Smartpick: Workload Prediction for Serverless-enabled Scalable Data Analytics Systems

Anshuman Das Mohapatra
University of Nebraska at Omaha
Omaha, Nebraska, USA
adasmohapatra@unomaha.edu

Kwangsung Oh
University of Nebraska at Omaha
Omaha, Nebraska, USA
kwangsungoh@unomaha.edu

Abstract

Many data analytic systems have adopted a newly emerging compute resource, serverless (SL), to handle data analytics queries in a timely and cost-efficient manner, i.e., serverless data analytics. While these systems can start processing queries quickly thanks to the agility and scalability of SL, they may encounter performance- and cost-bottlenecks based on workloads due to SL's worse performance and more expensive cost than traditional compute resources, e.g., virtual machine (VM). In this paper, we introduce *Smartpick*, a SL-enabled scalable data analytics system that exploits SL and VM together to realize composite benefits, i.e., agility from SL and better performance with reduced cost from VM. Smartpick uses a machine learning prediction scheme, decision-tree based Random Forest with Bayesian Optimizer, to determine SL and VM configurations, i.e., how many SL and VM instances for queries, that meet cost-performance goals. Smartpick offers a *knob* for applications to allow them to explore a richer cost-performance tradeoff space opened by exploiting SL and VM together. To maximize the benefits of SL, Smartpick supports a simple but strong mechanism, called *relay-instances*. Smartpick also supports event-driven prediction model retraining to deal with workload dynamics. A Smartpick prototype was implemented on Spark and deployed on live test-beds, Amazon AWS and Google Cloud Platform. Evaluation results indicate 97.05% and 83.49% prediction accuracies respectively with up to 50% cost reduction as opposed to the baselines. The results also confirm that Smartpick allows data analytics applications to navigate the richer cost-performance tradeoff space efficiently and to handle workload dynamics effectively and automatically.

CCS Concepts: • **Computer systems organization** → **Cloud computing**; • **Computing methodologies** → *Machine learning approaches*; *Model development and analysis*; *Distributed computing methodologies.*

Keywords: serverless-enabled, machine learning, prediction model, cost-performance tradeoff, relay

ACM Reference Format:
Anshuman Das Mohapatra and Kwangsung Oh. 2023. Smartpick: Workload Prediction for Serverless-enabled Scalable Data Analytics Systems. In *24th International Middleware Conference (Middleware '23), December 11–15, 2023, Bologna, Italy*. ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3590140.3592850

1 Introduction

1.1 Motivation

Many Internet applications are running on cloud environments and generating large-scale data, e.g., Facebook [15], Twitter [46] and Google [19]. For these Internet applications, analyzing high volume of data is one of the most important workloads. For example, Facebook and Twitter analyze users' posts, users' activity logs, systems' logs to query trends, make advertising decisions, and check overall cluster health. Since the results of data analytics queries are usually used for making important decisions that affect revenues and system health, the queries must be processed promptly without a performance bottleneck.

To meet the performance goals, data analytics systems may deploy *redundant* compute resources, e.g., virtual machines (VMs), *a prior*. While this approach is simple and works well, this will incur additional cost ($) for idle VMs. To avoid cost for unused compute resources, many previous works [1, 2, 17, 21, 40, 47, 61, 70, 77, 82] focused on determining optimal configurations, e.g., the number of VM instances and their types, and storage types, by predicting required compute resources for workloads. With these systems, additional VMs can be deployed to handle incoming queries without the performance bottleneck and idle VMs can be terminated to reduce cost based on workloads, i.e., scalable data analytics systems. These systems, however, may not handle the *latency-sensitive* queries promptly due to the unavoidable overhead of VM, i.e., boot-up latency (> 55 seconds) [30, 48]. If queries cause peak workload due to a lack of compute resources, they must wait until additional VM instances are fully deployed to be processed.

Many recent works [28, 31, 34, 38, 41, 64, 71, 72] focused on adopting a newly emerging compute resource, *serverless* (SL), such as Apache OpenWhisk [12], AWS Lambda [3], Azure Functions [51], and Google Functions [24], for data analytics

Middleware '23, December 11–15, 2023, Bologna, Italy

Table 1. Comparison between SL and VM with the same amount of compute resources (2 vCPU with 2 GB RAM)

	SL	VM
Agility (Boot latency)	**High (< 100 ms)**	Low (> 55 seconds)
Performance	Varying based on memory size	**Relatively constant**
Cost Efficiency	**High (Pure pay-as-you-go : only when executed)**	Low (Pay-as-you-go : when deployed)
Unit Time Cost ($)	Expensive (up to 5.8X)	**Cheaper**

to avoid the cold-boot latency problem, i.e., serverless data analytics (SDA). Since SL offers *agility*, very small boot-up time (< 100 ms), and a *pure-pay-as-you-go* cost model[1], SDA systems can deploy SL instances[2] immediately and handle incoming queries without overprovisioned VMs in a cost-efficient way. These SDA systems, unfortunately, may still encounter cost- and performance bottlenecks based on data analytic workloads because SL offers worse performance and more expensive cost than VM [43, 44, 59].

Table 1 shows the comparisons between SL and VM, which represents different cost-performance points. While data analytics systems may choose either one based on their resource demands and goals, it would be highly desirable for them to achieve composite benefits (**bold** in Table 1), i.e., agility and cost-efficiency from SL and better performance and cheaper cost from VM. However, determining compute resources configurations, e.g., how many SL and VM instances, is challenging due to the complexities: 1) heterogeneous compute resource characteristics, 2) workload prediction (how long a query will be executed), 3) diverse cost-performance goals, and 4) dynamics from workloads. While some recent works [28, 31, 58, 59, 68, 84] tried to exploit SL and VM together but they could not address these challenges as they have focused on either simple workload (independent tasks) or simple assumption without workload prediction. Thus, they may not work well for data analytics.

In this paper, we introduce *Smartpick*, a serverless-enabled data analytics system (SEDA), that helps data analytics applications achieve desired cost-performance goals by addressing aforementioned challenges. To determine *optimal* cloud configurations of SL and VM, Smartpick uses a machine learning technique, decision-tree based Random Forest (RF) coupled with Bayesian Optimizer (BO), that predicts data analytic workloads using historical information. Smartpick provides a *knob* that allows applications to easily explore the cost-performance tradeoff space opened by exploiting SL and VM together. Smartpick supports a simple but strong mechanism called *relay-instances* to further improve performance with reduced cost. To handle workload dynamics, Smartpick uses an event-driven approach that triggers a model retraining task to automatically evolve prediction models.

[1]Most popular cloud providers charge for SL only when the code is executed at either 1 millisecond (AWS) or 100 millisecond (GCP) granularity.

[2]We use the term serverless instances to refer serverless code invocations.

Table 2. Feature comparison with state-of-the-art. △ indicates that metric is considered but with limitations

	Cocoa	SplitServe	Smartpick
Exploiting SL & VM	✓	✓	✓
Workload Prediction			✓
Handling Dynamics			✓
Segueing (Relay-instances)		△	✓
Cost-performance Tradeoff	△		✓

A Smartpick prototype implementation was built on the Spark [13], so that Spark applications can easily utilize our system by setting diverse Smartpick's properties *without any modification*. We evaluated Smartpick on live-testbeds, Amazon AWS and Google Cloud Platform (GCP), using well-known benchmarks: TPC-DS [54], Word Count [74], and TPC-H [73]. Evaluations show that Smartpick can accurately characterize the TPC-DS workload performance with accuracies of 97.05% on AWS and 83.49% on GCP. The experimental results show that Smartpick can reduce cost by up to 50% without performance degradation by using the relay-instances mechanism. The results also confirm that Smartpick allows applications to easily explore the richer cost-performance tradeoff space with a simple knob and to handle workload dynamics by retraining the prediction model automatically.

1.2 Research Contributions

Table 2 compares Smartpick approach to two recent SEDA systems, i.e., Cocoa [59] and SplitServe [31]. While these systems utilize both SL and VM, they do not predict queries' workloads but just rely on external workload prediction systems [1, 2, 17, 21, 40, 47, 61, 70, 77, 82]. However, these prediction systems may not work well in SEDA due to their SL-agnostic approach and workload dynamics, which significantly affect overall cost and performance. Thus, we designed the workload prediction module to easily work with any SEDA system that needs performance prediction. Since using SL for a long time would incur additional cost without performance improvement [31, 59], Smartpick judiciously and dynamically terminates SL instances using the mechanism called *relay-instances*. While SplitServe [31] uses a similar technique called segueing, they use a static approach, which leads to significant cost inflation. While Cocoa considers exploring the cost-performance tradeoff space like Smartpick, its performance is highly dependent on several static parameters that may be hard to tune in SEDA.

To summarize, the research contributions are as follows:

- The design and implementation of Smartpick, the first scalable data analytics system (to the best of our knowledge) that predicts data analytics workloads with consideration of SL and VM together to determine optimal compute resource configurations.
- Flexibility that allows *unmodified data analytics applications and other SEDA systems* to reap the benefits.

• A simple way to easily explore the cost-performance tradeoff space using diverse mechanisms embedded within the workload prediction.
• Event-driven re-training of the prediction model to handle workload dynamics, e.g., varying data size and new queries.
• Thoughtful empirical evaluations on AWS [4] and GCP [25], showing the efficacy of Smartpick.

