Automated Configuration for Agile Software Environments

Negar Mohammadi Koushki, Sanjeev Sondur, and Krishna Kant
Computer and Information Sciences
Temple University
Philadelphia, USA
{koushki|sanjeev.sondur|kkant}@temple.edu

***Abstract*—The increasing use of the DevOps paradigm in software systems has substantially increased the frequency of configuration parameter setting changes. This is generally a very challenging problem due to the complex interdependencies between various configuration parameters and calls for an automated mechanism that can both run quickly and provide accurate settings. In this paper, we propose an efficient discrete combinatorial optimization technique for this purpose that makes two unique contributions: (a) an improved and extended metaheuristic that exploits the application domain knowledge for fast convergence, and (b) the development and quantification of a discrete version of the classical tunneling mechanism to improve the accuracy of the solution. Our extensive evaluation using available workload traces that do include configuration information shows that the proposed technique can provide a lower-cost solution (by ∼60%) with faster convergence (by ∼48%) as compared to the traditional metaheuristic algorithms. Also, our solution succeeds in finding a feasible solution in approximately 30% *more cases* than the baseline.**

***Index Terms*—Configuration Modeling, Resource Allocation, Resource Provisioning, Machine Learning, metaheuristics, Simulated Annealing**

I. Introduction

The DevOps transformation of IT services is fueling a radical change in how cloud services are conceptualized, designed, and implemented [1]. The service oriented architecture (SoA) and its manifestation in the emerging microservices paradigm [2] attempt to reduce dependencies between services to make their development and deployment more agile and allow for easier maintenance and scalability. The motto of the DevOps phenomenon is *Continuous Integration and Development* (CI/CD) [3]; that is, the *deployed services* are constantly being enhanced and tuned both with respect to their code and run-time settings. Furthermore, the services are increasingly being deployed in their own lightweight containers which allow even the hardware resources assigned to the service fungible at run-time.

The run-time settings, popularly known as *Configuration Parameters* (CPs), play a key role in the functioning of most software systems and are easily misconfigured due to a variety of reasons including a lack of clear documentation/understanding of what they do, interdependencies across them and lack of robust automated mechanisms to set them. In traditional systems, their incorrect setting (or "misconfiguration") are responsible for up to 80% of down-times [4] and up to 85% of the incidents [5], with *one week on average* for root cause identification [4, 5].

The CI/CD process substantially increases this complexity and the resultant potential for ill effects due to the need to constantly tune the CPs as the service modules continuously undergo changes. In the current agile software systems, configuration changes happen all the time; e.g., reported thousands of times a day in Facebook [6]. Most of these are related to "tuning" the CPs rather than drastic changes. Thus, an automated mechanism that can quickly define the configuration settings (or verify proposed configuration settings) becomes a crucial ingredient of the emerging DevOps drives software development processes that continue to get adopted across a large variety of application domains. The purpose of this paper is to develop such a mechanism focused on the performance and cost aspects.

Our Contributions: Owing to the complexity of performance (and possibly cost) functions, we adopt a metaheuristic-based discrete combinatorial optimization to solve our problem. Our key contributions in this regard are: (a) extending the metaheuristic for efficiently solving the constrained optimization problem that exploits the application domain knowledge in grouping and choosing the parameters for perturbation, (b) developing the classical notion of "tunneling" in the context of implicit, discrete performance function, and (c) explore how and when the tunneling helps in an efficient solution. Our extensive evaluation using available workloads shows that the proposed technique can provide a lower-cost solution (by ∼60%) with faster convergence (by ∼48%) as compared to the traditional metaheuristic algorithms. Also, our solution succeeds in finding an average of 30% *more additional* solutions than the baseline.

The rest of the paper is organized as follows. Section II reviews the configuration selection techniques and challenges. Section III provides an overview of our proposed method. In section IV, we describe the approach for selecting configuration. Section V addresses the issues in configuration modeling and then in section VI, we present our experimental evaluation approach. We conclude the paper in section VIII.

II. Current Art on Configuration Selection

The current state of the art explored during our work shows that configuration issues are related to resource provision-

ing and resource management [7, 8] techniques to optimize latency, task completion time, data replication, and impact on cache capacity, delay, and energy consumption. Our work addresses *recommending* a suitable resource allocation (e.g., storage, compute, bandwidth) to achieve the desired goal (e.g., workload performance, energy, cost, size, etc.). Cloud resource allocation is a challenging job since the final outcome like Service Level Agreement (SLA) & Quality of Service (QoS) requirements of workloads derives from the provisioning of appropriate resources to cloud workloads. Ref. [8] highlights various challenges and cites that the discovery of the best workload–resource pair based on application requirements of cloud users is an optimization problem. Ref. [9] addresses this by using a metaheuristics approach to provision Cloud resources for satisfying QoS.

To overcome the difficulty in characterizing the behavior outcome (e.g predicting performance), several studies have used Classification Regression Trees (CART)-based model [10] and ML techniques to design a performance influencing model (PIM) [11]. In our study, PIM is only the first step to building a surrogate function to solve the combinatorial optimization problem. Our work focuses on *choosing a set of CPs* that satisfy user workload/performance demands under given constraints. Supporting the complexity in our configuration work, authors in [12] use an example of Cloudlet infrastructure to show that configurations of Cloudlets are very challenging because of the many unknowns pertaining to the software mechanisms and controls.

Challenges in Configuration and Resource Allocation: In studying the importance of the resource allocation mechanisms in Cloud/Edge infrastructures, authors in Ref. [7] highlight the algorithmic challenges in efficiently using the Cloud resources (such as available computing, storage, and networking infrastructures) to serve the workload and data. They show that a sheer number of Cloud resources makes it difficult for users to effectively optimize their parameters (i.e., resource allocation) that ensures required QoS/SLA (e.g., performance, latency, execution time) while keeping the associated costs low. In Ref. [13], authors show that Cloud operators need to increase resource utilization while maintaining good performance. They state that it is difficult to achieve optimal resource allocation because of: (i) uninformed over-provision, (ii) the diverse/dynamic nature of applications, and (iii) that the performance depends on multiple resources.

We address the *dynamic resource allocation problem* capable of allocating multiple resources (such as CPU, memory, network bandwidth, and storage (I/O) bandwidth, etc.) to achieve the required QoS/SLA (e.g., throughput, latency, execution time) with minimal costs (metrics like deployment/maintenance cost ($), energy, power, etc.). Instead of relying on simulation tools, we demonstrate the effectiveness of our solution using publicly available data-sets (see Table. II) from real-world environments like Cloud & Edge applications, HPC workloads, application services, etc.

