Adaptive Performance Modeling of Data-intensive Workloads for Resource Provisioning in Virtualized Environment

HOSEIN MOHAMAMDI MAKRANI, University of California, Davis, USA
HOSSEIN SAYADI, California State University, Long Beach, USA
NAJMEH NAZARI, University of Tehran, Iran
SAI MNOJ PUDUKOTAI DINAKARRAO and AVESTA SASAN, George Mason University, USA
TINOOSH MOHSENIN, University of Maryland, Baltimore County, USA
SETAREH RAFATIRAD, George Mason University, USA
HOUMAN HOMAYOUN, University of California, Davis, USA

The processing of data-intensive workloads is a challenging and time-consuming task that often requires massive infrastructure to ensure fast data analysis. The cloud platform is the most popular and powerful scale-out infrastructure to perform big data analytics and eliminate the need to maintain expensive and high-end computing resources at the user side. The performance and the cost of such infrastructure depend on the overall server configuration, such as processor, memory, network, and storage configurations. In addition to the cost of owning or maintaining the hardware, the heterogeneity in the server configuration further expands the selection space, leading to non-convergence. The challenge is further exacerbated by the dependency of the application's performance on the underlying hardware. Despite an increasing interest in resource provisioning, few works have been done to develop accurate and practical models to proactively predict the performance of data-intensive applications corresponding to the server configuration and provision a cost-optimal configuration online.

In this work, through a comprehensive real-system empirical analysis of performance, we address these challenges by introducing ProMLB: a proactive machine-learning-based methodology for resource provisioning. We first characterize diverse types of data-intensive workloads across different types of server architectures. The characterization aids in accurately capture applications' behavior and train a model for prediction of their performance.

Then, ProMLB builds a set of cross-platform performance models for each application. Based on the developed predictive model, ProMLB uses an optimization technique to distinguish close-to-optimal configuration to minimize the product of execution time and cost. Compared to the oracle scheduler, ProMLB achieves 91% accuracy in terms of application-resource matching. On average, ProMLB improves the performance and resource utilization by 42.6% and 41.1%, respectively, compared to baseline scheduler. Moreover, ProMLB improves the performance per cost by 2.5× on average.

Authors' addresses: H. M. Makrani and H. Homayoun, University of California, Davis, Davis, CA; emails: hmakrani@ucdavis.edu, hhomayoun@ucdavis.edu; H. Sayadi, California State University, Long Beach, Long Beach, CA; email: hossein.sayadi@csulb.edu; N. Nazari, University of Tehran, Tehran, Iran; email: Najme.nazari66@gmail.com; Sai Mnoj Pudukotai Dinakarrao, A. Sasan and S. Rafatirad, George Mason University, Fairfax, VA; emails: spudukot@gmu.edu, asasan@gmu.edu, srafatir@gmu.edu; T. Mohsenin, University of Maryland, Baltimore County, Baltimore, MA; email: tinoosh@umbc.edu.

2376-3639/2021/03-ART18 $15.00
https://doi.org/10.1145/3442696

CCS Concepts: • **Computer systems organization** → **Cloud computing**; *Multicore architectures*; • **Theory of computation** → Support vector machines;

Additional Key Words and Phrases: Performance, cloud, optimization, big data

ACM Reference format:
Hosein Mohamamdi Makrani, Hossein Sayadi, Najmeh Nazari, Sai Mnoj Pudukotai Dinakarrao, Avesta Sasan, Tinoosh Mohsenin, Setareh Rafatirad, and Houman Homayoun. 2021. Adaptive Performance Modeling of Data-intensive Workloads for Resource Provisioning in Virtualized Environment. *ACM Trans. Model. Perform. Eval. Comput. Syst.* 5, 4, Article 18 (March 2021), 24 pages.
https://doi.org/10.1145/3442696

1 INTRODUCTION

The continuous increase in the volume of data due to the rise of social media [20], Internet-of-Things, and multimedia has produced an overwhelming flow of data referred to as big data [34, 57]. To efficiently process such massive data, scale-out architectures have gained interest as a promising solution that is designed to provide a massively scalable computer architecture. Recent improvements in the networking, storage, energy efficiency, and infrastructure management have made cloud a preferable approach to respond to the challenges associated with big data [43].

Cloud computing is a significant paradigm shift in service for enterprise applications and has become a powerful platform to perform large-scale computing [11]. The advantages of cloud computing include the virtualized environment, parallel processing, security, and scalable data storage. Some of the first adopters of big data in cloud computing are the users who deployed MapReduce, SQL-like languages, deep learning, and in-memory analytics clusters in computing environments provided by vendors, such as IBM SoftLayer (has been renamed to IBM Cloud), Microsoft Azure, and Amazon AWS. The structure of execution environments of such big data analytics is a cluster of virtual machines (VM) [30].

Virtualization is a process of resource sharing and isolation of underlying hardware to increase computer resource utilization, efficiency, and scalability. Considering the fact that the cloud service providers offer a wide range of cloud configuration choices such as VM instances with a variety of CPUs, memory, disk, and network configurations and also customized VMs for analytics applications, determining the best cloud configuration for a given application by brute-force search is expensive and exhaustive. Choosing the right cloud configuration is essential, as a non-optimal configuration results in more cost for the same performance target as different analytic jobs have diverse behaviors and resource requirements. A more challenging problem regarding big data analytics is that the behavior and resource requirements of these applications vary during different phases of execution [12]. As the application behavior varies with time, a reactive resource allocation methodology achieves a sub-optimal performance gains due to computational complexity and involved latencies. Therefore, a dynamic proactive approach is needed to determine and allocate optimal resources at different phases for a running application.

In response to these challenges, we propose ProMLB-a methodology that proactively predicts the future behavior of running applications by dynamically generating a cross-platform performance model for all the available hardware resources and provisions a near-optimal configuration that maximizes performance per cost, while introducing a low search overhead. The overhead of generating the performance model and finding the optimal configuration is negligible as ProMLB is implemented on a separate server as centralized cluster management, and it does not interfere with on-going application executions.

To the best of our knowledge, previous works did not fully address all the following challenges together: proactive behavior prediction of data-intensive applications, selecting a highly accurate machine learning (ML) technique for performance modeling based on real empirical characterization, consideration of VM live migration time overhead, optimizing the performance w.r.t the cloud cost, and considering various optimization techniques to achieve fairness among jobs at the same time.

In this work, we propose a practical framework called ProMLB that can address all the aforementioned challenges simultaneously. To this goal, we first analyze various applications' architectural characteristics. Based on this information, a database is built and used for training the prediction models (using time series neural network, K nearest neighbors (KNN) regression, and hidden Markov model (HMM)), generating performance models (using multilayer perceptron, and Support Vector Machine (SVM)), and applying different optimization techniques (Knapsack algorithm and Cobb Douglas utility function) in terms of performance/cost efficiency and fairness. To be as close as to the real-world cloud providers, we utilize IBM's SoftLayer pricing list to derive a cost model for server platforms in a scale-out environment. The developed cost model takes into account the processor, memory, and disk configurations.

The novelty of this work is outlined in a threefold manner:

- An Ensemble learning-based proactive phase prediction with high accuracy based on the behavior of the application and underlying execution hardware is devised.
- A non-linear performance model to efficiently estimate the actual behavior of the applications running on different hardware platforms considering the architectural parameters, as well as real-time constraints such as migration time, is deployed.
- A cost-performance tradeoff is achieved by employing the Bounded Knapsack algorithm. To solve the problem of proactive resource allocation in data centers with minimal processing overheads, a hierarchical approach based decision-maker is employed in the last stage.

The evaluation results show that the phase predictor of ProMLB achieves 92% to predict the future phase change of workloads correctly. On average, ProMLB improves the performance/cost (performance per unit cost) and performance by 2.5× and 42% on an average (up to 70%), respectively. It needs to be noted that this improvement in speedup is only achieved by efficient resource allocation and without any software or framework tuning overheads. Based on the evaluation results, ProMLB increases CPU utilization efficiency (averaged across all cores), DRAM bandwidth, and memory capacity utilization efficiency by 36%, 53%, and 39%, respectively.

The remainder of this article is organized as follows: Section 2 reviews the related works and presents our motivation for this study. Section 3 introduces the ProMLB framework and the technical details of various components of ProMLB. Section 4 presents our experimental setup and the implementation of ProMLB. Section 5 presents the performance analysis of ProMLB and compares it with existing works in terms of resource utilization, efficiency, improvement in performance, and performance per unit of cost. Finally, we derive the conclusions in Section 6.

2 MOTIVATION AND BACKGROUND

2.1 Motivation

The number of performance and cost optimization tuning knobs available for data-intensive applications is large compared to traditional CPU applications [CPU2006]. Hence, we consider an application as a generic black-box to simplify the model of the application's behavior for correlating its architectural signature with available hardware resources.

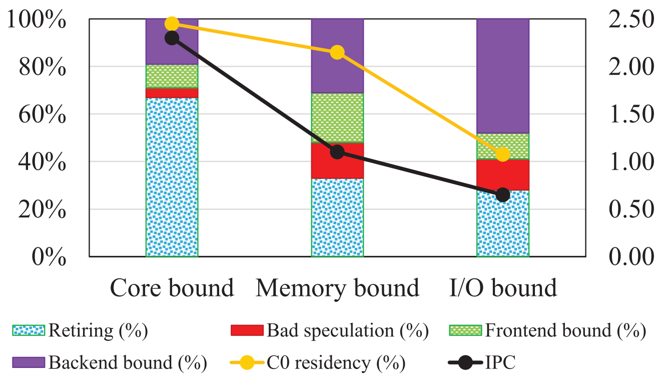

Fig. 1. Micro-Architectural break-down of workloads for different phases.

Based on Top-Down methodology [53], applications' behavior can be classified into three categories such as I/O bound, core bound, and memory bound. Top-Down methodology chooses the micro-op (μop) queue of an out-of-order server as a dividing point between a core's front-end and back-end and uses it to classify μop pipeline slots in four broad categories: Retiring, Front-end bound, Bad speculation, and Back-end bound. Of these, Retiring is classified as "useful work" and the rest prevent the workload from utilizing the full core width. We apply this approach to our big data workloads. An application may transition multiple times between these phases during its execution time. Figure 1 illustrates the microarchitectural differences between those three phases. The main difference between memory bound and I/O bound is in C0 residency. This can be explained as follows: If the application is I/O bound, then the core is waiting for I/O; hence the core changes its state to save power. Therefore, C0 residency drops.

