Search-based Methods for Multi-Cloud Configuration

Malgorzata Lazuka*,†, Thomas Parnell*, Andreea Anghel*, Haralampos Pozidis*
*IBM Research Europe, Zürich, Switzerland
†ETH Zürich, Switzerland
Email: {mal, tpa, aan, hap}@zurich.ibm.com

Abstract—Multi-cloud computing has become increasingly popular with enterprises looking to avoid vendor lock-in. While most cloud providers offer similar functionality, they may differ significantly in terms of performance and/or cost. A customer looking to benefit from such differences will naturally want to solve the multi-cloud configuration problem: given a workload, which cloud provider should be chosen and how should its nodes be configured in order to minimize runtime or cost? In this work, we consider possible solutions to this multi-cloud optimization problem. We develop and evaluate possible adaptations of state-of-the-art cloud configuration solutions to the multi-cloud domain. Furthermore, we identify an analogy between multi-cloud configuration and the selection-configuration problems that are commonly studied in the automated machine learning (AutoML) field. Inspired by this connection, we utilize popular optimizers from AutoML to solve multi-cloud configuration. Finally, we propose a new algorithm for solving multi-cloud configuration, CloudBandit. It treats the outer problem of cloud provider selection as a best-arm identification problem, in which each arm pull corresponds to running an arbitrary black-box optimizer on the inner problem of node configuration. Our extensive experiments indicate that (a) many state-of-the-art cloud configuration solutions can be adapted to multi-cloud, with best results obtained for adaptations which utilize the hierarchical structure of the multi-cloud configuration domain, (b) hierarchical methods from AutoML can be used for the multi-cloud configuration task and can outperform state-of-the-art cloud configuration solutions and (c) CloudBandit achieves competitive or lower regret relative to other tested algorithms, whilst also identifying configurations that have 65% lower median cost and 20% lower median runtime in production, compared to choosing a random provider and configuration.

Index Terms—multi-cloud, optimization, search-based methods, cloud configuration, machine learning

I. INTRODUCTION

We live in the era of cloud computing, in which cloud providers compete to offer computing resources (servers, storage, managed software) to businesses and consumers via the internet. Cloud is a high-growth segment: according to [11], worldwide end-user spending on public cloud services is forecast to grow 16.2% in 2022 to total $474 billion. For enterprises, there are many benefits to using cloud services, e.g., increased flexibility, reduced costs, and rapid scaling. Coupled to the rise of cloud is the growing popularity of containers and container orchestration platforms. Containers can be deployed quickly from one computing environment to another, making them well-suited to the world of cloud computing. Many of today’s workloads, such as machine learning (ML) training, are distributed in nature and require the deployment and management of multiple containers, all working together in parallel. Container orchestration platforms such as Kubernetes [4] enable running such systems resiliently, providing features like auto-scaling and failure management.

Most cloud providers offer a Kubernetes service that a user can utilize to specify the desired number of nodes and the nodes’ configuration (how many CPUs, memory, etc.). Given a Kubernetes cluster, one can run a complex distributed workload, whilst remaining agnostic to which cloud provider is providing the hardware resources. This level of abstraction presents the user with a new optimization opportunity, which we will refer to as the multi-cloud configuration problem. Namely, given a workload, which cloud provider should be selected, and how should the nodes in the cluster be configured, in order to minimize runtime or cost?

In practice, this optimization problem is of particular interest to users with complex, distributed workloads that must be run repeatedly. Consider a business with an ML-based recommendation engine that must be trained, in a distributed fashion, once every hour. By solving the multi-cloud configuration problem for such a recurring workload, the business will be able to exploit price/performance differences that may exist between cloud providers and save time or money with every run of the workload. However, any optimization algorithm will involve making a certain number of evaluations (e.g., running the workload on a particular cloud provider, with a particular node configuration), and these evaluations themselves incur a certain additional cost. In order for the savings offered by the optimized deployment to outweigh the expense of extra evaluations, effective algorithms with fast convergence are of particular interest in this context, and are the primary focus of this paper.

Existing solutions for solving the cloud configuration problem can be broadly separated into two categories: predictive methods and search-based methods. We provide a detailed overview of their pros and cons in Section II. In the first part of this work, we evaluate the performance of multiple such state-of-the-art methods, most of them adapted for the first time to the multi-cloud setting. In particular, we are interested in understanding 1) whether search-based methods are superior to predictive methods, and 2) how search-based methods developed in the single-cloud context should be adapted to the multi-cloud setting. Furthermore, motivated by an interesting
connection between the multi-cloud configuration problem and
the selection/configuration problems studied in the automated
machine learning (AutoML) field, we address yet another novel
research question, namely how AutoML solutions perform
when applied to the multi-cloud setting. Finally, we propose
a new search-based algorithm that outperforms the existing
methods in terms of both convergence speed and cost savings.
In summary, the contributions of our work are 4-fold:

• As a result of a detailed review of the state-of-the-art, we
have identified a set of promising predictive/search-based
methods that we have implemented and adapted (when
necessary) to multi-cloud configuration.

• We have applied (to our knowledge, for the first time) three
popular AutoML methods to the multi-cloud configuration
problem.

• Inspired by existing multi-armed bandit methods, we
propose CloudBandit – the first (to our knowledge) search-
based solution to multi-cloud configuration, which is faster
and more efficient than other analyzed methods.

• We perform an extensive performance evaluation of
CloudBandit, as well as of various predictive/search-based
methods and AutoML solutions using a one-of-a-kind
data set with real cloud data collected from from 60 multi-
cloud configuration tasks, across 3 public cloud providers.

II. STATE-OF-THE-ART

In this section, we will review the state-of-the-art in the field
of predictive and search-based cloud configuration methods.
We will also discuss methods that try to solve an analogous
optimization problem in the cloud scheduling literature.

A. Predictive Methods

Given a target workload, predictive methods use a statistical
model to predict how the workload will perform on a set
of cloud configurations. Using these predictions, the cloud
configurations can be ranked and the optimal configuration
estimated. Methods vary in terms of what type of model is
used, as well as with what kind of data the model is trained.