III. Overview of Proposed Method

In this paper, we focus on the problem of determining configuration parameters that minimize some cost functions subject to some minimum performance requirements. Essentially the same methods apply if instead, we try to maximize performance for a given cost constraint. In a virtualized environment, the cost may refer to either the actual cost charged by the provider (e.g., AWS) or costs that we assign to various resources including CPU cores, memory BW and size, I/O rate, storage space allocated, etc. The appropriate configuration settings must often be determined quickly and automatically to cater to the CI/CD needs. Our work addresses some specific questions raised by the DevOps team and further, supported in Ref. [8], wiz: (i) How to design the resource allocation mechanism to provide dynamic scalability at CPU, network, application-level, etc.?, & (ii) How to minimize the cost and optimize the resource allocation simultaneously?

Regardless of whether the performance is used as an objective function or constraint, it is likely to be a very complex function of various configuration parameters; thus accurate analytic or simulation models are likely to be very difficult to construct and time-consuming to run. Therefore, we turn to a machine learning (ML) model that can generally be queried very fast to satisfy the needs of CI/CD. The main drawback of an ML model is the need for substantial amounts of data for training that mostly covers the parameter ranges that are like to be used in practice. A beneficial side-effect of the configuration dynamism in DevOps environments is the availability of data with many different configuration settings. At the same time, the dynamism is likely to be limited to sensible ranges for an operational system. Thus the ML model for performance as a function of configuration parameters that are routinely retrained can fulfill our needs well (commonly referred to as PIM [11] and shown as "forward problem" in Fig. 2a). Although the cost model could also use the same approach, it is likely to be much simpler, and thus simple analytic expressions for it are generally adequate. We assume this to be the case in the rest of the paper.

A. Metaheuristics Based Modeling

It has been demonstrated in [14] that it is difficult to build an ML model directly for the reverse problem of setting the configuration parameters (shown as "backward problem" in Fig. 2a); therefore, we will devise a method that uses the performance and cost models directly to estimating optimal configuration parameter values. Because of the lack of convexity of the performance function in general, a suitable approach is to use a discrete combinatorial optimization using metaheuristics [15]. All methods aim to explore the state space efficiently while avoiding being trapped in the local minima, of which there could be many. Fig. 1 illustrates the search process pictorially with the x-axis representing the iterations and the y-axis the solution obtained in each iteration. The algorithm will keep track of the minima obtained so far and may or may not discover the global minima until the maximum iteration count (a hyper-parameter of the algorithm) is reached. Using

public domain data from several Cloud environments, we demonstrate the effectiveness of our solution with two distinct, yet equally important metrics: (i) the speed in searching the huge configuration space and "quickly" selecting a suitable solution, & (ii) ensure that the selected solution is a minimum cost solution. We explain this further in the evaluation section (Section VI-A).

Since our problem involves constrained optimization, we also need to ensure that any accepted solution is feasible. In such a setting, it is always helpful to avoid generating infeasible solutions in the first place, but this is not always possible. Regardless of the metaheuristics used, the stochastic optimization techniques move from the current best solution to the next solution that is both feasible (i.e., satisfies the constraints) and is better. Ideally, we would like to choose the next solution that is likely to have these characteristics without considering solutions that are unlikely to be useful. This is where domain knowledge is crucial. Often, domain knowledge consists of an abstract relationship between CPs or rules of thumb that can be evaluated easily. However, since they are fuzzy and not strictly required, they cannot be used as formal constraints. Another kind of domain knowledge concerns the varying influence of parameters that can guide which parameters need to be perturbed and by how much to get to the next proposed solution. Yet another aspect concerns an estimate of the amount by which one needs to move to get out of the region of local optimality to land in another region that can possibly provide a lower local optimum.

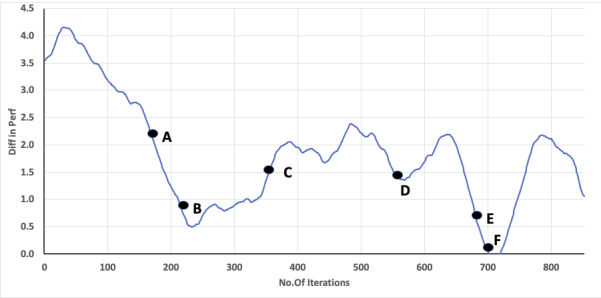


Fig. 1: Stochastic Tunneling.

B. Tunneling in Metaheuristics

A useful notion in stochastic optimization is *tunneling* [16] illustrated in Fig. 1, where we "tunnel" from local minima to a deeper local minima directly [17]. Traditional tunneling works in the continuous parameter state space with explicit objective function $f(x)$ where x is the input vector (i.e., configuration vector in our case). It also assumes that $f(x)$ has the first two derivatives. The method works in two steps: (a) find the local minima, say x^*, using the steepest descent algorithm from the current point, (b) minimize the modified function $h(x) = [f(x) - f(x^*)]/\|x - x^*\|^\alpha$ with $\alpha \geq 1$ to determine the next solution, say x'. It can be seen that with larger α, nearby points are penalized. (The choice of α can be problematic, and other approaches have been investigated [18]). Thus if we choose x' based on the gradient of $h(x)$ away from x^*, we are more likely to go towards a deeper minima than at x^*.

Although such a mechanism cannot be applied this technique directly to our problem, due to implicit $f(x)$ and discrete parameter space, we show that such a technique can improve the solution performance considerably.

IV. Combinatorial Optimization Based Configuration Selection

A general formulation of the configuration selection problem is as follows. Let CP denote the configuration defined as the vector of user-settable parameters $\vec{x}$ and vendor-selected (usually hidden) parameters $\vec{y}$. These along with the workload parameters $\vec{w}$ determine the desired objective function ϕ subject to some constraints. That is,

$$\vec{CP} = \{\vec{x}, \vec{y}\} \tag{1}$$

$$\phi = f(\vec{w}, \vec{CP}) \tag{2}$$

$$g_i(\vec{w}, \vec{CP}) \geq 0 \quad i = 1, 2, .., K \tag{3}$$

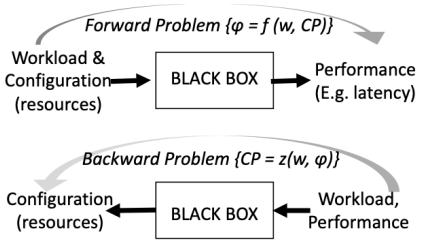
where $f()$ could represent performance or cost, and $g_i()$ is the ith constraint involving the workload and configuration parameters. The functions $f()$ and $g_i()$'s are usually quite complex and may not be expressible explicitly. Our study is supported by Ref [19], wherein authors state that performance modeling is complex since the running time (performance or throughput) is affected by the number of resources in the Cloud configuration in a non-linear way, and performance under a Cloud configuration is not deterministic. It is also worth noticing that the configuration space Ω is often discrete, with intermediate values being practically infeasible, even if they are conceptually meaningful. For example, if the memory modules for the systems at hand have a minimum granularity of 16GB, an installed memory of 24GB is infeasible. Thus, defining continuous or differentiable extensions of the functions $f()$ and $g()$ is neither straightforward, nor meaningful. Thus, the traditional tunneling structure is not possible; also, while one could estimate the local gradient by evaluating the functions at nearby feasible points, the value of local search is less clear.

