Optimizing Resource Configuration for Edge Storage

Sanjeev Sondur, Anis Alazzawe and Krishna Kant

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

Abstract. Edge storage (ES) system provides the edge-clients an impression of unlimited local storage by connecting with cloud storage on the backend. As ES is often deployed in environments that impose stringent limits on cost, space, cooling needs, etc., effective allocation of compute and storage resources is crucial for achieving the desired performance from ES. In this paper, we address the problem of optimally configuring such a system in the presence of multiple, often conflicting, workload and QoS requirements. We also seek to obtain multiple solutions to enable more flexible choice by the administrators. Our solution uses stochastic optimization along with principal component analysis (PCA) to focus on the most relevant configuration parameters, and it evaluates the choices by using a Neural Network that can predict performance with configurations as inputs. The approach, when used with a metaheuristic such as Generic Algorithm (GA), provides much smoother evolution of the objective function and thus can be terminated much ealier than an uninformed GA.

Keywords: Edge Storage, Configuration Management, Principal Component Analysis, Decision Trees, Genetic Algorithms, Feature Extraction

1 Introduction

Enormous volume of raw data generated by edge-applications such as IoT devices, smart transportation solutions, health-care systems, environment monitors, etc. need to be persisted on the Cloud reliably and securely. *Storage at the Edge* or Edge Storage (ES) is built using commodity hardware and deployed in close proximity to the originating source - to capture, process and retain data at the "edge" of the network, and store/retrieve such data from far-off data-centers. As the high volume of edge data can overwhelm a traditional storage system and test the limits of unpredictable network bandwidth to the Cloud, ES administrators are challenged with designing an optimal solution to achieve the right workload/performance demands. An optimal solution (i.e. resource allocation) should satisfy both the workload/ performance demands of the edge application and the imposed constraints such as cost, space, heat dissipation and cooling needs. Obviously, throwing extra resource does not overcome the performance bottleneck or satisfy the constraints.

Edge Storage systems are based on Cloud Storage Gateway [12] that translate *edge-applications* SCSI based block IO requests to REST based object-store APIs on the Cloud side. This gives the impression to the edge-clients (i.e IoT devices, applications, users, etc.) of unlimited storage, even though ES compute and storage resources are very limited. Configuration and optimal resource allocation for ES is essential to avoid IO timeouts and slow performance. Managing configuration of an ES involves not only the right choice of the many parameters that influence performance but also to satisfy the constraints imposed by limited space, cooling needs and deployment costs.

Using a commercial ES software, we studied the problem of configuring the hardware (CPU, memory, storage, network, local I/O rate, etc.) and resources (data cache area, meta-data area) for given workload characteristics (number of files, file size, meta-data, etc.) and application goals (e.g., required performance). Because of the complexity of the problem and the numerous parameters involved, we studied the problem using statistical machine learning (SML). We use the insight from the performance prediction (p-SML) model, to recommend an optimal configuration (i.e. hardware and resource) that satisfies both the workload/performance and user given constraints.

Based on empirically data collected from real world experiments, our p-SML model achieves an accuracy around 97%. We used Principal Component Analysis (PCA) and extracted Feature Importance (FI) to provide a reasonable metric to understand the variance of a parameter and its relative contribution towards the performance. Our solution to recommend an optimal configuration (i.e. compute and storage resource allocation) uses a stochastic model such as Genetic Algorithm (GA) enhanced with PCA and FI metrics. The enhancement results in much fast convergence; the modified algorithm reached maximum fitness function 50% earlier than the standard algorithm.

This paper is organized as follows. State of current art is given in section 2. We briefly describe the characteristics of the Edge Storage system in section 3 and formulate the configuration research questions in section 4. We present our solution approach in section 5. Section 6 discusses the evaluation methodology, implementation, and results. We explore potential future work and conclude the paper in section 7.

2 State of Current Art

Klimovic and Costa [3] [7] support our complexity problem involved in analysing the workload data streams and a wide choice of configuration space to be explored in a cloud storage system. In designing Selecta, Klimovic address the storage configuration for Data Analytics workload using TPC traces on the block storage devices on the data center side, while our work studies the Object-store based ES configuration on the edge side using vendor provided workloads. Costa [3] state that configuring a storage system for desired deduplication performance is extremely complex and difficult to characterize. Rao [13] show that a traditional control theoretic framework is inadequate to capture the complex-

ities of resource allocation for VMs. Ofer [9] study come very close to our work both in application of machine learning for Cloud based object storage systems. While their study applies deep learning to cache eviction/refresh techniques in Object-store, we explore the configuration management of ES which is also based on Object-store.

