2020 IEEE 22nd International Conference on High Performance Computing and Communications; IEEE 18th International Conference on Smart City; IEEE 6th International Conference on Data Science and Systems (HPCC/SmartCity/DSS)

DiHi: Distributed and Hierarchical Performance Modeling of Multi-VM Cloud Running Applications

Hamidreza Moradi, Wei Wang and Dakai Zhu
The University of Texas at San Antonio
{hamidreza.moradi, wei.wang, dakai.zhu}@utsa.edu

***Abstract*—Performance fluctuation of cloud running applications, due to collocations of different tenants on the same machine, is one of the cloud users' concerns. To alleviate users' concerns with insightful information, performance modeling and prediction of their applications is deemed necessary. In this paper, we propose two hierarchical and a monolithic performance modeling and prediction frameworks for multi-VM applications running on clouds. Hierarchical frameworks have been considered to distribute the required processing power for performance modeling and prediction over a group of collaborating VMs for better scalability, and to reduce the load of transferring unprocessed profiling data. Several unsupervised learning algorithms and existing single-VM models have been incorporated into the hierarchical frameworks to evaluate their effects on models' accuracy. Then, the results are compared with a baseline approach where the most intuitive monolithic framework requires the accumulation of all profiling data on a central VM. We have considered a comprehensive set of micro-benchmarks to probe the contention in various resources of all collaborating VMs. Then, using the proposed frameworks, the experienced contention levels by each VM and the overall performance of the applications are modeled and predicted. The results of our experiments in a private and two public clouds show that the distributed and hierarchical frameworks can predict the overall applications' performance effectively with comparable accuracy to the monolithic framework with an average prediction error of 5% for different cluster sizes and clouds.**

I. Introduction

As reported by Berkeley view of cloud computing, performance variation has been one of the main concerns of cloud running applications [17]. Performance variation stems from the collocation of multiple cloud tenant VMs on the same hardware competing for the available resources. Although there are efforts to control performance variation caused by collocation [36, 37], studies show that there is still an average of 20% to 25% performance variation for cloud running applications [33, 34, 40]. Moreover, performance variation accounts for nearly 25% of users' created support tickets [28], wasting users' time and requiring providers' specialized support teams. To address users' concerns, the accurate performance modeling and prediction of their cloud running applications can significantly help them to set their performance expectations, find slow-running VMs that affect applications' progress, and make insightful scheduling and auto-scaling decisions [5, 38, 39].

However, performance modeling and predictions of cloud applications is a challenging task due to the distinct intensity and source of interference experienced by every single VM, uniquely affecting the performance of applications. Moreover, the source and intensity of interference experienced by a VM can often vary based on the changes in its collocated VMs and their running applications on the same host [13]. These changes will result in different contention levels, subsequently impacting the overall performance of multi-VM applications. To model and predict the performance of multi-VM cloud applications under different contention levels experienced by VMs in a group, an accurate resource profiling of all collaborating VMs is required. Architecture-level profiling with CPU's Performance Monitoring Unit (PMU), although well studied [13, 32], is not accessible by ordinary cloud users. OS-level profiling, using the operating system offered resource utilization, does not provide an accurate measure of resource contention. Finally, application-level profiling, using a framework provided execution graph, does not apply to all cloud running applications and is not suitable for ordinary cloud users [6, 47, 48].

To address the resource profiling issue, prior studies have used micro-benchmarks to estimate the intensity of resource contention in different resources such as CPU, Memory, I/O, and cache [4, 40, 46]. Leitner et al. used resource profiling to model the average performance of single-VM cloud running applications not suitable for runtime prediction [40]. Baughman et al. used actual application performance to model the average execution time with different data input sizes [4]. Varghese et al. [46] used user-provided weights for the profiled resources to rank a list of VM instances. However, some models [4, 40] provide the average performance of applications on different cloud instances, but do not considering the change in contention levels. And some models [46] rely on inaccurate weights provided by the users, while others have only considered a limited set of resources to profile [40]. To address the existing limitations, in our previous works [33, 34] we have used micro-benchmarks for in-situ performance modeling and prediction of cloud running applications. A set of devised micro-benchmarks are executed before the target application to collect contention information and corresponding application performance. Then, the collected data is used for accurate in-situ performance modeling and prediction of single-VM cloud applications.

Note that the studied models can predict the performance of

978-1-7281-7649-9/20/$31.00 ©2020 IEEE
DOI 10.1109/HPCC-SmartCity-DSS50907.2020.00002

single-VM applications, which does not consider the contention in intra-VM communications for multi-VM applications. To address this problem, in this paper, we propose ***DiHi***, a framework for distributed and hierarchical in-situ performance modeling and prediction of multi-VM cloud applications. We use a comprehensive set of micro-benchmarks to profile the contention experienced by each VM in different resources including the intra-VM communications (i.e. CPU, Memory bandwidth/latency, I/O, L1, L2, L3 caches, and Network bandwidth/latency). The execution of micro-benchmarks on all collaborating VMs in parallel will be followed by the actual application execution, providing the experienced contention levels by each VM and the corresponding performance of the application. Then, the collected performance data is used for model training and evaluation. Once the performance model is built for a specific instance (VM-type/flavor) and application, considering the periodic execution of cloud applications [28], it can be reused as needed in the future.

