Optimal Configuration of High Performance Systems

Sanjeev Sondur
Temple University, USA
sanjeev.sondur@temple.edu

Anis Alazzawe
Temple University, USA
aalazzawe@temple.edu

Krishna Kant
Temple University, USA
kkant@temple.edu

***Abstract*—Large scale systems often have numerous configuration parameters that must be set properly in order to achieve maximum performance under given resources constraints (or equivalently minimum resource requirements for acceptance performance level). The key difficulty in properly setting the parameters is the complex inter-dependencies between the consistent setting of the parameters and the overall impact of a setting on the resulting resource requirements or performance. In this paper we discuss an approach that combines machine learning and domain knowledge driven combinatorial optimization to allow the recommendation or one or more configurations from which the administrators can make the appropriate choice. We illustrate the methodology using the example of the cloud storage gateway (CSG), which forms an essential infrastructure service in edge computing and other scenarios involving remote access to large amounts of cloud storage. An extensive evaluation using real world vendor provided workloads demonstrates the effectiveness of the approach in providing fast solutions to the optimal configuration problem.**

Keywords–Edge Storage, Configuration Management, Principal Component Analysis, Machine Learning, Genetic Algorithms, Feature Extraction, Meta-heuristics,

I. Introduction

The behavior of all cyber-systems in a data center or an enterprise system largely depends on their *configuration* which describe the resource allocation to achieve the desired goal under given constraints. The ill-effects of misconfiguration (or poor resource allocation) has been widely articulated as unavailability [15], financial burden [7], security breach [2], etc. However, configuration settings cannot be classified as simply “correct” or “incorrect”; instead, the overall behavior of a device and that of the entire system depends on their setting and interactions between them. In particular, the interactions between various parameter settings and their nonlinear and often non-monotonic impact on performance rules out simple approaches, such as setting up a convex optimization problem with explicit objective function and constraints, and solving using traditional techniques such as hill climbing. Instead, we exploit machine learning and meta-heuristics that exploits the domain knowledge concerning the problem at hand to determine good configuration settings that satisfy the given objectives. It is also worth noting here that while traditional optimization methods provide a single solution to the problem, multiple solutions are often necessary in practice so that the administrator can choose from them based on considerations that are difficult to formalize.

While the methods explored in this paper are general and can be applied to almost any configuration problem, the domain knowledge is necessarily problem specific and is often crucial in obtaining sensible solutions. Therefore, we ground the analysis in this paper by considering the problem of configuring a local storage system backed up by a large remote storage system. The prime example of this is a Cloud Storage Gateway (CSG) [19] that couples an amount of locally available storage in a data center with a large but remote Cloud storage to create the impression of essentially unlimited storage. Doing so requires a proper setting of many configuration parameters of the local system as discussed later. Similar situations arise in many other storage contexts such as Network Attached Storage (NAS), Containers (VM image, Dockers), Cloudlets [22], etc. In particular, an Edge storage has to contend with many constraints (e.g., space, power, cooling, etc.) in addition to the strict QoS requirements, and thus forms an ideal context for studying the configuration of Cloud backed local storage.

We used a commercial CSG product for conducting our experiments and collecting the data for our configuration study. The details of the experimental data collection and its analysis is presented in our earlier work [23]. This work showed that while it is possible to build an accurate statistical machine learning model (SML) for the problem of predicting performance for a given configuration (hereafter referred as the ”forward problem”), it is very difficult to achieve acceptable accuracy from such a model for predicting configuration for a given performance/cost level (hereafter referred as the ”backward problem”). Furthermore, any such model will realistically apply to the configuration of only a small set of parameters. Thus the approach that we explore in this paper is to use the forward model as an *oracle* that is intended to be used sparingly along with meta-heuristics to determine multiple ”good” configurations. Our observational study from the experiments gave us insight to apply relevant domain knowledge to solve the “backward problem.” The meta-heuristics is guided by the domain knowledge so that it does not make entirely random choices and thus can converge substantially faster and yield better results than an unguided meta-heuristics.