Hsu designed Inside-Out [6] to predict performance in a distributed storage system. They study low-level system metrics (e.g., CPU usage, RAM usage and network I/O) as a proxy for measuring high-level performance. Cao [2] evaluated few popular black box auto-tuning techniques for storage using macro-workloads generated by Filebench. Both Hsu and Cao comparative study supports our research in that optimal configurations depend by hardware, software, and workloads and that no one technique is superior to all others. Ularu [17] use Decision Trees to configure an application and highlight the use of Decision Trees on solving a configuration problem because of the wide solution space to be explored. Wang [19] presents a survey of work on the impact of edge caching capacity, delay, and energy efficiency on system performance.

Task offloading and migration schemes commonly used in Mobile Edge Computing largely pertain to radio and computational resource allocation, however in ES servers these options are not viable since ES has to service requests (or tasks) locally. Current art surveyed relate to performance prediction for a given workload or workload scheduling, placement etc., while our work focuses on *choosing a set of configuration parameters* that satisfy user workload/performance demands under given constraints.

3 Characteristics of an Edge Storage System

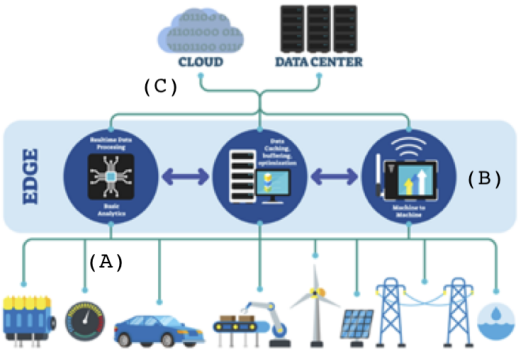


Fig. 1. Edge Computing/ Edge Storage [1]

Edge Storage (ES) system (marked 'B' in Fig. 1) is a new paradigm in storage strategies that addresses the challenges involved in data storage and retrieval in Edge/Fog Computing. ES systems provide a *local cache buffer* to bridge the gap between the high throughput demands of the latency sensitive edge applications and the low/unpredictable network connectivity to the Cloud. ES connects the edge-applications to an Object-store on the Cloud because of the inherent

advantages for Object-store in the cloud model. Edge client applications operate using small blocks of data (of 4KB or 16KB size) at SCSI speeds, shown as (A) in Fig. 1. On the Cloud side, marked as (C) in the same figure, ES has to interface with Cloud Object-store over an unpredictable network (low throughput, high latency) with *varied object sizes* that are largely dependent on the applications. In addition, ES should address reliable communication such as IO block acknowledgements or retry an error-ed IO block on the SCSI side and acknowledge or re-fetching the entire object on Cloud side. To address this imbalance, ES has to satisfy key requirements such as (i) protocol translation from/to SCSI to/from http/REST based services, (ii) map 4KB/16KB block IO requests to Object-store APIs for objects of varied size (or vis-versa), (iii) satisfy high throughput/low latency edge-client request over a low throughput/high latency connection to the Object-store Cloud service, (iv) manage storage overheads like security, meta-data management, reliability, rotating log file, garbage collection, etc.

In an Object-store system, every object is associated with corresponding metadata that is maintained by a metadata server. When an object is initially requested by ES, this metadata also must be brought in from the cloud. It is generally desirable to keep the metadata longer than the data so that if the object is evicted and then re-requested, the ES can avoid small IOs associated with metadata accesses. However, a proper balance must be maintained between the space allocated to the data and metadata for optimal performance. Thus, both workload access pattern and metadata management determine the performance experienced by the user. As part of our study in ES configuration recommendation and resource allocation, we will explore the space allocated for data cache and metadata cache. The architecture and capabilities of ES along with Object-store functionality are discussed in our previous paper [14].

3.1 Modeling the behavior of an Edge Storage System

ES performance p is defined as the throughput experienced by the edge-clients, and is generally expressed as either MB/s or as objects/sec [10]. ES architecture involves the complexities inherent in storage systems, cache allocation, satisfying IO demands, and unpredictable network bandwidth [16]. Mathematical modeling of the behavior of an ES is difficult because of the complex inter-dependencies between the numerous parameters and the poor understanding of these relationship and their impact on the overall performance. Protocol translation, security and meta-data operations, workload interfaces (block IO, Object-store APIs), etc. consume computational resources that are defined by core speed, number of cores, memory, and memory bandwidth. Satisfying latency sensitive edge-client IO request needs data buffering, intelligent cache operations, etc. that demand both storage and memory resources. The limited storage resource in an ES has to be efficiently partitioned for data-cache, meta-data and other operational overheads (e.g. swap space, log files). Unpredictable nature of back-end Cloud connectivity raises additional challenges in cache eviction, refresh vs. prefetch,

efficient bundling of object requests, exploiting workload patterns etc. The inter-dependencies and complex behavior of these parameters (e.g. CPU, memory, cache-space, etc.) make analytic modeling of performance very difficult [18].