In *DiHi*, to distribute the required processing for scalable model training and prediction, and to reduce the amount of profiling data to transfer between the collaborating VMs, hierarchical models are used [51]. For hierarchical performance modeling and prediction, the first level of VMs is responsible for pre-processing and clustering of the profiling data into different contention levels. Then, the experienced contention levels by each VM is sent to a managing VM for aggregation and final modeling. In this way, micro-benchmarks execution results will be extracted, normalized, and clustered using unsupervised learning algorithms into different contention levels on each collaborating VM. In this step, clustering algorithm is used given that the effect of experienced contention by each collaborating VM on the final application performance is still unknown. Next, the contention level information from all the VMs will be sent to the managing VM for aggregation.

Finally, the effect of different contention levels experienced by each collaborating VM on the overall performance of the application will be modeled by the managing VM using feed-forward Neural networks (NN). This approach will make performance modeling and prediction more scalable by reducing the extensive processing load on a single VM. Moreover, training the model in a distributed fashion speeds-up the modeling and prediction process and reduces the load of transferring the unprocessed profiling data to a central VM. We have also considered to incorporate the previously built single-VM models, as used in *uPredict* [33], into the *DiHi* hierarchical model. The single-VM model will replace the clustering algorithm in *DiHi* to estimate the experienced contention levels in local resources of each collaborating VMs. Since the single-VM model won't consider the effect of contention in intra-VM communications, its output will be sent in conjunction with measured communication contention to the managing VM for aggregation and modeling.

Experiments have been conducted on a private cloud, Chameleon cloud [7], and Google Compute Engine (GCE) [20] using six representative cloud applications from HiBench [24], CloudSuite [16], and Nas Parallel Benchmarks (NPB) [3]. Various clustering algorithms have been considered in the *DiHi* framework to evaluate their effects on the final models' accuracy, including KMeans [22], Affinity Propagation [18], Mean Shift [49], and Birch [53] clustering. Moreover, feed-forward neural networks [23], as the most accurate model for performance modeling and prediction [33], has been used for the final aggregation of the clustering results on the managing VM.

The results of our experiments show that *DiHi* framework with the considered set of micro-benchmarks can provide a precise measure of different contention levels experienced by collaborating VMs, leading to an accurate performance modeling and prediction with only 12.6%, 3.4%, and 2.8% average prediction errors with KMeans clustering algorithm on private cloud, Chameleon cloud, and GCE respectively. We have compared the accuracy of *DiHi* framework with monolithic feed-forward neural networks requiring the collection and processing of all performance data in a central location. The results show that *DiHi* is slightly less accurate than the monolithic model with only 1.6% more prediction error while decreasing network communications and distributing computation for performance modeling and prediction. The main contributions of this work include:

1) A distributed and hierarchical performance modeling and prediction framework, *DiHi*, is proposed to provide accurate performance modeling and prediction of multi-VM cloud applications.
2) A comprehensive set of micro-benchmarks (CPU, Memory bandwidth/latency, I/O, L1, L2, L3 caches, and Network bandwidth/Latency) is used to examine the contention in a wide range of system resources.
3) Integration of single-VM models into *DiHi* framework to evaluate the effectiveness of adopting the existing performance models.
4) An extensive evaluation of the proposed framework is conducted using six representative cloud applications on a private and two public clouds.

The rest of this paper is organized as follows. Section II provides the related works on performance modeling and prediction of cloud applications. Section III discusses the micro-benchmarks that are used to profile the resource contention on each VM. The proposed methodology and predictive frameworks are presented in Section IV. Section V discusses the experimental setup and evaluation results. Finally, Section VI concludes the paper.

II. Related Works

Proper Instance Selection: With the abundance of existing VM-configurations users have difficulty choosing the proper instance type. To assist the users, research studies have focused on predicting the average performance of user applications in a wide range of instance types and cloud providers [2, 10, 15, 42, 50]. Cherry Pick [2] selects several different instance

configurations for the actual application execution to predict the average performance across all the different instance types. In Cherry Pick, Bayesian optimization is used for instance selection to reduce the total number of required executions for performance modeling. Paris [50] models the application's behavior with different amounts of available system resources to recommend the instance type that better meets users' goals. They have considered a wide range of resources to profile using OS-level provided information to model and predict the application's performance. Moreover, Paris uses both offline and online performance modeling to reduce the cost and number of experiments required on the cloud. QuMan [42] is another tool that uses profiled application performance in isolation to estimate an appropriately sized VM on the cloud. However, the OS-level resources profiling used in Paris do not accurately reflect the experienced contention, and about 50% prediction error is observable. Moreover, performance profiling of users' applications in an isolation environment appears to be impractical, and modeled average performance seems imprecise.