In our performance prediction work [23], we showed that predicting the performance of a complex cyber-system is a challenging task because of complex interaction of a large number of parameters. However, it is possible to use machine learning techniques along with the domain knowledge to learn these relationships well enough to generate accurate performance predictions for given configuration settings. Unfortunately, the backward problem of recommending suitable configuration for given performance or cost targets remains unsolved, since the typical machine learning techniques are unable to achieve good accuracy and in any case a direct machine learning model would be specific to a particular combination of configuration parameters. This paper presents an efficient methodology to address the configuration question using an Edge storage system as an example. To the best of our knowledge, prior-art has addressed issues relating to resource provisioning and resource management, while we address the configuration problem as *finding or recommending* the required compute plus storage resource to satisfy a given condition (workload, performance, size, energy, etc.). We exploit the domain knowledge into meta-heuristics algorithm to reduce the (configuration) search space and converge at the required solution. In designing an approach to find a suitable configuration for a cyber-system, we use problem specific domain knowledge to enhance combinatorial optimization in order to converge at a solution (i.e. configuration state).

Our solution to recommend an optimal configuration (i.e. compute and storage resource allocation) uses Genetic Algorithm (GA) enhanced with principal component analysis (PCA) and feature importance (FI) metrics. The results from our modified Genetic Algorithm (**mGA**) algorithm show that they reach the maximum fitness (i.e. required configuration) about 22% faster than the generic versions of the algorithm, henceforth denoted as *gGA*.

The remainder of this paper is organized as follows. We describe the characteristics of the Edge Storage and configuration questions in section II. We present our solution approach with an enhanced GA - mGA in section III. Section IV gives a brief overview of the experiments and data collection. Section V discusses the evaluation methodology, implementation, and compares our solution results with the baseline algorithm. State of art and comparative study is given in section VI. We conclude the paper in section VII.

II. Configuration Management of Edge Storage

Given the importance of exploiting the domain knowledge in addressing the configuration management problem, it is important to understand some architectural details of the cloud storage gateway (CSG) that we experimented with and analyzed extensively for this work in the context of Edge computing (henceforth, referred as Edge Storage).

A. *Overview of Edge Storage Infrastructure*

Edge Storage Infrastructure (ESI) is shown in Fig. 1. It provides a *local storage 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. ESI 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 (D), ESI has to interface with Cloud Object-store over an unpredictable network (C) (low throughput, high latency) with *varied object sizes* that are largely dependent on the applications. In addition, ESI 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 the Cloud side. To address this imbalance, ESI 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.

Data Flow in an ESI: For a read operation, depending on the state of local data cache, ESI can either service the data locally or fetch the data from the Cloud Storage. ESI can achieve high read performance by pre-fetching data and associated metadata, but at the cost of occupying data-cache space. A write request from the edge application has to be persisted successfully to the Cloud Storage, first by persisting it locally on the data cache, and then transferring it successfully over to the Cloud. Both such read and write operations to the Cloud are limited by the amount of cache space (data cache plus meta-data area), the unpredictable network bandwidth, and overhead operations such as garbage collection, cache-eviction, meta-data operations, etc. However, the edge clients are transparent to these internal-details and view the ESI as a *virtual extension of the Cloud*. This virtualization is shown as the "logical data path" in Fig. 1.

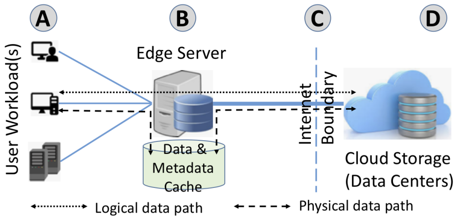


Figure 1: Workflow/ Data Path in Edge Storage Infrastructure.

Importance of Meta-Data: Although the Cloud storage could be block based, it is almost universally object based due to many advantages of Object Storage in the Cloud model [9]. In an Object-store system, every object is associated with the corresponding metadata that is maintained by a metadata server. When an object is initially requested by ESI, this meta-data 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 ESI 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 the workload access pattern and metadata management determine the performance experienced by the user. As part of our study in ESI configuration recommendation and resource allocation, we will explore the space allocated for data cache and metadata cache.

B. *Configuration Modeling for ESI*

ESI performance is defined as the throughput experienced by the edge-clients, and is generally expressed as either MB/s or as objects/sec [17]. ESI architecture involves the complexities inherent in storage systems, cache allocation, satisfying IO demands, and unpredictable network bandwidth [25].

