Towards Automated Configuration of Cloud Storage Gateways: A Data Driven Approach

Sanjeev Sondur and Krishna Kant(✉)

Temple University, Philadelphia, PA 19122, USA
{sanjeev.sondur,kkant}@temple.edu

Abstract. Cloud storage gateways (CSGs) are an essential part of enterprises to take advantage of the scale and flexibility of cloud object store. A CSG provides clients the impression of a locally configured large size block-based storage device, which needs to be mapped to remote cloud storage which is invariably object based. Proper configuration of the cloud storage gateway is extremely challenging because of numerous parameters involved and interactions among them. In this paper, we study this problem for a commercial CSG product that is typical of offerings in the market. We explore how machine learning techniques can be exploited both for the forward problem (i.e. predicting performance from the configuration parameters) and backward problem (i.e. predicting configuration parameter values from the target performance). Based on extensive testing with real world customer workloads, we show that it is possible to achieve excellent prediction accuracy while ensuring that the model is not overfitted to the data.

Keywords: Cloud storage gateway · Object store · Performance · Configuration management · Machine learning

1 Introduction

A Cloud Storage Gateway is an emerging concept in Cloud Storage Solutions; wherein the CSG application is installed on-premise and translates cloud storage object-store APIs such as SOAP or REST to the block I/O-based storage protocols such as SCSI, Fibre Channel, NFS or SMB.

Cloud Storage Gateway (CSG) concept was pioneered by Google [5], and subsequently offered by many leading industry vendors as a Cloud Storage solution. As shown in Fig. 1, CSG appliance connects the client applications running locally to an object store hosted in a remote cloud data center. Although the remote storage could be block based, it is almost universally object based due to many advantages of the cloud model. The advantage of CSG is that while the user data resides on the cloud storage devices, it makes the accesses appear locally going to a SCSI device.

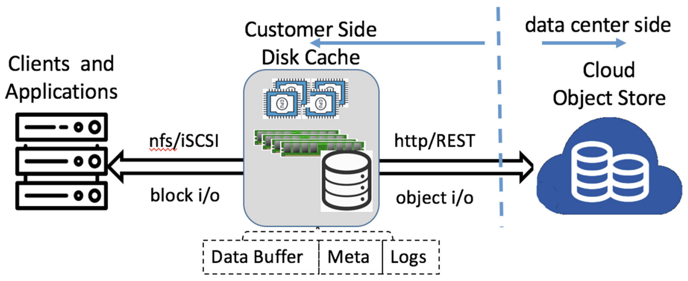

Fig. 1. Cloud storage gateway architecture.

Customers deploy CSG to expand storage capabilities of their local computing infrastructure. For example, a large video animation customer like Disney could work with hundreds of graphics files of size 1GB or more, a financial company may store a large number of fiscal records in medium size files (say 1–10 MB text files). The locally running business workloads would typically persist or retrieve a large amount of such data through the CSG. Since a single CSG may be used by many different business applications with different persist/retrieve patterns, a proper configuration of CSG is a very challenging problem.

Data center operators have a huge amount of operational data collected over time that can be exploited to understand the system configuration parameters and their influence on the operational behavior. We keep the discussion focused by studying the configuration challenges as pertaining to the customer side CSG system, and not the backend data center hosted cloud object store system in Fig. 1. We use a commercially available CSG available from a prominent market vendor. The main goals of our study are as follows:

- Design experiments to collect performance and configuration data for a large number of configurations of this CSG.
- Explore the use of suitable machine learning techniques to build models for solving the forward problem (predicting performance for given configuration parameters), and reverse problem (predicting certain configuration parameters based on target performance).
- Explore how the domain knowledge can be exploited to reduce the configuration space and enhance the accuracy of the predictions.

The key contribution of our work is to demonstrate that we can build robust models for relating user settable system and hardware parameters to the performance of cloud storage gateway and thereby *debugging* the configurations. Even in cases where such automated analysis fails to provide the optimal result, it is expected to yield configurations that are close to optimal and thus can be tuned further with far less effort and time than the prevalent manual approaches whose success entirely depends on the experience of the administrators. To the best of our knowledge, the prior work has predominantly considered performance as a function of workload parameters rather than the user tunable system parameters.

2 Motivation and Challenges

Proper configuration management of complex cyber-systems is a very challenging problem in the real-world, and yet very much under-appreciated in the research community. Misconfigurations in large enterprises often account for up to 80% of the malfunctions and attack vulnerabilities, and routinely consume days of engineer's time to diagnose and fix [10]. Configuration management of data center storage systems can be particularly complex and labor intensive task [3], and CSG is no exception. In addition, CSG configurations combine the complexities inherent in storage system configuration, cache configuration, unpredictable network traffic and the complexities of back-end cloud systems. Similar to other cyber-systems, CSG has many configuration parameters or "knobs" with little clarity on how to set them or what precise impact they have on the output end.