Cloud Workload Prediction: Long tail latency is a major concern among cloud users. To alleviate users' service disruption, several studies have predicted the incoming workload requests to make better VM collocation decisions in cloud data centers [9, 29, 52]. For instance, Kumar et al. used a pool of tasks (i.e. incoming workload requests) to extract the seasonal features for model training [29]. The trained model then can be used for future workload prediction to assist VM collocation. Although this approach seems promising, it requires the existence of a clear trend between the pool of tasks and future submissions. For hard-to-predict tasks without clear recurring patterns, Yu et al. proposed the use of a clustered pool of tasks based on workload characteristics [52]. Then, by using initial workload parameters for any submitted task the closest cluster can be found to predict the expected future workloads. Provided workload prediction can be used to make smart scheduling decisions, thus preventing any required instance migration. However, for this research category, a pool of incoming workloads with known characteristics and clear seasonality is required, though it may not always be accessible.

Interference Prediction and Prevention: To make better collocation decisions, numerous research studies have used Performance Monitoring Unit (PMU) counters to determine the sensitivity of applications to the source of interference [12, 13, 21, 32, 36, 37, 45]. DeepDive [36] uses PMU to detect the time and source of interference. In [45], the authors not only detected the source of interference using PMU counters but also alleviated contention by throttling background applications. Mage [37] first determines the sensitivity of the application to the source of interference. Then, bases on the sensitivity profile, it determines the proper placement of the application. However, all of these studies have considered architecture-level access to the system resources and CPU counters with control over the placement of the VMs. In contrast, PMU counters are not accessible by the cloud users, nor do they have any control over their applications placements, which thus makes the adoption of these solutions for cloud users impossible.

Framework-Specific Performance Prediction: In their studies, a number of researchers have focused on the application-specific performance modeling (e.g., Hadoop and Spark workloads) of cloud running applications by leveraging additional information about the job progress in the framework [6, 8, 47, 48]. In [47, 48], the authors used the Directed Acyclic Graph (DAG) information of a spark task to model the performance of different stages. Then, they predicted the application performance through the limited execution of newly submitted job and the existing model. Castiglione et al. [6] used Mean Field Analysis to analyze a large number of communications between the collaborating machines to provide a description of elements and their relations for modeling and prediction. However, these studies require information about the progress of task's different stages or communication patterns, which is not accessible for all cloud running applications. Moreover, it is difficult for ordinary cloud users to adopt such framework dependent techniques.

Application-Specific Performance Prediction: Several studies have considered different ways of application benchmarking for performance modeling and prediction [4, 31, 40, 44, 46, 50]. In [31], specific lines of code were inserted into several parts of the application's source code. The inserted codes were used to benchmark the progress of different execution stages. Then, the benchmarked progress with the corresponding application's performance under different cloud instances and settings were collected and used for modeling and later predictions. Tak et al. [44] implemented an easy-to-install PesudoApp that emulates the sequence of systems calls for a considered application. The execution of the PesudoApp under different cloud settings provided required benchmarking information for the performance prediction of the actual application. In [46], the authors use micro-benchmarks to profile different system resources for various cloud instances. Then, the profiling data with a set of weights provided by the user, representing the importance of each resource, will be used to generate two ranked lists of instances for the users with the consideration of performance and cost. Studies used benchmarking for performance modeling share some similarities to our approach. However, modifying the source code or emulating sequence of system calls may not always be practical for cloud users and can be difficult to adopt. Moreover, users do not have accurate information about the impact of different resources on the overall application's performance, which needs to be found. Finally, all these research studies focused on the average performance of the applications on different instances and have not considered the changed in the contention and corresponding application run-time performance. Our proposed framework, using micro-benchmarking, profiles the contention level of different resources and trains a model on the experienced contention levels and the corresponding application's

performance to accurately predict run-time performance.

III. Preliminary and Performance Profiling

Depending on the collocated VMs and their running applications on the same hardware, every single VM will experience unique levels of contention during its execution. It is required to estimate the contention level experienced by each particular VM in a group of VMs for performance modeling and prediction of multi-VM cloud running applications. Although PMU counters are widely adopted for performance modeling and interference detection [36, 37, 42, 48], they are not accessible to the cloud users. Moreover, not all applications or frameworks provide information about their progress. For these reasons, we considered using micro-benchmarks to probe the contention experienced by each collaborating VM. To estimate the contention in the host's local resources, we use a set of micro-benchmarks which were developed based on Lmbench [30] opensource benchmarking suite. Moreover, iperf [26] benchmarking suite is used for estimating the contention in intra-VM network communications. What follows are the implementation details for the micro-benchmarks and their considered execution time.

A. Contention Estimation with Micro-benchmarks

For the implementation details of our CPU, Memory Bandwidth, and Disk IO micro-benchmarks we refer the readers to the *uPredict* framework [33], and their output are represented as c_{cpu}, c_{mem}, and c_{disk}, respectively.