In [31], the authors considered the problem of finding the
optimal type and number of nodes for distributed analytics
workloads using a single cloud provider. The proposed method,
Ernest, trains a linear performance model for each node type,
using data collected from online evaluations of the target
workload. To minimize the cost of the online evaluations, the
workload is evaluated using a small subsample of the workload
input data. The trained model is then used to estimate the
performance of the workload using the full input data.

Rather than rely on online evaluations of the workloads,
other methods opt to train a predictive model using offline
data collected from benchmarking or profiling of historical
workloads. For instance, [25] proposed an offline approach
targeting the configuration of distributed scientific computing
workloads. To generate the offline dataset, several training
workloads were evaluated on a set of cloud configurations.
Using this data, a random forest (RF) model was trained
to predict the cloud performance using features related to
both the configuration and workload characteristics. In the
online phase, the same set of features are generated for the
target workload and the RF estimates are used to identify the
best configuration. RF models were also employed in [33]
to predict the performance of single-node workloads, such as
video compression and cloud data stores. The proposed method,
PARIS, trains an RF model using both historical data (collected
offline across two cloud providers) and online evaluations of the
target workloads on a small set of reference configurations. The
features used to train the model comprise configuration-specific
values as well as a fingerprint obtained by profiling the target
workload on the reference configurations. In [21] a similar
approach, combining offline data with online evaluations of the
target workload, was proposed to configure single-node
analytics workloads. Rather than using an RF model, latent
factor collaborative filtering was used to rank a set of cloud
configurations from a single cloud provider.

While predictive methods usually incur a relatively small
online search cost, they suffer from two main drawbacks.
Firstly, despite large amounts of offline data available, any
predictive model is inherently noisy – for example, PARIS [33]
achieves the relative RMSE between 15% and 65%. As a result,
inaccurate predictions can lead to a sub-optimal configuration
being chosen. Secondly, predictive methods require a significant
amount of offline data, which may be unavailable or time
consuming and expensive to collect.

B. Search-based Methods

An orthogonal research direction has explored search-based
methods that iteratively evaluate the target workload using
a sequence of different configurations. The length of this
sequence (i.e., the total number of configurations evaluated)
is referred to as the search budget. Similarly to predictive
approaches, search-based methods can use both offline data and
online evaluations. However, search-based methods additionally
use feedback from all previously evaluated configurations in
order to better select which configurations should be evaluated
next. Thanks to the additional feedback, search-based methods
are more likely to find the true optimal configuration and
typically do not require the offline phase. However, they may
incur a relatively large online search runtime and cost. Thus,
such methods are most appropriate for frequently recurring
workloads, such as daily log parsing or re-training of ML
models, for which the cost of the configuration search (which
should be run relatively infrequently) can be effectively
amortized. Recurring workloads are an important use-case:
in one production cluster, recurring workloads were found to
account for over 40% of jobs [9].

The simplest search-based method, and an important baseline,
is exhaustive search (i.e., trying all possible configurations
across all cloud providers). While this approach is guaranteed
to find the optimal configuration, it can be extremely time
consuming and expensive. Other simple baselines such as
random search (RS) and coordinate descent (CD), may be
less expensive than exhaustive search, but may fail to find the
optimal configuration for a small search budget. More efficient
search-based methods are desirable so that their cost can be more effectively amortized, allowing them to be run more frequently to adapt to changes in the workload characteristics.

In [1] the authors pose the cloud configuration problem as an optimization problem in which one would like minimize the cost of running a workload subject to a runtime constraint. To solve this, they propose an online system, CherryPick, that uses Bayesian Optimization (BO) with a Gaussian process (GP) surrogate, a Matern5/2 kernel and an Expected Improvement (EI) acquisition function that has been modified to account for the runtime constraint. Experiments are performed for 5 different distributed analytics workloads for 66 cloud configurations from a single cloud provider. It was demonstrated that CherryPick has a high probability of finding the optimal configuration, with a lower online search cost and runtime relative to random search, coordinate descent and Ernest.

BO applied to cloud configuration was further investigated in [3]. The authors frame cloud configuration as an unconstrained optimization problem in which the goal is to minimize either the runtime or cost of executing a workload in the cloud. Different BO surrogate functions were considered in addition to GP: RF, gradient-boosted regression trees (GBRT) and extra-trees (ET). The authors perform experiments for 12 distributed analytics and ML workloads across a maximum of 140 cloud configurations from a single provider. The various BO flavours are compared with other search-based solutions, such as stochastic hill climbing, simulated annealing and Tree-structured Parzen Estimator. Their conclusions were threefold: (a) in general, BO outperforms the other methods considered, (b) BO with GP and lower confidence bound (LCB) is the best choice when optimizing for cost, and (c) BO with GBRT or RF and probability of improvement (PI) is the best choice when optimizing for runtime.

Further improvements to BO-based methods were proposed in [14]. The authors propose augmenting BO with low-level information captured from the workloads during execution, such as CPU utilization, or working memory size. Additionally, the authors propose using the ET surrogate together with a different acquisition function: prediction delta. Experimental results were presented for 107 workloads across 88 different configurations from a single cloud provider. For 43% of workloads, the augmented form of BO was found to outperform naive GP-based BO, in terms of performance as well as search cost.

A number of works have also explored search-based methods other than BO. In [16] an approach was proposed, Scout, that leverages offline data to guide a online search algorithm based on pairwise-modelling. Scout was evaluated on 3 distributed big data analytics workloads across 69 configurations from a single cloud provider and was shown to outperform CherryPick, random search and coordinate descent in terms of execution runtime, deployment cost and search cost. The dataset from [16] was made publicly available, and was used in [15] to evaluate a search-based method, Micky, based on multi-armed bandits. Unlike all of the works mentioned so far, Micky tries to solve a different problem: finding the best cloud configuration for a group of workloads. It is shown that when the size of this group is large, Micky can find a satisfactory solution with a lower search cost than CherryPick.

To the best of our knowledge, there exist no prior works using search-based methods in the multi-cloud setting. A summary comparison of all aforementioned single-cloud search-based algorithms and predictive methods is provided in Table I. We compare them in terms of algorithms used, whether they involve offline or online evaluations, whether they leverage low-level metrics captured from the workload, whether multi-cloud environments were considered and finally, what baselines were used in their experiments.