The throughput p experienced by the edge-clients depends largely on the workload k, ES hardware h and the resources r allocated to the compute and storage layers of ES. An improper choice of these parameters will result in a poor experience by the end user such as IO timeouts (rejected requests) or poor throughput (low performance) or large unacceptable latency. To address the difficulty in detailed analytic characterization of the above parameters, we formulate the above parameters as a classification problem. Using domain expertise from ES administrators, we choose the most influencing parameters and classify or enumerate them into buckets, a sample of which is shown in Table 1.

Attribute	**#Classes**	**Example of Buckets or Enumeration**
Core Speed (GHz)	5	1.2, 1.8, 2.4 $\cdots$
Memory Capacity (GB)	5	16, 32, 64 $\cdots$
Data cache	7	25, 50, 100, 200, 500, 1000, $>$ 1000 GB
Metadata	5	25, 50, 100, 200 & 500GB
Observed Performance	10	uniform distribution (100Kbps,350Mbps)

Table 1. Sample Classification of Attributes.

Let nc denote the number of cores, cs the core speed, mc the memory size, bw the memory bandwidth, and di the disk IO rate. Also denote ar as the request arrival rate, rs the request size, and ms the metadata size. We then propose the following functions to represent the classification of hardware h and workload k:

$$h = f_1(nc, cs, mc, bw, di) \tag{1}$$

$$k = f_2(ar, rs, ms) \tag{2}$$

Note that in postulating these functions, we have included only a subset of the parameters that could potentially be relevant. In particular, we did not include in h other architectural details such as size/speed of L1, L2, L3 caches, since practically the choice would be limited to certain models of hardware platform from a given vendor. Also, while some parameters (e.g., the DRAM speed) could be selectable (with in an appropriate range), their level of influence does not warrant their consideration. This aspect necessarily involves the use of domain knowledge. Simply throwing in as many parameters into the model as possible can be self-defeating both in terms of data requirements for training the model and in diluting the model with weak dependencies that are difficult to characterize.

In addition to the hardware and workload characteristics, the performance achieved by a workload class also depends on the storage resources allocated to it. In particular, the total space r allocated to a workload class is simply the summation of data-cache size db, meta-data size md, and log size ls. (Obviously, r should be less than the total space available). Since the log size ls does not play a significant role in performance, we will ignore it here.

We can now express p in terms of workload class w, ES hardware class h and resource allocation class r as:

$$p = g(h, k, r) \tag{3}$$

4 Research Questions

Our research focuses on devising a suitable model for answering two key questions.

Q.1 Predict the performance p under given workload k, hardware h and resource allocation r.

Q.2 Predict an optimal configuration, i.e. hardware h **and** resource allocation r to satisfy the given workload k & performance p **and** user defined constraints.

Building a model using queuing theory or similar approach with many interrelated parameters is very difficult. A direct application of machine learning (ML) is ill-advised to predict some arbitrary set of parameters. In question Q.2, we *recommend* a set of parameters that identify the satisfying configuration for a given k, p. Answering Q.2 is particularly problematic for using ML for configuration prediction where we may want to predict some arbitrary set of configuration parameters, and furthermore, we want multiple solutions, since certain combinations may not be desirable or feasible from a practical perspective. These perspectives include availability of only certain discrete values (e.g., 2 or 4 cores only), availability of physical resources, difficulties in physical setup, etc. Further, they have to satisfy user defined constraints such as power limits, size, cooling needs, etc.

5 Solution Design

The above research topic raises a few additional questions that needs clear understanding to support our work. The first question is to decide on the design variables (a.k.a. feature-set in ML terminology), and what constitutes a satisfying data-set. The right choice of variables combines domain knowledge, careful selection of set-able or user-controlled variables, and influence on the accuracy of final metric (i.e. performance prediction accuracy). Then, we have to select the right ML model to learn the relationship between these variables. Finally, we need to investigate how additional insight from Q.1 can help in designing an optimal solution for Q.2.