Moreover, micro-architectural information of applications varies corresponding to their behavior, and it can be used as a signature for the applications. In this way, each application has multiple architectural signatures corresponding to its behavior. The architectural signature can be translated into the type of resources required for executing the application. Memory bound applications require more DRAM bandwidth and capacity. Core bound (compute intensive) applications require a high-performance processor with a high number of cores and core frequency. I/O bound applications require fast storage and network. There are hundreds of configurations available in a server farm that each of them can deliver different performance for a given application. The challenging problem to address is that an application may have various phases of execution. For each phase, it requires different resources to meet the performance requirements while maximizing the utilization of resources for the cost benefits.

Figure 2 demonstrate the phase change of PageRank application during a part of its execution. We can observe that there are several changes in the application's behavior. If we ask the programmer to provide the beginning or end of parallel regions or long-running API calls, then applications and frameworks must be redesigned. We designed ProMLB to address the optimal resource allocation problem without requiring any change in the application or the frameworks, and ProMLB is compatible with current frameworks.

2.2 Related Work

The prevalence of cloud computing has motivated several new studies for cluster management [7, 13–15, 32, 50, 54, 55]. A cluster management framework provides various services, including resource efficiency, security, fault tolerance, and monitoring capabilities. The proposed framework in this article, ProMLB, is a novel resource management engine that differs from previous works

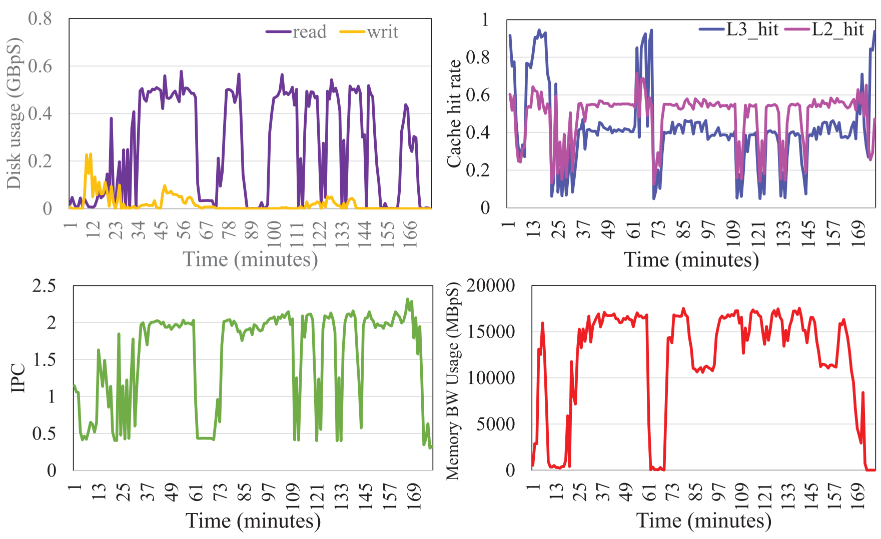

Fig. 2. Example of application's behavior and phase change.

Table 1. Comparison of States of the Art

System	Target	Complexity	Accuracy	Proactive	Dynamic	Domain	Cost aware
ProMLB	Performance/cost, Fairness	High	High	Yes	Yes	Big Data	Yes
BoPF (SIGMETRICS'19)	Fairness	Medium	High	No	No	Big Data	No
DAC (ASPLOS'18)	Performance	High	High	No	No	In-memory	No
PARIS (SoCC'17)	Performance	Medium	Medium	No	Yes	Broad	Yes
CherryPick (NSDI'17)	Performance	Low	Low	No	No	Big Data	No
MeNa (IISWC'17)	Performance/cost	Low	Low	No	No	Broad	Yes
HCloud (ASPLOS'16)	Cost	Medium	Medium	No	Yes	Scale-out	Yes
Ernest (NSDI'16)	Performance	Medium	High	No	No	Big Data	No
Heracles (ISCA'15)	Performance	Low	Medium	No	Yes	Latency-critical	No
Quasar (ASPLOS'14)	Performance	Medium	Medium	No	Yes	Scale-out	No
REF (ASPLOS'14)	Fairness	low	Low	No	Yes	Broad	No
Paragorn (ASPLOS'13)	Performance	Medium	Low	No	Yes	Scale-out	No

in many aspects. Table 1 summarizes the recent works and differentiates ProMLB from state-of-the-art studies.

Several recent studies [7, 13–15, 26, 50, 52] were proposed to address QoS-aware, performance aware, and cost-aware scheduling and resource allocation. One of the closest work to ProMLB is Quasar [14]. Quasar is an online scheduler that leverages historical performance data from scheduled applications to classify incoming applications and assign the application proper resources in a datacenter. It further relies on online adjustments of resource allocations to correct mistakes in the modeling phase in which it randomly samples a few applications, and injects microbenchmarks in the corresponding servers and performs live reclassification. If it detects significant deviations from the previous interference profile of a workload, then it considers whether migration or rescheduling is beneficial. In addition, Quasar does not consider the cost of running servers. Whereas, ProMLB continuously monitors all servers and proactively predicts their phase change ahead of time while considering the cost of the servers. As a result, ProMLB offers a more efficient resource allocation, and since the deployed predictors are lightweight, no high processing overheads are added to the system.

Reference [13] proposed a heterogeneous and interference-aware scheduling system in datacenters called Paragon. It is based on analytical methods that are built using previous information the system already has about applications to exhaustively schedule applications to maximize resource utilization in datacenters. Paragon deploys filtering techniques to identify some similarities between the current workload and the previously scheduled applications, and classify the unknown application based on the level of heterogeneity and interference in multiple shared resources in datacenter. Reference [7] proposed Cherrypick, a system that attempts to find a nearly optimal cloud configurations with high accuracy and low overhead in which it adaptively builds performance models customized for specific applications and cloud servers configurations to identify a near-optimal configuration.

Resource Elasticity Fairness (REF) [55] and BoPF [28] are resource provision methods to schedule a fair set of resources for each user at a computer architecture to cloud level by presenting fair resource allocation mechanisms that customized preferences to determine each user's fair share of the hardware. The researches in Reference [27] proposed Pliant, a lightweight cloud-based approach that employs the incremental and interference-aware approximation during periods of high contention to reduce interference in shared resource and ultimately tolerate some loss in output quality to boost the utilization of shared resources and servers at runtime. Bolt [16] is another research on cloud-based resource provisioning that proposed a scheduler system that leverages data mining techniques to detect the the type and characteristics of running applications on a cloud platform using the interference an adversary observes on the shared resources.

Kousiouris et al. [26] proposed to use a two-layer service in the cloud to translate high-level application parameters (workload and QoS based on Service Level Agreement) to resource level attributes. Their work did not consider any performance model to select the optimum configuration. Also, they have not considered the cost-efficiency.

Some systems adaptively allocate resources based on feedback. Rightscale [6] creates additional VM instances when a load of an application crosses a threshold for EC2. YARN [8] decides resource needs based on requests from the application. Other systems have explicit models to inform the control system. For example, the work in Reference [10] targeted achieving accurate control of the web application, by training the performance models on the production system making the model adapted to runtime changes in workload and performance characteristics of executed program. This work proposed to train the performance model using an exploration policy to collect sufficient data from different performance regions of the running application on the system pushing the system close to its capacity.

Wrangler [51] identifies overloaded nodes in map-reduce clusters and delays scheduling jobs on them. Interference is creating a challenge for accurate performance estimation. Recent works [31, 44] explore placing applications on particular resources to reduce interference, by co-scheduling applications with disjoint resource requirements [31]. However, users requesting VM types in cloud services like Amazon EC2 cannot usually control what applications are co-scheduled. None of these studies have focused on the influence of system parameters such as memory or storage on the performance and cost in the cloud.

There are other works that studied the performance of big data applications on modern processors [35, 37] and performed a set of comprehensive experiments to analysis the impact of memory subsystem on the performance of data intensive applications [33, 36]. Jackson et al. [24] and Barker et al. [9] analyzed high-performance computing (HPC) applications, latency-sensitive applications, scientific applications, and micro-benchmark applications on the cloud. Kanev [25]

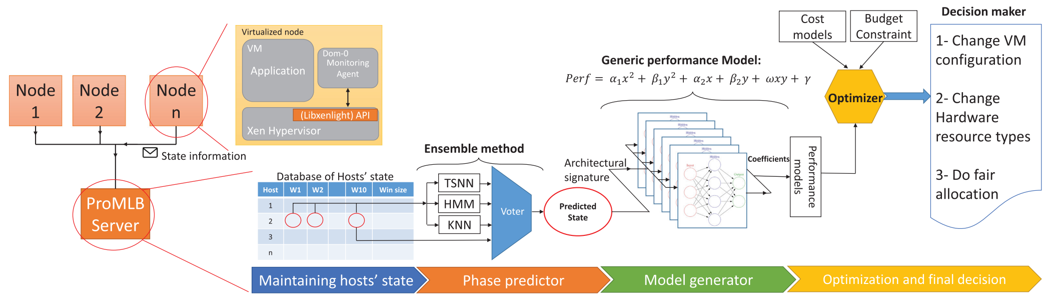

Fig. 3. ProMLB overview.

analyzed cloud-scale workloads to provide infrastructure-level insights. In particular, the authors in this work presented a detailed microarchitectural characterization of different real-world applications in datacenter jobs, collected from Google machines over a three year period. They found that cloud-scale workloads are significantly diverse, highlighting the necessity for computer architectures that can tolerate application variability without performance loss, while some patterns have shown the opportunities for hardware and software co-optimization. Last, Guevara et al. [18] studied the application of heterogeneous architectures for cloud-based workload optimization by exploring how deployment of heterogeneous platforms bring energy-efficiency for cloud applications.

Unlike ProMLB, none of the existing approaches aim at estimating the Performance/Cost of arbitrary data-intensive workloads on various server types at run-time and for each phase of the program to proactively provision the most efficient configuration in a heterogeneous scale-out environment.