Behavior $f()$ is expressed as the user expectation and can refer to performance, latency, throughput, etc. The constraints can represent the cost factor of such a system, heat dissipated or cooling needs, energy consumed, physical size, etc. Often, it is desirable to optimize multiple parameters simultaneously; however, in this paper, we consider objectives and constraints as a singular function.

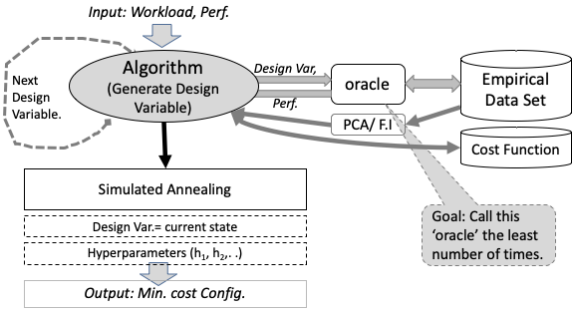
A. Basic Approach

Fig. 2 shows the overall scheme explored in this paper. Given a set of CPs, the first step is to define an "oracle," or a model for the **forward problem** of performance prediction based on the settings of CPs. As discussed in [20], statistical ML techniques work quite well for this. Authors in Ref. [14] have also shown that the ML techniques do not work well for the **backward problem** of configuration selection and require large amounts of training data. To overcome the above problem, we take advantage of the corresponding training data to determine a relative ranking of importance of various parameters via Principal Component Analysis (PCA) or similar

analysis as shown in Fig. 2b. This helps in both *confirming* and applying the domain knowledge such as (i) various relationships (e.g., higher CPU speed requires lower memory latency), (ii) generic rules of thumb (e.g., additional 64MB of memory per additional VDI client), (iii) system-specific ones that have been observed or (iv) can be deduced from the training data. It is important to underscore the importance of domain knowledge here since a blind application of these techniques is likely to yield incorrect or misleading results.



(a) Forward & Backward Problem



(b) Design of Algorithms

Fig. 2: Basic Approach: Design of Solution.

The problem to be solved is now to select a configuration $\vec{CP}$ (or $\vec{x}$ a few user-settable configurations) that provide a performance p above the lower bound (user desired performance p_u) while minimizing the cost of the solution. The choice of parameter values in $\vec{x}$ can directly or indirectly affect the objective function, i.e., ω cost of the configuration. This is shown as $z()$ depicting the "Backward Problem" in Fig. 2a.

We now define the constraint as the desired performance p_u:

$$p = \phi \geq p_u \tag{4}$$

where p is the expected performance from configuration $\vec{x}$. The objective is to find such a configuration $\vec{x}$ at a minimum cost.

$$\min(g(\vec{x})) \tag{5}$$

The "cost" of a configuration can represent a user's desired metric, such as the deployment cost, power consumption, cooling requirements, etc. Data for cost function can be derived from vendor specification for hardware server and allocated resources (e.g., disk capacity) or other suitable function $g()$. For example, k^{th} configuration $\vec{x_k}$ for some choice of parameters such as number of CPU cores, core speed, memory bandwidth, IO bandwidth, storage capacity, etc. has a cost $g(\vec{x_k})$.

Given the non-convex and non-monotonic influences of various parameters, the use of combinatorial optimization is natural for solving the *backward problem* (shown as $z()$ in Fig. 2a). In general, this optimization could be either *deterministic* or *stochastic*, where the latter allows for uncertainty in the objective function. Although our interest is in the deterministic case, uncertainties arise naturally in real-world problems (e.g., the cost of the solution better described by a distribution rather than a single value). In the stochastic case, the objective is generally to use a statistical measure (e.g., expected value) so that essentially the same methods apply in both cases.

All stochastic methods explore a sequence of next states to find a better solution with different techniques to make a trade-off between the expense of exploration (i.e., number of iterations, cost of evaluation) and the quality of the solution. For the latter, the algorithm must necessarily consider states where the objective function is worse than the optimum found so far, which means that a monotonic convergence is generally not possible.

Because of their stochastic nature, these algorithms do not have any guarantee that the result will be optimal for every run. Many comparative studies have been put forward to prove or disprove the efficiency of a particular method [21]. Therefore, the focus of our research work is *not* to compare algorithms; rather, we show that embedding domain knowledge into stochastic methods can help the algorithm to converge into a solution faster (than an uninformed algorithm). Our work focused on studying algorithms for their convergence speed and their capacity to find good objective function minima. On this front, we adapted our solution to a familiar stochastic method, wiz. Simulated Annealing (SA). We show the modifications in Table. I and explain such modifications in the following sections.

B. Emulating Tunneling

To apply the Tunneling approach to our context, we first seek the local minima (step 1), followed by an intelligent perturbation based on the sensitivity to take us further away from the current minima (step 2). To prepare for this, we first determine the relative ranking of each CP in terms of its influence on the objective function and the complex constraints. A pure data-driven method for doing this is the standard PCA and should work well assuming a sufficiently large data-set. Starting with the most dominant parameter, we form a group by pulling in other parameters known to be related to it (based on domain knowledge). We then start with the next most dominant parameter and form the next group until we have covered all the parameters. Then our approach with "smart" tunneling can be described by the following two steps:

1) We approach the local minima by considering the variation with respect to the leader of the first group. The gradient needs to be determined by taking a few samples.
2) The tunneling phase then perturbs each group leader in proportion of the PCA metric and adjusts all others in the group based on the known relationships

A stochastic optimization process works by randomly jumping from the current state x_i to a new state x_{i+1} based on some probability factor ρ, with an aim to find local minima x^* that

minimizes the objective function $f(x)$. We apply the above approach with tunneling by enabling the stochastic algorithm to circumvent the local minima points and rapidly move from an area of shallow minima (point (a) in Fig. 1) to a region of deeper minima (point (d) in the same figure), thereby allowing for faster exploration of solution space and faster convergence to a good solution. With a configuration problem at hand, as our objective function cannot be characterized by a direct analytical function, we use the performance prediction oracle (a black box) function as an objective function.