While working with the commercial vendor of a CSG product, we noticed that the most common problems were related to customer complaints about poor performance or I/O time-out errors. We invariably found that on further investigation that most of these complaints were a result of poor understanding of the workload (i.e. request streams) and the configuration parameters of the CSG. The main source of difficulties in configuration management are the numerous parameters with complex inter-dependencies that are mostly unknown or poorly understood with respect to their impact on the overall performance, availability or user experience [11].

This prompted our research into understanding the relationship between various parameters in the Cloud Storage Gateway environment. For example, there are few *uncontrollable* variables such as eviction rate, cloud storage response, internet throughput, etc. and some parameters under the user control such as: workload, hardware characteristics, cache configuration, etc.

It is well known that the storage system performance depends on the workload characteristics, deployed optimizations, and their specific configuration (See [3] & references therein). Configurations also "are often difficult and knowledge-intensive to develop, brittle to various environment and systems changes, and limited in capacity to deal with non-steady-state phenomena [7]."

Cloud storage gateway (CSG) is a relatively new paradigm in cloud storage solutions, and effective methods for its configuration management is largely unexplored.

3 Cloud Storage Gateway

Unlike the cloud storage services which they complement, CSGs use standard network protocols that integrate with existing applications and can also serve as an intermediary to multiple cloud storage providers. Increasingly, CSGs also provide many sophisticated services such as backup and recovery, caching, compression, encryption, storage de-duplication and provisioning. A CSG will typically serve multiple clients using a set of local storage devices (possibly a RAID but not necessarily) that is seen by the clients as a local block storage. All clients assigned to this local storage share the storage, although there might be some internal fine-grain storage allocation policies that are not revealed to the client. Each client will be allocated space for its data, metadata, and log files. The CSG can be viewed as two I/O layers (see Fig. 1): (i) front-end for local user I/O and (ii) the back-end for cloud storage I/O operations. At a minimum, the CSG will provide the ability of intelligently partitioning the space into data, metadata, and log files, and a suitable caching mechanism for each so that data transfers from the backend can be properly handled.

3.1 Characterizing the Behavior of a CSG

The performance and behavior of a CSG depends on the hardware platform architecture h (CPU, memory, storage, network, local I/O rate, etc.), workload characteristics k (incoming request rate/distribution, data writes/sec, data reads/sec, metadata reads/sec, metadata size, etc.) and application goals p. Application goal is predominately expressed as I/O performance (MBytes per sec). Generally, the application goals are achieved by: (a) most reads are satisfied locally (which is essential to match the higher I/O inject rate to the slower backend rate), and (b) maintain and batch writes locally so as to make the writeback more efficient. These functions along with the management of meta-data files, rotating log files, garbage collection etc. are normal storage system attributes that affect the behavior of CSG. Furthermore, workload characteristics such as burstiness are also important. Workload and performance may be specified either directly in terms of resource requirements, or in more abstract terms such as priority, latency, or resource combination (e.g., a "small" vs. "medium" vs. "large" configuration). In any case, these are ultimately translated to individual system parameters, either explicitly or via policies.

It is easy to see that if the workload characteristics k, system configuration h and CSG configuration r are not matched, the end users would likely experience undesired performance p which is usually defined as the read and write rates supported with certain maximum latency and without any I/O timeouts. Read/write operations beyond the acceptable range or complete rejection is considered I/O failure. As workloads change over time (i.e. as i^{th} workload k_i deviates from initial assumed pattern k_i), the initial user-defined configuration r_i may no longer support the demands of new workload(s) and cause undesired end-user experiences.

A major issue in properly configuring the CSG is that *the vendors invariably do not reveal most of the internal details, and instead expose a limited set of administrator controllable configuration parameters to tune the system.* Often, these administrator controllable parameters are not even the actual configuration parameters, but merely some sizing controls or decision variables that affect multiple internal parameters. In other words, the knobs visible to the enterprise are rather fuzzy with little knowledge of what exactly they do. Of course, this is partly done to simplify the job of the administrator; a vendor willing to expose all raw knobs would invariably make them unusable. Thus the phenomenon of fuzzy knobs with little visibility is an essential characteristic of real systems and cannot be wished away! It alone precludes simple analytic models for characterizing performance of a component like CSG or fine-tuning it for very specific workload or hardware. With absence of any quantifiable, well-defined correlation or closed loop representation, configuration management is more an art than science.