2 System Model and Motivation

2.1 System Model

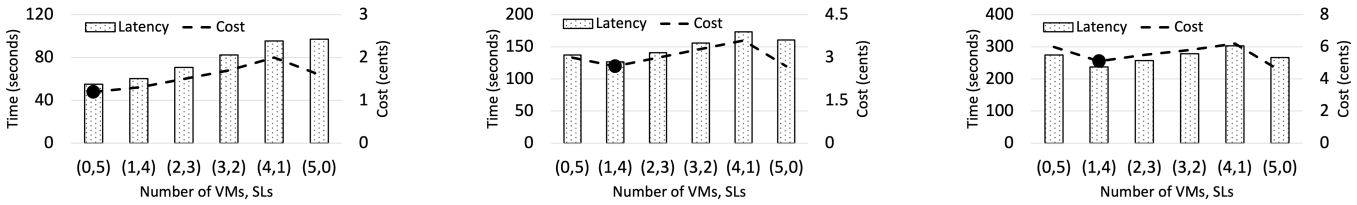
Data center (DC) setting and compute resources: We focus on a single DC environment in the public cloud, where the network is not a performance bottleneck [9] and infinite compute resources, i.e., serverless (SL) and virtual machine (VM), are available. Each compute resource has different characteristics in terms of performance, cost, and agility, as shown in Table 1. Such compute resources heterogeneity opens a rich cost-performance tradeoff space that applications can explore based on their demands. While data within a DC can be accessed and processed without a performance bottleneck, achieving memory-locality is important for performance improvement [9]. Exploiting SL in data analytics requires external storage systems, e.g., Redis [69] or AWS S3 [8], due to its limitations, e.g., limited network and storage, which may incur performance overhead. We assume that performance overhead from losing memory-locality is negligible as we target queries with several tens of seconds granularity. We will discuss potential performance improvement with improved memory locality in Section 7.

Data analytics applications: We consider data analytics applications that generate diverse classes of MapReduce-like queries, e.g., reporting, ad-hoc, iterative, and data mining, as classified in [62, 63]. These queries contain several map and reduce stages that cannot start until all their dependencies are resolved, i.e., dependent tasks. These queries can be processed by de-facto distributed data processing frameworks, e.g., Hadoop [11] and Spark [13]. While reporting queries are somewhat predictable as they are regularly generated based on the schedule, i.e., recurring (*static*) queries, the remaining classes of queries, especially ad-hoc queries, are impromptu and dynamically constructed to answer immediate and specific questions, i.e., *dynamic* queries. In this work, we mainly consider dynamic queries that may cause peak workloads. Applications may utilize infinite compute resources, e.g., redundant VM instances, to handle dynamic queries without the performance bottleneck, which incurs additional cost for under-utilized or idle compute resources [50]. We assume that they have limited operational budgets; thus, minimizing the cost of processing queries within their target performance goals is highly desirable.

Data analytics system (DAS): We assume that DAS deploys an optimal number of long-lived VM instances as *static compute resources* to handle static queries using workload prediction tools or systems [1, 2, 17, 21, 40, 47, 61, 70, 77, 82]. However, DAS may encounter a performance bottleneck due to peak workloads (lack of compute resources) caused by the dynamic queries, e.g., ad-hoc queries. While DAS can deploy additional VM instances to handle the dynamic queries, applications may not achieve the desired performance goals due to unavoidable overhead of VM, i.e., cold boot-up latency (> 55 seconds) [30, 48]. Instead, DAS may deploy SL instances to start processing queries immediately as done in previous works [28, 31, 34, 38, 41, 64, 71, 72], i.e., serverless data analytics (SDA). However, based on query workloads, SDA may encounter the cost-bottleneck for little (or no) performance improvement [59]. To handle dynamic queries in a timely and cost-efficient way, we consider DAS that uses a *hybrid approach exploiting SL and VM together* to achieve composite benefits, i.e., agility and cost-efficiency from SL, and better performance and cheaper cost from VM.

Determining optimal compute resource configuration problem: While recent works [28, 31, 58, 59, 68, 84] have introduced similar hybrid approaches, they adhere to simple assumptions or workloads, e.g., static parameters without workload prediction, dynamics-free prediction model, and independent tasks, which would not work well for serverless-enabled data analytics (SEDA). In this work, we focus on determining the optimal compute resource configurations, i.e., how many SL and VM instances need to dynamically be spawned to handle incoming queries. However, this is challenging because many metrics must be considered, e.g., query workload estimations (prediction), diverse applications' cost-performance goals, and heterogeneous compute resource characteristics. To determine optimal configurations, diverse approaches have been introduced to build performance prediction models using historical data [1, 2, 17, 21, 40, 47, 61, 70, 77, 82]. Unfortunately, these systems do not consider SL, but only VM for compute resources and thus do not work well for SEDA. Furthermore, with a large search space for optimality, novel approaches are required to navigate the solution space efficiently and ensure acceptable overhead/cost for the decision-making. In this work, we use a machine learning technique, decision-tree based Random Forest (RF), to predict data analytic workloads using historical information. To efficiently explore the large search space, we incorporate Bayesian Optimizer (BO) into our prediction model, i.e., RF + BO (Section 3). Given predicted workloads, we focus on minimizing cost while meeting target performance goals, i.e., exploring a cost-performance tradeoff space (Section 3.3).

Dynamics: We assume that applications may send new queries unknown to DAS at any time. In addition, data size can be changed as more data is aggregated. To predict workload correctly, the prediction model must be updated by incorporating these changes (Section 4.2).



(a) 100 tasks (SL-only is best) (b) 250 tasks (Hybrid is best) (c) 500 tasks (Hybrid is best)

Figure 1. Exploring resource determination and tradeoff. • indicates best performance.

2.2 Illustrative Example

Workloads in data analytics systems (DAS) have large variance on query completion times. This stems from the fact that each of them can have different query semantics and thus, dissimilar resource needs to process the given data. To account for such scenarios and to handle the incoming queries efficiently, we highlight the need for performance prediction through an interpretative example. Let's assume three classes of dynamic queries: short-, mid-, and long-running queries, that incur peak workload. These queries have 100 tasks (short), 250 tasks (mid), and 500 tasks (long) respectively. Since all static compute instances are busy handling regular queries, DAS needs to deploy additional compute resources to handle them. In this case, DAS must determine optimal configurations, i.e., *how many SL and VM instances*, that meet the applications' cost-performance tradeoff preference.

DAS has three options to deploy compute resources: 1) SL instances only (SL-only), 2) VM instances only (VM-only), and 3) both VMs and SLs (Hybrid). For the sake of comparison, we consider AWS t3.small instance (2 vCPUs and 2 GB memory) and AWS Lambda with 2GB memory. Note that AWS Lambda (2 GB) offers 2 vCPUs for each invocation. We take cost information from AWS [5, 6]. We consider storage cost for each VM (gp2 8 GB) and Redis [69] (external) storage cost (on master VM instance) whenever SL instances are involved. Note that we choose AWS t3 family for the same compute resources as SL instance, and we consider the burstable costs ($0.05 per vCPU-hour) in our model. For the performance of SL instances, we assume zero-boot latency and include 30% performance overhead to task execution time (based on experimental evidence as shown in Section 6.1). For the VM-only approach, we added 55 seconds to the query completion time as the cold-boot overhead [30, 48].

Figure 1 presents the expected query execution time and cost when DAS applies different approaches for an incoming query, assuming that 5 instances (either SL, VM or combined) are the optimal number of CPU cores. Here (0,5) and (5,0) represent the two extremes of compute resources configuration, i.e., SL-only and VM-only approach, respectively. For the short-query, the SL-only approach offers the best performance with reduced cost, thanks to the agility of SL. For mid- and long-queries, however, the SL-only approach inflates cost without performance improvement, while the hybrid approach leads to better performance with the average cost. Interestingly, the VM-only approach outperforms the SL-only approach for long-running query due to the heterogeneity between SL and VM, as discussed in Section 1. The results clearly show that a workload prediction scheme is extremely important to determine the optimal configurations of VMs and SLs for varying query classes. The results also indicate that there is a richer cost-performance tradeoff space based on the query workloads.

Relaying workload: In the hybrid approach, SLs can be invoked and used until a query is completed, which may incur additional cost without performance improvement due to SL's characteristics, as discussed in Section 2.1. To avoid this, SLs can be terminated when corresponding VM instances are ready to avoid cost inflation and performance degradation, i.e., *relay-instances* mechanism. For example, for a long-running query (500 tasks), 5 SLs and 5 VMs can be allocated simultaneously. The 5 SLs start running the tasks quickly and will be terminated when the corresponding 5 VMs are ready for the rest of the tasks, i.e., after VM's cold-boot time. This approach results in performance improvement to 198.8 seconds with a reduced cost of 5¢, which is a better approach than simply using SLs throughout the query execution. We will discuss the relay-instances mechanism in Section 4.3 in detail.

3 Determining Optimal Configurations

3.1 Workload Prediction

While many workload prediction systems have been proposed [1, 2, 17, 21, 40, 47, 61, 70, 77, 82], none of these works have considered SL to determine compute resource configurations. In this section, we introduce how Smartpick predicts query workload to determine the optimal configuration.

Feature Determination: Precisely predicting the query completion time is one of the key aspects of Smartpick. To this end, we thoroughly analyzed what parameters uniquely determine query completion time. Based on multiple initial runs, we deduced the rich set of features that govern this behavior, which are summarized in Table 3. When new

queries are submitted to an already trained model, the *query-duration* feature will act as the best estimation for completion time. Likewise, different *instances* will be traversed, and the best combination of VMs and SLs will be determined for efficiently executing a new incoming job. Having determined the features, we next explored several approaches [16, 61, 77] for modeling these parameters into query completion time, however, all of these approaches rely heavily on the implicit relationship across the parameters, which can be very difficult to model. Therefore, in our design, we incorporate black-box model for optimal compute-resource determination.

Problem Formulation: We choose decision-tree based Random Forest (RF) technique for quantifying the query completion time. This is preferred over other deep learning neural networks because it is computationally less intensive and requires significantly less training data [10, 23, 37, 75]. Moreover, it reduces model over-fitting through the technique of ensemble learning [32]. Equation 1 provides the formulation for the RF regressor, where β is the rich set of identified features and RF_t is the expected completion time.

$$f(\beta) = RF_t \tag{1}$$

Although this regressor can accurately model the underlying system, the search space involved for exhaustive navigation is huge. Our initial experiments show around *1 minute* of prediction latency when both VMs and SLs are involved for optimality determination. Given the time-sensitivity of data analytics workloads, exhaustive search proves a hindrance for efficient model performance. Therefore, we add a Bayesian Optimizer (BO) module to navigate the search space effectively. The BO in its raw form cannot be used for workload prediction of ad-hoc queries since this leads to a significant compute cost for the resource determination. We discuss these challenges in detail in Section 3.2. Hence, we modify the BO technique to tune it in accordance with cost-effectiveness.