We explore ways to tunnel through the shallow minima (point (b) in the same figure) and avoid the slow dynamics of the complex objective function. Such tunneling mechanisms should invariably use the domain knowledge to intelligently jump the local barriers and avoid uninteresting space (i.e., avoid points (d and e) in the same figure). We group the design variables together as a discrete space tunneling mechanism to aid the algorithm from being trapped at a local minimum and (jump through or) tunnel out of the minimum. We present the details in the next sections in context to the algorithm we explore.

C. Generic Design Approach Summarized

The approach described above can be generalized independent of the data-set and domain as:

1) Run experiments to collect the data with relevant configurable parameters and observable outcomes.
2) In the absence of a clear analytical function to describe the relationship between the configurable parameters to the outcome, use a suitable ML model to design an "oracle" as a prediction engine (a.k.a PIM).
3) Use PCA metrics from the data, determine the relative importance of design variables and group attributes based on domain knowledge to avoid exploring undesired spaces.
4) Use the above PIM model to represent the objective and/or constraint function in a stochastic algorithm.
5) To explore new design states, use PCA metrics as probability factors and perturbative the variables in groups.
6) Verify new state satisfies constrain using ML-based oracle as a tool.
7) Accept/Reject the current design state based on satisfying criteria.

We explain the modification to a familiar metaheuristics-based stochastic process, i.e., SA, below.

D. Modified Simulated Annealing (*mSA*)

SA is a general probabilistic local search algorithm generally used to solve difficult optimization problems. The pseudocode [22] for generic SA (gSA) is given in Algorithm 1. In SA, a state refers to a set of design variables, and a neighboring state refers to a set of values relatively closer to current design variables. In SA, entropy is represented as the cost function that has to be minimized. An acceptable state is a solution to the problem that is being solved.

The SA method has been widely used since the cost function can be easy to put into practice [22]. Our SA algorithm uses

Algorithm 1: Pseudocode for SA [22]

```
1  initialize(temperature T, random starting point) ;
2  for  i in T do
3  |   p = select_point_from_neighborhood(i) ■ ;
4  |   currentCost = compute_currentCost_at(p) ;
5  |   δ = currentCost - previousCost ;
6  |   if δ ≤ 0 then
7  |   |   accept_neighbor_point(p) ;
8  |   else
9  |   |   accept with probability exp(-δ/T) ♦ ;
10 |   T = β * T ■ ;
```

design variables from the configuration ($\vec{x}$) to represent the state. The entropy of the system is defined as the cost of the current state (i.e., cost of the configuration $g(\vec{x})$). The gSA steps in Algorithm 1 can be summarized as follows: (i) we first start with an initial annealing temperature (T_0) and a random design state (line 1), (ii) we search for the next state depending on the annealing temperature T_k and random distribution (line 4), (iii) we compute the difference in entropy (δ) between the current state and past state (line 5), and probabilistically accepting the current state depending on Boltzmann probability factor (line $6 \cdots 9$). In line 9, if the current solution is accepted, we apply tunneling logic to search for better local minima. The annealing scheme is defined in line 10. The algorithm stops after reaching a defined cooling temperature (line 2).

Our solution is based on very fast simulated annealing (VFSA) presented by Xu [23], which enhances both the annealing temperature (line 10) and the perturbation model (line 4). Lee [24] and others have discussed VFSA in detail and show the advantages of VFSA over SA. To speed up the convergence rate of SA, VFSA uses the Cauchy distribution function as the perturbation [24] which is able to realize a narrower search as the iterative solution approaches an optimum solution, which accelerates the convergence speed [23]. Our enhancements to the basic generic algorithm are illustrated in Table. I.

TABLE I: Very Fast Simulated Annealing Functions

Entity	**gSA**	**mSA**
Annealing temp T_k	$T_0 * exp(-\alpha(k-1)^{1/n})$	
Entropy change δ	$exp(\frac{c_i - c_{i-1}}{T_k})$	$(\frac{c_i - c_{i-1}}{T_k})^3$
Acceptance Probability ρ	1,if $p_i \geq p_u$ 1,if $\delta \leq U(0,1)$ 0,otherwise	1, if$p_i \geq p_u$&$c_i \leq c_{i-1}$ 1 ,if $\delta \leq U(0,1)$ 0 ,otherwise
Perturbation Model ζ_j	$T_k(\mu - 0.5)\left[\left(1 + \frac{1}{T_k}\right)^{\lvert 2\mu - 1\rvert} - 1\right](B_j - A_j)$	
Selecting neighboring state (s_{i+1})	$\begin{cases} \text{random\_new\_state}(), & \text{if } \rho = 1 \\ \zeta_j, & \text{otherwise} \end{cases}$	
Design Variables	Individually varied	Varied as a group

We discuss the supporting functions of gSA and mSA

in Table I using the following notations: k is the current iteration, n is the number of design variables, T_0 is the initial annealing temperature, α is the damping coefficient ($0 < \alpha < 1$), μ and U are uniform random variables between 0 and 1, $(B_j - A_j)$ is the range of j^{th} design variable ($1 \leq j \leq n$), $\vec{x_i}$ is the configuration (design variables) at i^{th} state, c_i is the configuration cost at i^{th} state, p_i is the predicted performance of configuration $\vec{x_i}$ at i^{th} state, and p_u is user given performance.

mSA uses the annealing scheme T_k and Cauchy distribution perturbation model ζ from VFSA (see Table I). For acceptance probability ρ, mSA makes a slight modification to accommodate the case where the next solution has the same performance but lower cost. If the acceptance probability for the current state is 1, a new random state is chosen (to avoid getting stuck in local minima) else a new state in the neighborhood is chosen.

V. Configuration Modeling Issue

Given the importance of addressing the configuration problem, we applied the above design (i.e., combinatorial optimization based configuration selection) to several publicly available data-sets (See Table. II). As these are published data-sets from various studies, we have no control over the data collected; experiments run, CPs, variability, etc. We explain these data-sets briefly below.

A. Cloud/Edge Storage Data-set (ES)

Edge computing [7] offers computation & storage at the very edge of the network, close to where data is produced and has recently emerged as a way to reduce latency and limit the load that is carried to higher layers of the infrastructure hierarchy. These platforms (henceforth referred to as ES) are constrained by limited resource capacity and placed between the Edge/IoT/user applications and the Cloud platform [7, 20]. ES is usually deployed at a branch office or remote location and has access to a rather limited local compute/storage and is connected to a Cloud data center over the Internet. ES essentially uses local storage as a cache for the remote Cloud storage to bridge the gap between the demand for low-latency/high-throughput local access and the reality of high-latency connection to the Cloud with unpredictable and usually low throughput [7, 20]. Resource allocation for the ES is challenging since they have to dynamically adapt to the requirements of the end-users' applications and end-devices (e.g., cars, drones), taking into account the resources' characteristics in terms of processing latency, capacity, security, location, and cost.