3 PROMLB

3.1 Overview

Figure 3 shows the overview of ProMLB framework. To continuously monitor each server's state, a monitoring agent runs on each host. These agents periodically send the host's state, such as architectural information and resource utilization, to the ProMLB server. ProMLB server maintains a database of per-host state and updates it on each interval. ProMLB predicts the future state of application based on the current and previous states. Based on the predicted state and corresponding architectural signature of application, ProMLB generates the application's performance model for all available platforms in the cloud. Afterward, ProMLB solves an optimization problem to find the best platform and configuration that maximize the performance/cost for a specific application at a given budget. Then, ProMLB uses Cobb–Douglas utility function to achieve fair allocation.

ProMLB is designed to maximize the performance per cost of running a data-intensive application on a distributed platform. To achieve this goal, our approach is to use bounded knapsack algorithm. In the Knapsack algorithm, by giving a set of items each with a weight and a value, we must determine the number of each item to include in a collection so that the total weight is less than or equal to a given limit and the total value is as large as possible. In our problem, we consider each resource as an item and their value is the performance gain that they can add to the system. In our problem, the cost of each resource is equal to the weight of an item in the original problem. The given limit is equal to the total budget of the user to provision resources for its application. We have another restriction that from each item, we must select at least a minimum amount. Because an application cannot start execution unless it gets a core, memory, storage, and

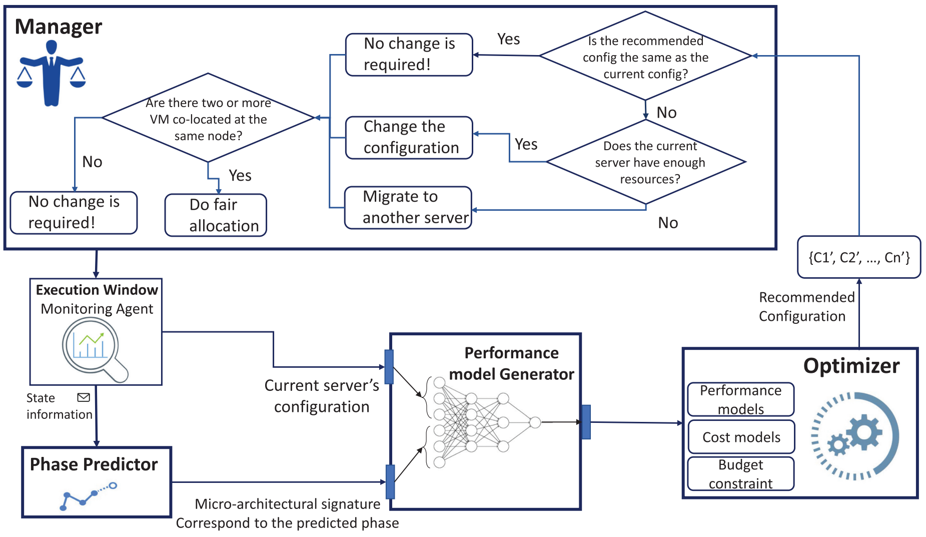
Manager
Are there two or more VM co-located at the same node?
No
Yes
No change is required!
Do fair allocation
No change is required!
Change the configuration
Migrate to another server
Yes
Is the recommended config the same as the current config?
No
Yes
Does the current server have enough resources?
No
{C1', C2', ..., Cn'}
Recommended Configuration
Execution Window Monitoring Agent
State information
Phase Predictor
Current server's configuration
Micro-architectural signature Correspond to the predicted phase
Performance model Generator
Optimizer
Performance models
Cost models
Budget constraint

Fig. 4. Block diagram of ProMLB.

network resources. To solve the optimization problem, we need a performance and cost model to determine the performance gain and the cost of adding each resource to the system.

The important point is that as we must assign all resources at the same time, we need a performance model that correlates the performance of adding a resource to all other resources. For example, if we want to add one core to the system, then the performance gain of that extra core is dependent to the current resources, and it is not independent from them. Therefore, we need performance models to accurately calculate the performance gain. However, the cost of adding each resource is independent from other resources. Therefore, the cost models are much simpler than the performance models. ProMLB server consists of four components to deliver all the above functionalities: Phase predictor, performance model generator, optimizer, and decision-maker.

Figure 4 shows the block diagram of ProMLB and explains how aforementioned components work together, e.g., how the optimizer and the manager take the knowledge of the predicted results and models in their configuration and allocation decisions.

3.2 Monitoring Agent

The monitoring agent has been implemented in a privileged VM in Xen hypervisor called Dom-0. Alternatively, datacenter operators may decide to host it on the application's VM. The monitoring agent periodically reports the state of the host to ProMLB server. The duration of the period, which is referred to as Window, depends on the application's characteristics. The monitoring agent reports the current state, architectural signature, and the duration of window to the ProMLB server.

3.2.1 Architectural Signature. The monitoring agent extracts architectural information of application during each window and reports it to the server. This architectural information is collected through the Intel Performance Counter Monitor tool (PCM) [3] to understand the memory and processor behavior. The information that we use to study the behavior of applications are: available virtual, physical, and shared memory, the cache, buffer space, memory bandwidth utilization, ratio of free to total disk space, storage bandwidth utilization, network Bytes sent and received, L2 and Last Level Cache (LLC) hits ratio, instruction per cycle, core C0 state residency, and CPU idle, system, user time. We consider this information as an architectural signature of an application.

In this work, the resource usage pattern is called application's behavior. We showed that the behavior of an application changes if it becomes memory-bound, core-bound, I/O-bound, or idle. Each of those distinct behavior is called a "phase." As an example, when we say an application is in its memory bound phase, it means its memory usage pattern shows the application is memory-intensive. Hence, if the application's behavior changes, it means that its phase is changed, too. Therefore, a period of time that application shows a distinct behavior is a phase. We define "window" as follows: a fixed duration of time that monitoring agents periodically sends its state report to the ProMLB server (master node). During each window, an application may have multiple phases. We label the window with a phase that consumes most of the time of that window. For example, if two thirds of a window are memory-bound, we call that window memory-bound. Basically, what our predictors will predict would be the major phase in the next window.

3.3 Phase Predictor

We equipped ProMLB with a phase predictor to be proactive to act before a significant change happens in the behavior of the application and degrades the application's performance. Phase predictor will predict the future behavior of the application based on the current and previous behavior. For this purpose, we employ three techniques, such as time series neural network, HMM, and KNN. Each of these techniques has its tradeoffs. We observed during simulations that accuracy is limited for unseen applications. Therefore, we employ Ensemble learning to boost accuracy. We use the ensemble method, which uses a combination of multiple learning algorithms to obtain better predictive performance than could be obtained from any of the constituent learning techniques alone. The accurate prediction is important as we can allocate enough resources to the application before the performance degrades.

3.3.1 Time Series Neural Network. Time series neural network (TSNN) [56] is an eager learning technique. The training of TSNN is done offline by our database. The time series neural network module is based on a nonlinear autoregressive network with exogenous inputs network. The following equation is utilized to predict future behavior:

$$Y(t) = F(Y(t-1), Y(t-2), \ldots, Y(t-n)).$$

In this work, we used a 10-layer fully connected NN with the following architecture [35, 105, 80, 80, 75, 60, 65, 45, 15, 1] for the prediction. The number of neurons in each hidden layer and the number of layers are decided through Grid Search [47] to reach the highest possible accuracy.

3.3.2 Hidden Markov Model. The HMM is another eager technique employed for effective prediction. The HMM is used extensively for performance modeling and performance-prediction analysis, where the HMM can predict the future state of a target system based on its current state. In reality, as the relationship between the observed time and the observed state is not one to one, a group of probability distributions for two stochastic processes are involved, called the HMM. In an HMM, the states are not observable, but when we visit a state, an observation is recorded that is a probabilistic function of the state.

3.3.3 K Nearest Neighbors Regression. KNN [48] is a lazy learning technique that does not require training. Suppose the dataset has m samples that each sample x_i is described by n input variables and an output variable y_i such as $x_i = \{x_{i1}, \ldots, x_{in}|y_i\}$. The goal is to learn a mapping function F: $x -> y$ known as a regression function that captures and models the relationship between input variables x and an output variable y. The KNN regression estimates the function by taking a local average of the dataset. Locality is defined in terms of the k samples nearest to the estimation sample. As the performance of KNN algorithm strongly depends on the parameter k, finding the best values of k is essential. A large k value decreases the effect of noise and minimizes the prediction losses. However, a small k value allows simple implementation and efficient queries.

3.3.4 Ensemble Method. Ensemble learning [46] is a branch of ML that is used to improve the accuracy and performance of general ML predictors. We use ensemble learning to enable the use of both eager learning techniques (TSNN and HMM) and lazy learning techniques (KNN), which does not require training. Using the Lazy learning technique enables ProMLB to be more flexible and have better accuracy for unknown applications. ML has been used in many areas [22, 41, 42] and developers select an appropriate technique based on their field's requirements [21, 40, 45]. In this work, we use Bagging or Bootstrap Aggregation [46], which is an ensemble learning model that is used for predictions. It is a statistical prediction technique where a future state of the application is estimated from voting of prediction results of three models. Each model is exploited to make a prediction, and the results are voted to give a more robust and generalized prediction. If the prediction of all three ML techniques is different from each other, then the voter will select the current state as the final result.

3.4 Performance and Cost Model Generator

In this part, we formulate the performance and cost analysis for different applications in a scale-out environment. The first part of this section is devoted to performance modeling. The second part is to formulate the dependency of the price that a subscriber must pay for utilizing different server configurations. This cost model is based on the bare metal servers' cost offered by the IBM SoftLayer cloud located in Washington, DC. We then present the developed models to formulate the performance improvement of each application with respect to the baseline hardware configuration. These models will be exploited by the optimizer in the next step to select the most performance- and cost-efficient server configuration for a given application.