Cache: To evaluate the contention level in the cache subsystem three micro-benchmarks are devised for L1, L2, and L3 caches, respectively. These three micro-benchmarks will run on each VM sequentially with threads equal to the number of available vCPUs to access an array of size 256KB, 2MB, and 20MB with the stride size of 128 respectively. The total number of times each micro-benchmark with all the threads can access the resources in a fixed period of time will be considered as the progress indicator (c_{L1}, c_{L2}, and c_{L3}). Intuitively, faster progress (more accesses in a fixed time) will be an indicator of lower contention levels, and slower progress will be an indicator of higher contention levels in a specific duration of time. The size of the array each micro-benchmark will access is set proportionate to the amount of cache available in the targeted cache level. With the first access to each data element, it will be brought to the cache, and the next few accesses will be faster for the fetched data. However, if due to contention some data elements have been pushed out of the cache, the next access will be slower in order to fetch the data back to the cache, resulting in longer access time and lower number accesses in a fixed period of time by the micro-benchmarks.

Memory Latency: Modern applications' performance has shown more sensitivity toward memory latency [35]. Memory latency micro-benchmark estimates the contention by accessing an array of 2GB with the stride size of 128 with only one thread to do a pointer-chasing. In pointer-chasing, each memory location contains the address of the next memory location to access, and one thread is utilized to prevent inter-thread contention. The number of memory accesses ($c_{latency}$) in a fixed duration of time will be considered as an indicator of contention in memory latency. Intuitively, with larger latency, the number of accesses to memory will decrease, and vise versa.

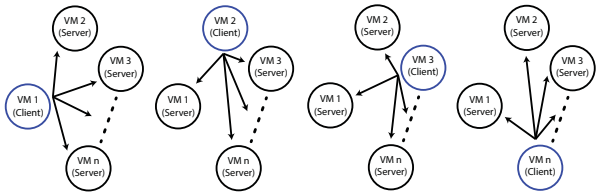


Fig. 1: Network Bandwidth/Latency Profiling for All Possible Pairs of VMs

Network Bandwidth/Latency: Network contention can be represented by change in network bandwidth or latency, affecting the communication time of collaborating VMs and consequently their performance. The source of network contention can be an adapter shared between multiple hosted VMs on the same hardware or data center network traffic on VMs' communication route. To measure existing network bandwidth/latency between two VMs we use iPerf3 micro-benchmark suite [26]. For iPerf3 to measure network bandwidth/latency one VM acts as a server, listening for the incoming data on a specific port from the client VM. IPerf3 floods the network bandwidth with data that will be sent from the client to the server to measure the available network bandwidth and corresponding connection latency. The results of measured network bandwidth and latency for a pair of client and server will be denoted as $c^{i,j}_{Net-Band}$ and $c^{i,j}_{Net-Lat}$, in which i and j are the client and server VM numbers correspondingly. For each VM as a client, the experiment will repeat with all the other collaborating VMs as the server, generating a tuple of measured network bandwidth/latency. Equation 1 shows such a tuple for network bandwidth where n is the total number of available VMs in the group and i is the client VM number.

$$c^{i}_{Net-Band} = (c^{i,1}_{Net-Band}, \cdots, c^{i,i-1}_{Net-Band}, c^{i,i+1}_{Net-Band}, \cdots, c^{i,n}_{Net-Band}) \quad (1)$$

An increase in network bandwidth and a decrease in connection latency represents a decrease in network contention affecting the performance of collaborating VMs favorably, and vise versa. In this work, network bandwidth and latency between all pairs of the VMs have been measured in both directions (From client to server and server to client) by changing the client and server roles as shown in Figure 1.

B. Profiling Duration

Longer profiling duration may better reflect variations of changes in the system contention level, but it will impose high

overhead. Therefore, shorter profiling duration is favorable to reduce the overhead as much as possible. Yet, it may not be able to capture all contention information, affecting the prediction accuracy. For this reason, to balance the profiling overhead and model's accuracy, all considered micro-benchmarks are capable of sub-second profiling duration. We have investigated the effects of profiling duration on model's prediction accuracy, the results show that 0.4 second profiling duration will provide accurate enough information about the probed resources for performance modeling and prediction. Moreover, increasing the profiling duration up to 3 seconds only has negligible improvement on the model's accuracy. As a result, in all the experiments, 0.4 second profiling duration has been considered.

All micro-benchmarks evaluating the contention in local resources (CPU, Memory bandwidth/latency, I/O, L1, L2, and L3 cache) will be executed sequentially on each VM, and the process happens for all collaborating VMs in parallel. However, for profiling the contention in shared resources between all the VMs in the same group (Network bandwidth/latency), the micro-benchmarks will execute on each pair of VMs one after the other to prevent the intra-VM profiling contention and conflict. Right after the execution of all the micro-benchmarks, the target application executes on all the collaborating VMs in the group. This will provide the resource contention experienced by each collaborating VM and corresponding application performance. The results will be used for model training and future predictions which will be discussed in Sec. IV.