3.2 Complexities in Configuration Control

Generally, as the size of the local storage (henceforth referred to as "cache") increases, we expect the CSG throughput to increase because more I/O can be handled locally. The benefit is entirely dependent on how well the caching mechanism keeps and prefetches "hot" data. Since the overall space available for caching is *shared* by multiple clients, there is interaction across clients. For example, giving more space for some clients hurts others, and the net effect is very complex to predict. This caching mechanism is hidden by the vendor and not controllable by the end-user, adding to the complexity of the configuration control. These comments apply to both data and metadata but with different effects. The metadata needs to be consulted for every I/O regardless of whether the corresponding data is in the cache or not. Depending on the workload and the granularity of access, metadata caching becomes more dominant than data caching. At the same time, metadata is generally much smaller than data, and thus it is much easier to provide generous amounts of storage for metadata. We explore the data cache vs. metadata size in our research through varied workload and configurations studies. The log size should have no influence on performance except that writebacks of the log would take up some backend I/O bandwidth. This initial description paves the way for understanding the complexity of configuring the CSG system.

Incorrect configurations could result in significant competition between the following three activities:

1. Eviction of modified pages requiring writeback to the backend object storage which is likely to experience high latency, limited I/O bandwidth due to network issues, and perhaps a significant write amplification due to the need to write the entire object.
2. Cache misses from client requests thereby requiring reading of backend object storage, which experiences similar issues as writebacks (e.g., significant read amplification and latency due to transfer of entire objects, for which the CSG need to make adequate room).
3. Local read/writes performed by the clients which are expected to be much more frequent and expect a low latency.

Note that the cache eviction is not complete until the write confirmation is received from the cloud storage, and the data must be kept in the cache until then. Consequently, a more aggressive eviction would only result in fewer entries in the cache to handle new data requests that must be fetched from the cloud. The CSG may also need to retry the *entire object* operation if unsuccessful. In addition to the whole object transfers on the backend, the transfer normally uses HTTP which adds considerable overhead. The varying size of the object could interfere with the SLA guaranteed to the user about upload/inject rates. We quantify SLAs based on local SCSI update traffic generated by the client, and any SLA violation is seen by the client as I/O timeout or errors. Note that increasing the cache size does not solve the writeback problem; in fact, it could even make it worse.

4 Configuration Problem Formulation

The CSG system in Fig. 1 serves as a 'disk cache' to buffer the incoming user request and match it with the cloud storage uploads (or downloads). User requests come over a SCSI bus at a high arrival rate λ_{in} (high throughput, high bandwidth, low latency) and the backend cloud storage presents a low eviction rate system λ_c (high latency/low bandwidth/higher error retries). Resource allocation for the disk cache needs to match the k_n incoming request streams and the service rates. We model CSG Controller of Fig. 1 as a queuing system. The incoming request stream k can be represented as:

$$k = f(ar, rs, rm) \tag{1}$$

where: k the request stream is a function of: ar the request arrival rate, rs the size of the request, and rm the metadata size. These request streams are characterized by the real world customer workloads as given in Table 2. Each of these requests consumes hardware and disk cache resource for servicing. Note in Fig. 1 that all n request streams share the same resource of CSG controller. There is no known functional relationship to analyze the queue behavior, and the controller has no user controllable factors to allocate resources per request stream. The relationship between the input stream, requested resource, queue and the disk cache is unknown. This leads us to conclude that there is no clear optimization or queuing technique that can model the serviceability of the incoming streams. For example, if the cache size is small and the eviction rate to backend cloud storage is high, then the system should be able to handle high requests. The converse means the requests will be dropped.

To add to this complex analysis, the CSG runs on a hardware platform characterized by core speed, number of cores, memory capacity, disk I/O capacity and network I/O throughput. Here, network I/O throughput represents the measurable I/O throughput to upload an object from the client system to the cloud storage. The hardware characteristics of the CSG platform is represented as:

$$h = f(cs, nc, me, m_{bw}, di, th) \tag{2}$$

where h the hardware characteristics is a function of: cs core speed, nc number of cores, me memory capacity, m_{bw} memory bandwidth, di disk I/O rate and th I/O throughput measured between the on-premise CSG and cloud object store. For example, one disk partition d_j on three independent disks has a better I/O rate compared to three individual partitions on one SCSI disk. Similarly a network card capacity of 10 Gbps would have better eviction rate than a network card capacity of 1 Gbps. Another evident characteristic is the performance boost from using a SSD versus a HDD disk. Again modeling the complex interactions between different these limited parameters and their relative effect on the *CSG disk cache performance* is unknown.

The disk cache is split into three distinct partitions: data buffer db, metadata md and log space ls (Sect. 3.2 & Fig. 1). These are bound by the total resource disk space available ds_{max}, such that:

$$ds_{max} \geq db + md + ls \tag{3}$$

Each of these parameters db, md, ls coupled with the request stream k influences the behavior of the system as explained earlier.