Because of the complexity explained earlier, we studied the problem using statistical machine learning (SML). A successful application of SML to complex systems problems *mandates judicious choice of attributes* to both limit the model complexity and to achieve good accuracy. In particular, simply including all attributes can yield poorer results than a carefully chosen proper subset. We tested various choices of attributes (or feature-set) that were compatible to the parameters in Eq. 1 and Eq. 2. We defined the hypothesis for predicting performance p for a given workload k, on hardware h and resource configuration r as:

$$\begin{aligned} \text{Hypothesis:} \quad & \phi(ar, rs, rm, cs, nc, me, bw, di, th, db, md, ls) \\ \text{Output:} \quad & \gamma() = p \end{aligned} \tag{4}$$

The results from the different choice of the design variables and ML performance prediction model is explained in section 6.1.

Predicting a satisfying configuration parameters for Q.2, based on user workload and target performance is a harder problem, since it involves determining a large set of complex inter-dependent variables that satisfy the given condition. Besides, there could be more than one solution that satisfy the required constraints. That is, there could be various combinations of hardware (CPU, memory, etc) and resource (data-cache size, meta-data size) that satisfy the user given workload/ performance under given constraints (e.g. minimum heat dissipation, size).

Our stochastic process based solution, in this case a Genetic Algorithm (GA) aims at narrowing down the large search space and output multiple satisfying solutions which can then be filtered for the user defined constraint. GA algorithm defines the design variables as a population, that is evaluated for fitness and then undergo random cross-over and mutated to derive at a new state. Each population is represented by a chromosome that maps to a design state, (i.e. a set of design variables). Our solution uses the performance prediction model from Q.1 as the fitness function to determine if the current state (i.e. chromosome) satisfies the user required performance. To achieve higher efficiency, a good solution would result in a smaller number of calls to such an oracle. Instead of default random mutation in GA, our enhanced approach aims to intelligently mutate the chromosome population to jump to the new state in a controlled manner. We used additional insight from PCA objects derived from the ML model to control the gene mutation probability. We explain the metrics from PCA objects, selection of principle design variables based on ML features importance in section 6.3.

Constraints can be expressed as cost functions to represent the deployment cost, power consumption, cooling requirements, etc. Cost function is represented as the *normalized* cost of a configuration based on the design variables. For example, k^{th} configuration cfg_k for some choice of hardware h_i and resource r_j, is represented as $cfg_k = \{cs_i, nc_i, bw_i, \cdots db_j, md_j, ..\}$ and has a cost C_{ij}. Data for cost function can be derived from vendor specification for hardware server and allocated resources.

We analyzed the empirical data collected from our experiments (given in section 6) to answer the above research topics in Q.1 & Q.2.

5.1 Implementation Details

We implemented the algorithms in Python using *scikit-learn* [11] library for Machine Learning components such as Principal Component Analysis, Classifiers, ML metrics (e.g. accuracy, precision etc.), Feature Importance etc. For stochastic algorithms, we used *NSGA-II* [5] Genetic Algorithm from Platypus library [4]. NSGA-II algorithm (discussed by Deb et al. in [5]) gives flexibility to define fitness function, define objectives and constrains, variable bounds, chromosome construction, crossover and mutation, solution-set, etc.

6 Experimental Evaluation

Our work in this paper is based on the experimental setup, vendor provided edge workload patterns, data collection, and partial analysis done in our earlier work [14] for answering the research question Q.1 listed above. Therefore, we describe the experimental setup briefly.

We used a commercially available ES product operating with a set of vendor provided workload patterns, that are reflective of real-world ES user population. For example, workload patterns for a smart-health monitoring system is characterized as 'Tiny' defined by image size of 4KB, about 10,000 images/24 hrs, with associated meta-data on date, time, location etc. Another workload pattern for health-care (e.g. Pathology) is characterized as 'Huge' with image size 1GB, about 200 images/24hrs, with meta-data about patient ID, hospital ID, etc. We executed 100s of workloads on various configurations and resource allocation schemes using different hardware servers. For each of these experiments, we collected performance p metrics along with hardware $(nc, cs, mc \cdots)$ and resource allocation $(dc, md \cdots)$ details. Empowered with this empirical data, we proceeded to address the research questions posed above. We implemented ES constraint data as a normalized value in the cost function based on hardware manufacturer's server specification data, such as power rating, cooling BTU, size, etc. For example, cost $C_{ij} = 0.475$ is the cost for hardware server h_i (e.g. 2 x 1.8GHz, 32GB mem, 100K IOPS, etc.) and resource r_j (e.g. 500GB data cache, 100GB meta-data).