### C. Related Problems

A number of works have studied a similar optimization problem in the context of cloud scheduling and other fields, both for single-cloud [29] and multi-cloud environments [30]. In this line of research, a workload is typically modelled as a directed acyclic graph, in which the nodes represent individual tasks, and the edges represent dependencies between tasks. Each task can be assigned to a different cloud, and assigned a different configuration, and the goal is to minimize the makespan of the workload (i.e., end-to-end runtime), possibly under various constraints. Unlike the predictive and search-based methods reviewed above, which evaluate the performance of workloads via real cloud measurements, these methods assume that the runtime of each task on a given configuration is either known a-priori or is given by some simple theoretical model based on the compute capability of the node in GFLOPS [8]. This assumption gives rise to a variety of different solutions: graph-based heuristics [8, 29, 34], genetic algorithms [24, 28], integer linear programming [30], and multiple criteria decision making [27]. To the best of our knowledge, none of these

| Paper | Type | Algorithms | Offline | Online | Low-level Info. | Multi-cloud | Baselines |
|-------|------|------------|---------|--------|-----------------|-------------|-----------|
| [31]  | Predictive | Linear Regression (Ernest) | ✗ | ✓ | ✓ | ✗ | n/a |
| [25]  | Predictive | Random Forest | ✓ | ✗ | ✗ | ✗ | [33] |
| [33]  | Predictive | Random Forest (PARIS) | ✓ | ✓ | ✓ | ✗ | [33] |
| [21]  | Predictive | Collaborative Filtering (Selecta) | ✓ | ✓ | ✗ | ✓ | [33] |
| [1]   | Search | Bayesian Opt. (CherryPick) | ✗ | ✓ | ✗ | ✗ | RS, CD, [31] |
| [3]   | Search | Bayesian Opt., SHC, SA, TPE | ✓ | ✓ | ✗ | ✗ | [1] |
| [14]  | Search | Augmented Bayesian Opt. (Arrow) | ✓ | ✓ | ✓ | ✗ | [1] |
| [16]  | Search | Pairwise Modelling (Scout) | ✓ | ✓ | ✗ | ✓ | RS, CD, [1] |
| [15]  | Search | Multi-armed Bandits (Micky) | ✓ | ✓ | ✗ | ✗ | [1] |
| Ours  | Search | Rising Bandits, Hyperopt, SMAC, CloudBandit | ✗ | ✓ | ✓ | ✓ | RS, [31], [33], [1], [3] |

TABLE I: State-of-the-Art Summary
methods have been evaluated in a real cloud environment, but on simulation data.

III. SEARCH-BASED MULTI-CLOUD CONFIGURATION

A. Problem Statement

The runtime/cost of a workload in the cloud depends on many factors, such as the number of nodes used, the vCPU count, the amount of memory allocated to each node, the manufacturer/generation of the backend CPU, the network on which the nodes communicate and the region in which the nodes are deployed. While some of these parameters (e.g., vCPU count and amount of memory) are, on the surface, common across different cloud providers, most providers do not give the option to set these parameters freely. Instead, one must select from a set of available VM types, parameterized into categories like family, type or size, that define which CPU will be used, how many vCPUs and how much memory will be assigned to the node, as well as what network interfaces are used. These categories are significantly different across cloud providers, making it difficult to construct a multi-cloud optimization problem over parameters that are common across all cloud providers (e.g., vCPU count) without introducing complex constraints.

Instead, it is much easier to view the multi-cloud configuration problem as an optimization over a hierarchical domain, where the domain for each cloud provider comprises a unique set of categorical parameters. Let $K = \{1, 2, \ldots, K\}$ denote the set of available cloud providers, and let $X^{(k)} = C_{1}^{(k)} \times \cdots \times C_{s_{k}}^{(k)}$ denote the set of parameter configurations offered by the $k$-th cloud provider, where $C_{i}^{(k)}, i = 1, \ldots, s_{k}$ denotes the set of values of the $i$-th configuration parameter of $k$-th cloud provider and $s_{k}$ denotes the number of configuration parameters available for $k$-th cloud provider. Additionally, let $N \subset \mathbb{Z}_{\geq 1}$ denote the set of valid sizes of the Kubernetes cluster in terms of number of nodes. Then, the multi-cloud configuration problem can be posed as the following hierarchical optimization problem:

$$
\min_{k \in K} \left( \min_{n \in N, x \in X^{(k)}} f_{k}(n, x) \right),
$$

where objective function $f_{k}(n, x) \in \mathbb{R}^{+}$ returns the execution runtime (or cost) of the target workload on a Kubernetes cluster deployed on the $k$-th cloud provider, comprising $n$ nodes of type $x$. As runtime and cost of running the workload in cloud with a specific configuration is not deterministic, function $f_{k}$ can return the target value obtained in a single measurement or use any chosen metric based on multiple measurements, such as the mean or the 90th percentile.

In this work, we are interested in developing search-based methods that try to solve problem (1). The search for the best configuration will be constrained by a search budget $B \in \mathbb{Z}_{\geq 1}$, specified in terms of the maximum number of evaluations of the objective functions $f_{k}$.

B. Adapting State-of-the-Art to Multi-Cloud

As discussed in Section II-B, the majority of related work was focused on applying black-box optimization methods, in particular BO [1], [3], [14], to the single-cloud configuration problem. It is thus natural to ask how such methods might be adapted to solve the multi-cloud configuration problem defined in (1). Below we consider two such approaches, which are illustrated visually in Figure 1.

1) Flattening the domain: The main challenge when applying off-the-shelf BO to (1) is that the domain of the inner minimization depends on the choice of cloud provider in the outer minimization. One straightforward approach is to effectively flatten the optimization domain by defining a new objective function that includes the cloud provider selection as an argument:

$$
\min_{k \in K, n \in N} \left( \min_{x \in X^{(k)}} f(k, n, x) \right).
$$

This approach is illustrated visually in Figure 1a. While optimization problem (3) can be readily solved using standard BO software packages, it has two key downsides. Firstly, the dimensionality of the flattened domain $K \times N \times X^{(1)} \times \cdots \times X^{(K)}$ becomes large as the number of cloud providers increases. It is generally understood that BO does not perform well in this limit [32], often being beaten by simple baselines like random search. Secondly, for a given value of $k$, the choice of $x^{(k')}$ for $k' \neq k$ will have no effect on the objective function (2) and the optimizer may well waste valuable evaluations trying to understand these non-existent relationships.