Finally, if the configuration r of the CSG is optimized on a given hardware h, the input stream/ workload k will experience a performance or service rate of p. We denote this performance as:

$$p = g(h, k, r) \tag{4}$$

denoting that the performance (or service rate) p depends on hardware characteristics h, service rate (or workload) k and CSG configuration r. Note that Eqs. 1, 2 and 4 is a multidimensional vector. Changing any one of the above parameters will affect the performance output p.

4.1 Research Questions

We define our research problem using a specific example relating to configuration challenge faced by a system administrator. Suggest a CSG configuration that satisfies the required performance p (10 MBps), given a specific hardware architecture h (2×1.2 GBps cores, 64 GB RAM, 1 GBps NIC card, 0.25 GBps network speed) and a workload w (5 concurrent users, 10 files, avg 5 GB size, upload time: 24 h). In general, we define the following:

1. $P = \{p_1, p_2, \cdots p_n\}$: performance constraints for each of the n applications, predominantly defined by expected I/O rate, e.g.: 100MBps min for both I/O write and reads.
2. $K = \{k_1, k_2, \cdots k_n\}$: workload characteristics for each of the n applications. We used real-world workload patterns observed from end customers of a commercial industry vendor (See Table 2), predominantly defined by average file size, number of files, users, sub-directory hierarchy etc.
3. $H = \{h_1, h_2, \cdots h_m\}$: m hardware characteristics, i.e. the machines running the CSG application, e.g.: core speeds, memory, local storage disk characteristics, network I/O bandwidth, etc.
4. $R = \{r_1, r_2, \cdots r_l\}$: l cloud gateway configurations each specified in terms of system configurations e.g.: 100GB data cache, 25GB log space, 50GB metadata space, 10 concurrent threads etc.

We can now ask two key questions:

Q.1 What should be the *cache configuration* to satisfy the ‘k’ request streams.
Q.2 What is the *maximum performance* from ‘k’ streams given certain cache configuration constraints (e.g. allocation of resources).

We are interested in a mechanism that draws a relationship between the workload characteristics K, system architecture H, configuration R, performance P and can *answer* the following.

1. Verify Configuration - given the system architecture, workload characteristics, and CSG configuration, determine if the performance constraints are met with a high probability.

$$[H_{new}, K_{new}, R_{new}] \Rightarrow P_{new} \text{ is satisfied} \tag{5}$$

In most cases, only a few parameters are new; however, because of the dependencies and nonlinear interactions, it may or may not be possible to exploit the unchanged parameters. This represents Q.2 of our discussion.

2. Configure - given the system architecture, workload characteristics, application goals and performance constraint, propose a configuration R_{new} that satisfies all the constraints. That is, given H_{new}, K_{new}, and P_{new} find R_{new}.

$$[H_{new}, K_{new}, P_{new}] \Rightarrow R_{new} \tag{6}$$

   This problem is the reverse of the first problem and is substantially harder. As in the last problem, only a few parameters may be new but it may or may not be possible to exploit the unchanged parameters.

3. Predict - Based on a time series of performance data for a given configuration, with changing workload, predict if the current configuration is likely to fail.

$$[H_{old}, R_{old}] and [K_i, P_i], i = 1, 2, ...n \Rightarrow \text{ Failure} \tag{7}$$

5 Solution Approach

As stated earlier, the complexity of the relationships between the user settable CSG parameters and the performance precludes a modeling or simulation based characterization. Therefore, we turn to machine learning based methods to learn various relationships along with our domain knowledge into the functioning of the CSG. Machine learning (ML) is, of course, no panacea; it often requires a significant amount of training data and the learned model may be "overfitted", and thus may be unable to accurately predict behavior when the inputs (workload or configuration parameter values) are sufficiently different from those for the training data. We will address this aspect carefully in our analysis.

5.1 Feature Vector

The feature vector used to support our research is built from the above equations Eqs. (1, 2, and 3) and represented as below. We have included throughput (a.k.a performance in bytes/sec) in the feature vector. As explained earlier in Sect. 4, our work defines QoS and SLA in terms of latency. Any latency exceeding the limit is experienced as I/O time-out. We do not explicitly include latency in the feature vector since a configuration that leads to time-outs will be rejected right away and is not relevant for performance analysis. We discuss some tests that violated SLA and unacceptable latency in Sect. 6.3; these were attributed to wrong configuration choice for specific workload constraints. Learning configurations that cause time-outs (and thereby avoid them) is a reasonable goal, but much harder and beyond the scope of this work. Therefore, all our tests used in the analysis are for valid SLA conditions (Table 1).