6.1 Influence of Chosen Feature Sets on Performance

As stated earlier, including the configuration parameters (or feature set) without a proper consideration of their relevance not only makes the model more complex but also interferes with the accuracy of the model. We demonstrate this in the following by studying the performance p as a function of configuration parameters (k, h, r). Fig. 2 shows the prediction accuracy results for various choice of attributes. In Fig. 2, Feature Set 3 includes high level attributes $\{k, h, r\}$ (Eq.3) and Feature Set 4 includes additional attribute by expanding the resource $\{k, h, ds, md\}$. Performance prediction accuracy using these two limited feature-set is about 93%. Feature Set 10 comprise of $\{cs, nc, mc, bw, di, ar, rs, ms, ds, md\}$, this results in a higher prediction accuracy of 97%. We verified the results by expanding the feature set with additional attributes.

A blind inclusion of more attributes is labelled as Feature Set 13, which includes additional attributes of network bandwidth (nw) and logfile size (ls). These additional attributes add undesired noise in the data and results in poorer predication accuracy (down to 91%). Based on our extensive experience and domain knowldge with Edge Storage, we know that this noise is the result of adding unpredictable network bandwidth (nw) and logfile size (ls), both of which do not contribute to the ES performance. These results reinforce our earlier comments regarding the selection parameters in Eq. 1 and Eq. 2.

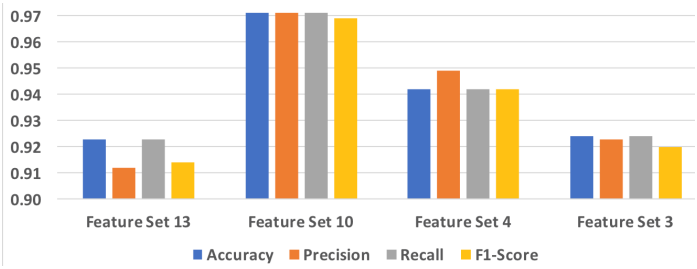


Fig. 2. Performance prediction accuracy with various feature set.

There is no auto-solution for improving efficiency of ML algorithms, as they depend on the application domain, careful selection of attributes (feature set), and hyper-parameters like regularization parameters, learning rate, etc [15]. Therefore, we tried several types of models and ultimately settled on Decision Tree (DT) for Q.1, as it consistently performed the best (see Fig.3). Building a performance prediction model for Q.1 based on Decision Trees yielded an accuracy around 97% for various test-train data combinations (k-fold validation, k=5). An extensive analysis of the model ensured that it does not suffer from under-fit or over-fit.

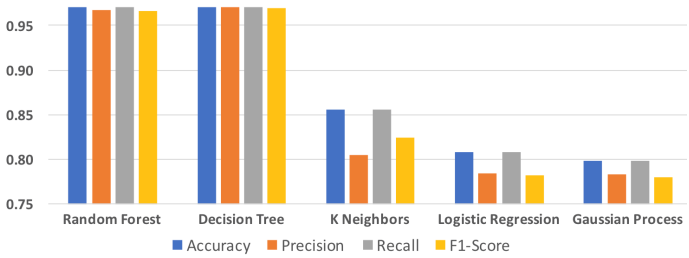


Fig. 3. Performance prediction accuracy with various ML predictors.

6.2 Extracting Feature Importance

PCA is dimensionality reduction technique that projects the data from its original p-dimensional space to a smaller k-dimensional subspace. PCA maximizes the variance accounted by the first k components and thereby attempts to include those components that have the most influence on the output. The k-dimensional subspace considered by PCA involves components that are linear combinations of the original variables; therefore, we still need to identify the most relevant original variables. In PCA terminology, the contribution of each variable to each principal component is described by *Loadings* [8], which can be easily extracted. Large loadings (positive or negative) indicate that a particular variable has a strong relationship to a particular principal component. The sign of a loading indicates whether a variable and a principal component are positively or negatively correlated.

Feature ablation is a technique for calculating feature importance (FI) that works for all machine learning models. A feature with a high importance has a greater impact on the target variable. We compared both FI from the DT model

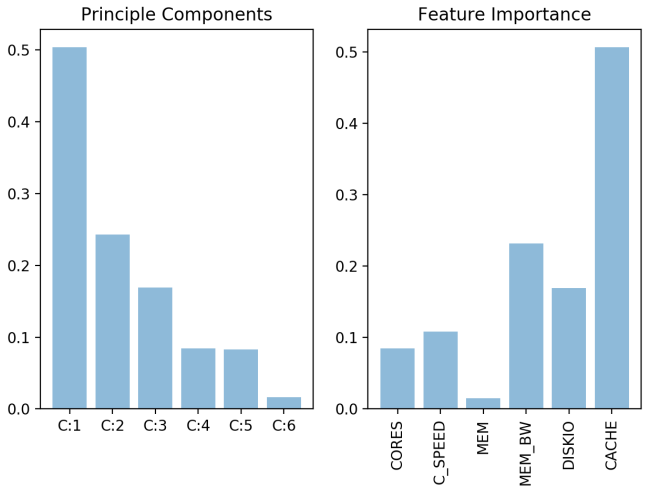


Fig. 4. PCA and Feature Importance.