Two primary components are associated with the BO, i.e., objective and surrogate functions. Equation 2 defines the objective function which is tailor-made for Smartpick. In this equation, RF_t is the predicted query completion time from the RF regressor and δ is the noise value which follows normal distribution. The surrogate function is chosen to be a *Gaussian Process Regressor*, since they demonstrate several remarkable characteristics. First, the variance in prediction accurately models the noise in observations, and second, it can precisely generate values for newer data points [56].

$$maximize : -(RF_t + \delta) \tag{2}$$

For the acquisition function, there are several choices - Expected Improvement (EI), Probability of Improvement (PI) and Upper Confidence Bound (UCB) [81]. For Smartpick, we incorporate PI over the other options because it is similar to EI and simpler [49], as well as, it is one of the most widely used acquisition functions for optimizers [36]. Thus, PI helps in efficiently exploiting/exploring the search space for optimal/near-optimal compute resource configurations in the form of tuples: $\{nVM, nSL\}$, where nVM is the desired number of VMs and nSL is the desired number of SLs. The termination criteria of the search are aligned with the improvement to (estimated) query completion time. If the improvement does not increase by 1% for 10 consecutive searches, the model returns the accomplished core configurations for VMs and SLs.

Table 3. Features for Workload Prediction

Feature	Comments
instances	Number of VMs and SLs used
input-size	Size of input in bytes
start-time-epoch	Initial job submit time in epoch
total-memory	Total memory of available workers
available-memory	Available memory of available workers
memory-per-executor	Memory assigned to each executor
num-waiting-apps	Number of applications in wait state
total-available-cores	Number of available cores
query-duration	Completion time of a given query

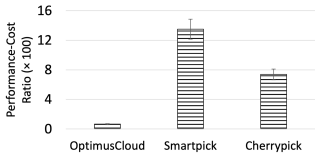


Figure 2. Comparison with known resource determination techniques (higher is better)

3.2 Why RF + BO is better than others?

Techniques proposed in latent factor collaborative filtering [40], machine learning models [82], online fitting [61], Bayesian optimization [2], sampling [77], and a mix of other tools [17] - work great when the search space involves only one type of compute resource (i.e., VMs). Some recent works utilized RF and BO to predict the workloads, e.g., OptimusCloud [47] uses RF and CherryPick [2] uses BO. Since they considered a single instance type as compute resource, they may simply add SLs as a new instance type in order to incorporate them. This approach, however, will lead to a huge search space for optimality, which **cannot** be traversed in a timely and cost-efficient way as they use RF and BO separately. To understand the benefits of the RF + BO approach, we tune our prediction model for OptimusCloud (RF-only) and CherryPick (BO-only) to incorporate both VMs and SLs. To compare different approaches, i.e., RF-only, BO-only, and RF + BO, we use performance-cost ratio (PC_r) [84] that can be computed as shown in Equation 3. Here, *Time* denotes the inference latency, whereas *cost* denotes the compute charges incurred for model creation.

$$PC_r = \frac{1/Time}{1 + cost} \tag{3}$$

We put same inputs (features) to each prediction model 10 times to see how each model works. Figure 2 shows our preliminary simulation results that is scaled to a multiple of 100 (higher is better). It is evident that OptimusCloud [47] gives the worst PC_r value because of the large overhead arising from search complexity. Moreover, CherryPick [2] has better search complexity because of the surrogate design (of BO) but incurs a higher cost from the projected execution runs on live VM and SL instances. Overall, we observed the best PC_r values for Smartpick since it not only reduces the search time complexity but also incurs a lower cost from the enhanced *RF + BO* approach.

3.3 Optimal Configurations with Preferences

Although optimal resource determination leads to minimum query latency, this may not be feasible for some applications that are sensitive to budget requirements. For these applications, some additional query latency would be tolerable for reducing operational cost, i.e., cost-performance tradeoff. Therefore, Smartpick supports a cost-performance tradeoff knob (ϵ) that can be tuned as per the application's target cost-performance goals. Given the knob, Smartpick may proportionally scale down the determined SLs and VMs. For example, setting the ϵ value to 0.5 halves the numbers of SL and VM instances from the optimal configurations determined for best performance. While this approach is simple, we observed that this would lead to significantly high query completion times without a smoother navigation of cost-performance tradeoff.

Instead, Smartpick optimizes resource determination based on the tolerance level set i.e., ϵ. Smartpick uses a list of estimated times (ET_l) to track the candidate solutions explored for the final optimum. This list is traversed before the final resource determination to meet desired cost-performance goals. Equation 4 shows the objective function that is modeled for finer and more precise control of tradeoff; $T_{est.}$ is the estimated time under consideration, t_{vm} is the estimated VM time, t_{sl} is the estimated SL time, C_{vm} denotes compute cost per instance of VM, C_{sl} denotes compute cost per instance of SL, C_{best} is the cost value associated with optimal configuration and T_{best} is the optimum time determined by Smartpick.

$$\begin{aligned} \max_{t} \quad & T_{est.}; \; T_{est.} \in ET_l \\ \text{s.t.} \quad & nVM \times t_{vm} \times C_{vm} + nSL \times t_{sl} \times C_{sl} \leq C_{best} \\ & T_{best} \times (\epsilon + 1) \geq T_{est.} \end{aligned} \tag{4}$$

It aims to find higher query estimation times ($T_{est.}$) that is within the specified limits, i.e., tolerable additional latency (2^{nd} constraint), but draws minimum compute cost (1^{st} constraint). For instance, $\epsilon = 0.2$ specifies a tolerance level of 20% above the optimum value (T_{best}), but the actual cost could be lower for a reduced query latency. This is not always guaranteed though and the optimization problem helps ascertain the required values as shown in Section 6.4.

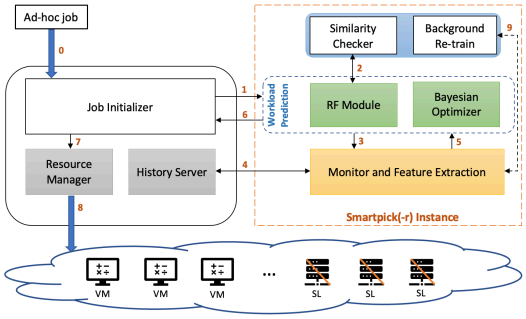


Figure 3. Smartpick Architecture

4 Smartpick Overview

In this section, we present an overview of Smartpick.

4.1 Smartpick Architecture & Workflow

Figure 3 shows the Smartpick architecture in which the numerical values show the order of query execution when a new query is sent to Smartpick.

- The workload prediction (WP) component comprises two sub-modules, i.e., RF and BO, that work together to efficiently explore the large search space as discussed in Section 3.
- Similarity Checker (SC) parses the alien (unknown) queries for extracting meaningful information such as the number of tables, columns, and subqueries inferred in the request.
- Monitor and Feature Extraction (MFE) monitors job execution, and maintains a trained RF model and query features.
- History Server (HS) captures and stores the metrics outlined in Table 3.
- Resource Manager (RM) spawns and manages SL and VM instances based on optimal compute resource configurations.
- Background Re-train creates a new model when the current model is outdated due to workload dynamics.

Workflow: When a new query is received (step 0), *Job Initializer* (JI) asks WP to determine the optimal number of VMs and SLs required for the job (step 1). To efficiently predict query workload, WP maintains a list of queries against which the current operating model is trained. If WP realizes that the incoming query is not in the queries list, i.e., unknown query, WP asks the SC to find the closest query/workload (in step 2). To determine optimal configurations, WP needs a trained RF model and query features as inputs except for *instances* and *query-duration*, as explained in Section 3.1. WP acquires these inputs from MFE (step 3) that pulls historical data from the History Server (step 4). When all the inputs are available (step 5), WP can determine the optimal number of SLs and VMs. If the cost-performance tradeoff knob (ϵ) is set to greater than 0, WP iterates the Estimated Time list (or ET_l)

to find a configuration that meets the cost-performance goal as explained in Section 3.3. From our evaluation, WP can determine compute configuration asynchronously (without blocking the Spark [13] execution flow) within 1.5 seconds for a known query and less than 2.5 seconds for an unknown (alien) query. We assume that this overhead is ignorable as we consider queries that take several tens of seconds. WP returns the resource requirements of incoming query to JI (in step 6). JI asks RM to spawn VMs and SLs based on the determination (step 7). RM spawns the desired number of VMs/SLs on the chosen cloud provider (step 8), following which the query execution begins. If the prediction error in query execution (examined by MFE on job completion in step 9) is higher than the threshold, the prediction model is retrained by Background Re-train.

4.2 Handling Dynamics

Workload dynamics could occur due to several reasons. For example, data analytics applications may need to write new queries to meet their needs [39]. In addition, applications on the cloud store data in enormous volumes for decision-making and health checks [53], i.e., increased data size. Smartpick is designed to handle such dynamics automatically.

Similarity check for alien queries: Determining compute resources for alien queries is challenging since the prediction model is completely unaware of their resource needs. To make a reasonably accurate prediction for such unknown queries, Smartpick maintains the known queries' identifiers and their attributes, such as the number of tables, columns, subqueries, and map tasks. When queries are sent, Smartpick extracts these attributes from the incoming queries and computes the *spatial cosine similarity* to search for the closest known-query identifier. This reference identifier, along with other inputs (as discussed in Section 3.1), is then used to deduce the request's resource-needs. We will show that Smartpick with similarity can help achieve good performance with reduced cost for similar yet alien queries in Section 6.5.1.