IV. Multi-VM Performance Modeling and Prediction

A. Overview

The impact of resource contention depends on its source and intensity, creating varying contention levels. For multi-VM cloud applications, their overall performance depends on the experienced contention level by each collaborating VM in a group. A VM may experience high contention levels and slow down the whole group, negatively affecting the performance of the application. The effect of contention levels experienced by the collaborating VMs on application performance can be modeled using machine learning algorithms. This process consists of two phases, performance modeling and prediction.

In the modeling phase, in each iteration, micro-benchmarks will execute on all the collaborating VMs before the target application execution to measure the contention experienced by the VMs and corresponding application performance. The results of this step is denoted as $\{p^1_{vm}, p^2_{vm}, \cdots, p^i_{vm}, \cdots, p^n_{vm}, t_{app}\}$ where p^i_{vm} is a tuple of micro-benchmarks execution results on i^{th} VM $(c^i_{cpu}, c^i_{L1}, \cdots, c^i_{Net-Band}, c^i_{Net-Lat})$, n is the number of collaborating VMs in the group, and t_{app} is the measured application performance (e.g., execution time) for the profiled level of resource contention. This process will be repeated for multiple iterations to collect required performance data. Then, the collected performance data will be provided to the machine learning algorithm for model training. Using the trained model, future predictions can be made by providing the profiling results ($\{p^1_{vm}, p^2_{vm}, \cdots, p^i_{vm}, \cdots, p^n_{vm}\}$) to the model as inputs for performance prediction ($\{t_{app}\}$). The performance model described above can be built using different techniques described in the following sections.

B. Predictive Models

Centralized Neural Network: Feed-forward Neural Networks (NN) have been shown to have great accuracy in modeling polynomial and non-polynomial behaviors of cloud running applications [14, 33]. Using NN, parameters for function f presented in Equation (2) can be found to model the application's performance for later predictions.

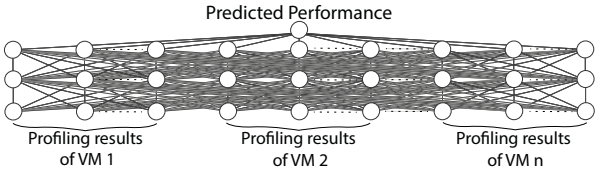
$$t_{app} = f(p^1_{vm}, p^2_{vm}, \cdots, p^n_{vm}) \tag{2}$$

However, to train a NN it is required to send all the profiling information collected on all the collaborating VMs to a central VM for modeling and prediction. Figure 2.a shows a NN with profiling information collected from all the collaborating VMs as inputs. The number of neurons in the first layer is equal to the number of profiled resources in all the collaborating VMs. Due to the sensitivity of NN to hyper-parameters (e.g., layers, neurons, dropout, etc.), we utilized HyperOpt [25], a hyperparameter optimization library, which will find the optimal parameters using a Tree-structured Parzen Estimator (TPE) algorithm [43]

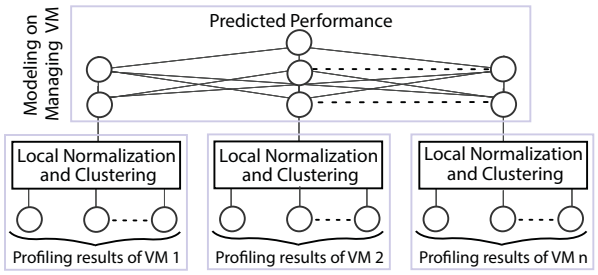
Distributed and Hierarchical Clustering: To distribute the required processing for model training among the collaborating VMs, and to reduce the large volume of data transfer to a central location for model training and prediction, we use hierarchical models [51]. As shown in Figure 2.b each VM will independently extract the information from the unprocessed profiling data, preprocess the data by normalization, and feed the normalized data to an unsupervised clustering algorithm. Then, instead of sending unprocessed profiling data, the output of clustering algorithm will be sent to the managing VM for aggregation and final modeling or prediction. In the modeling phase, the clustering algorithm g^i on i^{th} VM , as shown in the equation 3, will assign a group number to each local profiling data tuple in such a way that data tuples in each group will be more similar to each other. The group numbers assigned represent the contention levels experienced by i^{th} VM for the profiled duration of time.

$$cl^i = g^i(p^i_{vm}) \tag{3}$$

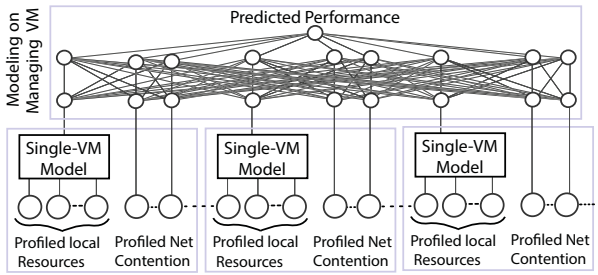
Then, group numbers, instead of actual profiling data, will be sent to the managing VM for final performance modeling using NN, as shown in equation 4. Here, the NN will model the correlation between the contention level experienced by each collaborating VM and the overall application performance.



a. Centralized Neural Network



b. Distributed and Hierarchical Clustering



c. Single-VM Model with Hierarchical Framework

Fig. 2: Frameworks for Performance Modeling and Prediction

$$t_{app} = f(cl^1, cl^2, \cdots, cl^n) \tag{4}$$

In the prediction phase, the contention in resources of each collaborating VM (i.e. i^{th} VM) will be profiled by the execution of the micro-benchmarks. Then, the tuple of profiled information for different resources will be fed to the locally trained clustering algorithm (g^i) which will output the experienced contention level (cl^i) by the VM. Finally, the experienced contention levels by all the VMs need to be sent to the managing VM for aggregation and performance prediction using the previously trained NN model (f).