We address the configuration selection problem in Edge platforms using data published in Ref. [20][1]. The authors capture the performance throughout the ES server for various configuration settings (combination of CPU cores, core speed, memory, storage allocation & network capacity, etc.) for various data-transfer request sizes, i.e. different workloads such as Health monitors, MRI/CT Scans, Mammography images, etc. The observed performance of the ES denoted as ϕ, is influenced by its CPs ($\vec{x}$ in Eq. 2) and the given workload ($\vec{w}$). The CPs include computing resources (cores, CPU-speed, memory capacity, etc.), IO path (memory bandwidth, disk IO bandwidth, etc.), buffer space allocation (cache space, metadata space), etc. A full description of the ES system, various CPs influencing the behavior, workloads, etc., is given in Ref. [20]. We represent the ES configuration as a combination of required compute and storage resource - number of cores nc, core speed cs , memory capacity mc, memory bandwidth bw, and disk IO rate di. Workload can be defined by the request arrival rate ar, request size rs, and the metadata size ms. Now, in the context of an ES system being studied, we can express Eq. 2 more clearly as: $\vec{x}$= $\{nc, cs, mc, bw, di\}$ and $\vec{w}$ = $\{ar, rs, ms\}$. The research question would then be to find the suitable values for $\vec{x}$ for a given $\vec{w}$ that satisfies the given constraint (Eq. 4) at a minimum cost (Eq. 5).

B. Configuration Modeling for BitBrains Data Center (BB)

The other publicly available data-set used in this research is BitBrains[2] workload trace [25] containing the performance logs of 1,750 VMs from a distributed data center from BitBrains, which are collected over 5000 cores and 5 million CPU hours accumulated over four months. This data-set (see Table II) provides specialized interactive services and batch processing workloads in a Cloud environment for managed hosting and business computation, including leading banks, insurance companies, credit card operators, etc.

Workloads for Evaluating BB: Authors [25] conduct a comprehensive characterization of both requested and actually used resources, using data corresponding to CPU, memory, disk, and network resources. The initial configuration CP of each VM present in these traces is characterized by the attributes shown in Table II. With limited knowledge of the details of the data-set, we formulate the BB VM configuration as a combination of required compute, storage, and network resource - number of cores nc, memory capacity mc, network receive bandwidth $nwrd$, and network transmit bandwidth $nwwr$. We characterize the workload as the load on the storage disks as read request rate $dskrd$ and write request rate $dskrd$ and observed behavior (ϕ) as the CPU usage (%).

We can now represent the configuration problem as selecting the right combination of configuration values (i.e. resources $\vec{x}$ = $\{nc, mc, nwrd, nwwr\}$) for a given workload ($\vec{w}$ = $\{dskrd, dskwr\}$) to satisfy the user defined conditions (Eq. 4 and Eq. 5).

C. Configuration Modeling for Enterprise Data-set (EE)

We evaluate our work using three Cloud applications (Apache, SQLLite, and Berkeley DB) from Ref. [11, 26, 27, 29], by commonly grouping them as Enterprise Data-sets[3,4] (EE). Apache HTTP Server is a highly popular web server. Xu et al. [30] report that the Apache server has more than 550

[1] [ES] https://www.kkant.net/config_traces/CHIproject

[2] [BB] http://gwa.ewi.tudelft.nl/datasets/gwa-t-12-bitbrains (RND500)

[3] [EE] https://github.com/ai-se/Reimplement/tree/cleaned_version/Data

[4] [EE] https://goo.gl/689Dve (RawData/PopulationArchives)

TABLE II: CPs, Workload and Output of the Data-Sets.

Data-Set	**Domain**	**CPs** $\vec{x}$	**Workload Characteristics** $\vec{w}$	**Output** ϕ
ES [20]	Cloud Storage	No. of Cores, Core Speed, Memory Capacity, Memory Bandwidth, Disk IO Rate	Request Arrival Rate, Request Size, Metadata Size	Performance
BB [25]	Virtual Machines	No. of Cores, Core Speed, Memory Capacity, Network Data Rcvd., Network Data Transmit	Disk Read Throughput, Disk Write Throughput	CPU Usage (%)
Apache [11]	Web Server	Base, KeepAlive, Handle, HostnameLookups, EnableSendfile, FollowSymLinks, AccessLog, ExtendedStatus, InMemor1	N/A	Performance
SQLLite server [26]	SQL Server	SetCacheSize, StandardCacheSize, LowerCacheSize, HigherCacheSize, LockingMode, ExclusiveLock, NormalLockingMode, PageSize, StandardPageSize, LowerPageSize, HigherPageSize, HighestPageSize$\cdots$ $\cdots$	N/A	Performance
Berkeley DB C [27]	Embedded database	havecrypto, havehash, havereplicatio0, haveverif1, havesequence, havestatistics, diagnostic, pagesize, ps1k, ps4k, ps8k,ps16k, ps32k, cachesize, cs32mb, cs16mb,cs64mb, cs512mb	N/A	Performance
MIT [28]	HPC Env.	CPU Frequency, Resident Memory Size, Virtual Memory Size	Amount of Data ReadWrite (MB)	CPU Util. (%)

parameters and many of these parameters have dependencies and correlations, which further complicates the configuration problem we address here. Reference [11, 29] narrows the CPs down to only nine CPs as given in Table. II. Berkeley DB (C) [27] is an embedded key-value-based database library that provides scalable high performance database management services to applications. SQLLite [26] is the most popular lightweight relational database management system used by several browsers and operating systems as an embedded database. In producing the data-set, the authors [29] stress the application to maximum workload and observe performance data for various configurations. Authors [27, 26] have used 18 CPs for BDBC and 29 CPs for SQLLite data-sets. This expands the configuration space Ω to a larger degree, and the results show the efficacy of the proposed solution in such a large configuration space.