Retraining prediction models: While Similarity Checker works well for alike queries, it does not account for workloads that are completely different from the trained queries. Thus, in the event of new/changed workloads, that is, when the accuracy is below an acceptable threshold, we need to retrain the prediction model. To achieve this, Smartpick monitors the difference between actual- and predicted- query execution time. If the difference is greater than a specified threshold, then Smartpick will spawn an asynchronous model retraining task that will re-tune the prediction models (in background) for handling dynamics. In addition, this re-training needs to be highly configurable so that any application with specific needs can reap the maximum benefits out of it. We will discuss these configurable options in detail in Section 5.

Table 4. Smartpick Properties

Key	Default Value
smartpick.cloud.*compute.provider*	AWS
smartpick.cloud.*compute.instanceFamily*	t3
smartpick.cloud.*compute.relay*	True
smartpick.cloud.*compute.knob*	0
smartpick.train.*max.batch*	100
smartpick.train.*pref.sameInstance*	False
smartpick.train.*min.ram.gb*	4
smartpick.train.*errorDifference.trigger*	50

4.3 Relay Instances

To reap the benefits from the hybrid approach, i.e., SL + VM, they should be used in coordination. This is because utilizing SL instances until when a query is completed may incur an additional cost with little (or no) performance improvement due to SL's more expensive cost and worse performance than VM, as discussed in Section 1. To avoid this, Smartpick uses a simple but efficient mechanism, *relay-instances*, with which the SL instances start running the tasks quickly, and will be terminated when corresponding VMs are ready for the rest of the tasks. That is, SLs are only used during the VM's cold-boot time, and then terminated to maximize the benefits of the hybrid approach, i.e., agility from SL and better performance with reduced cost from VM. Consequently, Smartpick's prediction model incorporates the relay-instances mechanism, and thus, the VM and SL resources determined (which may be unequal but optimal) account for these relaying workloads.

SplitServe [31] offers a similar approach, called segueing. However, their approach relies on a static threshold to terminate SLs, which may be costly with limited performance improvement. In addition, they use the same numbers SL and VM, which may not be optimal for a query. For example, SLs can be idle during the static timeout in segueing, which inflates overall cost significantly with limited performance improvement. We present the benefits of relay instances and cost-performance comparison between relay instances and segueing in Section 6.3.

5 Smartpick Implementation

Smartpick is implemented on top of Spark 2.2.1 [13]. Table 4 shows Smartpick's properties that applications can easily set. Spark applications can easily utilize Smartpick by setting these properties without any modification. We will explain each property from the following explanation in detail. Most components in Smartpick are implemented in Python 3.0 [66] if not otherwise specified.

Workload prediction module: We designed and implemented the workload prediction module as a separate process (server) using Thrift RPC [14]. Thus, other SEDA systems can get benefits from Smartpick, i.e., workload prediction and the cost-performance tradeoff feature. We will show how

two recent SEDA systems, i.e., Cocoa and Smartpick, utilize Smartpick as an external prediction system in Section 6.3.2.

Training prediction model: To kick-start Smartpick, the first model training is invoked through a CLI (Command Line Interface) script, tailor-made to initialize and create models from scratch. When a prediction model needs to be trained either initially or in handling dynamics, we devise a heuristic to vary each training sample in the range of ± 5% and create a reasonable dataset comprising around 10x samples (x being the original size). This task ensures that Smartpick can function quickly and effectively with as small as 100 representational workloads. Finally, the data burst is preceded and succeeded by random shuffling so that eventually, when the entire dataset is split into training and test sets, an unbiased selection is performed [55].

Optimal cloud configurations: To determine the optimal cloud configuration with the prediction, *compute.knob* can be set. If the best performance is preferred regardless of cost, it can be set to 0. Or it can be set any greater number than 0 to explore the cost-performance tradeoff space discussed in Section 3.3. Applications can set *compute.instanceFamily* property to increase memory locality for further performance improvement, as discussed in Section 7.

Query similarity check: To parse the alien queries, the similarity checker (SC) uses the sql-metadata library [65], which helps extract meaningful information such as the number of tables, columns and subqueries inferred in the request. Next, a 4-dimensional list is computed having all of the features (along with the number of map tasks), followed by the determination of spatial cosine similarity with respect to the known queries that helps filter out the best match. Thus, the closest query identifier is returned to the WP module, which then uses it to deduce the request's resource-needs.

Prediction model updates: Background re-training is necessary when the model is out of course and the predictions deviate from actual values beyond a pre-defined threshold, i.e., *errorDifference.trigger*. An independent monitor thread in the MFE evaluates this condition and if required, creates a new model with *warm_start*, which is built as a pickle object for up-to-date reference. On completion, the monitor replaces this model in the referred directory, and all new workload predictions point to this object. Smartpick allows users to select where the new model will be trained based on user's preferences, i.e., *pref.sameInstance* and *min.ram.gb*. If the same instance re-training is configured (*pref.sameInstance*) and minimum memory (*min.ram.gb*) is available, Smartpick spawns a new sub-process for re-training. Otherwise, a new instance is started and used for this purpose. Smartpick also supports batch-based re-training (batch size given by the key *max.batch*) that works independently to keep the model incrementally up-to-date.

Metrics collection and history server: To capture the metrics outlined in Table 3, Spark's implementation of listener classes (along with the dependent modules) are modified and monitoring data is stored in JSON format. Once this model is in place, any subsequent request for data processing triggers asynchronous system-level events that have no (little) overhead to the ongoing job. The history server provides internal DNS (Domain Name System) as APIs for other components, e.g., MFE, to request and process the targeted metrics.

Managing compute instances: Resource manager (RM) is implemented on JDK 8 [60] using SDK libraries of AWS [7] and Google Cloud [27]. Applications can point to the primary cloud provider by setting a Smartpick property - *compute.provider*. RM communicates with the respective cloud interface and launches the determined numbers of VMs and SLs. Once these instances are up and running, it tracks their charging statuses for statistics on cost monitoring to be used later for performance/cost evaluation.

Relay-instances mechanism: To make the relay-instance mechanism active, the property *compute.relay* can be set to "True". SLs are terminated when relayed VM instances are ready to execute tasks. To this end, RM will use mapping between REQUEST ID (for SL) and INSTANCE ID (for VM) after sending requests to cloud providers. When a VM instance is ready to be used and connects to RM with its INSTANCE ID, RM will find the corresponding target SL (REQUEST ID) using INSTANCE ID and let the task scheduler stop assigning tasks to it. After checking that no task is running on the SL, RM sends a termination message to it.

Cost estimation: To estimate the cost for queries, we modified Spark workers to send instance information such as ID, cloud provider, region, type, storage type, and storage size to the RM when they connect to it. While most information is static, thus hard-coded in the images, IDs are generated dynamically when Smartpick sends requests to cloud providers, e.g., REQUEST ID for SL and INSTANCE ID for VM. To identify each worker, a boot script for VM and a function code for SL acquire these IDs and set them as an environment variable. Using these IDs, Smartpick tracks instances' execution time and calculates overall compute resource cost for queries. Since VM instances are charged only when they are in the "Running" state, Smartpick uses a dedicated thread that checks their statuses. In our implementation, each VM instance uses 8 GB (SSD) storage which is charged per second. While SL does not charge for its volatile storage (2048 MB), the external storage cost, e.g., AWS t3.xlarge or GCP e2-standard-4 for Redis, is added to the total cost if at least one SL instance is running for a query. Note, data transfer within a DC is free of charge in most cloud providers.

6 Evaluation

In this section, we present a detailed discussion of our evaluation to show the efficacy of Smartpick.

6.1 Experimental Setup

Compute resource setting: We deployed Smartpick prototype implementation on live test-beds of AWS [4] (US East region) and GCP (US East region) [25]. On AWS, we use *t3.xlarge* instance (4 vCPUs and 16 GB RAM) for the Spark master, Spark driver, and the external Redis server. For workers that are dynamically deployed at run-time, we use *t3.small* instances (2 vCPUs and 2 GB RAM) for VM and Lambda [3] 2 GB RAM for SL. Note that each Lambda instance provides 2 vCPUs. That is, each VM and SL instance offer the same amount of CPU cores and memory in our evaluation. On GCP, we use a similar compute resource setting to AWS, i.e., *e2-standard-4* (4 vCPUs and 16 GB RAM) for the master, the driver, and the Redis server, and *e2-small* (2 vCPUs and 2 GB RAM) and Function [24] with 2 GB RAM for workers. All experimental results are an average of 10 runs, plotted with 90% confidence intervals. For cost, we use cost information on AWS and GCP web pages for VMs and SLs. We consider storage cost, e.g., local disk storage of VM and external storage (Redis) instance for communication among SLs as explained in Section 5. We also consider burstable costs of $0.05 per vCPU-hour as we use the *t3* instance family. Note that burstable costs of GCP e2-small is free of charge, but users cannot control it.

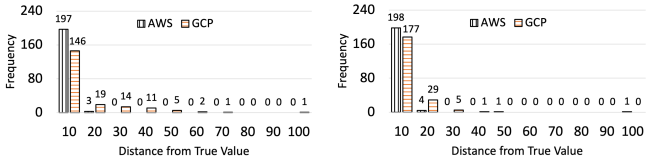
Applications: For workloads to evaluate Smartpick, we use three popular benchmarks, TPC-DS [54], TPC-H [73], and Word Count (WC) [74]. TPC-DS suite comprises compute and I/O intensive workloads with a high number of dependent map and shuffle stages (6 ∼ 16). TPC-H benchmark has SQL-like query benchmarking (moderated compute and I/O) with a lesser sequence of stages (2 ∼ 6). Lastly, we use Word Count as a simple query with I/O requirement. For input data, we generate 100 GB of data in both AWS S3 and Google storage for each benchmark. While we observed similar patterns of results from these benchmarks, we mainly show the results from TPC-DS queries due to space constraints. We use WC and TPC-H benchmarks as new queries to evaluate Smartpick's performance on workload dynamics. In addition, we generate separate 500 GB data for benchmarks to see how Smartpick reacts with changes to data size.

