resource assignment.

References

1. Curino, C., Difallah, D.E., Pavlo, A., Cudre-Mauroux, P.: Benchmarking OLTP/Web Databases in the Cloud: the OLTP Bench Framework. In: Proceedings of the 4th International Workshop on Cloud Data Management, pp. 17-20. ACM, New York (2012)
2. Binnig, B., Kossmann, D., Kraska, T., Loesing, S.: How is the Weather Tomorrow? Towards a Benchmark for the Cloud. In: Proceedings of the 2nd International Workshop on Testing Database Systems, pp. 9:1-9:6. ACM, New York (2009)
3. Wang, H. Li, J., Zhang, H., Zhou, Y.: Benchmarking Replication and Consistency Strategies in Cloud Serving Databases: HBase and Cassandra. In: Big Data Benchmarks, Performance Optimization, and Emerging Hardware 4th and 5th Workshops, pp. 71-82. Springer, Switzerland (2014)
4. Chang, F., Dean, J., Ghemawat, S., Hsieh, W.C., Wallach, D.A., Burrows, M., Chandra, T., Fikes, A., Gruber, R.E.: Bigtable:A Distributed Storage System for Structured Data. ACM Transactions on Computer Systems. 26, 4:1-4:26 (2008)
5. Rabl, T., Gómez-Villamor, S., Sadoghi, M., Muntés-Mulero, V., Jacobsen, H.A., Mankovskii, S.: Solving Big Data Challenges for Enterprise Application Performance Management. PVLDB, 5(12), 1724-1735 (2012)
6. Mohammad, S., Breß, S., Schallehn, E.: Cloud Data Management: a Short Overview and Comparison of Current Approaches. In: 24th GI-Workshop on Foundations of Database, pp. 41–46. CEUR-WS, Aachen (2012)
7. Cooper, B. F., Silberstein, A., Tam, E., Ramakrishnan, R., Sears, R.: Benchmarking Cloud Serving Systems with YCSB. In: 1st ACM Symposium on Cloud computing, pp. 143–154. ACM Press, New York (2010)
8. Kotanchek, M., Smits, G., Vladislavleva, E.: Trustable Symbolic Regression Models: Using Ensembles, Interval Arithmetic and Pareto Fronts to Develop Robust and Trust-aware Models. Genetic Programming Theory and Practice V. pp. 201–220. Springer, US (2008)
9. Matsunaga, A., Fortes, J. A. B.: On the Use of Machine Learning to Predict the Time and Resources Consumed by Applications. In: 10th International Conference on Cluster, Cloud and Grid Computing, pp. 495–504. IEEE Press, New York (2010)
10. Lakshman, A., Malik, P.: Cassandra - A Decentralized Structured Storage System. Operating Systems Review. 44(2), 35–40 (2010)
11. Aplin, P.: Benchmarking Cassandra on Violin - Violin Memory. Technical report, Violin Memory (2013)
12. Shi, Y., Meng, X., Zhao, J., Hu, X., Liu, B., Wang, H.: Benchmarking Cloud-based Data Management Systems. In: 2nd International Workshop on Cloud Data Management, pp. 47–54. ACM Press, New York (2010)
13. Piao, J. T., Yan, J.: Computing Resource Prediction for MapReduce Applications Using Decision Tree. In: Web Technologies and Applications, pp. 570–577. Springer, Heidelberg (2012)

14. Noorshams, Q., Bruhn, D., Kounev, S., Reussner, R.: Predictive performance modeling of virtualized storage systems using optimized statistical regression techniques. In: Proceedings of the 4th International Conference on Performance Engineering, pp. 283-294. ACM, New York(2013)
15. Yanggratoke, R., Kreitz, G., Goldmann, M., Stadler, R.: Predicting Response Times for the Spotify Backend. In: Proceedings of the 8th International Conference on Network and Service Management, pp. 117-125. International Federation for Information Processing, Laxenburg (2013)
16. Kundu, S., Rangaswami, R., Dutta, K., Zhao, M.: Application Performance Modeling in Virtualized Environment. In: Proceedings of the 16th International Symposium on High Performance Computer Architecture (HPCA), pp. 1-10. IEEE, (2010)
17. Chen, X., Ho, C.P., Osman, R., Harrison, P.G., Knottenbelt, W.J.: Understanding, Modeling and improving the performance of Web Applications in Multicore Virtualised Environments. In: Proceedings of the 5th International Conference on Performance Engineering, pp. 197-207. ACM, New York (2014)
18. Kraft, S., Casale, G., Krishnamurthy, D., Greer, D., Kilpatrick, P.: Performance Models of Storage Contention in Cloud Environments. Software & Systems Modeling. 12, 681-704 (2013)
19. Ganapathi, A., Chen, Y., Fox, A., H. Katz, R., Patterson, D.A.: Statistics-Driven Workload Modeling for the Cloud. In: Workshops Proceedings of the 26th International Conference on Data Engineering (ICDE), pp. 87-92. IEEE, California (2010)
20. Polo, J., Castillo, C., Carrera, D., Becerra, Y., Whalley, I., Steinder, M., Torres, J., Ayguadé, E.: Resource-aware Adaptive Scheduling for Mapreduce Clusters. In: Proceedings of the 12th International Conference on Middleware, pp.187-207. Springer, Berlin (2011)
21. Xiong, P., Chi, Y., Shenghuo, Z., Moon, H.J., Calton, P., Hacigümüş, H.: Intelligent Management of Virtualized Resources for Database Systems in Cloud Environment. In: Proceedings of the 27th International Conference on Data Engineering (ICDE), pp. 87-98. IEEE, Washington (2011)
22. Chi, Y., Moon, H.J., Hacigümüş, H.: iCBS: Incremental Cost-based Scheduling Under Piecewise Linear SLAs. Proc. VLDB Endow. 4, 563-574 (2011)
23. Lim, H., Han, Y., Babu, S.: How to Fit when No One Size Fits. In: Proceeding of the 6th Biennial Conference on Innovative Data Systems Research (CIDR), Online Proceedings (2013)
24. Herodotou, H., Lim, H., Luo, G., Borisov, N., Dong, F., Bilgen, F., Babu, S.: Starfish: a Self-tuning System for Big Data Analytics. In: Proceedings of the 5th Biennial Conference on Innovative Data Systems Research (CIDR), pp. 261-272. Online Proceedings (2011)
25. Aboulnaga, A., Salem, K., A. Soror, A., Minhas, U.F., Kokosielis, P., Kamath, S.: Deploying Database Appliances in the Cloud. IEEE Data Eng. Bull. , 32, 13-20 (2009)
26. Goiri, I., Le, K., Nguyen, T. D., Guitart, J., Torres, J., Bianchini, R.: GreenHadoop: Leveraging Green Energy in Data-Processing Frameworks. In: the 7th European Conference on Computer Systems, pp. 5770. ACM, New York (2012)