There are many parameters relevant to CSG operation that could potentially affect the performance; however, it is neither possible, nor practical to consider them all. There is no escape from applying the domain knowledge to consider only those parameters that are likely to be controllable or relevant. One such

Table 1. Subset of feature vector supporting the problem.

ar	Request arrival rate	rs	Size of the request
rm	Request metadata size	cs	Core speed
nc	Number of cores	me	Memory capacity
m_{bw}	Memory bus bandwidth	di	Disk I/O rate
th	Network I/O throughput	db	Data buffer
md	Metadata	ls	Log size
p	System performance (measured as throughput)		

example is the ubiquitous use of NFS for users to mount the remote storage device (but still within the local data center boundaries). NFS has many mount options (rsize, wsize, etc.) that can be chosen for individual mounts. However, these parameters are invariably set at default values and unlikely to be changed. Similarly, although the storage interconnect speeds (PCI bus speed, SATA/SAS interface speed, etc.) are potentially important, their selection happens at a much more basic level (i.e., when deploying the storage device/system) rather than for performance optimization. Therefore, consideration of these aspects is beyond the scope of this paper.

5.2 Research Hypothesis

We use statistical machine learning, classification, and optimization mechanism to *learn* these relationships. Our prediction model is expressed as a function of the above feature vector. Let $\boldsymbol{x} = \{x_1, x_2, \dots x_k\}$ denote the vector configuration parameter values. Let $\phi(\boldsymbol{x})$ denote the *hypothesized functional relationship* to be learned and $\gamma(\boldsymbol{x})$ is the true observed output for given values of the input $\boldsymbol{x}$.

We now have the basics to answer our research question Q.1 and Q.2 by using the above features and to accurately design our hypothesis $\phi(x)$ and output γ.

For a given workload, a service rate k_i, a set of constraints on hardware h_i and CSG configuration r_i, we *predict* the maximum performance p_i by the following hypothesis (i represents i^{th} variation).

Hypothesis:

$$\begin{aligned} &\phi(ar, rs, rm, cs, nc, me, m_{bw}, di, th, db, md, ls) \\ &\text{Output:} \quad \gamma() = p \end{aligned} \tag{8}$$

Similarly, rearranging the features, and re-writing the hypothesis, we *predict* the data cache configuration r required to achieve a given performance p for a given workload (service rate) k under a set of hardware constraints h.

Hypothesis:

$$\begin{aligned} &\phi(p, ar, rs, rm, cs, nc, m, m_{bw}, di, th) \\ &\text{Output:} \quad \gamma() = (db, md, ls) \end{aligned} \tag{9}$$

5.3 Classification Problem

In Eq. 8, we compute a single parameter p (performance) for a given set of constraints, and in Eq. 9, we compute multiple parameters db, md, ls (data cache size, metadata and log size). The former is called single label classification and the latter is called multi-label classification [6]. Computing a single label/parameter is relatively easier than computing three inter-related labels/parameters. For our end results, we predict performance, data cache size etc. as multiple classes (e.g. performance = {class 1, class 2, . . .} or data cache size = { class 1, class 2, . . . }). These are called multi-class prediction (for Eq. 8) and multi-class multi-label prediction (for Eq. 9). We will quantify this while discussing workload design and results in Sects. 6.2 and 8.

6 Implementation Details

6.1 Test Environment

Our test environment (Fig. 2) is comprised of (i) Dell PowerEdge R320 with 4 cores @ 1.8 GHz, 16 GiB memory, 3 ATA Disks- each of 500 GB and one 1 GB Ethernet interface (ii) Dell PowerEdge R730xd with 8 cores @ 2.1 GHz, 32 GiB memory, one SCSI disk of 5495 GB and one 1 GB Ethernet interface. Both servers have Ubuntu 14.04 with required tools and connected to local network. We used different hardware configurations to study the influence of cores, core speeds, disk, and memory configurations. On each of these servers, we partitioned the disk for several cache configurations. The server is connected to the HDD volumes on a remote cloud object store service, as NFS mounts. We used C++ and Python scripting tools for executing the workload and collecting metrics.

The algorithms were implemented in Python using *scikit-learn* [4] library. This tool gives both the algorithms, tuning parameters, cross validation and the associated metrics such as accuracy, prediction error, etc. For machine learning algorithm efficiency metrics, see [1].

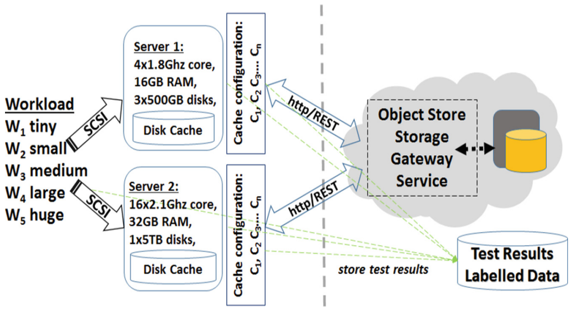

Fig. 2. Cloud storage gateway - test environment.