ESI consumes computational resources for protocol translation, client authentication and capability based storage access management, meta-data operations, workload interfaces (block IO, Object-store APIs), etc. Thus, platform parameters such as core speed, number of cores, memory size, and memory

TABLE I: Sample Classification of Design Variables.

Attribute	**No. of 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 size (GB)	7	25, 50, 100, 200, 500, 1000, >1000
Metadata size (GB)	5	25, 50, 100, 200 & 500
Observed Performance	10	Uniform distribution (100Kbps,350Mbps)

bandwidth all become crucial. Satisfying latency sensitive edge-client IO request needs data buffering, intelligent cache operations, etc. that demand cache resources (both storage and memory). The limited storage resource in an ESI has to be efficiently partitioned for data-cache, meta-data, and other operational overheads (e.g. swap space, log files). Unpredictable network connectivity to the back-end Cloud 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 accurate and tractable analytic modeling of performance very difficult.

The throughput (p) experienced by the edge-clients depends largely on the workload (k), ESI hardware (h) and the resources (r) allocated to the compute and storage layers of ESI. 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 the detailed analytic characterization of the above parameters, we formulate the above parameters as a classification problem. Although, in theory, most parameters can take a large range of values, practical limitations, and sensitivity considerations usually confine the feasible values to a small set of discrete values. Table I provides an illustrative example in this regard.

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. This again is based on domain knowledge, since simply throwing in arbitrary platform parameters may actually dilute the model and lead to worse results, as presented in section V-A.

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 throughput p in terms of workload class w, ESI hardware class h and resource allocation class r as:

$$p{=}f_3(h{,}k{,}r) \tag{3}$$

Research Questions: The forward and backward problems introduced earlier could now be concretely defined as follows:

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

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

C. Problem Formulation

The complex relationship between various user settable parameters (compute capacity, storage resource, etc.) and vendor provided latent parameters (cache eviction rate, data replacement logic, meta-data operations, etc.) makes analytic models intractable. Therefore, we used machine learning techniques to learn the various relationships that influence the outcome (performance). We provide detailed analysis for RQ.1 and the proposed solution in our earlier paper [23]. For completeness, we present the salient aspects of the performance prediction model in section V-A, and use such an oracle to solve the configuration question in RQ.1.

Predicting satisfying configuration parameters for RQ.2, based on user workload and target performance is difficult 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 satisfies 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 solution (i.e selected configurations state Ψ) should satisfy the user given condition (i.e. performance p_{user}) at a minimum cost possible. We define a constraint function as:

$$p{\geq}p_{user} \tag{4}$$

where p is the expected performance from configuration Ψ. The objective to find such a configuration Ψ at a minimum cost.

$$\min(cost(\Psi)) \tag{5}$$

The objective can be expressed as the deployment cost, power consumption, cooling requirements, etc. The cost function is represented as the *normalized* cost of a configuration Ψ based on the design variables. For example, k^{th} configuration Ψ_k for some choice of hardware h_i and resource r_j, is represented as $\Psi_k{=}\{cs_i{,}nc_i{,}bw_i{,}{\cdots}db_j{,}md_j{,}..\}$ and has a cost $cost(\Psi_k)$. Data for cost function can be derived from vendor specification for hardware server and allocated resources (disk capacity). We approach the above question RQ.1 and RQ.2 in two phases: (i) a statistical machine learning (p-SML) model to predict the behavior (e.g. performance) of the system, and (ii) a combinatorial optimization guided by the domain knowledge to recommend a suitable configuration that satisfies given goals and constraints for user given workload/performance.

III. Determining Optimal Configuration

There are numerous algorithms for combinatorial optimization [1] designed to sample the state space in some way and improve the solution. Because of the inherent randomness in the way the states are explored, no algorithm yields universally better result than the others; instead, the quality of results on making use of the characteristics of the problem to improve the search. In this paper, we choose the popular genetic algorithm (GA) ([11, 21]). Algorithm 1 presents pseudocode of GA [21] and is based on evolutionary biology to optimizing a user given fitness function of a chromosome (solution) as the solutions evolve over time (lines 1-4). The fitness corresponds to how well a solution performs; i.e. inputs with the highest fitness are desired. The fittest chromosomes generate the next generation of children by random mutation and cross-over. Each element, or gene, of every chromosome is mutated with a probability defined by the mutation rate (line 5). The design variables in Eq.1 and Eq.2 form the genes in the chromosome in GA. The efficiency of the algorithm can be perceived in multiple ways, for example as the quality of the output chromosomes (defined by the fitness function) or how fast the optimal solution is obtained (defined by the number of iterations to reach the solution).