D. Configuration Modeling for MIT Cloud Data Set (MIT)

MIT published a rich data-set[5] from their Supercloud petascale cluster [28] running various HPC workloads. This huge (2TB data) data set contains time-series data of the scheduler, file system, compute nodes, CPU, GPU, and sensor data from physical monitoring of the facility housing the cluster itself. We used the 2^{nd} partition data-set with 480 CPU nodes (2x24-core Intel Xeon Platinum 8260 processor), each with 192GB of RAM and a Lustre high-performance parallel file system running on a 3-petabyte Cray L300 parallel storage array (See Table.IV Slurm Time Series Data). The data attributes in this work comprises: CPU frequency, residual memory, virtual memory size, CPU utilization, disk IO, etc. We now propose the resource allocation (configuration) question as: "finding the design variables (node number, VM-Size, CPU Frequency, RSS Memory Size) for HPC workload (given as ReadMB & WriteMB) for a required CPU Utilization (Constraint)." We demonstrate the efficacy of the proposed solution for large data-sets.

[5][MIT] https://dcc.mit.edu

We refer readers to the detailed literature at Ref. [11, 26, 27, 28, 29] for a full description of the data-set(s). With our problem at hand, the problem (Eq. 2 and Eq. 3) reduces to finding the best configuration ($\vec{x}$) for a given workload ($\vec{w}$) and a user given performance (p_u) at minimum possible cost.

VI. Evaluation

A. Metrics for Evaluation

Using the above data-sets, we evaluate the efficacy of our solution in finding a satisfying solution with two key metrics: (**M1**) the number of calls to the performance function (a.k.a oracle), and (**M2**) the minimum cost of the selected configuration (i.e Eq. 5). Metric M1 is important as it relates to how fast the algorithm can *find an optimal set of parameters* from the vast configuration space Ω. Metric M2 may refer to the monetary cost ($$) of the selected physical configuration, resource consumption of the selected configuration in a virtualized environment, or some other attribute (e.g., energy consumption, provisioning difficulty, etc.). Naturally, metric M2 is generally much more important than M1 (the computation time), but there are two situations that make M1 very important: (a) frequent changes in configuration, which is quite common in current Clouds, selection/change happens frequently, and (b) models (oracles) with long running times.

We executed 100s of test cases across all the data-sets, each test case T_i refers to a unique combination of $\vec{w_j}$ and ϕ_k in the data-set. We discuss the evaluation results using M1 and M2 metrics w.r.t the three approaches discussed above, i.e. (a) Generic Simulated Annealing (gSA), (b) Modified Simulated Annealing (mSA), and (c) Modified Simulated Annealing with Tunneling (mSA(T)).

B. Performance Oracle

The efficiency of ML algorithms depends on a variety of factors, including the input attributes and hyper-parameters (e.g., regularization parameters, learning rate, etc.), and it is generally not possible to characterize which algorithm works the best in a given situation [31]. Therefore, we tried several

models and ultimately settled on Logistic Regression, as it consistently performed well and beat others in most workloads. An extensive analysis (e.g. k-fold validation) of the model ensured that it does not suffer from under-fit or over-fit.

C. Using Domain Knowledge to Group Attributes

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, but the settings across groups can be done independently. In theory, such grouping can be done purely in a data-driven manner (e.g., by using clustering techniques), but this is likely to result in spurious groups unless we have a large amount of data covering full ranges of various CPs and the clustering algorithm does not result in anomalies. The value of domain knowledge is to do a suitable grouping either entirely manually or by coercing the clustering algorithm to prefer certain groupings over others.

TABLE III: Grouping Design Variables for Various Data-Sets

Data-Set	Group	Design Attribute Pairs
ES	Group G1	Number of Cores, Memory capacity
	Group G2	Core speed, Memory bandwidth
	Independently varied	Data cache, Disk IO rate
BB	Group G1	Number of Cores, Memory capacity
	Independently varied	CPU capacity, Network data transmit, Network data received
Apache BDBC SQLLite	Independently varied	All CPs
MIT	Group G1	Virtual memory used by process, Resident memory footprint set size
	Independently varied	CPU clock frequency

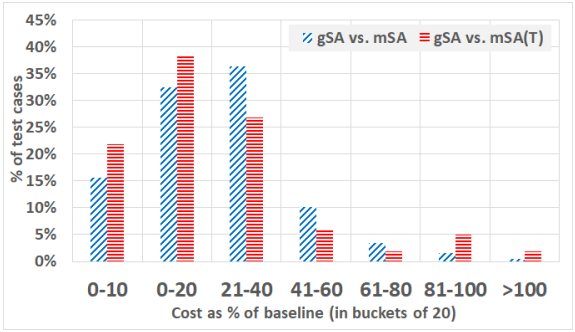


Fig. 3: Histogram of Cost for ES

In any configuration context, we are likely to have several generic and usage-specific insights into the system. For example, in computing infrastructure, a faster CPU must be paired with a faster DRAM; else, the CPU will simply stall waiting for the memory. A faster disk is also important, but much less so, since the IOs involve a context switch whereas memory access does not. Similarly, more CPU cores doing independent work will likely need more memory, and for workloads involving remote IO, both network and IO speeds

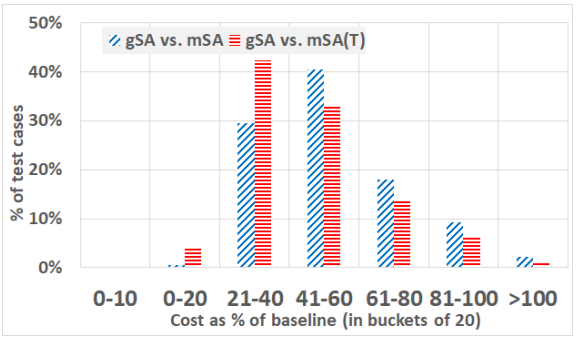


Fig. 4: Histogram of Cost for MIT

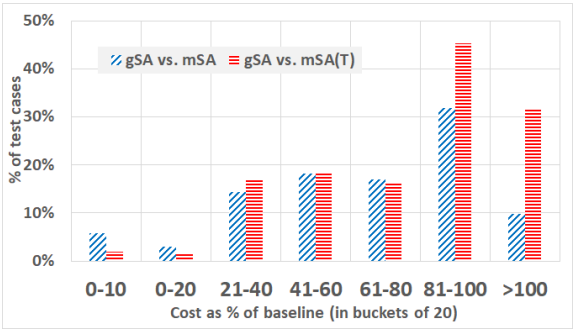


Fig. 5: Histogram of Cost for BB

must increase in tandem. *Grouping* of CPs 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. As shown in Table III, ES, BB, and MIT data-sets are grouped according to the interdependence between design variables. Since the