For our hierarchical model, four clustering algorithms have been considered to evaluate their effects on the final models' accuracy. The considered clustering algorithms are KMeans [22], Affinity Propagation [18], Mean Shift [49], and Birch [53] clustering. Each clustering algorithm uses different techniques to group the data points. KMeans assigns the points to different clusters with the goal of minimizing variance within the cluster, Affinity Propagation identifies a subset of representative examples using message passing concept, Mean Shift uses the density of data points in feature space to identify clusters, and Birch uses the tree structure for forming the clusters. Similarly here, NN and clustering algorithms have been optimized using HyperOpt hyper-parameters optimization library to find the best parameters for modeling.

Single-VM Model with Hierarchical Framework: We have also considered the approach of adopting the existing single-VM performance models by incorporating into the hierarchical framework. Incorporating single-VM models help aggregate the contention experienced in local VM resources with almost no training in lower-level VMs. For details of the steps involved to generate a single-VM model, we refer the readers to *uPredict* framework [33]. In the modeling phase, first, micro-benchmarks will be executed on all the collaborating VMs. Then, the normalized profiling results of the local resources will be fed as inputs to the single-VM model on each VM. Here, the output will represent the contention level experienced in the system local resources without the consideration of contention in intra-VM communications (Network bandwidth/latency). However, the experienced contentions in intra-VM communications are required for accurate performance modeling and prediction of multi-VM applications. For this reason, in addition to the contention level estimated in local resources by the single-VM model, the measured contention of inter-VM communications will be sent to the managing VM for final aggregation and modeling using NN. In the prediction phase, profiling results of local resources will be fed into the single-VM model for aggregation. Then, the output along with the measured network contention will be sent to the managing VM for performance prediction. Figure 2.c shows the hierarchical performance modeling and prediction by re-utilizing pre-existing single-VM model.

V. Evaluations and Discussions

We have conducted extensive experiments to evaluate the feasibility of using *DiHi* framework for performance modeling and prediction of multi-VM cloud running applications. First, we provide our experimental setups. Then, we discuss the models' accuracy in predicting the performance of applications running on a group of 2 VMs. Finally, the scalability of *DiHi* framework is evaluated considering its accuracy in predicting the performance of applications running on a group of 4 VMs.

A. Experiments Setup

Representative Benchmarks: We considered a total of six benchmark applications from Intel HiBench [24], CloudSuite [16], and NAS Parallel Benchmarks (NPB) [3]. It includes *Bayesian classification (bayes)* and *K-means clustering (kmeans)* with small inputs from Intel HiBench. *In-Memory analytics (inMem)* and *Graph analytics (graph)* are selected from CloudSuite. For *Graph analytics* benchmark, the only default data input, and for the *In-Memory analytic*, the largest data input were used. From NPB, *lu* and *ep* with class C input size were considered. These benchmark applications

have been selected to representing a wide range of Multi-VM cloud running applications. HiBench benchmarks are selected to represent Hadoop workload requiring many disk operations, CloudSuite benchmarks are selected to represent Spark inMemory operations used in businesses for many data analytic workloads, and NPB benchmarks as HPC applications utilizing CPU heavily. Note that all the selected benchmarks are setup to run on a group of VMs.

VM Configurations: We have conducted experiments on three different clouds: private, Chameleon and Google Compute Engine (GCE). For the private cloud, OpenStack Ocata was installed on two servers, each with two Intel Xeon E5-2630 processors (with a total of 16 cores on each server) and 128GB memory. For experiments conducted on the private cloud, due to the limited number of available servers, only a group of 2 VMs are utilized to execute the applications with each VM hosted on one of the servers. This setup enables the control of background interference experienced by each VM separately. Each VM will utilize 8 vCPUs and 16 GB of memory to run the selected applications (total of 16 vCPUs and 32 GB of memory on both VMs). Resource contention is introduced to the system by launching up to seven background VMs with the same configuration on each server. Interference configuration will be chosen randomly at run-time, and 100 data points will be collected, then the interference configuration will change. This process will repeat for 10 iterations to collect a total of 1000 data points. The interfering VMs will be chosen randomly at runtime to execute one of the CPU, Memory, I/O, and network intensive applications from iBench [11], FIO [27], and GNU Wget [19]. Wget will download a 1GB Linux Mint iso image from a host that is directly connected to the same network switch. For Chameleon and GCE, two sets of experiments have been conducted with 2 and 4 VMs group sizes. On Chameleon as a scientific cloud [7], for group of 2 VMs *m1.xlarge* instance has been considered with 8 cores and 16GB of RAM, and *m1.large* VM instances with 4 cores and 8 GB of RAM has been selected for a cluster of 4 VMs, with both instances having 40GB of disk drive. For GCE, *e2-standard-8* and *e2-standard-4* instances with 8 and 4 cores and 16 and 8 GB RAM with 60 GB of disk drive are considered for cluster sizes of 2 and 4 VMs, respectively. These instances are chosen as the closest configuration to our private cloud experiment. For all the experiments Ubuntu Server 16.04 is used as the OS for the VMs.