Algorithm 1: Pseudocode of Generic GA [21]

```
1 initialize the population;
2 evaluate population;
3 while (!stopCondition) do
4     select the best-fit individuals for reproduction;
5     breed new individuals through crossover and
        mutation operations ■;
6     evaluate the individual fitness of new individuals;
7     replace least-fit population with new individuals;
```

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 RQ.1 as the fitness function to determine if the current state (i.e. chromosome) satisfies the user required performance (Eq.4). *p-SML* performance prediction oracle is consulted to predict the performance of each chromosome (configuration state or design variable).

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 (at line 5), we propose to do it intelligently. We use additional insights from principal component analysis (PCA) derived from the p-SML model to control the gene mutation probability (marked as ■ in line 5 in Algorithm 1). We explain the metrics from PCA, probability factors, selection of principle design variables based on features importance in evaluation section V-C.

In addition to PCA, we incorporate domain knowledge in the algorithm by dividing the design variables into groups based on their level of interdependencies. That is, the design variables within a group show strong interdependence and thus should be set collectively, where the settings across groups can be done independently.

TABLE II: Grouping Design Variables

Group	Design Attribute Pairs
Group G1	Number of Cores, Memory capacity
Group G2	Core speed, Memory bandwidth
Independently varied	Data cache, Disk IO rate

For example, in the context of a computer system, it is well understood that a faster CPU should be paired with a faster DRAM, else the CPU will simply stall waiting for the memory. *Grouping* of configuration variables based on insights avoids exploration of states that are unlikely to be useful and thus is expected to both speed up the convergence and lead to better solutions within a given number of iterations.

With such enhancements, we show that our mGA solution with an "informed" approach can converge to the required configuration at least 22% faster than GA.

IV. Experiments and Empirical Data

We explain our evaluation methodology in three steps: (i) collecting empirical data based on industry workload, (ii) implementation details, and (iii) evaluation results.

A. Real World Workload

In absence of publicly available ESI data patterns or workload streams or trace dumps, we used a set of vendor provided workload patterns (shown in Table III), that are reflective of real-world ESI user population. Meta-data shown in the table refer to attributes that influence both the meta-data operations and the performance. These are shown in the meta-data column as ownership (O), sub-directory depth (F=Flat,D=Deep), and object-permission (P). These workloads are equivalent to studies from Yi [29], Varma [26], and YCSB [3].

TABLE III: Sample Workload Type and Applications.

Request Id	#users x objects x object_size	Meta-Data	Sample Appl. [26, 17, 29]
W1	25 x 10,000 x 4 KB	O,D,F,P	Health Monitors
W2	25 x 10,000 x 256 KB	O,D,F,P	MRI/ CT Scans/ Traffic Images
W3	5 x 10,000 x 1 MB	O,D,F,P	DICOM Visible Light
W4	5 x 1,000 x 10 MB	O,D,F,P	Mammography/ Street Video(1 min.)
W5	2 x 200 x 1 GB	O,D,F,P	Pathology

For example, workload patterns for a smart-health monitoring system is characterized as "W1" defined by image size of 4KB, about 10,000 images/24 hrs, with the associated meta-data on date, ownership, location, etc. Another workload pattern for health-care (e.g. Pathology) is characterized as "W5" 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. We used two hardware servers of the following configurations: (i) Server A comprising of 4 core x 1.8 GHz CPU, 16GB memory, 3 HDDs of 500 GB each & 1GB NIC cards and (ii) Server B with 8 core x 2.1 GHz CPU, 32GB memory, 1 HDDs of 1 TB each & 1GB NIC cards. For each of these experiments, we collected performance p metrics along with hardware (nc,cs,$mc\cdots$) and resource allocation (dc,$md\cdots$) details. The design variables (a.k.a problem attributes) were classified into buckets, a sample of which is shown in Table I. Observed metrics, such as performance was classified into buckets, i.e. performance of 467175 Bps and 21293642 Bps was classified as throughput $class$ 1 or throughput $class$ 4 respectively.