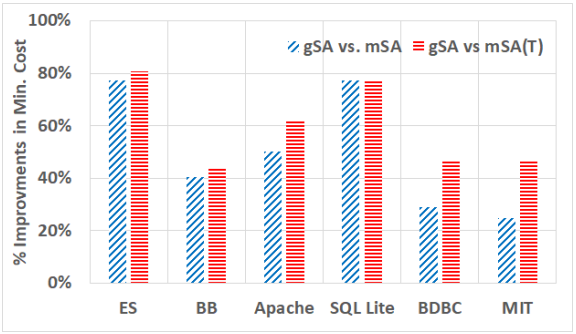


Fig. 6: % Improvement in Solution Cost

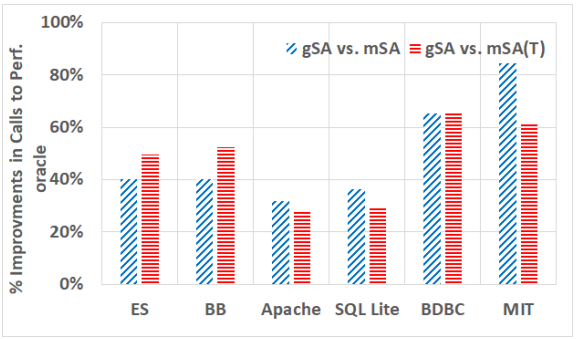


Fig. 7: % Improvement in #calls to Oracle

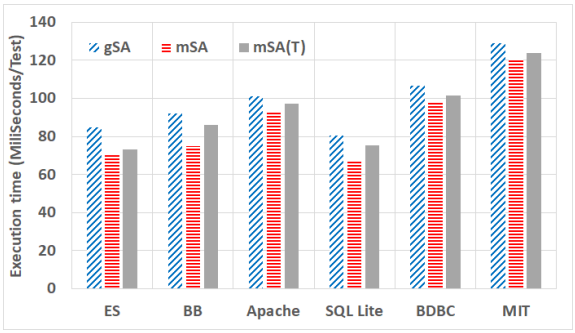


Fig. 8: Execution Time for Different Data-sets

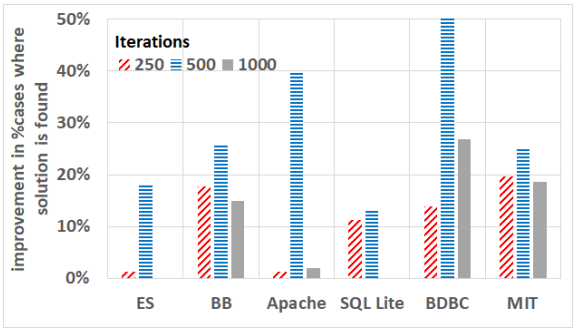


Fig. 9: % Improvement in % cases that provide a solution (gSA vs. mSA(T))

trace description doesn't give much information about the configuration or workload, we have grouped the other data-sets independently (Apache, BDBC, and SQLLite).

D. Efficacy of the algorithms (gSA, mSA and mSA(T))

We use the gSA algorithm as the baseline, as it presents the naive (or *uninformed*) stochastic method of searching a wide configuration space for a set of suitable CPs.

In our evaluation for metric M2, the minimum cost of the solutions with mSA(T) was much less (hence better/desired) than the solution found by mSA or gSA (shown in Fig 6).

Detailed results are shown in Fig. 3, 4, 5 for a few data-sets (ES, MIT, and BB respectively), where we plot the improvement in the solution cost provided by our two algorithms (mSA and mSA(T)) over the baseline uninformed algorithm gSA. Here the X-axis refers to the cost of mSA and mSA(T) divided by the solution cost of gSA, and expressed as a percentage. That is, the buckets for $\leq 100\%$ represent an improvement over gSA, and $> 100\%$ represent a degradation. The y-axis is the count of test cases whose solution cost falls into that bucket – again, normalized so that it all adds up to 100% of the test cases. These charts are produced by considering 100s of test cases and thus represent an extensive exploration of the configuration space.

Fig. 3 in ES results shows that in the 1st group (0-10%), mSA achieved the solution with $\leq 10\%$ of the baseline cost for 15% of test cases; and mSA(T) further improved this to 22% of test cases. The maximum improvement observed was in a few cases where the cost of mSA(T) was only 2% of the cost provided by gSA! Fig. 4 in MIT results also shows that in the 2nd group, mSA achieved the solution with $\leq 20\%$ of the baseline cost for 0% of test cases, but mSA(T) improved on that to 4% of test cases.

Similarly in Fig. 5 in BB results, in the 6th group (81 to 100%), mSA cost was about 61-80% of the cost of the baseline in about 17% test-cases; and mSA(T) improved this in about 22% of test-cases. The final group (> 100) in all the sub-graphs show cases where gSA cost was better than mSA or mSA(T); however, these cases were small in the case of an ES and MIT. With BB, the evaluation showed that mSA(T) failed to get minimum cost in about 30% of the cases (compared to gSA). Note that because of the inherent randomness in the way the states are explored, no stochastic algorithm can provide a universally better result in all cases.

Fig. 6 shows that both of our algorithms (mSA and mSA(T)) provide a lower-cost solution in comparison with gSA by $\sim$50% and $\sim$60%, respectively. The solution cost for the ES, BB, Apache, SQLLite, BDBC, and MIT data-sets is improved in range 25% – 77% by mSA, and in range 47% – 81% by mSA(T).

Fig. 7 shows the improvement in the number of calls to the performance Oracle. It is again seen that mSA/mSA(T) consult the performance Oracle significantly fewer times, which can be significant if running the performance model becomes expensive.

Fig. 8 shows the run-time of the three algorithms (gSA, mSA, and mSA(T)) for various workloads. (This is the time elapsed until the solution stops improving, but limited to cases where the solution is indeed found.) The results show that mSA and mSA(T) beat gSA even here, although by rather small amounts of 12% and 6% respectively. The more significant observation, however, is that the run-times are fairly small in all cases, which is essential for frequent configuration changes. Thus we expect that even with much more complex situations, the mechanism would be able to determine the suitable configuration rather quickly thereby satisfying the needs of DevOps related auto-configuration.

Finally, Fig. 9 compares the ability to find a solution within certain number of iterations. For this, we choose 250, 500, and 1000 as the limits on #iterations. The key reason to consider 3 different values is to ascertain that the results are not an artifact for a given iteration count. For 1000 iterations, gSA is successful in only 75% of the cases, but mSA/mSA(T) are successful in 98% and 97% of the cases respectively. Fewer iterations show an even better improvement of mSA/mSA(T) over gSA, although the absolute success rate will surely decrease with #iterations. All in all, unlike gSA, mSA/mSA(T) succeed in finding the solution in almost all cases, find solutions of significantly lower cost (see Fig 6), and even run somewhat faster (see Fig. 8).