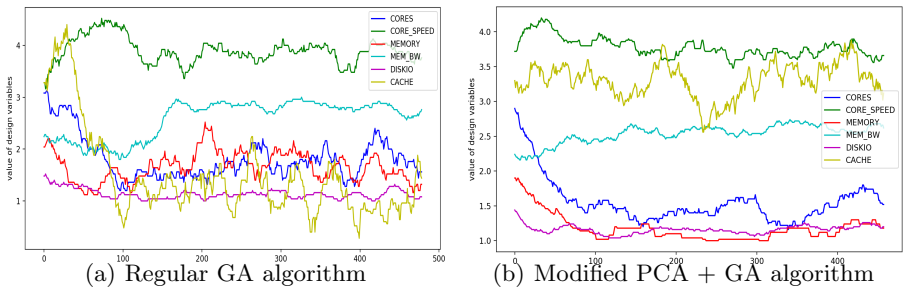
and PCA & Loadings from the PCA objects to gain confidence in ranking the predominant attributes that contribute to ES performance. The *scree plot* of PCA and FI for our data-set is given in Fig. 4. In the figure, the left sub-graph shows PCA values for different orthogonal components (C1$\cdots$C6) on x-axis, and the right sub-graph shows FI values for the design variables (on x-axis). Based on above metrics, PCA & FI provided a reasonable metric to understand the variance of a parameter and its relative contribution towards the performance. Instead of randomly mutating the set of genes to generate a new population set (i.e. new configuration state), we focused on a deterministic way to control the cross-over and mutation process. We used the above metric from PCA and FI to probabilistically mutate the genes in the modified PCA+GA approach and generate a 'controlled' new state. The results of our solution with enhanced GA algorithm is given below.

6.3 Recommending an Optimal ES Configuration

The design variables (CPU, memory, IO bandwidth, etc.) plus the workload properties (file size, meta-data, no. of files, etc.) formed the chromosome (i.e. gene pool) that represent a population. The GA population thus obtained, satisfies the hypothesis in Eq. 4 to define the behavior of an ES. Note that during mutation, we do not vary the workload variables (ar, rs, rm) as these are user given properties for predicting the required configuration. The fitness function (FF) defined by the DT from section 6.1 selects the right population (design variables) that satisfies the user defined performance. To ensure efficiency, this performance predicting oracle has to be consulted sparingly for rapid convergence. In our implementation, we used an initial population size of 10, and Tournament selector to pick top two best fit population for cross over function.

In a standard GA approach, the design variables (i.e. gene pool) are randomly mutated to get to a new state (i.e. new population set). The population set is continuously evaluated for fitness and the *best fit population* is selected as the suitable solution (i.e. population with predicted performance equal to user

defined performance). Uncontrolled mutation may result in design variables being randomly selected from a wide range and this may result in finding a suitable solution after a considerable time (usually measured as number of iterations). Our goal is to enhance the GA algorithm to intelligently mutate the gene pool such that the desired solution (i.e. fitness function) is reached faster (i.e. less number of iterations).