2) Independent optimizers: An alternative, perhaps even simpler, way to adapt off-the-shelf BO to (1) is simply to use $K$ independent optimizers, one for each cloud provider. Thus, the $k$-th optimizer will solve the inner optimization problem:

$$
\min_{n \in N, x \in X^{(k)}} f_{k}(n, x).
$$

This approach is illustrated visually in Figure 1b. In order to make a fair comparison against the flattened approach, it is necessary that the search budget be split equally amongst each of the $K$ optimizers. Specifically, if the single optimizer over the flattened domain is given budget $B$, then each of the $K$ independent optimizers should be given budget $B/K$. The approach has the advantage that it acknowledges the hierarchy that is present in the problem, albeit in the most naive way, and, as a result, each optimizer is operating over a significantly smaller domain. However, it has the disadvantage that all cloud providers will be assigned equal budget, and thus during the search it cannot learn to discard cloud providers that do not look promising. Additionally, the parameter corresponding to the number of nodes is optimized independently for each cloud provider, when it is in fact a parameter that is common to all.

C. Applying Methods from AutoML

In [3] the authors make an analogy between cloud configuration and hyperparameter tuning in ML. They point out important differences between these two problems. Firstly,
cloud configurations consist mostly of integer or categorical variables, whereas hyperparameters in ML are often continuous. Secondly, cloud configurations are typically complex and expensive to evaluate (involving spinning up a cluster of nodes), whereas hyperparameter configurations may be easily evaluated on a single node. Despite these differences, the authors found that methods that work well for hyperparameter tuning (such as BO with RF or GBRT) are also highly effective for solving the cloud configuration problem.

The key insight of this part of our work is to go one step further and notice that, due to its hierarchical nature, the selection-configuration problem defined in (1) bears a striking resemblance to the Combined Algorithm Selection and Hyperparameter Optimization (CASH) problem at the heart of AutoML. Namely, in AutoML one must select which ML model to use (e.g., neural networks vs. gradient-boosted decision trees), and decide how to configure each model (e.g., how each hyperparameter should be chosen). Having established this analogy, it makes sense to ask the question: can existing methods from AutoML be applied to the multi-cloud configuration problem defined in (1)?

HyperOpt and SMAC are both examples of sequential model-based optimization (SMBO) algorithms based on BO, commonly used in AutoML. SMAC, introduced in [17] and recently extended in [23], implements a form of BO with an RF surrogate. HyperOpt was introduced in [2] and uses the Tree-Structured Parzen Estimator (TPE) as a surrogate. Both optimizers are able to take into account the natural hierarchy present in selection-configuration problems. SMAC achieves this by modeling the hierarchical domains using an RF, whereas HyperOpt models it as a graph-structured generative process, which sets valid configuration parameters sequentially. They are both actively maintained GitHub projects and provide APIs that can be used to solve arbitrary selection-configuration problems (rather than being specific to AutoML). Therefore, they can be easily applied to multi-cloud configuration.

Another solution used to solve the CASH optimization problem and capable of utilizing the hierarchical structure of the domain is Rising Bandits (RB), a best-arm identification approach proposed in [22]. RB defines multiple arms, each corresponding to a different ML model. Pulling an arm corresponds to running a fixed number of iterations of BO to find the best hyperparameter setting for the corresponding ML model. The goal of the optimization is to identify the best arm, which maximizes the reward (given by the best validation accuracy found by the optimizer). To have an efficient algorithm, the goal is to achieve this whilst minimizing the total number of arm pulls. To that end, RB makes a theoretical assumption about the way the model’s accuracy converges as a function of BO iterations. Specifically, RB assumes that after a certain number of pulls all arms will reach a point of diminishing returns. When this point is reached, the authors derive a theoretical criterion for when the arm is guaranteed to never outperform another available arm, at which point it is eliminated. Thanks to arm eliminations, more budget can be devoted to more promising ML models.

Adapting a best-arm identification scheme such as RB to the multi-cloud configuration use case is relatively straightforward. Firstly, each arm is associated with a different cloud provider. Each arm pull corresponds to running an optimizer searching for the best cloud configuration and number of nodes for the respective cloud provider. Rather than trying to maximize accuracy, in this context the goal of RB is to minimize runtime or cost. Unfortunately, the theoretical assumption made by RB is not guaranteed to hold in a multi-cloud configuration scenario. As a result, RB may incorrectly choose the arm to be eliminated. Despite this, we include RB in our experimental work.

D. A Novel Bandit-based Approach: CloudBandit

Rising Bandits is promising in the AutoML context but its assumptions cannot be translated to the multi-cloud scenario. Therefore, we develop a new bandit-based algorithm, CloudBandit (CB), specifically designed for the multi-cloud configuration use case. CB is different to Rising Bandits in three important aspects. Firstly, each arm pull corresponds to running an optimizer searching for the best cloud configuration and number of nodes for the respective cloud provider. Rather than trying to maximize accuracy, in this context the goal of RB is to minimize runtime or cost. Unfortunately, the theoretical assumption made by RB is not guaranteed to hold in a multi-cloud configuration scenario. As a result, RB may incorrectly choose the arm to be eliminated. Despite this, we include RB in our experimental work.
exponentially more than those that are eliminated. Finally, CB uses a simple empirical method for eliminating arms, rather than relying on any theoretical assumptions.