In order to examine the influence of configuration parameters (compute power, memory resource, cache disk space, etc.) on the final outcome (performance), we executed our tests on different hardware servers of varied configurations given earlier. Disk space r on each of these machines was partitioned for different data cache (dc) size from 25 GB to 1000 GB. Hard disk (i.e. the data cache) in the servers was nfs mounted and connected to Object-store on Cloud OSS. Each of the servers had Ubuntu 14.04, required tools to run the experiments and collect the metrics.

For each of the test execution, the scripts collected the design variables used and metrics observed of Eq. 1,2, 3. Along with performance observed p (throughput in bytes/sec) for a given workload k and resource allocation r, we collected the configuration details h (e.g. cores, core speed, memory, disk capacity, etc) and resources r (i.e. data cache area, metadata, log size). We decided to suspend data collection after reaching a performance prediction accuracy of about 95%. We implemented ESI cost function as a normalized value based on the 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).

B. Implementation Details

We implemented the algorithms in Python using *scikit-learn* [18] library for Machine Learning components such as Principal Component Analysis, Classifiers, ML metrics (e.g. accuracy, precision, etc.), Feature Importance etc. For combinatorial optimization, we used *NSGA-II* [6] Genetic Algorithm from Platypus library [5]. NSGA-II algorithm gives the flexibility to define fitness function, define objectives and constraints, variable bounds, chromosome construction, crossover and mutation, solution-set, etc.

C. Hyper-parameters of GA

For combinatorial optimization algorithms like GA, it is difficult to determine hyper-parameters a priori [11]. We choose the hyper-parameters after several executions and choose the values that gave the best results. For GA & mGA, we set the initial population of 170 with the tournament selector to choose the top 8 "fit candidates" for next generation mutation. To enable meaningful comparison of different algorithms, we choose to count the number of calls to *performance-prediction-oracle*, and as explained earlier, a good solution would result in a smaller number of calls to such an oracle. The count of calls to the oracle is shown as iterations (y-axis) in the results.

V. Evaluation Results

In this section, we discuss experimental results for both the forward and backward prediction problems discussed earlier. For the latter, we present the two independent results based on the domain knowledge enhanced meta-heuristics processes.

A. Influence of Chosen Feature Sets on Performance

As stated earlier, including the configuration parameters (or feature set) without 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 a 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 attributes 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 labeled 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 prediction accuracy (down to 91%). Based on our extensive experience and domain knowledge 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 ESI performance. These results reinforce our earlier comments regarding the selection parameters in Eq. 1 and Eq. 2.

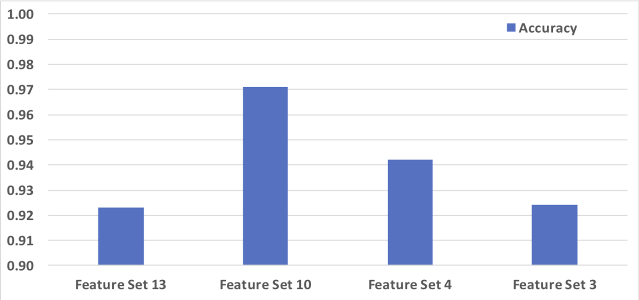


Figure 2: Performance prediction accuracy and feature set.

There is no auto-solution for improving the 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. [24]. Therefore, we tried several types of models and ultimately settled on Decision Tree (DT) for RQ.1, as it consistently performed the best (see Fig.3). Building a performance prediction model for RQ.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.

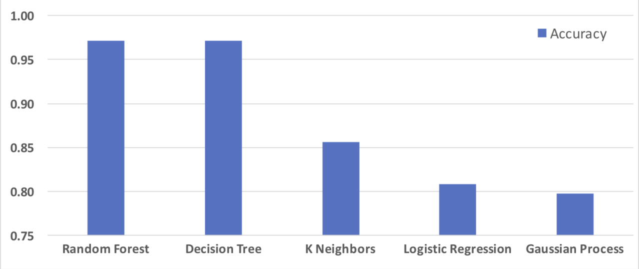


Figure 3: Performance prediction accuracy and ML predictors.

B. Extracting Feature Importance

Principal Component Analysis (PCA) is a 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* [14], which can be easily extracted. Large loadings (positive or negative) indicate that a particular variable has a strong relationship with a particular principal component. The sign of a loading indicates whether a variable and a principal component are positively or negatively correlated.