Baselines: We compare Smartpick's hybrid approach with two extreme approaches, i.e., SL-only and VM-only. To mimic VM-only and SL-only approaches, we tweak Smartpick's workload prediction module to choose either SL-only or VM-only for comparison purposes. For the baselines, we compare the Smartpick against two state-of-the-art serverless-enabled data analytics systems, Cocoa [59] and SplitServe [31]. Note that we obtained the source code of Cocoa and SplitServe and integrated them into Smartpick's implementation on Spark for seamless comparisons.

Building Prediction Models: To train the prediction models, we run 20 randomly selected configurations of VMs and SLs for each of the 5 TPC-DS queries i.e., 11, 49, 68, 74, and 82, as representational workloads, short-, mid-, and long-running queries. We generate 1000 data samples, i.e., different SLs + VMs configurations, by the heuristic approach discussed in Section 5. We use 800 samples to build prediction models and 200 samples to evaluate the accuracies of the models (Section 6.2). We build two prediction models, *Smartpick* without relay-instances and *Smartpick-r* with the relay-instances for comparison purpose.

Table 5. Performance comparison between GCP and AWS

Provider	Cloud Storage (MiB/s)	VM I/O (writes/s)	VM I/O (reads/s)	Memory (1k-ops/s)	VM CPU (events/s)	SL CPU (events/s)
AWS	117.53	771.06	1156.59	4675.66	1109.07	811.13
GCP	51.64	764.14	1146.21	4182.49	906.67	714.87



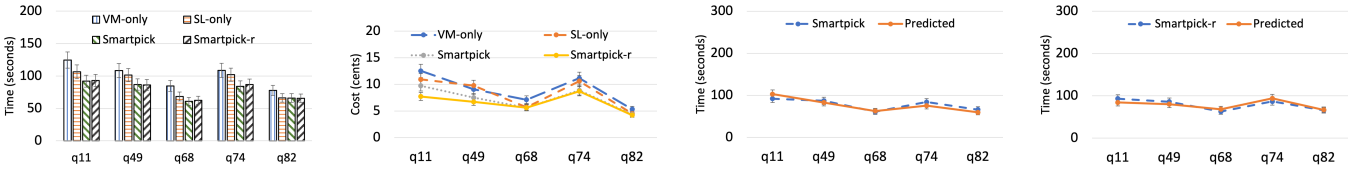
(a) Smartpick (b) Smartpick-r

Figure 4. Accuracy on test dataset

Performance Comparison between AWS and GCP: To clearly understand the experimental results, we first describe the performance difference between AWS and GCP. Table 5 shows benchmark results between AWS and GCP; S3 and Storage for cloud storage, t3.small and e2-small for VM, and Lambda and Function for SL. Both of these VM and SL compute resources have 2 GB memory with dual vCPUs. In order to collect the bandwidth information for Cloud Storage accesses, we upload a 1.5 GB text file onto AWS S3 and GCP Storage and then capture the time taken for download through a Python [66] script. For the remaining measures, we use the Sysbench [42] with identical parameters on both the cloud providers. The table shows that AWS S3 provides better data transfer rate (bandwidth), which can affect overall query performance as input data is read from these cloud storage. For CPU performance on VM, i.e., I/O, Memory, and VM CPU, AWS offers better performance than GCP. We observe that there is no significant difference in the boot-up time of VM as both require 31 ∼ 32 seconds. Similarly, for CPU comparisons on SLs, AWS offers better performance than GCP. Additionally, SL workers on GCP [24] do not have ephemeral storage for source files other than the configured RAM [26], which further reduces the available memory for computation. In summary, the query execution times in GCP are comparably higher than that in AWS, which offers better performance for cloud resources we used in our evaluation.

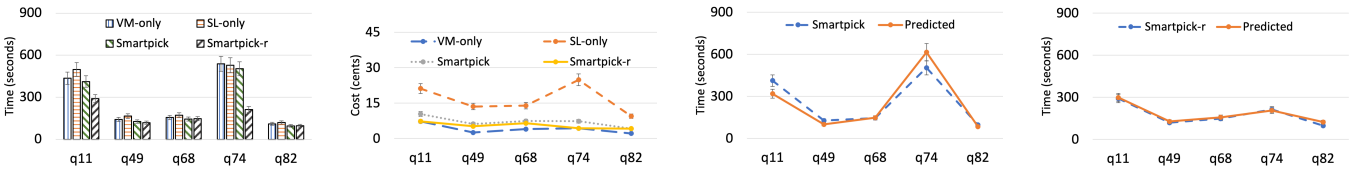
6.2 Workload Prediction

In this experiment, we show how accurately Smartpick and Smartpick-r models predict given queries' workloads with



(a) Performance Comparison (b) Cost Comparison (c) Accuracy of Smartpick (d) Accuracy of Smartpick-r

Figure 5. Evaluation on AWS. (a), (b) - Lower is better. (c), (d) - Compactness is better.



(a) Performance Comparison (b) Cost Comparison (c) Accuracy of Smartpick (d) Accuracy of Smartpick-r

Figure 6. Evaluation on GCP. (a), (b) - Lower is better. (c), (d) - Compactness is better.

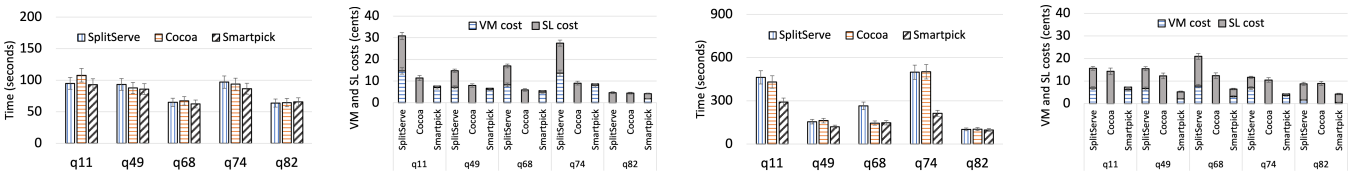
the initial prediction models explained in Section 6.1. We capture different key statistics from the model training phase. First, we see a reasonable Root Mean Squared Error (RMSE) for both the models, i.e., Smartpick and Smartpick-r. On AWS, we get RMSE scores of 6.2 and 8.2 respectively, where as on GCP, we get the same as 12.8 and 7.59 respectively. Based on the extensive statistical analysis, we take 2 times the standard error as an accurate enough prediction, since it considers both the directions of error (positive and negative) [83]. Thus, we plot graphs for each of the above cases by considering the distance from truth values on the test dataset.

Figure 4 shows the frequency of test samples (200/1000 in our experiments with an 80:20 hold-out split for training and testing respectively) at varying distances from the truth values in seconds. It is observed that for Smartpick on AWS, 98.5% of the predicted samples lie within 10 seconds difference of the actual query execution times, which shows that the model yields accurate predictions [83]. Likewise, Smartpick-r provides a prediction accuracy of 97.05% on AWS. Smartpick and Smartpick-r on GCP give prediction accuracies of 73.4% and 83.49%, respectively, which is due to higher query execution time on GCP that incurs more variance. We assume that these results are reliable enough for prediction systems [33, 45, 57, 76]. Besides, the prediction model will become more accurate as Smartpick considers workload dynamics (Section 6.5.2).

6.3 Performance and Cost Comparisons

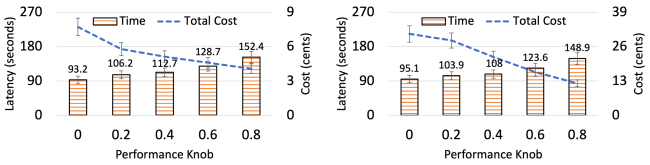
6.3.1 Comparisons with other approaches. In this experiment, we compare the performance of Smartpick and Smartpick-r to two baselines, i.e., VM-only and SL-only approaches. Note that the cost-performance knob (ϵ) in this experiment is set to 0, i.e., the best performance. Figure 5 shows the results on AWS. Figure 5(a) and Figure 5(b) show query completion times and cost, respectively for five TPC-DS queries (11, 49, 68, 74, and 82) with 4 different approaches, i.e., VM-only, SL-only, Smartpick, and Smartpick-r. The results clearly show that both Smartpick models achieve better performance to that of VM-only and SL-only approaches with reduced cost. While we can see similar performance from Smartpick and Smartpick-r, Smartpick-r incurs less cost as expensive SLs are terminated when corresponding VMs are ready, which shows the benefits of the *relay-instances* mechanism. Figure 5(c) and Figure 5(d) show predicted and actual query completion times using Smartpick and Smartpick-r respectively. These figures show that Smartpick can predict given queries' execution times accurately. Figure 6 shows the similar patterns of results on GCP with more variance than AWS due to the different performance characteristics as explained in Section 6.1. For query 49 on GCP, we see a slightly better performance/cost compared to other queries, which is due to the persistent behavior of workload and significantly lesser variance. The VM-only cost on GCP is lower than other approaches as the burstable feature is free of charge on GCP. Overall, Smartpick-r shows better/similar performance with reduced cost compared to other approaches. In the rest of experiments, we use Smartpick to refer to Smartpick-r, unless otherwise mentioned.

6.3.2 Comparisons with State-of-the-art Systems. In this section, we compare Smartpick with state-of-the-art systems, i.e., Cocoa [59] and SplitServe [31]. Since they rely on external workload prediction (WP) systems, we tweak our WP module to choose VM instead of SL + VM, and plug-in the module into Cocoa and SplitServe respectively as we discussed in Section 5. Figure 7 shows the evaluation on AWS



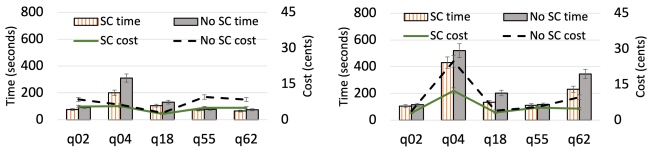
(a) Performance on AWS (b) Cost on AWS (c) Performance on GCP (d) Cost on GCP

Figure 7. Performance and cost comparisons with state-of-the-art (Cocoa and SplitServe). Lower is better.