We describe CB in detail in Algorithm 1. The algorithm maintains an active set of arms $A$, which is initially equal to the complete set of all arms (i.e., all cloud providers $K$). Pulling an arm corresponds to running a single iteration of an arbitrary black-box optimizer searching for the best configuration and number of nodes for the corresponding cloud provider. Let $p_{k,t} \in X^{(k)} \times N$ denote the best configuration and number of nodes found by pulling the $k$-th arm $t$ times and let loss $L_{k,t} = f_k(p_{k,t})$ denote the runtime or cost for this (configuration, number of nodes) pair. CB performs a number of rounds equal to the number of providers $K$. In the $m$-th round (for $m = 1, \ldots, K$), each arm in the active set $A$ is pulled $b_m$ times, i.e., we run $b_m$ iterations of an arbitrary black-box optimizer to find the best configuration and number of nodes for the corresponding cloud provider. Therefore, in the $m$-th round CB finds the best (configuration, number of nodes) pair $p_{k,b_m}$ and corresponding loss $L_{k,b_m}$ for each active arm $k \in A$, where $b = \sum_{i=1}^{m-1} b_i$ denotes the total number of pulls performed on each active arm in past rounds. Next, the arm in the active set with the highest loss $L_{k,b_m}$ is identified and eliminated from the active set $A$. Before proceeding to the next round, the search budget is increased by a multiplicative factor $\eta$. At the end of the $K$-th round, the best identified (configuration, number of nodes) pair is returned, for the sole remaining provider.

**Algorithm 1 CloudBandit**

1. Initialize set of arms (cloud providers): $A = K = \{1, \ldots, K\}$ and $b = 0$. Choose component black-box optimizer $O$.
2. Set initial budget $b_1$ and budget growth factor $\eta$.
3. for round $m = 1, \ldots, K$ do
   4. for cloud provider $k \in A$ do
      5. Run $b_m$ iterations of optimizer $O$ for provider $k$.
      6. Receive best (configuration, nodes) pair $p_{k,b_m}$ and corresponding loss: $L_{k,b_m} = f_k(p_{k,b_m})$.
   7. end for
   8. Identify the worst arm and eliminate it from the active set: $A = A \setminus \{\arg \max_{k \in A} L_{k,b_m}\}$.
   9. Increment $b = b + b_m$ and set budget for next round: $b_{m+1} = \eta \cdot b_m$.
10. end for
11. Output best provider $k^* = \arg \min_{k \in A} L_{k,b}$ and its best (configuration, nodes) pair $p_{k^*,b}$.

The algorithm has two hyperparameters: the initial budget $b_1$ and the growth factor $\eta$. The total search budget of CB can be expressed in terms of these two hyperparameters: $B = \sum_{m=1}^{K} (K - m + 1) b_1 \eta^{m-1}$. By using relatively small $b_1$ and setting $\eta > 1$, it is hoped that the algorithm can eliminate slow (or expensive) cloud providers very quickly and devote exponentially more budget to the more promising cloud providers.

### Table II: Details of the optimization tasks and cloud configuration parameters used in the dataset.

| Cloud configuration | Datasets                                      | Targets          |
|---------------------|-----------------------------------------------|------------------|
| AWS                 | buzz in social media [20], credit card clustering, quantile transformer, standard scaler, xgboost | cost, runtime    |
| Azure               | family: D_y2, D_y3                             |                  |
| GCP                 | type: standard, highmem, highcpu, vcpu: 2, 4   |                  |

| Nodes               | 2, 3, 4, 5 |

### IV. EXPERIMENTAL RESULTS

**A. Building an Offline Benchmark Dataset**

Comparing search-based methods in an online setting requires time consuming and expensive evaluations each and every time an algorithm is run. Therefore, in order to compare the different methods described in Section III in a runtime and cost-effective way, it was necessary to first construct an offline dataset that can be used for benchmarking. This dataset could then be used to evaluate all of the different methods, including many repetitions with different random seeds, by simulating the behaviour of the algorithms. Specifically, when the algorithm needs to evaluate a workload using given configuration on a given provider, rather than going and creating a Kubernetes cluster and deploying the workload, we can simply read the runtime or cost from the offline dataset that was previously collected. While a similar dataset has previously been published for the single-cloud configuration problem [16], there exists no public data regarding multi-cloud configuration.

**Workloads and optimization tasks.** When constructing our dataset, we considered a variety of distributed data analytics tasks implemented using the Dask framework [7]. Dask was selected since it has a clean integration with Kubernetes, and supports a wide variety of tasks including ML training, as well as data pre-processing. We tried to select a set of tasks with significantly different characteristics. For example, distributed training of gradient-boosted decision trees (XGBoost [5]) involves plenty of branching logic and complex communication patterns, whereas distributed clustering tasks like k-means are typically compute-bound with minimal communication [10]. Each selected Dask task was used to generate optimization tasks, each comprising: (a) a workload defined as a (Dask task, input dataset) pair, and (b) an optimization target which can be either runtime or cost. We collected data for 30 workloads (10 Dask tasks running on 3 input datasets) and 2 optimization targets, leading to a total of 60 optimization tasks. Details regarding all optimization tasks can be found in Table II.

**Cloud provider and configuration space.** Each of the 30 workloads was then deployed on a variety of different configurations across three different cloud providers: Amazon

---

1 Our dataset has been made public and is available at [https://github.com/IBM/multi-cloud-configuration-dataset](https://github.com/IBM/multi-cloud-configuration-dataset).
Web Services™ (AWS™), Microsoft Azure™ and Google Cloud Platform™ (GCP™). For each cloud provider, we selected the region that was closest to our research lab geographically. For AWS™, we generated 24 different configurations by varying the cluster size, together with the family and size options. For Azure™, we varied family, cpu_size together with the cluster size to generate 16 different configurations. Finally, for GCP™ we generated 48 different configurations by varying the cluster size, family, type and vcpu. This resulted in a total of 88 different multi-cloud configurations, specific details of which are also provided in Table II. This was the largest configuration space we could evaluate given our research budget constraints. For each workload on each configuration, we measured the total execution runtime. Additionally, we estimated the execution cost in the same manner as [1], [31] and [33] by multiplying the runtime by the listed price-per-hour for each type of node, together with a factor equal to the number of nodes. While this is an imperfect estimate, and will not take into account additional data transfer costs, more precise estimates are difficult to obtain since cloud billing typically occurs monthly in an aggregate manner.

B. Experimental Methodology and Implementation Details

Using the offline dataset we have collected, we evaluated various predictive and search-based methods from the cloud configuration literature adapted to the multi-cloud setting, search-based methods from the AutoML literature applied to multi-cloud configuration, as well as the new search-based algorithm we have proposed in this paper (CloudBandit). For each method, experiments were performed for all 30 workloads, for both runtime and cost optimization targets. For the search-based methods, we vary the search budget $B = 11, 22, \ldots, 88$ and perform 50 repetitions using different random seeds. Methods are compared in terms of their regret: the relative distance to the true minimum, averaged over all seeds and workloads. Below, we provide details regarding how each of the methods was implemented.