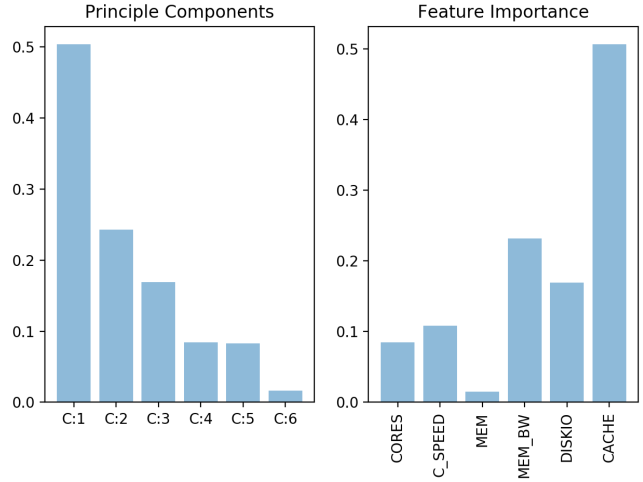


Figure 4: PCA and Feature Importance.

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 and PCA & Loadings from the PCA objects to gain confidence in ranking the predominant attributes that contribute to ESI 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···C6) on x-axis, and the right sub-graph shows FI values for the design variables (on x-axis). Based on the 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 the mutation process. We used the above metric from PCA and FI to probabilistically mutate the genes in the modified (PCA+GA) mGA approach and generate a 'controlled' new state. The results of our mGA solution is given below.

Next, we discuss the solution for the backward problem (BP), relate to finding the near-optimal configuration Ψ that satisfies a given workload/performance criteria (Eq. 4) at minimal possible cost (Eq. 5). We compare the functions against the baseline and validate how quickly an algorithm converges at such a required configuration state.

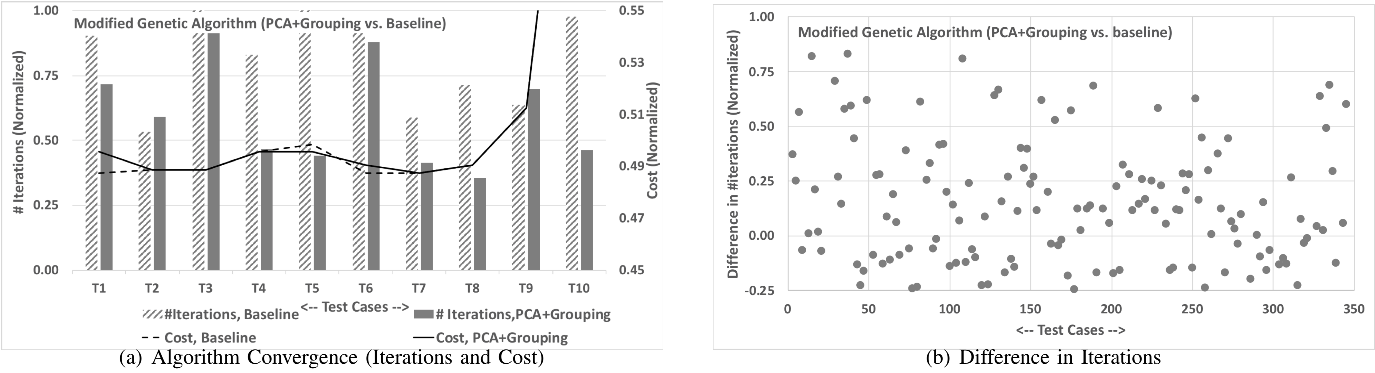
C. Recommending a Configuration using mGA

The design variables (CPU, memory, IO bandwidth, etc.) plus the workload properties (file size, meta-data, no. of files, etc.) forms the chromosome in the gene pool that represents a population. 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 Decision Tree from section V-A selects a sub-set of the population (design variables) that satisfy the user defined performance. To ensure efficiency, this performance predicting oracle has to be consulted sparingly for rapid convergence.

In GA, 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 a suitable solution (i.e. population with predicted performance equal to user defined performance). An uncontrolled mutation may result in the design variables being randomly selected from a wide range and this may result in finding a suitable solution after a considerable time (measured as the number of calls to the oracle). 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).