VII. Discussions

...Add here...

VIII. Conclusions

In this paper, we presented an efficient methodology to recommend optimal configurations for the emerging DevOps

environments with implicit performance function and cost constraints. We propose an improved metaheuristics-based approach enhanced by both the domain knowledge and smart tunneling techniques. We applied the technique to several real-world traces from various publicly available Cloud environments where configuration information was included in the data-set. The results show that the proposed mechanism outperforms a standard naive uninformed approach by 44-81% (depending on the domain and data-set) in terms of the cost of the solution, converges faster by 28-65%, and still run somewhat faster.

The key reasons for such performance gains include the following. First, we compute entropy as a quadratic function to give us a wider choice of acceptance which is able to better avoid getting stuck at local minima, Second, we intelligently group the attributes which avoids exploring unnecessary portions of the search space. Third, by adding the tunneling logic, mSA(T) avoids *jumping out of local minima too quickly*; instead, it explores a few additional states *closer to current local minima*. We also show that the proposed approach can determine desired configuration very quickly, which is essential in highly dynamic DevOps and microservices environments.

REFERENCES

[1] M. Shahin, "Architecting for devops and continuous deployment," in *Proc. of ASWE*, ACM,2015, pp.147–148.

[2] S. Newman, *Building microservices – Designing Fine Grained Systems,2nd Edition*, "O'Reilly Media", 2021.

[3] J. Humble and D. Farley, *Continuous delivery: reliable software releases through build, test, and deployment automation*, Pearson Education, 2010.

[4] F. Connolly, "Production operations – the last mile of a devops strategy," *LMC Report*, Mar 2014.

[5] W. Cappelli, "Causal analysis makes availability and performance data actionable," *Gartner Report*, Oct 2015.

[6] C. Tang, T. Kooburat, P. Venkatachalam, A. Chander, Z. Wen, A. Narayanan, P. Dowell, and R. Karl, "Holistic Configuration Management at Facebook," in *Proc. of SOSP*, ACM, 2015, pp. 328–343.

[7] P. Soumplis, P. Kokkinos, A. Kretsis et.al., "Resource Allocation Challenges in the Cloud and Edge Continuum," in *Advances in Computing, Informatics, Networking and Cybersecurity*, Springer, 2022, pp. 443–464.

[8] S. Singh and et al, "Cloud resource provisioning: survey, status and future research directions,"*Knowledge and Information Systems*, vol. 49, no. 3, pp.1005–1069, 2016.

[9] S. S. Gill and Et al., "Chopper: an intelligent qos-aware autonomic resource management approach for cloud computing," *Cluster Computing*, vol. 21, 2018.

[10] M. Wang and Et al., "Storage device performance prediction with CART models," in *MASCOTS*, 2004.

[11] N. Siegmund, A. Grebhahn, S. Apel, and C. Kastner, "Performance-influence models for highly configurable systems," in *Proc. of ESEC/FSE*, 2015.

[12] M. Satyanarayanan, "The emergence of edge computing," *Computer*, vol. 50, no. 1, pp. 30–39, 2017.

[13] Y. Sfakianakis, M. Marazakis, and A. Bilas, "Skynet: Performance-driven Resource Management for Dynamic Workloads," in *IEEE CLOUD*, 2021.

[14] S. Sondur, K. Kant, S. Vucetic, and B. Byers, "Storage on the edge: Evaluating cloud backed edge storage in cyberphysical systems," in *IEEE Intl Conf. on MASS*, 2019, pp. 362–370.

[15] K. Hussain, M. N. M. Salleh, S. Cheng, and Y. Shi, "Metaheuristic research: a comprehensive survey," *Artificial Intelligence Review*, 2019.

[16] A. V. Levy and A. Montalvo, "The tunneling algorithm for the global minimization of functions," *SIAM J. Sci. and Stat. Comput.*, vol. 6, no. 1, p. 15–29, 1985.

[17] F. Schoen, "Stochastic techniques for global optimization: A survey of recent advances," *J. of Global Optimization*, vol. 1, no. 3, pp. 207–228, 1991.

[18] Z. Li and Y. Yang, "A modified tunnelling algorithm for global minimization with box constrained," in 5th Intl. conf on Computational Sciences & Optimization, 2012, pp. 423–427.

[19] O. Alipourfard, H. H. Liu, J. Chen et. al., "Cherrypick: Adaptively unearthing the best cloud configurations for big data analytics,", Proc. of NSDI, 2017, pp. 469–482.

[20] S. Sondur and K. Kant, "Towards automated configuration of cloud storage gateways: A data driven approach," in Intl. conf on Cloud Computing, Springer, 2019, pp. 192–207.

[21] M. Barrette and Et al., "Statistical multi-comparison of evolutionary algorithms," *Bioinspired Optimizaiton Methods and their Applications*, 2008.

[22] K. P. Ferentinos, K. G. Arvanitis, and N. Sigrimis, "Heuristic optimization methods for motion planning of autonomous agricultural vehicles," *J. Global Optimization*, vol. 23, no. 2, pp. 155–170, 2002.

[23] Y. Xu, Q. Ye, and G. Meng, "Hybrid phase retrieval algorithm based on modified very fast simulated annealing," Intl. J. of Microwave & Wireless Technologies, vol. 10, no. 9, pp. 1072–1080, 2018.

[24] C.-Y. Lee, "Fast simulated annealing with a multivariate cauchy distribution and the configuration's initial temperature," *J. Korean Physical Society*, 2015.

[25] A. Iosup and Et al., "The grid workloads archive," *Future Generation Computer Systems*, vol. 24, 2008.

[26] V. Nair, Z. Yu, T. Menzies, N. Siegmund, and S. Apel, "Finding faster configurations using flash," *IEEE Transactions on Software Engineering*, 2018.

[27] V. Nair, T. Menzies, N. Siegmund, and S. Apel, "Faster discovery of faster system configurations with spectral learning," *Automated Software Engineering*, 2018.

[28] S. Samsi, , and Et al., "The MIT Supercloud Dataset," in *IEEE HPEC*, 2021.

[29] V. Nair, T. Menzies, N. Siegmund, and Apel, "Using bad learners to find good configurations," in *Proc. of Joint Meeting on Foundations of Software Eng.*, 2017.

[30] T. Xu and Et al., "Hey, you have given me too many knobs!" in *Proc. of Joint Meeting on Foundations of Software Eng.*, 2015.
[31] M. S. Sorower, "A literature survey on algorithms for multi-label learning," *OSU, Corvallis*, vol. 18, 2010.