Linear Predictor. We implemented a purely-online predictive method that builds a linear model for each workload and each cloud configuration using the features defined by Ernest [31]. Unlike Ernest, which trains the model using online evaluations taken using a small subset of the workload input dataset, we train the linear model using online evaluations of the workload using the full input dataset following a leave-one-out approach. Specifically, for a given workload, cloud provider and a provider-specific configuration, we estimate the runtime/cost for a given cluster size $n \in \mathcal{N}$ by training the linear model using evaluations on all clusters of size $n' \in \mathcal{N} \setminus n$. Relative to Ernest, this approach has a larger online cost, but should provide a strictly best-case scenario in terms of predictive performance.

Random Forest Predictor. Following a similar approach to PARIS [33], we implement a predictive approach which builds a set of RF models, one for each cloud provider, to predict the runtime/cost of a given workload on a given configuration. Each model is constructed over features related to the configuration, as well as fingerprint features corresponding to the performance of the workload on 2 reference configurations. Unlike [33], our fingerprints simply contain the execution runtime (or cost) of the workload, rather than low-level information captured from the cluster, as our work is focused on a scenario where the workload is a black box. To make predictions for a target workload $w$, the model is first trained using feature vectors corresponding to all workloads $w' \neq w$ in the offline dataset. Then, in order to construct the fingerprint for the target workload so that a prediction can be made, it is necessary to perform online evaluations using a total of 6 reference configurations, 2 for each cloud provider.

Random Search (RS). We implemented the simplest search-based baseline. For a given budget $B$, we select $B$ configurations at random (with replacement, across all cloud providers) and select the best one.

Bayesian Optimization (BO). We used BO tools provided by scikit-optimize [13] to implement the search-based scheme proposed in CherryPick [1], and the search-based schemes suggested by Bilal et al. [3]. For CherryPick, we use a GP surrogate, a Matern5/2 kernel and the EI acquisition function. For the schemes from [3], we use a GP surrogate and LCB acquisition function when optimizing for cost, and an RF surrogate and PI acquisition function when optimizing for runtime. In order to adapt the BO methods to the multi-cloud scenario, we consider the flattened method defined in Section III-B1, as well as the approach using independent optimizers, as defined in Section III-B2.

SMAC [23] and HyperOpt [2]. We applied both SMAC and HyperOpt using the code provided on GitHub, in both cases taking care to make sure that the conditional dependencies resulting from the hierarchical nature of the domain are specified correctly.

Rising Bandits (RB) [22]. We implemented Rising Bandits ourselves, as no public code was available. To that end, for the component optimizer we use the BO implementation provided by scikit-optimize [13] with default settings (GP surrogate and gp-hedge acquisition function), as no details regarding the used BO method were provided. We implement lower and upper confidence bounds of each arm’s runtime or cost which guide arm elimination analogously to the author’s confidence bounds of the ML model’s accuracy.

CloudBandit (CB). We set CloudBandit’s hyperparameter $\eta = 2$ and use multiple values of $b_i$ to vary the total search budget $B$. We consider 2 component black-box optimizers: CherryPick and RBFOpt [6]. RBFOpt is a black-box optimizer based on the Radial Basis Function method, originally proposed in [12], and is considered in our experiments because it has been shown to outperform BO in a wide variety of scenarios [26].

C. Comparing Methods from Cloud Configuration Literature

In Figure 2, we present a comparison between the predictive methods, random search and BO methods from the cloud configuration literature. As there is no well-defined notion of search budget for predictive methods, we present their results as
horizontal lines. For the BO methods, we compare the approach using the flattened domain (denoted ‘x1’), together with the approach using independent optimizers for each cloud provider (denoted ‘x3’) for a range of search budgets $B$. From these results, we can draw a number of interesting conclusions:

- While the predictive methods, in particular the RF-based approach, identify a relatively good configuration, search-based methods are able to identify configurations with a significantly lower regret, particularly when optimizing for runtime.
- Both adaptations of state-of-the-art BO-based methods (CherryPick and the solution by Bilal et al.) are outperformed by random search in the majority of cases. They beat random search only for the cost optimization target with lower budgets.
- While in most cases the approach using independent optimizers (‘x3’) achieves lower regret than the approach using the flattened domain (‘x1’), it is not consistently better. Thus, it seems that neither approach is capable of adequately capturing the structure that is inherent to the multi-cloud configuration domain.

### D. Comparing Hierarchical Methods

In Figure 3 we compare the performance of the three search-based methods from the AutoML literature (SMAC, HyperOpt and Rising Bandits), against CloudBandit with both component optimizers (CB-CherryPick and CB-RBFOpt). Additionally, we include the ‘x1’ and ‘x3’ adaptations of CherryPick, as well as random search, for reference. From these results, we can draw the following conclusions:

- By explicitly acknowledging the hierarchy that is present in the problem, SMAC is able to outperform almost all methods in terms of regret, for both small and large budgets.
- HyperOpt performs well for large budgets, consistently outperforming both adaptations of CherryPick, but it is beaten by both CB-RBFOpt and SMAC. We attribute this to the fact that unlike SMAC, HyperOpt can repeat configurations, which effectively wastes precious evaluations.
- Rising Bandits performs relatively well for small budgets, but is significantly worse than the other AutoML methods for large budgets. We believe this is evidence that the theoretical criterion that is used to eliminate arms does not translate well to the multi-cloud setting.
CB-RBFOpt consistently beats HyperOpt, Rising Bandits and CherryPick. It also outperforms SMAC for the cost optimization target with small search budgets and is only slightly worse than SMAC for the runtime optimization target.

- Embedding CherryPick inside CloudBandit significantly improves its performance, consistently beating both the ‘x1’ and the ‘x3’ adaptations. This, together with excellent results of CB-RBFOpt, demonstrates that CloudBandit is capable of identifying the least promising providers early on in the search, and thus it can devote more budget to more promising providers and find the best configurations faster.

- SMAC and CB-RBFOpt are the only optimizers that consistently beat random search across all budgets and targets.