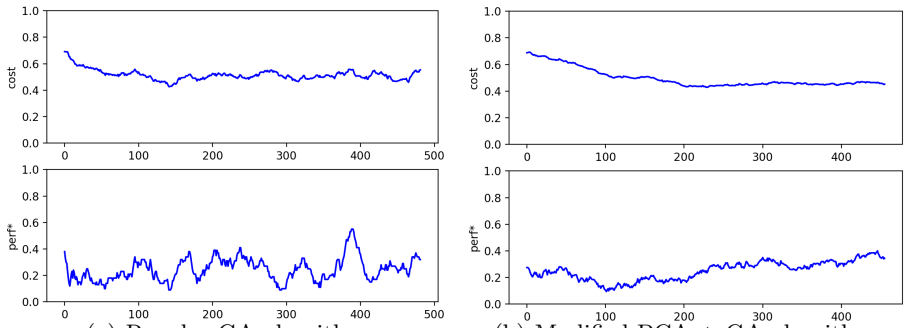


(a) Regular GA algorithm (b) Modified PCA + GA algorithm

Fig. 5. Comparison of Design Variables

We extracted additional data from ML objects, PCA model and feature importance (FI) as explained in section 6.2. A high FI metric relates to a high relevance of the variable towards the output. For example, Fig 4 shows that data-cache value of 0.506 has the highest relevance to the final ES performance. We used this *data relevance* to probabilistically mutate different genes. Using data from Fig. 4, gene representing data-cache undergo mutation with 0.506 probability, and the gene representing core speed undergo mutation with 0.108 probability and so on. This disciplined mutation allows the PCA+GA algorithm move to a new state (i.e new population set) in a controlled fashion. The results of this modification is visible in Fig. 5(a) & 5(b) across various iterations (x-axis). In Fig. 5(a), the design variables (genes) undergo random mutation and hence spread across a wide spectrum. All variables including data-cache (the influencing parameter) take a random values, thereby giving a random output (i.e. performance in Fig. 6(a)). In our modified PCA+GA approach, the effect of disciplined change of variables is visible in Fig. 5(b). Design variables with lesser influence tend to settle down quickly and the influencing variable (data cache, memory bandwidth) span 'within a limited' range searching for a satisfying solution (i.e. user desired performance). This intelligent control of gene-mutation results in reaching the solution-set faster (i.e less number of iterations).

Genetic algorithms results in a 'multiple solution-sets' that satisfy the fitness function, which can be further refined or filtered for desired results. In our approach, the solution set should satisfy the user given constraints, normally defined as a minimum cost function. That is, we extract the most satisfying configuration state (chromosome in solution set) based on cost function. We evaluated the PCA+GA approach to verify convergence into a satisfying config-



(a) Regular GA algorithm (b) Modified PCA + GA algorithm

Fig. 6. Cost and Performance Comparisons

uration with minimal cost. Fig. 6(a) shows the normalized values of cost function (top graph) and fitness function (perf*) for unmodified standard GA solution. Fitness function is distributed across various performance ranges because of the design variables being randomly varied. The cost function is equally unsettling. With our controlled gene mutation in PCA+GA solution, the design variables find the satisfying fitness function (performance) faster at a minimum cost in less number of iterations. Further, the cost function settles down faster, as shown in Fig. 6(b).

Our modified GA+PCA based approach converges to a solution faster than the traditional approach. Evaluation results shown in Fig. 7 over various test cases show that GA+PCA outpaced standard approach by 50%, i.e. modified algorithm reached maximum fitness function 50% earlier than standard algorithm. For example, test case T1 is a query to suggest an optimal configuration for: Workload class:8, Perf class:5 (i.e. Large Workload: 1000 files of 10 MB size, 5 users, Required Perf.: 250MBps). The GA+PCA based approach converges to a solution after 80 iterations with a solution: 2 cores x 3.2GHz, 16 GB Mem, 3.2GB Mem bus, DiskIO = 10K IOPS, Normalized Cost = 0.4995. The same query to a standard GA takes about 160 iterations to find a minimum cost solution. We observed similar improvements in other sample queries shown as T2, T3, T4 in Fig. 7, with GA+PCA reaching the solution in half the time as standard GA.

7 Conclusions and Future Work

In this paper, we presented a methodology for resource allocation and configuration of an Edge Storage System (ES), which is of crucial importance in efficiently supporting edge services in the highly resource constrained environment where the ES typically operates. Because of the large number of configuration parameters and inter-dependencies among them, quantifying the influence of configuration parameters on the performance is a challenging problem. We proposed a meta-heuristics based approach (in this case Genetic Algorithm) and machine learning techniques aided by domain knowledge. We have shown that such an

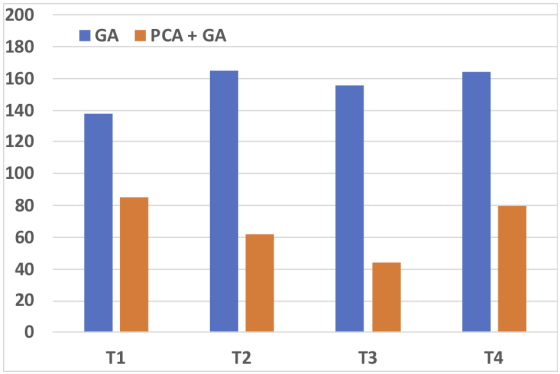


Fig. 7. GA Test Results

approach can robustly identify suitable values of configuration parameters that satisfy the target performance and deployment constraints.

Our ongoing work is focused on using other stochastic optimization techniques such as Simulated Annealing (SA) to find the configuration states that answer (Q2) above, and compare the results with GA. As with GA, we will design the Simulated Annealing so that it can intelligently jump to a new random state by exploiting the domain knowledge regarding the behavior of performance as a function of various parameters (e.g., monotonicity or unimodel behavior).