Data Collection: We chose 0.4 second profiling duration as our experiments indicate that increasing the profiling duration up to 3 seconds has a negligible effect on the models' accuracy. For the chosen profiling duration, all the micro-benchmarks will execute prior to the execution of the target application in all collaborating VMs in parallel, except the network bandwidth and latency micro-benchmarks. The execution of network-related micro-benchmarks will initiate from each VM one by one to prevent the interference generated by the concurrent execution. In this way, the profiling information will provide us the in-situ contention level experienced by each collaborating VM, followed by the application's performance executed on all the VMs in the group. This process will repeat to collect 1,000 data points for each application, which was ran over the course of 3 months on the private cloud and a month on Chameleon and CGE.

For each benchmark application, 80% of the collected data points are randomly selected to train the model (60% for training and 20% for validation), and 20% have been used for model testing.

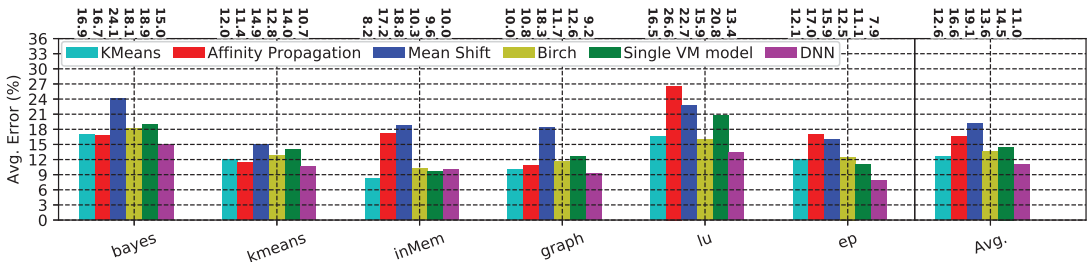
Model Implementation: To implement the predictive models we used existing open-source libraries. For clustering algorithms, scikit-learn version 0.19.2 [41] was used, and the NN models were implemented using TensorFlow version 1.12 [1]. Due to the sensitivity of the models to the hyper-parameters, HyperOpt optimization library version 0.1.1 [25] - was employed, where 100 iterations were used to find the best configurations from the provided search space.

B. Evaluation of DiHi with 2 VMs:

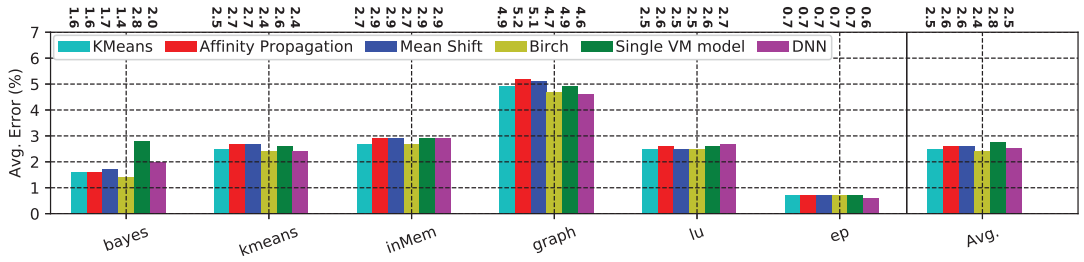
We have evaluated the *DiHi* framework in a private cloud, Chameleon and GCE with 2 VMs to run the specified benchmarks. Figure 3 shows the prediction error of the considered predictive models in *DiHi* for six benchmarks. Here, for each benchmark, 80% of the collected data is randomly selected for model training and evaluation, and the remaining 20% is used for testing the model's accuracy.

By comparing Figure 3 part a, b, and c, it can be observed that the most intuitive approach, collecting all the performance data from collaborating VMs and feeding as inputs at the same time to a NN with hyper-parameter optimization, performs the best on average for the data collected in all three different clouds. A possible reason for the better accuracy, compared to other predictive schemes, can be the ability to better tune the model in a central location with all available training data. However, the other five predictive models based on *DiHi* show very close accuracy. In addition, *DiHi*-based models reduce the load of transferring unprocessed data to a central location and distribute the required processing power among all collaborating VMs.