3.4.1 Performance Modeling. One of the novel contributions of this article is to generate a performance model for each phase of applications dynamically. This leads to a more accurate model and helps the optimizer to select the best configuration. Figure 5 is an example to illustrate that each application has a different performance model depending on its phase and the server platform. Offline analysis of our applications shows that the performance of data-intensive applications is a convex function of servers' parameters such as core count and core's frequency. Based on the analysis of our characterization, a generic performance model can be developed. However, this generic model has to be adopted for each application. As a panacea to automatically tune the generic model depending on the architectural signature of the application, we employ Artificial Neural Network (ANN) here [19]. ANNs are a class of ML technique that maps a set of input parameters to a set of target values.

We formulate each server's performance as the product of per-processor performance and the number of processors in each server. Regarding servers' configuration, the parameters that can be configured are core count, core frequency, DRAM bandwidth and capacity, storage bandwidth. Therefore, there are nine different performance models from the combination of those parameters. As the performance does not scale linearly with the parameters, such as the number of cores, a nonlinear modeling is required. The following equation demonstrates the generic model:

$$Perf = \alpha_1 x^2 + \beta_1 y^2 + \alpha_2 x + \beta_2 y + \omega xy + \gamma,$$

where $x, y \in \{core, freq, DRAMBW, DRAMcap, StorageBW\}$ and $x \neq y$.

To capture this non-linearity effectively, we chose a SVM [49] to fit the performance models. SVM analysis is a popular ML tool for regression. Based on our database, we fit these models using SVM to find the coefficients. Once the coefficients are calculated, we use them for training the ANNs to map the architectural signature of applications to those coefficients.

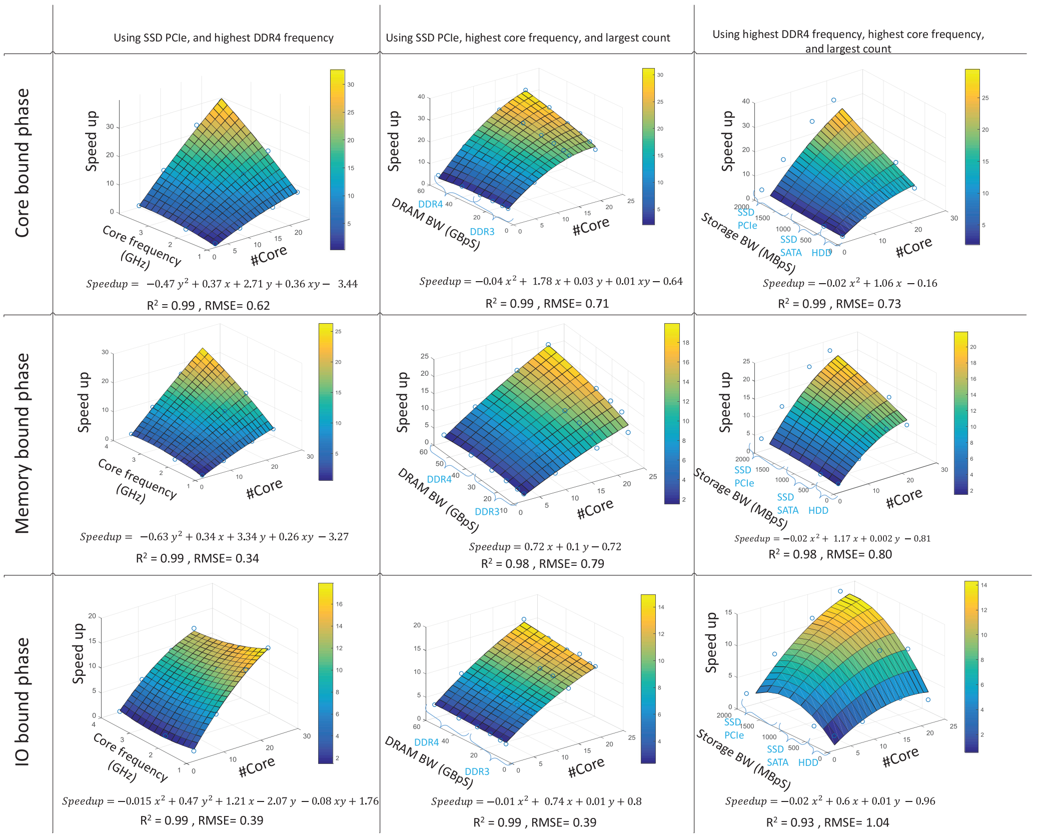

Fig. 5. Performance model generated for Graph analytics workload from Flink framework.

Nine three-layer fully connected ANNs were trained by our training database to adopt the generic performance model for each application based on the architectural signature. We started with a simple three fully connected layer neural network. We found out this model achieves our desired accuracy. Therefore, we did not use a more complex model, such as a convolutional neural network, or binary neural network [39]. Each ANN has 17 inputs, 230 hidden neurons, and six outputs. We used Grid Search to find the best number of hidden neurons. Inputs of neural networks are the architectural signature. Each output neuron stands for a coefficient. ANNs generate the performance models in parallel. In the next section, these models will be used to find the best platform and configuration. Figure 5 shows a subset of generated performance models for three different phases of graph analytic application in the Flink framework. In each sub-figure, X represents the number of cores and Y stands for the other parameter.

The advantage of this approach is that we can accurately model the performance of applications at each phase of their execution for various type of servers and improve the server selection. An appropriate resource provisioning will decrease the execution time of subscriber's job, increases the resource utilization of scale-out infrastructure and eventually brings economic benefits for both subscriber and provider. This is important, because performance improvement in datacenters translates into millions of dollars revenue per year for cloud provider and also it decreases the cost for subscriber and make cloud services more attractive for the end users.

Table 2. Values of Processor Cost's Formula

Parameter	δ	θ	ζ	κ	R^2
Value	−353.5	208.1	31.4	54.9	0.82

3.4.2 Cost Modeling. We first analyze the parameters that influence the pricing of a server. Our goal is to establish a relationship between the performance of studied applications and the cost of a server running those applications. The server price is determined as a function of server configuration as follows:

$$C_{server} = C_{processor} + C_{memory} + C_{disk} + C_{network}.$$

The per-server costs include configurable DRAM, configurable processor, disk, and network costs. In this work, we do not consider configuring the network for performance optimization. Therefore, to establish a relationship between performances of applications as well as the price, we are simply treating the network cost as constant. We extracted the price data for 30 available server configurations in IBM SoftLayer bare metal servers. We used the regression technique to derive a cost equation for storage, memory, and processor.

All the below cost equations are the predicted charge that subscribers must pay in dollar for renting a bare metal server (on a monthly basis) on the IBM SoftLayer (data are collected in January 2019), which includes the power, cooling, and maintenance related costs of the server. The equation for price per server based on the server's processing configuration is as follows:

$$C_{processor} = \delta + \theta N_{socket} + \zeta Core + \kappa Frequency,$$

where $Frequency < 4\ GHz, Cache = 2.5\ MB/Core$, and $Core < 26$ per socket.

The values presented in Table 2 are coefficients of parameters and eventually can be translated to the cost in dollar. We used MATLAB's regress library to fit our models with the price data that we collected from IBM SoftLayer in January 2019. R-squared is a goodness-of-fit measure for linear regression models. This statistic indicates the amount of the variance in the dependent variable that the independent variables explain collectively. R-squared measures the strength of the relationship between our model and the dependent variable on a convenient [0,1] scale. 0 represents a model that does not explain any of the variation and 1 represents a model that explains all of the variation in the response variable around its mean.

For the cost of memory, we derived two different equations. The first considers the effect of memory frequency and the number of channels on the cost of each memory module. These parameters determine the available DRAM bandwidth for the processor. The maximum capacity of each available memory module is 32 GB. This is the maximum available DRAM module in the market (at the time of this research). In this work, we consider one module per DIMM,

$$C_{module} = [(9 \times Capacity) \times (Mem.Frequency - 0.31)] - 5 \times N_{channel},$$

where the memory frequency is in GHz and memory capacity is in GB.

Beyond 32 GB, the memory cost is estimated using the following equation:

$$C_{memory} = (1.81 \times Capacity) + 364.$$

For the cost of storage, three types of storage are available such as SSD PCIe, SSD SATA, and HDD. To change the capacity or the bandwidth of storage, ProMLB can aggregate multiple disks together. In this way, the cost of storage is as follows:

$$C_{storage} = (N_{SSD-PCIe} \times Cost_{SSD-PCIE}) + (N_{SSD-SATA} \times Cost_{SSD-SATA}) + (N_{HDD} \times Cost_{HDD}).$$

3.5 Optimizer

For a given application and workload, our goal is to find the optimal or a near-optimal server configuration that simultaneously satisfies the performance requirements with minimal operational cost. For this purpose, we use the Bounded Knapsack algorithm to solve the aforementioned optimization problem.

3.5.1 Bounded Knapsack Solution. This solution was introduced in MeNa [32] to select the best memory configuration to maximize the performance/cost. In this work, we use the Bounded Knapsack solution to select the optimal server configuration (not only memory). To apply that methodology, first, we have to identify the cost to increase performance by changing each server parameters. Hence, we need to define a quantity called performance-cost sensitivity. For example, the performance-cost sensitivity to the bandwidth of memory is defined as follows:

$$Sens(Mem.BW.) = ((\partial Perf)/(\partial Mem.BW))/((\partial Cost)/(\partial Mem.BW)).$$

Afterward, we calculate this quantity with respect to all server parameters, such as the number of cores, memory capacity, and all other configurable parameters using our performance and cost models that we have presented in the previous section. After the calculation of sensitivity quantity, we sort all sensitivity values and based on the most significant values, we put them into a FIFO. This FIFO helps to set a priority for each parameter when allocating resources. Then by using dynamic programming, we solve the bounded knapsack problem:

$$\text{Maximize } \Sigma_{i=1}^{n} Perf_i \times Conf_i$$
$$\text{Subject to } \Sigma_{i=1}^{n} Cost_i \times Conf_i \leq Budget \text{ and } min_i \leq Conf_i \leq max_i,$$

where $Conf_i$ represents the number or the value of parameter i, min_i and max_i are the minimum and the maximum available resource for parameter i. Also, $Cost_i$ present the cost corresponding to $Conf_i$. Similarly, $Perf_i$ present the performance improvement corresponding to $Conf_i$. The result of this optimization is the recommended configuration to the manager. The budget is a constraint that the user must provide. The result of solving the optimization problem is a set of configuration such as the number of sockets, number of nodes, the number of cores, core frequency, memory capacity, memory bandwidth, storage capacity, and storage bandwidth. The optimizer recommends this configuration to the manager. It is the responsibility of the manager to decide the action that is needed to take for scaling the current platform to make it as close as to the recommended configuration for the targeted VM.