Once the test scripts were ready, the evaluation setup for each experiment involved partitioning the disks for various configurations, allocate cache/metadata size (see Table 3), connecting the newly configured server to data center object store and running the workloads and of course collecting the metrics. We ran over 100 different combinations of configurations/ workloads and collected the metrics (around 990 tests). Compared to the huge configuration space and vast research boundary, this limited tests gave us enough data points to validate our approach (see Sect. 8).

6.2 Workload Execution

Since there is no public domain workload data or readily available storage traces for CSG systems, we conducted our research using real world customer workload patters. We ran complete experiments on CSG platform from a commercial industry vendor and using workload characteristics as observed by real-world customers. This is shown in Table 2. Additional variants of workloads influencing meta-data size such as sub-directory depth and hierarchy were also used but are not shown in the table. File size and number of files followed a Zipf distribution with α of zero for maximum variance.

6.3 Metrics Collected

Table 2. Real world customer workloads.

Workload	Average file size	No. of files	No. of users	Total file size
Tiny	4 KB	10,000	25	1 GB
Small	256 KB	10,000	10	25 GB
Medium	1 MB	10,000	5	50 GB
Large	10 MB	1,000	5	50 GB
Huge	1 GB	200	2	400 GB

We executed workloads on different servers and various configurations, and collected metrics on execution time, meta-data time (e.g. to create sub-directories, open and close files etc.), throughput in bit/sec. Alongside the workloads we captured the configuration information about the server (e.g. cores, core speed, memory, disk capacity etc.) and CSG cache configurations (i.e. data cache area, meta-data, log size). We captured the available network throughput independent of the CSG, using a special RESTAPI tool set. We attempted a few workloads that would result in I/O timeouts to study the boundary conditions. Since the performance or throughput metrics at boundary conditions were meaningless (i.e. zero throughput or IO timeouts), we discounted these metrics from our labeled data-set. We ran all the workloads on both servers for different combination of data cache/metadata size configurations. The different instances were tested both for the homogeneous case (all workloads identical) and the heterogeneous case (workloads with different file sizes and read/write ratios). To collect a wide range of samples (test-data), we ran over 100 combinations of workload and configurations, executed about 990 of test cases, i.e., approximately 9 10 different configurations per workload type. During the course of our experiments and data collection, we persisted over 8 million objects and populated over 5.5TB of cloud object store.

6.4 Data Pre-processing and Classification

For each experiment, we created the 'sample' using workload, configuration, compute servers characteristics, observed performance into the feature vectors as {2 cores, 1.2 GHz speed, 16 GB RAM, 100 GB cache, 50 GB metadata, 347 Mbps network I/O, 108 Mbps performance etc.}. We classified the discrete numbers into buckets to give us meaningful insight into the behavior of the gateway servers and configuration. For observed metrics, a discrete throughput data of 488175 bits/s and 21392622 bits/s was classified into bucket sets as throughput *class* 1 or throughput *class* 4. Similarly a configuration of cache data or metadata size of 450GB and 200 GB was classified as *class* 5, 4 and so on. Via such discretization, we classified the data into buckets as shown in Table 3. In terms of machine learning, this bucketization means that the regression problem is transformed into a classification problem.

7 Predicting Performance and Configuration

Table 3. Metric Classification.

Metric	#Classes	Bucket size
N/W Throughput	10	unif (100 Kbps, 350 Mbps)
Performance	10	unif (100 Kbps, 350 Mbps)
Data cache	7	25, 50, 100, 200, 500, 1000, > 1000 GB
Metadata	5	25, 50, 100, 200 & 500 GB

There are well proven algorithms in the field of machine learning and their applicability is specific to the data characteristics and domain [6]. Further, the efficiency and accuracy metrics of the algorithms depend on *both* the problem domain and associated parameters like learning rate, regularization parameter, etc. We explored a wide range of ML algorithms to find the best fit for our problem domain. We found that Decision Trees fitted the relationship between the performance and configuration parameters with higher accuracy. Using statistical machine learning methods, Decision Trees tries to infer a split of the training data based on the values of the available features to produce a good generalization. The algorithm can handle both binary or multi-class classification problems. The leaf nodes can refer to either of the K classes concerned. It is basically an approximation function working on a multi-dimensional Cartesian space using piece-wise constant functions. Decision trees have been used in other storage metrics predictions such as [8] where authors exploit the trees for response time prediction of a single disk across different workload parameters. The reverse problem in Eq. 9, is a multilabel multi-class classification prediction, where in the end result includes multiple parameters: data cache and metadata configuration.