We extracted additional data from ML objects, PCA model, and feature importance (FI) as explained in section V-B. 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 ESI 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 mGA process to move to a new state (i.e new population set) in a controlled fashion. 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 (Eq.4), normally at a minimum cost (Eq.5). In our implementation, we keep track of the number of iterations required to "find" a satisfying configuration such a minimal cost. The algorithm "records" the number of calls to the oracle needed to obtain the most satisfying configuration state (chromosome in solution set) at a "minimum" possible cost and time. Fig. 5(a) shows the normalized values of such convergence (iterations) for both mGA and gGA algorithms for various test cases (T1···T10) along with the cost function for the solution (Ψ). This figure shows a sub-set of 10 test cases (x-axis) from large number of test cases we executed. As seen in Fig. 5(a), with a controlled gene mutation in mGA algorithm, the design variables find the satisfying fitness function (performance) faster at the same minimum cost in less number of iterations. The cost of the



(a) Algorithm Convergence (Iterations and Cost)

(b) Difference in Iterations

Figure 5: GA and mGA (PCA+Grouping) Test Results (Iterations and Cost)

configuration from mGA algorithm matches the minimal cost obtained in the gGA algorithm (line graph in Fig. 5(a)).

Fig. 5(a) shows the test cases executed on the x-axis and the normalized number of iterations and costs on the y-axis. (The normalization is w.r.t. 500, which is the maximum number of iterations.) The lines for the costs refer to the minimal cost of solution (Eq.5) obtained by both mGA and GA. 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 mGA based approach converges to a solution after 225 iterations with a solution: 2 cores x 3.2GHz, 16 GB Mem, 3.2GB Mem bus, DiskIO = 10K IOPS, Normalized Cost = 0.4875. The same query to a GA takes about 333 iterations to find a minimum cost solution.

Fig. 5(b) shows a more comprehensive comparison between gGA and mGA. The metric on y-axis is [#iterations(GA) - #iterations(mGA)]/500, and the x-axis is simply the 400 cases that were run with different parameters. Note that the dots above the x-axis (positive values) show that mGA performs better than gGA, whereas negative values show the opposite. It is clear that mGA outpaces gGA in almost 90% of the cases. Furthermore, in the cases where mGA performs better than GA, it takes 22% fewer iterations than gGA on the average.

VI. State of Current Art

In a recent survey, Zhang [30] categories various techniques in resource management in Cloud and Edge computing as latency optimization, shorter task completion time, container placement, data replication, and Edge caching strategies. Wang [28] presents a survey on the impact of Edge caching capacity, delay, and energy efficiency on system performance in a mobile Edge server. Meta-heuristics based work has been proposed to solve Cloud resource provisioning problems by allocating applications among the virtual machines to satisfy user QoS [13]. In CHOPPER [8], authors present a resource scheduling and provisioning scheme based on QoS metrics (execution time, execution cost, energy consumption, and waiting time). Task offloading, workload scheduling, application placement, and migration schemes used in Cloud/ Edge Computing largely pertain to network and computational resource allocation. This flexibility is not available for ESIs as they have to fulfill the workload request "locally and immediately". Our work focuses on *choosing a set of configuration parameters* that satisfy user workload/performance demands under given conditions.

Klimovic and Costa [4] [12] confirm our observations regarding the difficulty of proper configuration in Cloud storage systems. In designing Selecta, Klimovic address the storage configuration for data analytics workload using TPC traces on block storage devices inside data centers, while our work studies the Object-store based ESI configuration on the Edge side using vendor provided workloads. Costa [4] state that configuring a storage system for desired deduplication performance is extremely complex and difficult to characterize. Rao [20] show that a traditional control theoretic framework is inadequate to capture the complexities of resource allocation for VMs. Ofer [16] study comes close to our work, but their study applies deep learning to cache eviction/refresh techniques in Object-store, rather than setting configuration parameters.

For the prediction of performance vs. workload parameters of a storage system, Wang [27] used a Classification Regression Trees (CART)-based model and showed a relative error between 17% and 38% for response time prediction. Hsu designed Inside-Out [10] to predict performance in a distributed storage system by studying low-level system metrics (e.g., CPU usage, RAM usage and network I/O) as a proxy for measuring high-level performance. Compared to Inside-Out performance prediction accuracy of 91%, our pSML model achieves an accuracy above 95%.