3.6 Manager

After finding the optimal configuration, the manager takes actions to allocate or adjust the resources assigned to the applications. Actions that can be executed by the manger are as follow:

The first action can be Dynamic voltage and frequency scaling, which is the adjustment of voltage and speed settings to increase or decrease CPU frequency. If it is required, then the manager can increase or decrease the number of CPU cores assigned to the application (hot-(un)plugging of resources such as memory and cores). Moreover, the manager can change VM configuration and add allocated storage or remove them. It is also feasible to increase or decrease memory capacity. The last action will be to migrate VM to a different node. Live migration is performed by an underlying mechanism (Xen Hypervisor). The manager migrates a VM when the migration latency is predicted less than half of the window's time and also when there are not enough resources on the current server.

3.6.1 Predicting Migration Time. The manager must take into account the Xen live migration time to decide whether to migrate the VM or not. Therefore, we adopt the performance model of

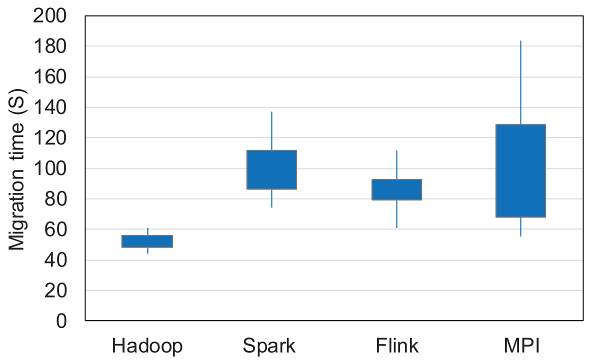

Fig. 6. Migration time.

Xen live migration proposed in Reference [38] to predict the time that takes to migrate a VM from node A to node B and resume the job.

Performance modeling of live migration involves three main factors: the size of VM memory (V_{Mem}), the memory dirtying rate (D), and network transmission rate (J). Live VM migration achieves negligible application downtime by iteratively pre-copying the pages dirtied at the previous round of transmission. Xen provides the ability to track memory accesses of guest VMs using a mechanism referred to shadow page tables. The shadow page tables are maintained by the hypervisor and translated from guest page tables on demand. In this way, the hypervisor is able to trap all memory updates within a VM and maintains a bitmap to mark the dirty pages. As VM live migration also works in shadow paging model, we can measure a VM's memory dirtying rate incidentally before the pre-migration phase. The data transmission rate for each round is determined by adding a constant increment to the previous round's memory dirtying rate (D) where the constant variable and its default value is empirically set at 100 Mb/s. Let λ denote the ratio of D to J. Then we have the migration latency:

$$T_{mig} = \sum_{i=0}^{n} T_i = \frac{V_{mem}}{J} . \frac{1 - \lambda^{n+1}}{1 - \lambda} . \tag{1}$$

Figure 6 shows the variation of migration time for different data-intensive frameworks. In this study, we set the window size equal to three times (3×) of migration time, because it gives better prediction accuracy (Figure 8 shows the impact of window size on accuracy).

3.6.2 Fair Allocation. Another aspect to consider is fair allocation. When running multiple VM on a node, the manager uses REF [55] to allocate the resources among VMs, as co-scheduling multiple VMs on a single server could result in interference. REF is a fair allocation mechanism that satisfies three game-theoretic properties (sharing incentives (SI), envy-freeness (EF), and Pareto efficiency (PE)) using Cobb–Douglass utility function. In this work, we begin with the space of possible allocations. We then add constraints to identify allocations with the desired properties as follow:

Suppose multiple VMs share a server with R types of hardware resources. Let $\mathrm{x}_i = \{\mathrm{x}_{i1}, \ldots, \mathrm{x}_{iR}\}$ denote ith VM's hardware allocation. Further, let $u_i(x_i)$ denote ith VM's utility. Following equation defines utility within the Cobb–Douglas preference domain:

$$u_i(x_i) = a_{i0} \sqcap_{r=1}^{R} x_{ir}{}^{a_{ir}} .$$

The parameters $a_i = \{a_{i1}, \ldots, a_{iR}\}$ quantify the elasticity with which a VM demands a resource. Let C_r denote the total capacity of resource r in the system. We can find a fair multi-resource allocations given Cobb–Douglas preferences with the following feasibility problem for N virtual machines and R resources:

Find x subject to:

1) $u_i(x_i) \geq u_i(x_j) \quad i, j \in [1,N]$
2) $\frac{a_{ir}}{a_{is}} \frac{x_{is}}{x_{ir}} = \frac{a_{jr}}{a_{js}} \frac{x_{js}}{x_{jr}} \quad i, j \in [1, N]; r, s \in [1,R]$
3) $u_i(x_i) \geq u_i(C/N) \quad i \in [1,N]$
4) $\sum_{i=1}^{N} x_{ir} \leq C_r \quad r \in [1,R]$,

where $C/N = \{C_1/N, \ldots, C_R/N\}$. In this formulation, the four constraints enforce EF, PE, SI, and capacity. The outcome of applying Cobb–Douglass utility function is a fair resource allocation among multiple VMs running on a server.

3.6.3 Resource Isolation. To decrease the side-effects of resource contention and interference, we enforce resource partitioning and isolation techniques. We employ core isolation (thread pinning to physical cores), to constrain interference context switching. We employ the Cache Allocation Technology available in Intel chips [5] to isolate LLC. The size of cache partitions can be changed at runtime by reprogramming MSR registers. We also use the outbound network bandwidth partitioning capabilities of Linux's traffic control. We employ the `qdisc` [4] to enforce bandwidth limits. To perform DRAM bandwidth partitioning, the manager monitors the DRAM bandwidth usage of each application using Intel PCM to co-locate jobs on the same machine where it can accommodate their aggregate peak memory bandwidth usage.

4 IMPLEMENTATION

In this section, we present our experimental system configurations and the setup. We first introduce the frameworks and the workloads we consider for evaluating the ProMLB. We then describe our hardware platform that runs the ProMLB server.

4.1 Workloads

In our study, we used Hadoop MapReduce version 2.7.1, Spark version 2.1.0 in conjunction with Scala 2.11, Flink version 1.3.3, and MPICH2 version 3.2 installed on Linux Ubuntu 16.04 LTS.

For a building and training of ProMLB, we target various domains of data-intensive workloads such that of microkernels, graph analytics, ML, E-commerce, social networks, search engines, and multimedia, totally 19 workloads. The size of input is in the range of 10 GB and 2 TB. We use BigDataBench [2] and HiBench [23] for the choice of big data benchmarking. The selected workloads have different characteristics such as high-level data graph and different input/output ratios. Some of them have unstructured data types and some others are graph based. Also, these workloads are popular in research and are widely used for demonstration of techniques. For validation of ProMLB, we used CloudSuite [17] workloads: Data Analytics, web search, Graph Analytics, and In-memory Analytics.

Figure 7 clarifies how we divided our workloads and dataset. First part is devoted for developing the system, and the second part is devoted for the evaluation of our entire system. During the development part, we used 19 workloads from two suites (BigDataBench and HiBench) to train our models. To evaluate our models during this part, we partitioned our dataset to two sets (unseen dataset for testing, and seen dataset for the training and validation). In this part, data from all 19 workloads are aggregated together and we randomly leave out 20% of the data for the unseen data points. Then we applied fivefold cross-validation technique on the 80% remaining data. The common schemes of cross-validation are m-fold cross-validation. In m-fold cross-validation, the dataset is randomly divided into m subsets or folds and repeated m times. Each time, one fold is reserved as a test dataset to validate the model and the remaining m-1 folds are used for training

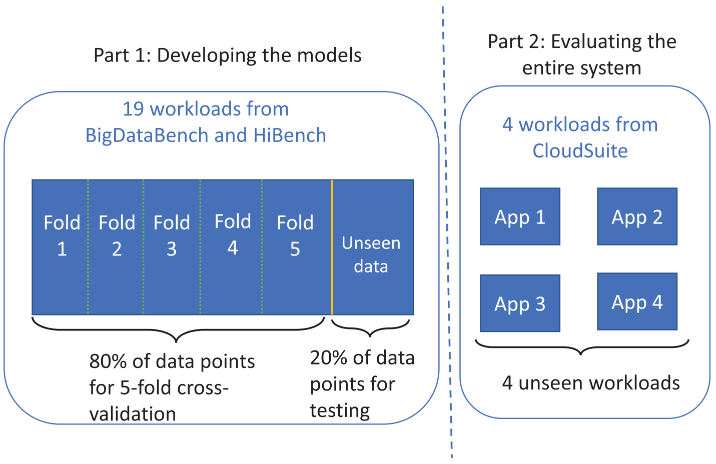
Part 1: Developing the models
19 workloads from BigDataBench and HiBench
Fold 1
Fold 2
Fold 3
Fold 4
Fold 5
Unseen data
80% of data points for 5-fold cross-validation
20% of data points for testing
Part 2: Evaluating the entire system
4 workloads from CloudSuite
App 1
App 2
App 3
App 4
4 unseen workloads

Fig. 7. Workloads and the dataset division for the training and testing phases of ProMLB.