Our ML based solution approach is illustrated in Fig. 3. The working data-set is the used to train the ML Classifiers. We used two different algorithms and regression classifiers to predict (i) single label performance value and (ii) multi-label configuration parameters. The algorithms use a *sub-set* of data to train the respective model, using hypothesis in Eqs. 8 and 9. The resulting weights of the model is used to predict end result for a *new query*. The algorithm returns the predicted values, accuracy metric and root mean square deviation. We present the prediction accuracy, cross validation results and algorithm efficiency below.

8 Evaluation Results

The evaluation results should tell us how suitable our approach is to predicting the right performance (Eq. 8) and a right configuration (Eq. 9).

Preparing Cross Validation Data Set

There are several ways in which the collected data can be split into the training set and the testing set to verify an algorithm for over-fit/under-fit (See Monte Carlo cross validation selection [9]). Towards this end, we first shuffle the data randomly and then divide it into three equal size buckets. We then use some part of each bucket for testing and use the rest of the bucket for training. We now consider two specific cases: (A) Use 90% of each bucket for training and the rest for testing, and (B) use 80% of each bucket for training and the rest for testing. For each case, we use only one bucket at a time, thereby giving three sub-cases in each case. These combinations are marked as suffix A.1, A.2, A.3, B.1, B.2, B.3 in the figures below.

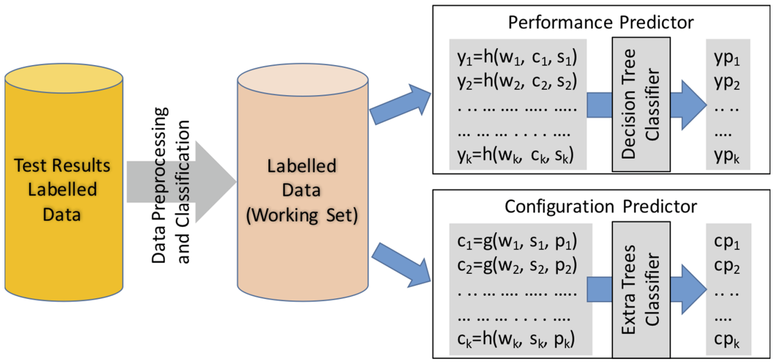

Fig. 3. Predicting performance and configuration.

Predicting Performance - Results

The performance prediction efficiency is shown in Fig. 4. The bar-graphs have different sample size and *train vs. test* data ratios, as shown in primary (left side) y-axis labels. X-axis indicates the different test cases. For these test cases, the prediction accuracy is on secondary y-axis (right side). The results show that our solution has prediction accuracy around 95% for various combinations (or about 5% prediction error) and do not suffer from under-fit or over-fit. As stated earlier, we are not aware of other public studies on characterizing the relationship between configuration parameters and performance of CSG or other systems, and thus a direct comparison against prior results is not possible directly. However, we compare our results against results from similar techniques used in a different storage context (e.g., performance vs. workload parameters). In particular, the study by Wang [8] using their CART-based models show a relative error between 17% and 38% for response time prediction. Using Inside-Out [2], Hsu reports a performance prediction error around 9%.

Predicting Cache Configuration - Results

The cache prediction efficiency is shown in Fig. 5. The bar-graphs have different sample size and *train vs. test* data ratios, as shown in primary (left side) y-axis labels. X-axis indicates the different test cases. For these test cases, the prediction accuracy is on secondary y-axis (right side). The results show that our solution has prediction accuracy around 75% for various combinations (or about 25% prediction error) and do not suffer from under-fit or over-fit. In cache prediction, we predict multiple parameters, i.e. data cache size, meta-data and log size. The reason for higher error is self-explanatory by the nature of complex multi-label multi-class parameter prediction with limited training data set.

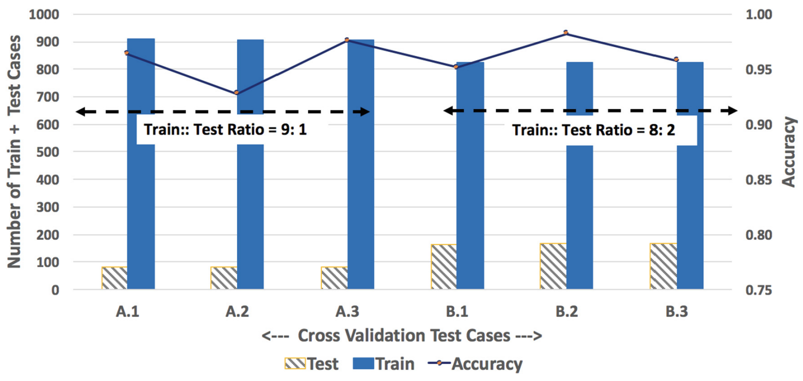

Fig. 4. Efficiency metrics for performance prediction.