We envision that future work based on similar approach can help resource allocation in other domains, such as configuration of VMimages, Docker containers, etc. These domains exhibit similar behavior and modeling challenges as highlighted in our work. On the positive note, data center operators have a huge amount of operational data collected over time that can be leveraged to understand the system behavior and recommend the required configuration parameters.

References

1. IEEE Innovation at work. Real-life use cases for edge computing. https://innovationatwork.ieee.org/real-life-edge-computing-use-cases/, 2019.
2. Zhen Cao, Vasily Tarasov, Sachin Tiwari, and Erez Zadok. Towards Better Understanding of Black-box Auto-tuning: A Comparative Analysis for Storage Systems. In *2018 {USENIX} Annual Technical Conference ({USENIX}{ATC} 18)*, pages 893–907, 2018.
3. L. B. Costa and M. Ripeanu. Towards Automating the Configuration of a Distributed Storage System. In *2010 11th IEEE/ACM International Conference on Grid Computing*, pages 201–208, Oct 2010.
4. Hadka D. Platypus - Multiobjective Optimization in Python. ttp://platypus.readthedocs.io/en/latest/, 2019.
5. K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, April 2002.
6. Chin-Jung Hsu, Rajesh K Panta, Moo-Ryong Ra, and Vincent W Freeh. Inside-out: Reliable Performance Prediction for Distributed Storage Systems in the Cloud. In

2016 IEEE 35th Symposium on Reliable Distributed Systems (SRDS), pages 127–136. IEEE, 2016.

7. Ana Klimovic, Heiner Litz, and Christos Kozyrakis. Selecta: heterogeneous cloud storage configuration for data analytics. In *2018 {USENIX} Annual Technical Conference ({USENIX}{ATC} 18)*, pages 759–773, 2018.
8. Pierre Legendre and Loic FJ Legendre. *Numerical ecology*. Elsevier, 2012.
9. Effi Ofer, Amir Epstein, Dafna Sadeh, and Danny Harnik. Applying deep learning to object store caching. In *Proceedings of the 11th ACM International Systems and Storage Conference*, SYSTOR '18, pages 126–126, New York, NY, USA, 2018. ACM.
10. Oracle. Performance Evaluation of Storage and Retrieval of DICOM Image Content ... . http://www.oracle.com/us/industries/healthcare/058477.pdf, 2010.
11. F. Pedregosa, G. Varoquaux, and A. et.al. Gramfort. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.
12. Anand Prahlad, Marcus S Muller, and Rajiv et.al. Kottomtharayil. Data object store and server for a cloud storage environment, including data deduplication and data management across multiple cloud storage sites, October 9 2012. US Patent 8,285,681.
13. Jia Rao, Xiangping Bu, Cheng-Zhong Xu, Leyi Wang, and George Yin. VCONF: A Reinforcement Learning Approach to Virtual Machines Auto-configuration. In *Proceedings of the 6th International Conference on Autonomic Computing*, ICAC '09, pages 137–146, New York, NY, USA, 2009. ACM.
14. Sanjeev Sondur, Krishna Kant, and Slobodan Vucetic. Storage on the Edge: Evaluating Cloud Backed Edge Storage in Cyberphysical Systems. In *2019 IEEE 16TH International Conference on Mobile Adhoc and Sensor Systems (MASS)*, 2019.
15. Mohammad S Sorower. A literature survey on algorithms for multi-label learning. *Oregon State University, Corvallis*, 18, 2010.
16. Yusuke Tanimura and Hidetaka Koie. Operation-level performance control in the object store for distributed storage systems. In *2015 IEEE International Conference on Data Science and Data Intensive Systems*, pages 111–112. IEEE, 2015.
17. Elena Geanina Ularu, Florina Camelia Puican, George Suciu, Alexandru Vulpe, and Gyorgy Todoran. Mobile Computing and Cloud maturity-Introducing Machine Learning for ERP Configuration Automation. *Informatica Economica*, 17(1), 2013.
18. Mengzhi Wang, Kinman Au, Anastassia Ailamaki, Anthony Brockwell, Christos Faloutsos, and Gregory R Ganger. Storage device performance prediction with cart models. In *The IEEE Computer Society's 12th Annual International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems, 2004.(MASCOTS 2004). Proceedings.*, pages 588–595. IEEE, 2004.
19. Shuo Wang, Xing Zhang, Yan Zhang, Lin Wang, Juwo Yang, and Wenbo Wang. A survey on mobile edge networks: Convergence of computing, caching and communications. *IEEE Access*, 5:6757–6779, 2017.