Table 3. Detailed Information of Local Cluster

Name	Server	Freq.	Socket	Core	Cache	Mem Capacity	Storage	Server type	Count
S1	Xeon E5-4669 V4	2.2	4	22	55	96	SSD PCIe	HPC	2
S2	Xeon E5-4667 V4	2.2	4	18	45	64	SSD SATA	HPC	2
S3	Xeon E5-4650 V4	2.2	4	14	35	32	SSD SATA	HPC	2
S4	Xeon E5-2690 V4	2.6	2	14	35	512	SSD / HDD	Memory opt.	4
S5	Xeon E5-2650 V4	2.2	2	12	30	256	SSD / HDD	Memory opt.	4
S6	Xeon E5-2667 V4	3.2	2	8	25	32	SSD PCIe	I/O opt.	4
S7	Xeon E5-2643 V4	3.4	1	6	20	32	SSD PCIe	I/O opt.	4
S8	Xeon E5-2660 V2	2.2	2	10	25	16	HDD	General purp.	6
S9	Xeon E5-2650 V2	2.6	2	8	20	16	HDD	General purp.	6
S10	Xeon E5-1630 V4	3.7	1	4	10	8	HDD	Power opt.	2
S11	Xeon E5-1680 V4	3.4	1	8	20	12	HDD	Power opt.	2
S12	Xeon E3-1270 V6	3.8	1	4	8	8	HDD	Power opt.	2

of the model. Then, the classification accuracy across all m trails is computed. Figure 10 is related to this part of our experiments. As you can see, the training subfigure is related to the accuracy of training folds (*m*-1 folds) and validation subfigure is related to the validation fold (the remained fold). Then the testing subfigure is related to those 20% data that we leaved out from dataset as unseen data points. After building the models and in the second part, to evaluate ProMLB with a completely new unseen workloads, we selected 4 workloads from CloudSuite that we never used them to train or build our models. Results presented in Figure 11(b) are related to this part of our dataset.

4.2 Hardware Platform

We tested ProMLB on our 40-node cluster. Our cluster includes servers of 12 different configurations shown in Table 3. We also show how many servers of each type we use. Note that these configurations range from high-end Xeon systems to low-end ones. There is a wide range of core counts, clock frequencies, storage type, and memory capacities and bandwidth in our cluster.

Table 4. Average Prediction Time of Each Predictor

	Average prediction time
KNN	74 ms
HMM	351 ms
TSNN	580 ms
Ensemble	22 ms

4.3 ProMLB Prototype

We implemented the ProMLB prototype as a Java application running on Linux Ubuntu 16.04 LTS. ProMLB is merged into Apache CloudStack [1], which is an open-source cloud management software for running a private cloud infrastructure. It has enterprise-class support for scaling out VMs on XenServer hosts and controls the XenServer host instances using the Java bindings of the XenServer Management API. XenServer is a Linux distribution that is based on the Xen hypervisor. Neural Networks used in this work are implemented with Keras in Python. The API designed for ProMLB includes functions to express the type of submitted workloads, and functions to check job status, revoke it, or update the constraints. At the current stage, ProMLB can not be used for container-based systems such as Kubernetes. We will address these issues in our future works. ProMLB currently can manage Hadoop, Spark, Flink, and MPI based data-intensive applications. At this stage of implementation, ProMLB does not support fault tolerance. This will be a straightforward extension if ProMLB server is used as a hot-spare mirroring to provide fault-tolerance, which requires a continuously replication of all system states between two servers. ProMLB does not explicitly consider latency-critical applications or dependencies between application components. It also does not enforce fine-grain priorities between application components or user requests, or optimize for shared data placement.

5 EVALUATION

In this section, first we report the experimental results in terms of overhead and accuracy. We then compare ProMLB scheduler with other schedulers.

5.1 Overhead

In this subsection, we report the overhead of ProMLB, including the time used to collect training data, training the performance models, and searching for optimum configurations. The collecting data has the highest overhead, 8.3 hours on average and up to 10.2 hours for each workload (using 1-GB input data). Model training of each predictor took 31 minutes on average and training of ANNs took 73 minutes using two Nvidia GeForce RTX 2080 on a 16 core processor. It should be noted that training time is a one-time cost and it is even an offline cost. It only is required when we are building the system. When the development of system is finished and the system is under the deployment, we do not have such overheads. Compared to the manual configuration this overhead is still attractive as the target of ProMLB is the big data applications that repeatedly run in data centers for months or even longer. In this usage scenario, this one-time cost is amortized with a very large number of runs. Therefore, the additional overhead per run is very low. Table 4 presents the average prediction time of predictors. It should be noted that the average of prediction time is around 607 ms.

ProMLB does not need to redo the process of data collection when a new application or a server comes. In this case, when the predictor encounters a new behavior, it saves the trace and reports it to the manager. New traces will be added to the database to retrain and update the models offline.

Table 5. Information of Scheduling Decisions (Average Results for Each Virtual Machine)

Applications	WC	Sort	NW	CC	KM	NB	PR	Grep	Average
Number of phase change	24	38	65	41	33	20	59	172	56.5
Number of change in VM configuration	22	33	56	37	28	18	53	151	49.7
Number of VM migration	5	6	10	8	7	4	11	32	10.3
Average VM migration latency (Second)	95	81	103	92	94	87	91	76	89.8
Standard deviation of migration latency	17	16	31	10	24	16	12	9	16.8
Average down time (millisecond)	378	264	571	342	389	355	367	294	370
Total migration time to total execution time (%)	6.2%	4.0%	5.1%	5.7%	6.4%	5.6%	5.4%	4.6%	5.3%

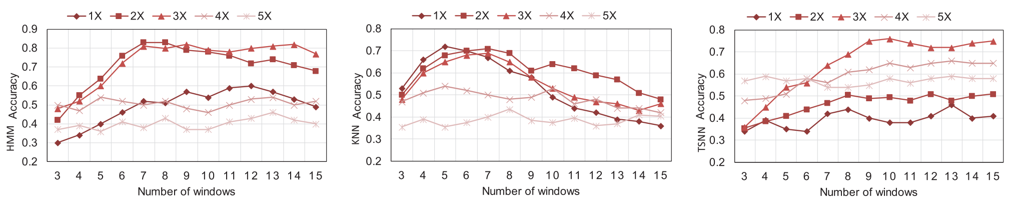

Fig. 8. Impact of window's size and number of windows on accuracy of predictors.

When the new model is ready, it can easily be replaced with the old one without any significant interrupt in the execution of the manager. Searching optimal configuration is done very fast, in seconds. This time is negligible compared to the window's size (few minutes). Moreover, this time will not be added to the execution time of workloads as this computation is being performed on ProMLB server while workloads are performing their normal execution without interrupt.

To show the results of scheduling decisions made by decision maker, we report the the number of migrations and VM resource changes for some studied applications for Spark framework in Table 5. The results show each VM migrated 10 times on average. The average migration latency is 89.8 s. Moreover, the ratio of total migration time to total execution time is around 5.3%, which is acceptable. Based on the results, the downtime of each migration is around 370 ms on average.

Corresponding to the resource utilization overhead of ProMLB, we should mention that all components of ProMLB such as predictor, performance model generator, optimizer, and manager are running on the ProMLB server (on the master node). The only component running on the worker nodes is monitoring agent. Therefore, when the ProMLB is managing the cluster, the overhead of total resource utilization of cluster is $1/N$ in which N is the number of nodes in the cluster. Because only one node is devoted to the ProMLB and the rest of the nodes are running the jobs. Hence, regardless of the memory usage of predictors or the CPU utilization of optimizer, these components are running on master node and they do not have any interfere with the worker nodes.

5.2 Accuracy

In this work, we used 20% of our data for testing the models as an unseen data. The rest of the data were used for training and validation through the fivefold cross-validation.

Figure 8 demonstrates how the window size affects the accuracy of ML techniques. In this work, each timestamp entry of time series neural network is a phase of application. The number of delay for our time series network is 10 windows. We use 8 windows of information to train the HMM. We then use this information to predict the next phase. After each window, the HMM model will be retrained and, therefore, training of our HMM is online. In this study, we set $k = 7$ as equal to the number observing windows for KNN.

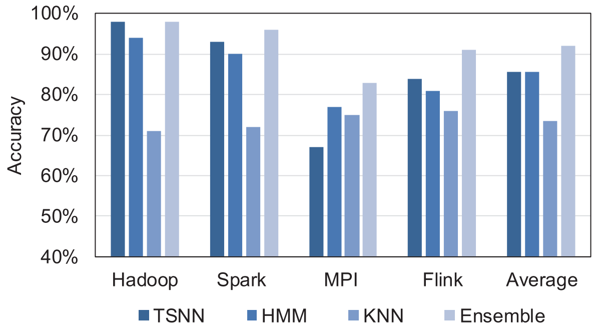

Fig. 9. Overall accuracy of predictors.

Each predictor is basically designed to predict what would be the major phase in the next window. For example, if it predicts the distribution of phases in the next window, then it is as follows: 40% of the window time memory bound, 25% of the time core bound, 20% of time I/O bound, and the rest is idle, and then the predictor concludes that the next window is memory bound, which means the application is memory intensive for the most of the time in the next window of execution. The accuracy of phase prediction is calculated as follows: Number of windows that predictor correctly predicted the phase / total number of windows. For example, if the prediction for 8 windows from a total of 10 windows is correct, then the accuracy is 80%. The results presented in Figure 9 are based in this type of calculation for accuracy.

Figure 9 summarizes a validation of the accuracy of the phase predictor engine in ProMLB. The result shows that the accuracy of ensemble method is much higher compared to each ML technique. Ensemble method achieves 92% accuracy to correctly predict the next phase of workloads. It is interesting to observe that the behavior of Hadoop, Spark, and Flink frameworks are more predictable, compared to MPI based applications. it should be noted that that without enforcing any isolation technique, the accuracy will drop to 78%. Therefore, it is important to avoid of interference between co-located VMs for better resource allocation.

Moreover, we also wanted to show that how much the major time prediction is accurate for each window. For example, suppose that the predictor predicts that the major phase in next window would be memory bound for 60% of the time. After executing the application in that window time, if the application were memory bound for 65% of the time instead of 60%, then we would have a 5% error in determining the amount of time that application was memory bound, but still, we were correct about the type of its phase. The results presented in Figure 10 show the actual value and the predicted value in a scatter view by our predictors. The *R* value clearly shows that the models are fairly accurate across the entire configuration space: All data points are located around the corresponding bisectore, indicating that the predictions and estimations are close to the real measurements. Overall, we find that there are not many outliers in our models, indicating that they can be used for optimizing the performance.

5.3 Performance and Cost Efficiency

To have a comprehensive comparison between ProMLB and other resource provisioners, we consider the following systems:

(1) *Oracle*: This is an ideal system that has a full prior knowledge of application behavior and therefore it allocates the optimal resources at runtime.

(2) *Default*: This is the default system without any manipulation from outside.