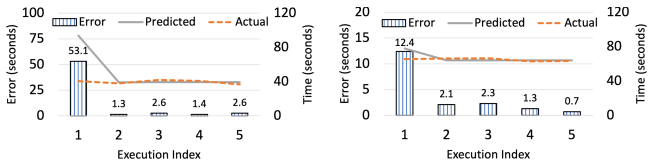
(a) Smartpick (b) SplitServe

Figure 8. Cost-performance tradeoff on AWS



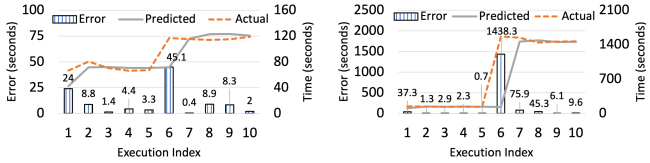
(a) New TPC-DS queries on AWS (b) New TPC-DS queries on GCP

Figure 9. Behavior with new TPC-DS queries



(a) Word Count on AWS (b) Word Count on GCP

Figure 10. Word Count problem on Smartpick



(a) TPC-H on AWS (b) TPC-H on GCP

Figure 11. TPC-H on Smartpick with change in data size

and GCP. We observe that SplitServe tends to give comparable query completion times, but at a high cost (VMs and SLs combined) due to the underlying design of segueing, i.e., the same number of SL and VM, and static timeout threshold for SL, as we discussed in Section 4.3. Similarly, Cocoa gives comparable query completion times, but we see inflated costs for Cocoa as well. This is because Cocoa tends to always favor SLs because of its dependency on other simply assumed static values, such as the execution time for each map/shuffle task, which significantly affects their decisions. Thus, Smartpick can offer better query completion times with much reduced cost than other systems.

6.4 Exploiting cost-performance tradeoff

For applications that have a limited budget, achieving the target performance goal with the minimum cost is an important task, as discussed in Section 3.3. In this experiment, we show how such applications achieve their cost-performance goals using Smartpick's property *compute.knob*. Additionally, systems, e.g., SplitServe [31] that did not account for cost-performance tradeoff, can also benefit from Smartpick's design. Figure 8 shows the behavior of Smartpick and SplitServe (for query 11) with different values of the newly introduced performance knob. As applications increase the value of this knob from 0.2 - 0.8, the cost reduces significantly by trading off the query latency, as discussed in Section 3.3. Figure 8(b) also shows that other systems, e.g., SplitServe, can benefit from Smartpick's feature by exploring the cost-performance tradeoff space. Note that we could see a similar pattern of results from other queries on AWS and GCP, but omitted to cite these results here due to space constraints.

6.5 Handling Dynamics

6.5.1 New Queries from TPC-DS workload. To see how Smartpick handles other queries of TPC-DS, we use the queries 2, 4, 18, 55, and 62, as unknown queries to Smartpick that have similar workloads with the queries used for building prediction models. Figure 9 shows the benefit from Similarity Checker (SC), which helps achieve the best query latency ($\epsilon = 0$) at a reduced cost for all new queries. This highlights the significance of SC module for similar workloads, which was discussed in detail in Section 4.

6.5.2 Handling new workloads and increase in size. One of the key aspects of Smartpick is to handle new queries

by retraining models with the characteristics of new workload. In this section, we use Word Count (WC) as a new workload to Smartpick. Based on the early trials, we observe that same instance re-training leads to an overhead on the ongoing job (which is expected), and therefore, advocate the use of different instance re-training (unless required otherwise). To trigger the model retraining, we set *errorDifference.trigger* to 10. That is, if the difference between actual query execution time and predicted time is more than 10 seconds, then model retraining is triggered. When the new query is submitted at first, Similarity Checker is invoked for each unknown query and the job proceeds to termination based on the closest match as discussed in Section 4.2. Upon job termination, an independent monitor thread triggers background re-training if the difference in predicted and actual values is higher than the configured threshold (*errorDifference.trigger*). Figure 10 shows that Smartpick's prediction model quickly converges to new values by efficient (data-burst based) re-training, as discussed in Section 4.2.

Another important aspect of handling dynamics is the change in workload size. We follow the same set-up as above, but instead use TPC-H query 3 workload as an alien query. In addition, after 5 executions, we change the database to point to a larger size of 500 GB and clean the event logs for existing query. While such significant changes may be rare in real environment, the dataset size grows eventually and consistently with increasing use of the application. Figure 11 shows the results observed for query 3. Clearly, when the data size shoots up, Smartpick can capture this change and quickly converges to the actual execution times. This support of handling dynamics asynchronously and quickly makes Smartpick a robust application with enhanced reliability even in the presence of workload dynamics. Note that the larger spike in the case of GCP is because of the slowness of cloud resources (as discussed in Section 6.1), which is further aggravated by the large input data size of 500 GB.

7 Related Work

Exploiting SL and VM together: LIBRA [68], aims to reduce the cost of hybrid deployments by utilizing cost indifference point, though actual costs can vary depending on the granularity of estimated completion time, where Smartpick comes into play. Cocoa [59] depends on static parameters and does not support relaying of workloads from SLs to VMs, which results in inflated cost. While SplitServe [31] incorporates segueing from SLs to VMs, it results in cost inflation due to its design. It also demands the end-user to employ a cost manager for determining the additional SL resources, which is burdensome work. SplitServe [31], MArk [84], FEAT [58] and Spock [29] aim at reactively launching the SL instances whenever free cores are unavailable. Conversely, Smartpick's resource determination scheme optimizes the choice of VMs and SLs together while meeting cost-performance goals.

Workload prediction for compute resource configurations: Numerous prior works [1, 2, 17, 21, 40, 47, 61, 70, 77, 82] have proposed methodical workload prediction schemes that help determine resource configurations for VM-based workloads. Adding SLs to the supported compute instance types leads to a huge search space for optimality and thus, renders these techniques time-consuming and ineffective. Interestingly, PerfOrator [67] uses hardware-level statistics to build performance model of big data queries, whereas Smartpick requires no advance knowledge of hardware settings and even supports the hybrid model of SLs and VMs.

Handling dynamics: CherryPick [2] relies solely on the BO model to incorporate cloud uncertainties into the decision-making. This works fine for VM instance families but is not well suited to the hybrid approach for ad-hoc alien queries. Jockey [22], Morpheus [35], and ARIA [78] dynamically tune resource allocations (based on historical data) to ensure time-critical jobs with stringent SLOs are provided with required compute resources. They are, however silent on types of compute resources and do not consider the cold boot-up time of VMs. Conversely, Optimus [61] does not depend on the historical information and imposes a checkpoint-inspired technique to handle changes in parameter servers, which can lead to a huge overhead due to multiple restarts. Quasar [20] updates its (VM) resource allocation approach based on active monitoring and sensitivity of the application's performance. Smartpick, instead, can handle unknown requests by employing spatial cosine similarity and course-grained dynamics, as shown in Section 4.2.

Enhancing memory locality: Many serverless-enabled data analytics systems [28, 31, 34, 38, 41, 64, 71, 72] have utilized external storage systems, such as Redis and AWS S3, to avoid SL's limitation, i.e., limited network. However, this may naturally cause performance degradation due to losing data (memory) locality. Some recent works [18, 52, 79, 80] showed that SL instances can communicate with each other directly using TCP hole punching and socket-related library replacement. We expect that using such techniques would improve performance for diverse queries, especially short-running queries. We plan to apply these techniques in Smartpick for further performance improvement without additional cost. To improve memory locality, we also consider using larger (expensive) VM instance types (and families). We could observe that applications can improve performance with additional cost by using larger VM instance family, e.g., AWS c3, which opens another richer tradeoff space. However, we omitted this result due to space constraints.

8 Conclusion

In this paper, we present Smartpick, a scalable data analytics system that determines optimal compute resource configurations for given queries by predicting workloads with consideration of hybrid compute resources, i.e., SL and VM.

Smartpick utilizes decision-tree based Random Forest to predict workloads and Bayesian Optimizer to efficiently explore the large search space for determining optimal configurations. Smartpick is mindful of cost-performance tradeoff space opened by exploiting SL and VM together, and incorporates workload dynamics. Experimental results on AWS and GCP indicate high-precision resource determination for Smartpick with prediction accuracies of 97.05% and 83.49% respectively. The results confirm that Smartpick enables applications to achieve their target cost-performance goals, handle workload dynamics automatically, and improve performance without additional cost compared to baselines. The results also show that other data analytics systems can benefit from Smartpick.

References

[1] Hani Al-Sayeh, Bunjamin Memishi, Muhammad Attahir Jibril, Marcus Paradies, and Kai-Uwe Sattler. 2022. Juggler: Autonomous Cost Optimization and Performance Prediction of Big Data Applications. In *Proceedings of the 2022 International Conference on Management of Data* (Philadelphia, PA, USA) *(SIGMOD '22)*. Association for Computing Machinery, New York, NY, USA, 1840–1854. https://doi.org/10.1145/3514221.3517892

[2] Omid Alipourfard, Hongqiang Harry Liu, Jianshu Chen, Shivaram Venkataraman, Minlan Yu, and Ming Zhang. 2017. CherryPick: Adaptively Unearthing the Best Cloud Configurations for Big Data Analytics. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*. USENIX Association, Boston, MA, 469–482. https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/alipourfard

[3] Amazon AWS. [n.d.]. https://aws.amazon.com/lambda/.

[4] Amazon AWS. [n.d.]. https://aws.amazon.com/.

[5] Amazon AWS. [n.d.]. https://aws.amazon.com/ec2/instance-types/t3/.

[6] Amazon AWS. [n.d.]. https://aws.amazon.com/lambda/pricing/.