E. Savings Analysis

We proceed to compare the best methods identified above in terms of cost and runtime savings that they can achieve in production. When a user chooses to use any of the multi-cloud configuration solutions discussed above, they agree to the additional, one-time expense (cost or runtime) carried by the optimization process. Then they proceed to run the batch workload $N$ times, achieving some savings each time thanks to the optimized cloud deployment. A natural question is whether the savings incurred during the $N$ production runs of the workload make up for the additional expense of the optimization process itself. If the optimization expense outweighs the savings, it is more beneficial for the user to simply select a random provider and configuration.

For that reason, we analyze the best algorithms developed in this work in terms of savings compared to choosing the cloud provider and configuration randomly. We define savings as follows:

$$S = \frac{TE_{\text{rand}} - TE_{\text{opt}}}{TE_{\text{rand}}} = \frac{N \cdot R_{\text{rand}} - (C_{\text{opt}} + N \cdot R_{\text{opt}})}{N \cdot R_{\text{rand}}}$$

where $TE_{\text{opt}}$ and $TE_{\text{rand}}$ denote the total expense (cost or runtime) incurred by the optimized or random configuration strategy, respectively. $C_{\text{opt}}$ denotes the one-time expense of the optimization process and $R_{\text{opt}}$ and $R_{\text{rand}}$ denote the expense of running the workload with the optimized or random cloud configuration, respectively. $N$ denotes how many workload production runs are performed before the optimizer is used to update the cloud configuration again.

In this study, we present results for the optimizers running once per a fixed number of $N = 64$ production runs, all given a fixed search budget $B = 33$. For the analysis we selected 4 best algorithms from all multi-cloud solutions evaluated previously: SMAC, HyperOpt, CB-CherryPick and CB-RBFOpt. As baselines, we present random search and exhaustive search. The savings were computed using each algorithm’s results averaged over all random seeds, separately for each workload. The box plots in Figures 4a and 4b present the distributions of savings of the analyzed algorithms across the workloads. The boxes mark the interquartile range (IQR) of the distribution (i.e., the 25–75 percentile range), with the additional line inside marking the median. The whiskers spread by at most 1.5 IQR in both directions.

For the runtime optimization target, the differences between the algorithms’ savings are not as distinct as for the cost optimization target, but the tendencies of how the algorithms compare to one another are the same for both targets. SMAC provides median savings of approximately 30% for the cost target and 10% for the runtime target, outperforming random search and exhaustive search in terms of savings for both optimization targets. HyperOpt achieves higher median savings than SMAC for both optimization targets: approximately 40% for the cost target and 15% for the runtime target. Finally, both CloudBandit schemes (CB-RBFOpt and CB-CherryPick) clearly outperform both SMAC and Hyperopt by achieving median savings of approximately 65% for the cost target and 20% for the runtime target. This is the result of CloudBandit identifying and focusing on the best cloud providers early on, thanks to which it reduces the number of evaluations made on the more expensive cloud providers. For the runtime optimization target, all 4 optimizers sometimes achieve negative savings, but only within the 0–25 percentile range. Additionally, for the cost optimization target, no algorithm has negative savings for any workload. This means that all 4 optimizers...
included in this analysis can be considered more profitable than simply selecting a random cloud provider and configuration. Finally, exhaustive search is dramatically outperformed by all other approaches and achieves strictly negative savings for both optimization targets. Testing all available configurations is therefore not a feasible approach to multi-cloud configuration.

V. CONCLUSION

In this paper, we have performed an in-depth study of the suitability of different predictive and search-based methods for the multi-cloud configuration problem. We have first shown that existing search-based methods from the single-cloud configuration literature, adapted to the multi-cloud optimization problem, outperform state-of-the-art predictive methods. However, they are still inferior to random search in the majority of cases, even when adapting them to take into account the hierarchical structure of the multi-cloud optimization domain. Furthermore, we have shown that an AutoML solution (SMAC), which natively leverages this hierarchy, exhibits excellent performance when compared to search-based methods from the cloud configuration literature, as well as random search. Lastly, we have introduced a novel search-based method, CloudBandit, specifically designed to solve the multi-cloud problem. The experimental results have demonstrated that CloudBandit has similar performance to SMAC when using RBFOpt as an internal black-box optimizer. CloudBandit has two advantages over SMAC: 1) it performs better for the cost optimization target with small search budgets, and 2) it provides larger cost and runtime savings in production. As future work, we are planning to perform a similar in-depth study, but for another category of workloads, namely ML inference applications. We further intend to enhance CloudBandit’s performance by using a component black-box optimizer that incorporates the workload’s low-level metrics, such as CPU usage or I/O pressure.

REFERENCES

[1] Omid Alipourfard, Hongqiang Harry Liu, Jianshu Chen, Shivaram Venkataraman, Minlan Yu, and Ming Zhang. Cherrypick: Adaptively unearthing the best cloud configurations for big data analytics. In 14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17), pages 469–482, Boston, MA, March 2017. USENIX Association.

[2] James Bergstra, Daniel Yamins, and David Cox. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. In Proceedings of the 30th International Conference on Machine Learning, volume 23 of Proceedings of Machine Learning Research, pages 115–123, Atlanta, Georgia, USA, 17–19 Jun 2013. PMLR.

[3] Muhammad Bilal, Marco Serafini, Marco Canini, and Rodrigo Rodrigues. Do the best cloud configurations grow on trees? An experimental evaluation of black box algorithms for optimizing cloud workloads. Proc. VLDB Endow., 13(12):2563–2575, July 2020.

[4] Eric Brewer. An update on container support on google cloud platform, 2014. https://cloudplatform.googleblog.com/2014/06/an-update-on-container-support-on-google-cloud-platform.html, accessed: 5.10.2021.

[5] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD ’16, page 785–794, New York, NY, USA, 2016. Association for Computing Machinery.

[6] Alberto Costa and Giacomo Nannicini. Rbfopt: an open-source library for black-box optimization with costly function evaluations. Mathematical Programming Computation, 10(4):597–629, 2018.

[7] Dask Development Team. Dask: Library for dynamic task scheduling, 2016.

[8] Juan J Durillo and Rada Prodan. Multi-objective workload scheduling in amazon ec2. Cluster computing, 17(2):169–189, 2014.