(3) *Matrix Completion*: Matrix completion (MC) method or collaborative filtering [29] proposed in Quasar [14].

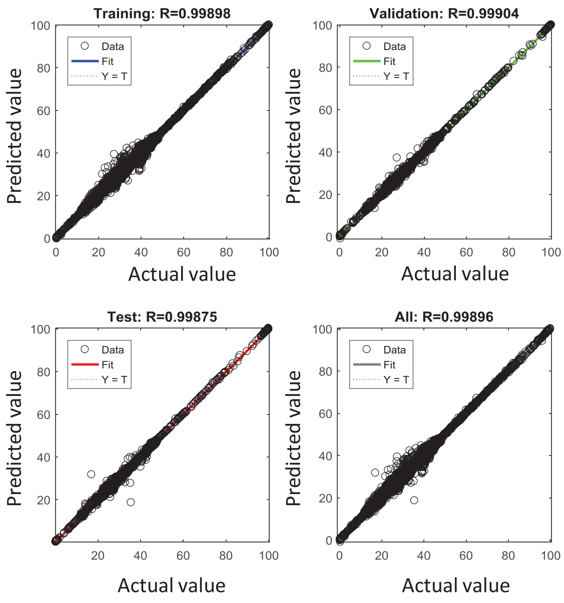

Fig. 10. Scatter plots of prediction values versus real measurements for 25,920 data points.

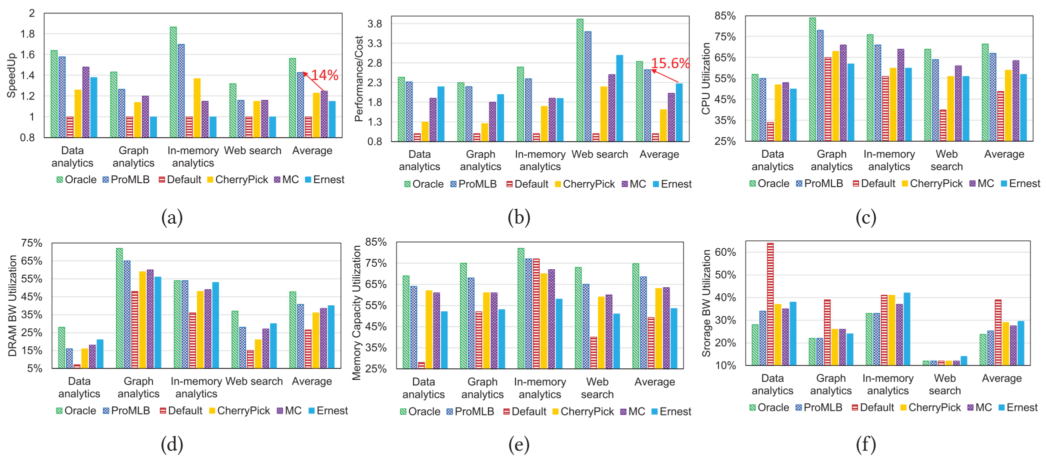

Fig. 11. Evaluation of different metrics using various schedulers: (a) Normalized speedup, (b) Normalized Perf/Cost, (c) CPU utilization across all servers, (d) DRAM bandwidth utilization across all servers, (e) DRAM capacity utilization across all servers, and (f) Storage bandwidth utilization across all servers.

(4) *Cherry-Pick scheduler*: One of the closest to ours is CherryPick, which uses a regression model for performance estimation and Bayesian optimization to find the right configuration.

(5) *Ernest scheduler*: Ernest [50] predicts the runtime of distributed analytics jobs as a function of cluster size and provisions a cost optimal configuration. However, Ernest cannot infer the performance of new workloads on a VM type without first running the workload on that VM type and also is not a dynamic approach.

5.3.1 Performance. Figure 11(a) shows the speed up of ProMLB over the baseline for CloudSuite workloads. On average, ProMLB improves the performance by 42% and up to 70%. This speed up was only achieved by efficient resource allocation, without any change in the applications or frameworks. ProMLB is performing 14% better than the state-of-the-art approaches, because it is proactive and dynamic as well. We observe that MC outperforms Ernest and CherryPick techniques as MC is able to dynamically change the configuration. Figure 11(b) shows the performance/cost improvement of studied workloads. On average, ProMLB can improve the performance/cost by 2.5× compared to default scheduler. The interesting observation is that Ernest performs better than MC and CHerryPick in terms of Performance/Cost. The reason is that Ernes is a cost aware approach but ProMLB is still performing 15.6% better than Ernest.

5.3.2 Utilization. Based on results presented in Figure 11(c), ProMLB increases CPU utilization (average across all cores) to 67% versus 49% with baseline, which is a 36% improvement. Figure 11(d)–11(f) shows the utilization of DRAM bandwidth, memory capacity, and storage bandwidth during the execution of workloads. Based on the evaluation results, ProMLB increases DRAM bandwidth and memory capacity utilization by 53% and 39%, respectively, on average. Our results indicate that the storage bandwidth utilization was dropped by 35% on average. ProMLB increases the available storage bandwidth to the applications by aggregating multiple disks together. By increasing the storage bandwidth and keep remaining the number of disk accesses, the disk utilization decreases, and this degradation does not have a negative impact on the performance.

6 CONCLUSION

In this work, we propose a proactive online resource provisioning methodology called ProMLB to address the challenge of resource allocation for data-intensive workloads in scale-out platforms. A wide range of server configuration choices are available in the cloud, and it creates a large search space to navigate for selecting an optimal configuration. Moreover, the applications' performance depends on the chosen configuration, and it makes the optimization problem even harder. In this work, to reduce the complexity, the application's behavior is first characterized into a core, I/O, or memory bound. Then, the ensemble learning based prediction is employed to predict the next phase of the application.

Further, the performance models for predicted application behavior across different platforms is derived. As the cost for chosen configuration plays a key role in resource allocation, ProMLB uses an optimization technique to distinguish a close-to-optimal configuration to maximize performance per cost. Compared to the oracle scheduler, ProMLB achieves 91% accuracy to allocate the right resource to workloads. ProMLB improves the performance and resource utilization by 42.6% and 41.1%, respectively, compared to baseline scheduler, on average. Moreover, ProMLB improves the performance per cost by 2.5× on average.

REFERENCES

[1] [n.d.]. *The Apache Software Foundation.* Retrieved from https://cloudstack.apache.org/.

[2] L. Wang, J. Zhan, C. Luo, Y. Zhu, Q. Yang, Y. He, W. Gao, Z. Jia, Y. Shi, S. Zhang, and C. Zheng. 2014. Bigdatabench: A big data benchmark suite from internet services. In *2014 IEEE 20th International Symposium on High Performance Computer Architecture (HPCA'14).* IEEE, 488–499.

[3] Thomas Willhalm and Roman Dementiev. [n.d.]. Retrieved from https://software.intel.com/en-us/articles/intel-performance-counter-monitor.

[4] [n.d.]. *Martin A. Brown. Traffic Control Howto.* Retrieved from http://linux-ip.net/articles/Traffic-Control-HOWTO/.

[5] 2014. *Intel R 64 and IA-32 Architecture Software Developer's Manual, Vol. 3B: System Programming Guide, Part 2.*

[6] 2017. *Rightscale Inc. 2017. Amazon EC2: Rightscale.* Retrieved from http://www.rightscale.com/.

[7] 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 *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation (NSDI'17)*, Vol. 2. 4–2.

[8] YARN Apache Hadoop. 2013. Yet another resource negotiator. *In Proceedings of the ACM Symposium on Cloud Computing (SoCC'13).*

[9] Sean Kenneth Barker and Prashant Shenoy. 2010. Empirical evaluation of latency-sensitive application performance in the cloud. In *Proceedings of the 1st Annual ACM SIGMM Conference on Multimedia Systems.* ACM, 35–46.

[10] Peter Bodik, Rean Griffith, Charles Sutton, Armando Fox, Michael I. Jordan, and David A. Patterson. 2009. Automatic exploration of datacenter performance regimes. In *Proceedings of the 1st Workshop on Automated Control for Datacenters and Clouds.* ACM, 1–6.

[11] C. L. Philip Chen and Chun-Yang Zhang. 2014. Data-intensive applications, challenges, techniques and technologies: A survey on big data. *Inf. Sci.* 275 (2014), 314–347.

[12] Jinchuan Chen, Yueguo Chen, Xiaoyong Du, Cuiping Li, Jiaheng Lu, Suyun Zhao, and Xuan Zhou. 2013. Big data challenge: A data management perspective. *Front. Comput. Sci.* 7, 2 (2013), 157–164.

[13] Christina Delimitrou and Christos Kozyrakis. 2013. Paragon: QoS-aware scheduling for heterogeneous datacenters. *ACM SIGPLAN Not.* 48 (2013), 77–88.

[14] Christina Delimitrou and Christos Kozyrakis. 2014. Quasar: Resource-efficient and QoS-aware cluster management. *ACM SIGPLAN Not.* 49, 4 (2014), 127–144.

[15] Christina Delimitrou and Christos Kozyrakis. 2016. Hcloud: Resource-efficient provisioning in shared cloud systems. *ACM SIGOPS Operat. Syst. Rev.* 50, 2 (2016), 473–488.

[16] Christina Delimitrou and Christos Kozyrakis. 2017. Bolt: I know what you did last summer... in the cloud. In *ACM SIGARCH Comput. Arch. News* 45 (2017), 599–613.

[17] Michael Ferdman, Almutaz Adileh, Onur Kocberber, Stavros Volos, Mohammad Alisafaee, Djordje Jevdjic, Cansu Kaynak, Adrian Daniel Popescu, Anastasia Ailamaki, and Babak Falsafi. 2012. Clearing the clouds: A study of emerging scale-out workloads on modern hardware. *ACM SIGPLAN Not.* 47 (2012), 37–48.

[18] Marisabel Guevara, Benjamin Lubin, and Benjamin C. Lee. 2013. Navigating heterogeneous processors with market mechanisms. In *Proceedings of the 2013 IEEE 19th International Symposium on High Performance Computer Architecture (HPCA'13).* IEEE, 95–106.

[19] Kevin Gurney. 2014. *An Introduction to Neural Networks.* CRC Press.