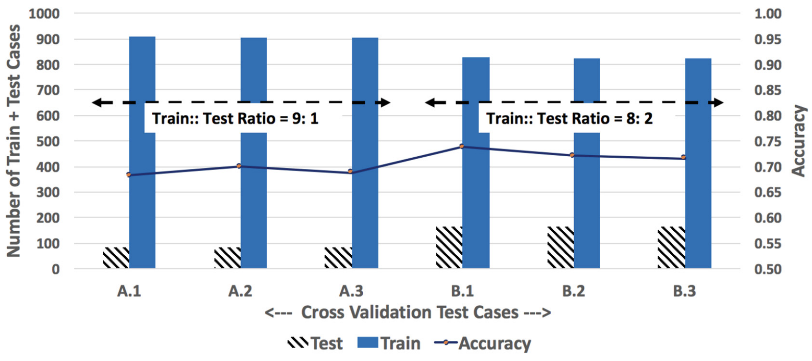

Fig. 5. Efficiency metrics for configuration prediction.

9 Conclusions and Future Work

In this paper, we present a methodology for the configuration and performance prediction of cloud storage gateway (CSG), which is an emerging system of crucial importance in providing scalable access to remote storage. Because of the large number of configuration parameters and inter-dependencies among them, modeling the influence of configuration parameters on the performance is a challenging problem. We show that machine learning techniques suitably aided by the use of domain knowledge can provide robust models which can be used for both the forward problem (i.e., predicting performance from the configuration parameters) and the reverse problem (i.e., predicting configuration parameters from the target performance).

We show that our models can provide performance prediction accuracies in the range of 5% without requiring large amounts of data. The prediction accuracies are worse when multiple configuration parameters are predicted, but still respectable (in 20% range). We believe that similar methodology can be applied to other systems as well, and we will examine this in our future work.

In future work, we will focus on robustness of the algorithms and extrapolation studies to look beyond test case boundaries. We plan to analyse the sensitivity cost to continuously monitor the performance and the workload in order to adapt the configuration gradually as the workload changes. Lessons learned from this research can be expanded to the auto-tuning of the hosted gateway solution and back end cloud based object store configurations.

Acknowledgements. This research was supported by NSF grant IIP-330295. Discussions with Dr. S. Vucetic of Temple University were highly valuable in devising the extended validation techniques presented in the paper.

References

1. Almseidin, M., Alzubi, M., Kovacs, S., Alkasassbeh, M.: Evaluation of machine learning algorithms for intrusion detection system. In: 2017 IEEE 15th International Symposium on Intelligent Systems and Informatics (SISY), pp. 000277–000282. IEEE (2017)
2. Hsu, C.-J., Panta, R.K., Ra, M.-R., Freeh, V.W.: Inside-out: reliable performance prediction for distributed storage systems in the cloud. In: 2016 IEEE 35th Symposium on Reliable Distributed Systems (SRDS), pp. 127–136. IEEE (2016)
3. Klimovic, A., Litz, H., Kozyrakis, C.: Selecta: heterogeneous cloud storage configuration for data analytics. In: 2018 {USENIX} Annual Technical Conference ({USENIX}{ATC} 2018), pp. 759–773 (2018)
4. Pedregosa, F., Varoquaux, G., Gramfort, A.E.: Scikit-learn: machine learning in Python. J. Mach. Learn. Res. **12**, 2825–2830 (2011)
5. Prahlad, A., Muller, M.S., Kottomtharayil, R.E.: Cloud gateway system for managing data storage to cloud storage sites, 2010. US Patent App. 12/751,953
6. Sorower, M.S.: A literature survey on algorithms for multi-label learning. Oregon State University, Corvallis **18**, 1–25 (2010)

7. Tesauro, G., et al.: Online resource allocation using decompositional reinforcement learning. AAAI **5**, 886–891 (2005)
8. Wang, M., Au, K., Ailamaki, A., Brockwell, A., Faloutsos, C., Ganger, G.R.: Storage device performance prediction with cart models. In: Proceedings of the IEEE Computer Society's 12th Annual International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems. (MASCOTS 2004). IEEE, pp. 588–595 (2004)
9. Xu, Q.-S., Liang, Y.-Z., Du, Y.-P.: Monte carlo cross-validation for selecting a model and estimating the prediction error in multivariate calibration. J. Chemom. J. Chemom. Soc. **18**(2), 112–120 (2004)
10. Xu, T., Zhou, Y.: Systems approaches to tackling configuration errors: a survey. ACM Comput. Surv. (CSUR) **47**(4), 70 (2015)
11. Yin, Z., Ma, X., Zheng, J., Zhou, Y., Bairavasundaram, L.N., Pasupathy, S.: An empirical study on configuration errors in commercial and open source systems. In: Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles, pp. 159–172. ACM (2011)