[7] Amazon AWS. [n.d.]. AWS SDK for Java. https://aws.amazon.com/sdk-for-java/.

[8] Amazon Simple Storage Service. [n.d.]. https://aws.amazon.com/s3/.

[9] Ganesh Ananthanarayanan, Ali Ghodsi, Scott Shenker, and Ion Stoica. 2011. Disk-locality in datacenter computing considered irrelevant.. In *HotOS*, Vol. 13. 12–12.

[10] Omer Anisfeld, Erez Biton, Ruven Milshtein, Mark Shifrin, and Omer Gurewitz. 2018. Scaling of Cloud Resources-Principal Component Analysis and Random Forest Approach. In *2018 IEEE International Conference on the Science of Electrical Engineering in Israel (ICSEE)*. 1–5. https://doi.org/10.1109/ICSEE.2018.8646134

[11] Apache Hadoop. [n.d.]. https://hadoop.apache.org/.

[12] Apache OpenWhisk. [n.d.]. http://https://openwhisk.apache.org/.

[13] Apache Spark. [n.d.]. https://spark.apache.org/docs/2.2.1/.

[14] Apache Thrift. [n.d.]. https://thrift.apache.org/.

[15] Avantika Monnappa. 2022. https://www.simplilearn.com/how-facebook-is-using-big-data-article.

[16] Janki Bhimani, Ningfang Mi, Miriam Leeser, and Zhengyu Yang. 2017. FiM: Performance Prediction Model for Parallel Computation in Iterative Data Processing Applications. https://doi.org/10.1109/CLOUD.2017.53

[17] Muhammad Bilal, Marco Canini, and Rodrigo Rodrigues. 2020. Finding the Right Cloud Configuration for Analytics Clusters. In *Proceedings of the 11th ACM Symposium on Cloud Computing* (Virtual Event, USA) *(SoCC '20)*. Association for Computing Machinery, New York, NY, USA, 208–222. https://doi.org/10.1145/3419111.3421305

[18] Roman Böhringer. 2022. *FMI: The FaaS Message Interface.* Master's thesis. ETH Zurich.

[19] Branka Vuleta. 2021. https://seedscientific.com/how-much-data-is-created-every-day/.

[20] Christina Delimitrou and Christos Kozyrakis. 2014. Quasar: Resource-Efficient and QoS-Aware Cluster Management. *SIGPLAN Not.* 49, 4 (feb 2014), 127–144. https://doi.org/10.1145/2644865.2541941

[21] Wei Fang, ZhiHui Lu, Jie Wu, and ZhenYin Cao. 2012. RPPS: A Novel Resource Prediction and Provisioning Scheme in Cloud Data Center. In *2012 IEEE Ninth International Conference on Services Computing*. 609–616. https://doi.org/10.1109/SCC.2012.47

[22] Andrew D. Ferguson, Peter Bodik, Srikanth Kandula, Eric Boutin, and Rodrigo Fonseca. 2012. Jockey: Guaranteed Job Latency in Data Parallel Clusters. In *Proceedings of the 7th ACM European Conference on Computer Systems* (Bern, Switzerland) *(EuroSys '12)*. Association for Computing Machinery, New York, NY, USA, 99–112. https://doi.org/10.1145/2168836.2168847

[23] Soumi Ghosh and Chandan Banerjee. 2020. A Predictive Analysis Model of Customer Purchase Behavior using Modified Random Forest Algorithm in Cloud Environment. In *2020 IEEE 1st International Conference for Convergence in Engineering (ICCE)*. 239–244. https://doi.org/10.1109/ICCE50343.2020.9290700

[24] Google Cloud. [n.d.]. https://cloud.google.com/functions.

[25] Google Cloud. [n.d.]. https://cloud.google.com/.

[26] Google Cloud. [n.d.]. https://cloud.google.com/functions/docs/concepts/execution-environment#file_system.

[27] Google Cloud. [n.d.]. Java Cloud Client Libraries. https://cloud.google.com/java/docs/reference/.

[28] J. R. Gunasekaran, P. Thinakaran, M. T. Kandemir, B. Urgaonkar, G. Kesidis, and C. Das. 2019. Spock: Exploiting Serverless Functions for SLO and Cost Aware Resource Procurement in Public Cloud. In *2019 IEEE 12th International Conference on Cloud Computing (CLOUD)*. 199–208.

[29] Jashwant Raj Gunasekaran, Prashanth Thinakaran, Mahmut Taylan Kandemir, Bhuvan Urgaonkar, George Kesidis, and Chita Das. 2019. Spock: Exploiting Serverless Functions for SLO and Cost Aware Resource Procurement in Public Cloud. In *2019 IEEE 12th International Conference on Cloud Computing (CLOUD)*. 199–208. https://doi.org/10.1109/CLOUD.2019.00043

[30] Jianwei Hao, Ting Jiang, Wei Wang, and In Kee Kim. 2021. An Empirical Analysis of VM Startup Times in Public IaaS Clouds: An Extended Report.

[31] Aman Jain, Ata F. Baarzi, George Kesidis, Bhuvan Urgaonkar, Nader Alfares, and Mahmut Kandemir. 2020. SplitServe: Efficiently Splitting Apache Spark Jobs Across FaaS and IaaS. In *Proceedings of the 21st International Middleware Conference* (Delft, Netherlands) *(Middleware '20)*. Association for Computing Machinery, New York, NY, USA, 236–250. https://doi.org/10.1145/3423211.3425695

[32] Tammy Jiang, Jaimie L. Gradus, and Anthony J. Rosellini. 2020. Supervised Machine Learning: A Brief Primer. *Behavior Therapy* 51, 5 (2020), 675–687. https://doi.org/10.1016/j.beth.2020.05.002

[33] Anshul Jindal, Mohak Chadha, Shajulin Benedict, and Michael Gerndt. 2021. Estimating the Capacities of Function-as-a-Service Functions. In *Proceedings of the 14th IEEE/ACM International Conference on Utility and Cloud Computing Companion* (Leicester, United Kingdom) *(UCC '21)*. Association for Computing Machinery, New York, NY, USA, Article 19, 8 pages. https://doi.org/10.1145/3492323.3495628

[34] Eric Jonas, Qifan Pu, Shivaram Venkataraman, Ion Stoica, and Benjamin Recht. 2017. Occupy the Cloud: Distributed Computing for the 99%. In *Proceedings of the 2017 Symposium on Cloud Computing*. Association for Computing Machinery, New York, NY, USA, 445–451. https://doi.org/10.1145/3127479.3128601

[35] Sangeetha Abdu Jyothi, Carlo Curino, Ishai Menache, Shravan Matthur Narayanamurthy, Alexey Tumanov, Jonathan Yaniv, Ruslan Mavlyutov,

Íñigo Goiri, Subru Krishnan, Janardhan Kulkarni, and Sriram Rao. 2016. Morpheus: Towards Automated SLOs for Enterprise Clusters. In *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation* (Savannah, GA, USA) *(OSDI'16)*. USENIX Association, USA, 117–134.

[36] Takuya Kanazawa. 2021. One-parameter family of acquisition functions for efficient global optimization. https://doi.org/10.48550/ARXIV.2104.12363

[37] Veena Khandelwal, Anand Kishore Chaturvedi, and Chandra Prakash Gupta. 2020. Amazon EC2 Spot Price Prediction Using Regression Random Forests. *IEEE Transactions on Cloud Computing* 8, 1 (2020), 59–72. https://doi.org/10.1109/TCC.2017.2780159

[38] Y. Kim and J. Lin. 2018. Serverless Data Analytics with Flint. In *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*. 451–455.

[39] Gabriela Kiryakova, Nadezhda Angelova, and Lina Yordanova. 2015. Application of cloud computing services in business. *Trakia Journal of Science* 13 (01 2015), 392–396. https://doi.org/10.15547/tjs.2015.s.01.067

[40] Ana Klimovic, Heiner Litz, and Christos Kozyrakis. 2018. Selecta: Heterogeneous Cloud Storage Configuration for Data Analytics. In *Proceedings of the 2018 USENIX Conference on Usenix Annual Technical Conference* (Boston, MA, USA) *(USENIX ATC '18)*. USENIX Association, Berkeley, CA, USA, 759–773. http://dl.acm.org/citation.cfm?id=3277355.3277429

[41] Ana Klimovic, Yawen Wang, Patrick Stuedi, Animesh Trivedi, Jonas Pfefferle, and Christos Kozyrakis. 2018. Pocket: Elastic Ephemeral Storage for Serverless Analytics. In *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation* (Carlsbad, CA, USA) *(OSDI'18)*. USENIX Association, Berkeley, CA, USA, 427–444. http://dl.acm.org/citation.cfm?id=3291168.3291200

[42] Alexey Kopytov. 2021. sysbench. https://github.com/akopytov/sysbench.

[43] J. Kuhlenkamp, S. Werner, and S. Tai. 2020. The Ifs and Buts of Less is More: A Serverless Computing Reality Check. In *2020 IEEE International Conference on Cloud Engineering (IC2E)*. 154–161. https://doi.org/10.1109/IC2E48712.2020.00023

[44] Hyungro Lee, Kumar Satyam, and Geoffrey Fox. 2018. Evaluation of Production Serverless Computing Environments. In *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*. 442–450. https://doi.org/10.1109/CLOUD.2018.00062

[45] Yao Lu, John Panneerselvam, Lu Liu, and Yan Wu. 2016. RVLBPNN: A workload forecasting model for smart cloud computing. *Scientific Programming* 2016 (11 2016), 1–9. https://doi.org/10.1155/2016/5635673

[46] Lu Zhang and Diuto Malife. [n.d.]. https://blog.twitter.com/engineering/en_us/\topics/infrastructure/2021/processing-billions-of-events-in-real-time-at-twitter-.