[20] Maryam Heidari and James H. Jr Jones. 2020. Using BERT to extract topic-independent sentiment features for social media bot detection. In *Proceedings of the IEEE 2020 11th Annual Ubiquitous Computing, Electronics & Mobile Communication Conference (UEMCON'20).*

[21] Maryam Heidari, James H. Jr Jones, and Ozlem Uzuner. 2020. Deep contextualized word embedding for text-based online user profiling to detect social bots on twitter. In *Proceedings of the IEEE 2020 International Conference on Data Mining Workshops (ICDMW'20).*

[22] Maryam Heidari and Setareh Rafatirad. 2020. Using transfer learning approach to implement convolutional neural network to recommend airline tickets by using online reviews. In *Proceedings of the IEEE 2020 15th International Workshop on Semantic and Social Media Adaptation and Personalization (SMAP'20).*

[23] Shengsheng Huang, Jie Huang, Jinquan Dai, Tao Xie, and Bo Huang. 2010. The hibench benchmark suite: Characterization of the mapreduce-based data analysis. In *Proceedings of the 2010 IEEE 26th International Conference on Data Engineering Workshops (ICDEW'10).* IEEE, 41–51.

[24] Keith R. Jackson, Lavanya Ramakrishnan, Krishna Muriki, Shane Canon, Shreyas Cholia, John Shalf, Harvey J. Wasserman, and Nicholas J. Wright. 2010. Performance analysis of high performance computing applications on the amazon web services cloud. In *Proceedings of the 2010 IEEE 2nd International Conference on Cloud Computing Technology and Science (CloudCom'10).* IEEE, 159–168.

[25] Svilen Kanev, Juan Pablo Darago, Kim Hazelwood, Parthasarathy Ranganathan, Tipp Moseley, Gu-Yeon Wei, and David Brooks. 2015. Profiling a warehouse-scale computer. In *Proceedings of the 2015 ACM/IEEE 42nd Annual International Symposium on Computer Architecture (ISCA'15).* IEEE, 158–169.

[26] George Kousiouris, Andreas Menychtas, Dimosthenis Kyriazis, Spyridon Gogouvitis, and Theodora Varvarigou. 2014. Dynamic, behavioral-based estimation of resource provisioning based on high-level application terms in cloud platforms. *Fut. Gener. Comput. Syst.* 32 (2014), 27–40.

[27] Neeraj Kulkarni, Feng Qi, and Christina Delimitrou. 2018. Leveraging approximation to improve datacenter resource efficiency. *IEEE Comput. Arch. Lett.* 17, 2 (2018), 171–174.

[28] Tan N. Le, Xiao Sun, Mosharaf Chowdhury, and Zhenhua Liu. 2019. BoPF: Mitigating the burstiness-fairness tradeoff in multi-resource clusters. *ACM SIGMETRICS Perf. Eval. Rev.* 46, 2 (2019), 77–78.

[29] Jure Leskovec, Anand Rajaraman, and Jeffrey David Ullman. 2014. *Mining of Massive Datasets*. Cambridge University Press.

[30] Haikun Liu, Cheng-Zhong Xu, Hai Jin, Jiayu Gong, and Xiaofei Liao. 2011. Performance and energy modeling for live migration of virtual machines. In *Proceedings of the 20th International Symposium on High Performance Distributed Computing*. ACM, 171–182.

[31] Amiya K. Maji, Subrata Mitra, Bowen Zhou, Saurabh Bagchi, and Akshat Verma. 2014. Mitigating interference in cloud services by middleware reconfiguration. In *Proceedings of the 15th International Middleware Conference*. ACM, 277–288.

[32] Hosein Mohammadi Makrani and Houman Homayoun. 2017. MeNa: A memory navigator for modern hardware in a scale-out environment. In *Proceedings of the 2017 IEEE International Symposium on Workload Characterization (IISWC'17)*. IEEE, 2–11.

[33] Hosein Mohammadi Makrani, Setareh Rafatirad, Amir Houmansadr, and Houman Homayoun. 2018. Main-memory requirements of big data applications on commodity server platform. In *Proceedings of the 2018 18th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID'18)*. IEEE, 653–660.

[34] Hosein Mohammadi Makrani, Hossein Sayadi, Sai Manoj Pudukotai Dinakarra, Setareh Rafatirad, and Houman Homayoun. 2018. A comprehensive memory analysis of data intensive workloads on server class architecture. In *Proceedings of the International Symposium on Memory Systems*. 19–30.

[35] Hosein Mohammadi Makrani, Hossein Sayadi, Devang Motwani, Han Wang, Setareh Rafatirad, and Houman Homayoun. 2018. Energy-aware and machine learning-based resource provisioning of in-memory analytics on cloud. In *Proceedings of the ACM Symposium on Cloud Computing*. 517–517.

[36] Hosein Mohammadi Makrani, Shahab Tabatabaei, Setareh Rafatirad, and Houman Homayoun. 2017. Understanding the role of memory subsystem on performance and energy-efficiency of hadoop applications. In *Proceedings of the 2017 8th International Green and Sustainable Computing Conference (IGSC'17)*. IEEE, 1–6.

[37] Maria Malik, Hassan Ghasemzadeh, Tinoosh Mohsenin, Rosario Cammarota, Liang Zhao, Avesta Sasan, Houman Homayoun, and Setareh Rafatirad. 2019. Ecost: Energy-efficient co-locating and self-tuning mapreduce applications. In *Proceedings of the 48th International Conference on Parallel Processing*. 1–11.

[38] Senthil Nathan, Umesh Bellur, and Purushottam Kulkarni. 2015. Towards a comprehensive performance model of virtual machine live migration. In *Proceedings of the 6th ACM Symposium on Cloud Computing*. 288–301.

[39] Nameh Nazari and Mostafa E. Salehi. 2020. Binary neural networks. In *Hardware Architectures for Deep Learning*. 95–115.

[40] Najmeh Nazari, Mohammad Loni, Mostafa E. Salehi, Masoud Daneshtalab, and Mikael Sjodin. 2019. TOT-Net: An endeavor toward optimizing ternary neural networks. In *Proceedings of the 2019 22nd Euromicro Conference on Digital System Design (DSD'19)*. IEEE, 305–312.

[41] Najmeh Nazari, Seyed Ahmad Mirsalari, Sima Sinaei, Mostafa E. Salehi, and Masoud Daneshtalab. 2020. Multi-level binarized LSTM in EEG classification for wearable devices. In *Proceedings of the 2020 28th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP'20)*. IEEE, 175–181.

[42] Katayoun Neshatpour, Hosein Mohammadi Mokrani, Avesta Sasan, Hassan Ghasemzadeh, Setareh Rafatirad, and Houman Homayoun. 2018. Architectural considerations for FPGA acceleration of machine learning applications in mapreduce. In *Proceedings of the 18th International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation*. 89–96.

[43] Rajiv Ranjan. 2014. Streaming big data processing in datacenter clouds. *IEEE Cloud Comput.* 1, 1 (2014), 78–83.

[44] Francisco Romero and Christina Delimitrou. 2018. Mage: Online interference-aware scheduling in multi-scale heterogeneous systems. *arXiv:1804.06462*. Retrieved from https://arxiv.org/abs/1804.06462.

[45] Hossein Sayadi, Hosein Mohammadi Makrani, Sai Manoj Pudukotai Dinakarrao, Tinoosh Mohsenin, Avesta Sasan, Setareh Rafatirad, and Houman Homayoun. 2019. 2smart: A two-stage machine learning-based approach for run-time specialized hardware-assisted malware detection. In *Proceedings of the 2019 Design, Automation & Test in Europe Conference & Exhibition (DATE'19)*. IEEE, 728–733.

[46] Hossein Sayadi, Nisarg Patel, Sai Manoj PD, Avesta Sasan, Setareh Rafatirad, and Houman Homayoun. 2018. Ensemble learning for effective run-time hardware-based malware detection: A comprehensive analysis and classification. In *Proceedings of the 2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC'18)*. IEEE, 1–6.

[47] Selmar K. Smit et al. 2009. Comparing parameter tuning methods for evolutionary algorithms. In *Congress on Evolutionary Computation*.

[48] Yunsheng Song, Jiye Liang, Jing Lu, and Xingwang Zhao. 2017. An efficient instance selection algorithm for k nearest neighbor regression. *Neurocomputing* 251 (2017), 26–34.

[49] Theodore B. Trafalis and Huseyin Ince. 2000. Support vector machine for regression and applications to financial forecasting. In *Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks (IJCNN'00)*, Vol. 6. IEEE, 348–353.

[50] Shivaram Venkataraman, Zongheng Yang, Michael J. Franklin, Benjamin Recht, and Ion Stoica. 2016. Ernest: Efficient performance prediction for large-scale advanced analytics. In *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation (NSDI'16)*. 363–378.

[51] Neeraja J. Yadwadkar, Ganesh Ananthanarayanan, and Randy Katz. 2014. Wrangler: Predictable and faster jobs using fewer resources. In *Proceedings of the ACM Symposium on Cloud Computing*. ACM, 1–14.

[52] 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*. ACM, 452–465.

[53] Ahmad Yasin. 2014. A top-down method for performance analysis and counters architecture. In *Proceedings of the 2014 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS'14)*. IEEE, 35–44.

[54] Zhibin Yu, Zhendong Bei, and Xuehai Qian. 2018. Datasize-aware high dimensional configurations auto-tuning of in-memory cluster computing. In *Proceedings of the 23rd International Conference on Architectural Support for Programming Languages and Operating Systems*. ACM, 564–577.

[55] Seyed Majid Zahedi and Benjamin C. Lee. 2014. REF: Resource elasticity fairness with sharing incentives for multiprocessors. *ACM SIGARCH Comput. Arch. News* 42, 1 (2014), 145–160.

[56] G. Peter Zhang. 2003. Time series forecasting using a hybrid ARIMA and neural network model. *Neurocomputing* 50 (2003), 159–175.

[57] Paul Zikopoulos and Chris Eaton. 2011. *Understanding Big Data: Analytics for Enterprise Class Hadoop and Streaming Data*. McGraw–Hill Osborne Media.

Received May 2020; revised October 2020; accepted December 2020