[9] Andrew D. Ferguson, Peter Bodik, Srikant Kandula, Eric Boutin, and Rodrigo Fonseca. Jockey: Guaranteed job latency in data parallel clusters. In Proceedings of the 7th ACM European Conference on Computer Systems, EuroSys ’12, page 99–112, New York, NY, USA, 2012. Association for Computing Machinery.

[10] George Forman and Bin Zhang. Distributed data clustering can be efficient and exact. SIGKDD Explor. News., 2(2):34–38, dec 2000.

[11] Gartner. Gartner says cloud will be the centerpiece of new digital experiences, 2021. https://www.gartner.com/en/newsroom/press-releases/2021-11-10-gartner-says-cloud-will-be-the-centerpiece-of-new-digital-experiences, accessed: 15.02.2022.

[12] H-M Gutmann. A radial basis function method for global optimization. Journal of global optimization, 19(3):201–227, 2001.

[13] Tim Head, Munju Kumar, Holger Nahasteidt, Gilles Louppe, and Iaroslav Shcherbatyi. scikit-optimize/scikit-optimize, September 2020.

[14] Chin-Jung Hsu, Vivek Nair, Vincent W. Freeh, and Tim Menzies. Arrow: Low-level augmented bayesian optimization for finding the best cloud vm. In 2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS), pages 660–670, 2018.

[15] Chin-Jung Hsu, Vivek Nair, Tim Menzies, and Vincent Freeh. Micky: A cheaper alternative for selecting cloud instances. In 2018 IEEE 11th International Conference on Cloud Computing (CLOUD), pages 409–416, 2018.

[16] Chin-Jung Hsu, Vivek Nair, Tim Menzies, and Vincent W Freeh. Scout: An experienced guide to find the best cloud configuration. arXiv preprint arXiv:1803.01296, 2018.

[17] Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In Proceedings of the 5th International Conference on Learning and Intelligent Optimization, LION’05, page 507–523, Berlin, Heidelberg, 2011. Springer-Verlag.

[18] Kaggle. Credit card fraud detection, 2018. https://www.kaggle.com/mlg-ulb/creditcardfrad.

[19] Kaggle. Santander customer transaction prediction, 2019. https://www.kaggle.com/c/santander-customer-transaction-prediction.

[20] F. Kawala, A. Douzal-Chouakria, E. Gaussier, and E. Dmert. Prédictions d’activité dans les réseaux sociaux en ligne. In Actes de la Conférence sur les Modèles et l’Analyse des Réseaux : Approches Mathématiques et Informatique (MARAMI), page 16, 2013. http://archive.ics.uci.edu/ml/datasets/Buzz+in+social+media+.

[21] Ana Klimovic, Heiner Ltz, and Christos Kozyrakis. Selecta: Heterogeneous cloud configuration for data analytics. In 2018 USENIX Annual Technical Conference (USENIX ATC 18), pages 759–773, Boston, MA, July 2018. USENIX Association.

[22] Yang Li, Jiawei Jiang, Jinyang Gao, Yingxia Shao, Ce Zhang, and Bin Cui. Efficient automatic cash via rising bands. Proceedings of the AAAI Conference on Artificial Intelligence, 34(04):4763–4771, Apr. 2020.

[23] Marius Lindauer, Katharina Eggensperger, Matthias Feurer, André Biedenkapp, Dihan Deng, Carolin Benning, René Sass, and Frank Hutter. Smac3: A versatile bayesian optimization package for hyperparameter optimization. arXiv preprint arXiv:2109.09831, 2021.

[24] Li Liu, Miao Zhang, Rajkumar Buyya, and Qi Fan. Deadline-constrained coevolutionary genetic algorithm for scientific workflow scheduling in cloud computing. Concurrency and Computation: Practice and Experience, 29(5):e3942, 2017.

[25] Giovanni Mariani, Andreea Anghel, Rik Jongerius, and Gero Dittmann. Predicting cloud performance for hpc applications before deployment. Future Generation Computer Systems, 87:618–628, 2018.

[26] Giacomo Nannicini. On the implementation of a global optimization method for mixed-variable problems. arXiv preprint arXiv:2009.02185, 2020.

[27] Arun Ramanurthy, Saket Saurabh, Manesh Ghare, and Sachin Lodha. Selection of cloud service providers for hosting web applications in a multi-cloud environment. In 2020 IEEE International Conference on Services Computing (SCC), pages 202–209, 2020.

[28] Tao Shi, Hui Ma, Gang Chen, and Sven Hartmann. Location-aware and budget-constrained service deployment for composite applications in
multi-cloud environment. *IEEE Transactions on Parallel and Distributed Systems*, 31(8):1954–1969, 2020.

[29] Haluk Topcuoglu, Salim Hariri, and Min-You Wu. Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE transactions on parallel and distributed systems*, 13(3):260–274, 2002.

[30] Ruben Van den Bossche, Kurt Vanmechelen, and Jan Broeckhove. Cost-optimal scheduling in hybrid iaas clouds for deadline constrained workloads. In *2010 IEEE 3rd international conference on cloud computing*, pages 228–235. IEEE, 2010.

[31] Shivaram Venkataraman, Zongheng Yang, Michael Franklin, Benjamin Recht, and Ion Stoica. Ernest: Efficient performance prediction for Large-Scale advanced analytics. In *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*, pages 363–378, Santa Clara, CA, March 2016. USENIX Association.

[32] Ziyu Wang, Masrour Zoghi, Frank Hutter, David Matheson, Nando De Freitas, et al. Bayesian optimization in high dimensions via random embeddings. In *IJCAI*, pages 1778–1784, 2013.

[33] Neeraja J. Yadwadkar, Bharath Hariharan, Joseph E. Gonzalez, Burton Smith, and Randy H. Katz. Selecting the best vm across multiple public clouds: A data-driven performance modeling approach. In *Proceedings of the 2017 Symposium on Cloud Computing*, SoCC ’17, page 452–465, New York, NY, USA, 2017. Association for Computing Machinery.

[34] Wei Zheng and Rizos Sakellariou. Budget-deadline constrained workflow planning for admission control. *Journal of grid computing*, 11(4):633–651, 2013.