VII. Conclusions

In this paper, we presented an approach to deciding optimal configuration by constructing a machine learning model for the performance and/or cost and then using as an implicit function to be optimized using combinatorial optimization based on genetic algorithm (GA). Furthermore, we enhance the GA by exploiting the domain knowledge in terms of the relative importance of various configuration parameters and their interrelationships. We explore a real-world storage gateway configuration using this methodology and show that it yields the results of same or better quality as the standard GA but in about 22% fewer iterations. In the future we plan to explore how the domain knowledge concerning the configuration parameter setting in different types of systems can be used with GA and other combinatorial optimization methods.

References

[1] Leonora Bianchi, Marco Dorigo, Luca Maria Gambardella, and Walter J Gutjahr. A survey on metaheuristics for stochastic combinatorial optimization. *Natural Computing*, 8(2):239–287, 2009.

[2] Richard Chirgwin. Suspicious BGP event routed big traffic sites through Russia, 2017.

[3] Brian F Cooper, Adam Silberstein, Erwin Tam, Raghu Ramakrishnan, and Russell Sears. Benchmarking cloud serving systems with ycsb. In *Proceedings of the 1st ACM symposium on Cloud computing*, pages 143–154. ACM, 2010.

[4] 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.

[5] Hadka D. Platypus - Multiobjective Optimization in Python, 2019.

[6] 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.

[7] Stephen Elliot. Amazon. com goes down, loses $66,240 per minute, 2017.

[8] Sukhpal Singh Gill, Inderveer Chana, Maninder Singh, and Rajkumar Buyya. Chopper: an intelligent qos-aware autonomic resource management approach for cloud computing. *Cluster Computing*, 21(2):1203–1241, 2018.

[9] Raúl Gracia-Tinedo, Josep Sampé, Edgar Zamora, Marc Sánchez-Artigas, Pedro García-López, Yosef Moatti, and Eran Rom. Crystal: Software-Defined Storage for Multi-tenant Object Stores. In *Proceedings of the 15th Usenix Conference on File and Storage Technologies*, pages 243–256. USENIX Association, 2017.

[10] 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.

[11] Alexander Kerr and Kieran Mullen. A comparison of genetic algorithms and simulated annealing in maximizing the thermal conductance of harmonic lattices. *Computational Materials Science*, 157:31–36, 2019.

[12] 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.

[13] Mohit Kumar, SC Sharma, Shalini Goel, Sambit Kumar Mishra, and Akhtar Husain. Autonomic cloud resource provisioning and scheduling using meta-heuristic algorithm. *Neural Computing and Applications*, 2020.

[14] Pierre Legendre and Loic FJ Legendre. *Numerical ecology*. Elsevier, 2012.

[15] Lily Hay Newman. How a tiny error shut off the internet for parts of the US, 2017.

[16] 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.

[17] Oracle. Performance Evaluation of Storage and Retrieval of DICOM Image Content ... , 2010.

[18] F. Pedregosa, G. Varoquaux, and A. et.al. Gramfort. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[19] 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.

[20] 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.

[21] Mahmood Rashid, M A Hakim Newton, Md Hoque, and Abdul Sattar. Mixing energy models in genetic algorithms for on-lattice protein structure prediction. *BioMed research international*, 2013:924137, 09 2013.

[22] M. Satyanarayanan. The emergence of edge computing. *Computer*, 50(1):30–39, 2017.

[23] Sanjeev Sondur and Krishna Kant. Towards automated configuration of cloud storage gateways: A data driven approach. In *International Conference on Cloud Computing*, pages 192–207. Springer, 2019.

[24] Mohammad S Sorower. A literature survey on algorithms for multi-label learning. *Oregon State University, Corvallis*, 18, 2010.

[25] 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.

[26] Ravi Varma. Storage media for computers in radiology. *The Indian journal of radiology and imaging*, 18:287–9, 11 2008.

[27] 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.

[28] 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.

[29] Shanhe Yi, Cheng Li, and Qun Li. A survey of fog computing: Concepts, applications and issues. In *Proceedings of the 2015 Workshop on Mobile Big Data*, Mobidata '15, pages 37–42, New York, NY, USA, 2015. ACM.

[30] Yuchao Zhang and Ke Xu. A survey of resource management in cloud and edge computing. In *Network Management in Cloud and Edge Computing*, pages 15–32. Springer, 2020.