[47] Ashraf Mahgoub, Alexander Michaelson Medoff, Rakesh Kumar, Subrata Mitra, Ana Klimovic, Somali Chaterji, and Saurabh Bagchi. 2020. OPTIMUSCLOUD: Heterogeneous Configuration Optimization for Distributed Databases in the Cloud. In *2020 USENIX Annual Technical Conference (USENIX ATC 20)*. USENIX Association, 189–203. https://www.usenix.org/conference/atc20/presentation/mahgoub

[48] Ming Mao and Marty Humphrey. 2012. A Performance Study on the VM Startup Time in the Cloud. In *2012 IEEE Fifth International Conference on Cloud Computing*. 423–430. https://doi.org/10.1109/CLOUD.2012.103

[49] MathWorks. 2022. Bayesian Optimization Algorithm. https://www.mathworks.com/help/stats/bayesian-optimization-algorithm.html.

[50] Avinash Mehta, Mukesh Menaria, Sanket Dangi, and Shrisha Rao. 2011. Energy conservation in cloud infrastructures. In *2011 IEEE International Systems Conference*. 456–460. https://doi.org/10.1109/SYSCON.2011.5929050

[51] Microsoft Azure. [n.d.]. https://azure.microsoft.com/en-us/services/functions/.

[52] Daniel William Moyer. 2021. *Punching Holes in the Cloud: Direct Communication between Serverless Functions Using NAT Traversal*. Ph.D. Dissertation. Virginia Tech.

[53] Manoj Muniswamaiah, Tilak Agerwala, and Charles Tappert. 2019. Big Data in Cloud Computing Review and Opportunities. *International Journal of Computer Science and Information Technology* 11, 4 (aug 2019), 43–57. https://doi.org/10.5121/ijcsit.2019.11404

[54] Raghunath Othayoth Nambiar and Meikel Poess. 2006. The Making of TPC-DS. In *Proceedings of the 32nd International Conference on Very Large Data Bases* (Seoul, Korea) *(VLDB '06)*. VLDB Endowment, 1049–1058.

[55] Thao Truong Nguyen, François Trahay, Jens Domke, Aleksandr Drozd, Emil Vatai, Jianwei Liao, Mohamed Wahib, and Balazs Gerofi. 2022. Why globally re-shuffle? Revisiting data shuffling in large scale deep learning. In *IPDPS 2022: 36th International Parallel & Distributed Processing Symposium*. IEEE, Lyon (virtual), France. https://hal.archives-ouvertes.fr/hal-03599740

[56] Rodolphe Le Riche Nicolas Durrande. 2017. *Introduction to Gaussian Process Surrogate Models*.

[57] Farhan Nisar, Samad Baseer, and Arshad khan. 2019. Survey on ARIMA Model Workloads in a DataCenter with respect to Cloud Architecture. In *2019 International Symposium on Recent Advances in Electrical Engineering (RAEE)*, Vol. 4. 1–4. https://doi.org/10.1109/RAEE.2019.8887075

[58] Joe H. Novak, Sneha Kumar Kasera, and Ryan Stutsman. 2019. Cloud Functions for Fast and Robust Resource Auto-Scaling. In *2019 11th International Conference on Communication Systems & Networks (COMSNETS)*. 133–140. https://doi.org/10.1109/COMSNETS.2019.8711058

[59] Kwangsung Oh and Myoungkyu Song. 2021. Cocoa: Towards a Scalable Compute Cost-aware Data Analytics System. In *2021 IEEE International Conference on Cloud Engineering (IC2E)*. 110–117. https://doi.org/10.1109/IC2E52221.2021.00025

[60] OpenJDK. [n.d.]. JDK 8. https://openjdk.org/projects/jdk8/.

[61] Yanghua Peng, Yixin Bao, Yangrui Chen, Chuan Wu, and Chuanxiong Guo. 2018. Optimus: An Efficient Dynamic Resource Scheduler for Deep Learning Clusters. In *Proceedings of the Thirteenth EuroSys Conference* (Porto, Portugal) *(EuroSys '18)*. Association for Computing Machinery, New York, NY, USA, Article 3, 14 pages. https://doi.org/10.1145/3190508.3190517

[62] Meikel Poess, Raghunath Othayoth Nambiar, and David Walrath. 2007. Why You Should Run TPC-DS: A Workload Analysis. In *Proceedings of the 33rd International Conference on Very Large Data Bases* (Vienna, Austria) *(VLDB '07)*. VLDB Endowment, 1138–1149.

[63] Meikel Pöss, T. Rabl, and Hans-Arno Jacobsen. 2017. Analysis of TPC-DS: the first standard benchmark for SQL-based big data systems. *Proceedings of the 2017 Symposium on Cloud Computing*.

[64] Qifan Pu, Shivaram Venkataraman, and Ion Stoica. 2019. Shuffling, Fast and Slow: Scalable Analytics on Serverless Infrastructure. In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*. USENIX Association, Boston, MA, 193–206. https://www.usenix.org/conference/nsdi19/presentation/pu

[65] PyPI. [n.d.]. sql-metadata. https://pypi.org/project/sql-metadata/.

[66] Python. [n.d.]. https://www.python.org/.

[67] Kaushik Rajan, Dharmesh Kakadia, Carlo Curino, and Subru Krishnan. 2016. PerfOrator: Eloquent Performance Models for Resource Optimization. In *Proceedings of the Seventh ACM Symposium on Cloud Computing* (Santa Clara, CA, USA) *(SoCC '16)*. Association for Computing Machinery, New York, NY, USA, 415–427. https://doi.org/10.1145/2987550.2987566

[68] A. Raza, Z. Zhang, N. Akhtar, V. Isahagian, and I. Matta. 2021. LIBRA: An Economical Hybrid Approach for Cloud Applications with Strict SLAs. In *2021 IEEE International Conference on Cloud Engineering (IC2E)*. IEEE Computer Society, Los Alamitos, CA, USA, 136–146. https://doi.org/10.1109/IC2E52221.2021.00028

[69] Redis. [n.d.]. https://redis.io/.

[70] Li Ruan, Yu Bai, Shaoning Li, Shuibing He, and Limin Xiao. 2021. Workload time series prediction in storage systems: a deep learning based approach. *Cluster Computing* (2021), 1–11.

[71] Vaishaal Shankar, Karl Krauth, Qifan Pu, Eric Jonas, Shivaram Venkataraman, Ion Stoica, Benjamin Recht, and Jonathan Ragan-Kelley. 2018. numpywren: serverless linear algebra. arXiv:1810.09679 [cs.DC]

[72] Spark on Lambda. [n.d.]. https://github.com/qubole/spark-on-lambda/.

[73] TPC. [n.d.]. *TPC-H Vesion 2 and Version 3.*

[74] RIP Tutorial. [n.d.]. *Word Count Example in Hive.*

[75] Rafael Brundo Uriarte, Francesco Tiezzi, and Sotirios A. Tsaftaris. 2016. Supporting Autonomic Management of Clouds: Service Clustering With Random Forest. *IEEE Transactions on Network and Service Management* 13, 3 (2016), 595–607. https://doi.org/10.1109/TNSM.2016.2569000

[76] Rafael Valero-Fernandez, David J. Collins, K.P. Lam, Colin Rigby, and James Bailey. 2017. Towards Accurate Predictions of Customer Purchasing Patterns. In *2017 IEEE International Conference on Computer and Information Technology (CIT)*. 157–161. https://doi.org/10.1109/CIT.2017.58

[77] Shivaram Venkataraman, Zongheng Yang, Michael Franklin, Benjamin Recht, and Ion Stoica. 2016. Ernest: Efficient Performance Prediction for Large-Scale Advanced Analytics. In *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*. USENIX Association, Santa Clara, CA, 363–378. https://www.usenix.org/conference/nsdi16/technical-sessions/presentation/venkataraman

[78] Abhishek Verma, Ludmila Cherkasova, and Roy H. Campbell. 2011. ARIA: Automatic Resource Inference and Allocation for Mapreduce Environments. In *Proceedings of the 8th ACM International Conference on Autonomic Computing* (Karlsruhe, Germany) *(ICAC '11)*. Association for Computing Machinery, New York, NY, USA, 235–244. https://doi.org/10.1145/1998582.1998637

[79] Michael Wawrzoniak, Ingo Müller, Rodrigo Bruno, Ana Klimovic, and Gustavo Alonso. 2022. Short-lived Datacenter. https://doi.org/10.48550/ARXIV.2202.06646

[80] Mike Wawrzoniak, Ingo Müller, Rodrigo Fraga Barcelos Paulus Bruno, and Gustavo Alonso. 2021-01. Boxer: Data Analytics on Network-enabled Serverless Platforms. https://doi.org/10.3929/ethz-b-000456492 11th Annual Conference on Innovative Data Systems Research (CIDR 2021); Conference Location: online; Conference Date: January 11-15, 2021; The conference lecture was held on January 12, 2021. Due to the Coronavirus (COVID-19) the conference was conducted virtually.

[81] James T. Wilson, Frank Hutter, and Marc Peter Deisenroth. 2018. Maximizing acquisition functions for Bayesian optimization. https://doi.org/10.48550/ARXIV.1805.10196

[82] Neeraja J. Yadwadkar, Bharath Hariharan, Joseph E. Gonzalez, Burton Smith, and Randy H. Katz. 2017. Selecting the Best VM Across Multiple Public Clouds: A Data-driven Performance Modeling Approach. In *Proceedings of the 2017 Symposium on Cloud Computing* (Santa Clara, California) *(SoCC '17)*. ACM, New York, NY, USA, 452–465. https://doi.org/10.1145/3127479.3131614

[83] Zach. 2019. Understanding the Standard Error of the Regression. https://www.statology.org/standard-error-regression/.

[84] Chengliang Zhang, Minchen Yu, Wei Wang, and Feng Yan. 2019. MArk: Exploiting Cloud Services for Cost-Effective, SLO-Aware Machine Learning Inference Serving. In *2019 USENIX Annual Technical Conference (USENIX ATC 19)*. USENIX Association, Renton, WA, 1049–1062. https://www.usenix.org/conference/atc19/presentation/zhang-chengliang