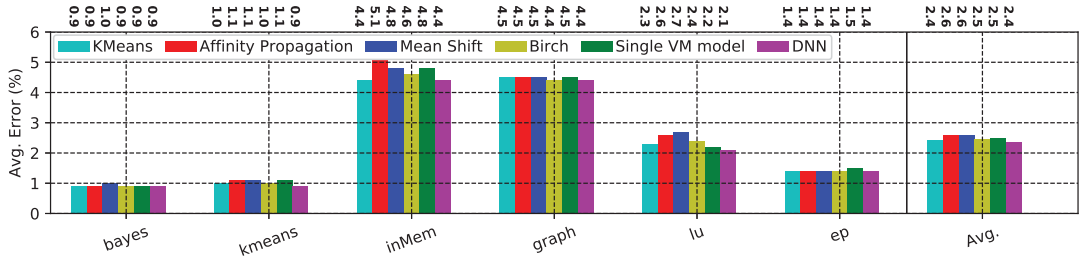
A closer look at the accuracy of predictive models in Figure 3 show that the clustering algorithms employed in *DiHi*, to group similar data points in each VM, have a quite large effect on the final model's accuracy. In particular, Figure 3.a shows a 6.5% increase in the model's accuracy by employing the KMeans clustering algorithm instead of Affinity Propagation, with KMeans clustering performing as the best algorithm that can be incorporated into *DiHi* framework. Moreover, by comparing the accuracy of the models in private (Fig 3.a) and public clouds (Fig 3.b and c), more stable results in a narrow fluctuation range can be noticed for public clouds. Lower contention experienced in public clouds can be the reason for better prediction accuracy of performance by all considered



a. Private cloud

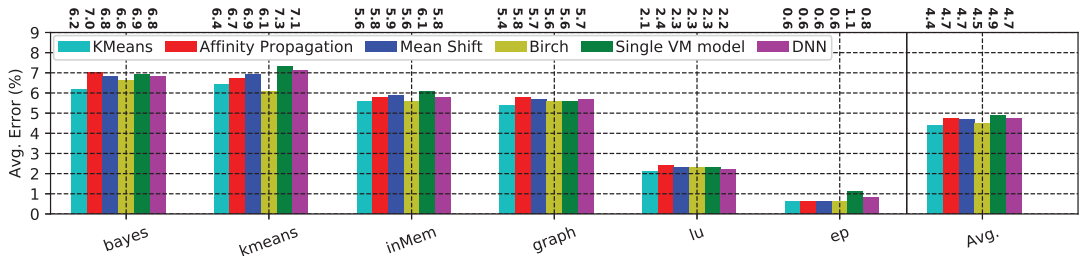


b. Chameleon

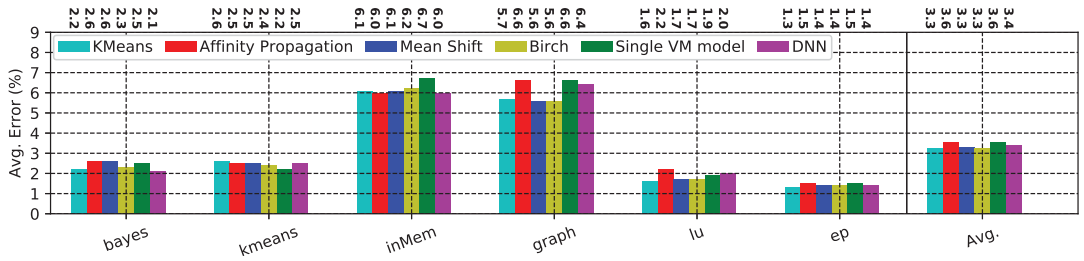


c. Google Compute Engine

Fig. 3: Prediction errors for different models with 2VMs of 8-core and 0.4-second profiling



a. Chameleon



b. Google Compute Engine

Fig. 4: Prediction errors for different models with 4VMs of 4-core and 0.4-second profiling

algorithms. Resource contention in public clouds accounts for an average of 20% to 25% performance fluctuation, however, in private cloud experiments contention rises to almost 300%. It should be noted that higher contention in private cloud is considered mainly to test the predictive models in a highly contentious cloud environment. In a high contention environment, more accurate predictive models and algorithms better show their effectiveness in modeling the performance. However, in low levels of contentions, even less sophisticated models can be found beneficial to model the applications' performance.

Figure 3 shows that *DiHi* framework employed with the single-VM model provides acceptable accuracy for performance modeling and prediction of multi-VM applications. The idea behind incorporating a single-VM model is to evaluate the possibility of re-utilizing the existing models trained on a single-VM execution data to provide a notion of experienced contention in local resources. Here, the single-VM model will aggregate the result of experienced contention in the local resources with almost no training, reducing the training time and required amount of data transfer to the managing VM. Figure 3.a shows that in private cloud, incorporating the single-VM model into *DiHi* results in 14.5% error on average, which is close to the results of *DiHi* with KMeans clustering algorithm, with less than a 2% increase in the error. Moreover, the performance of *DiHi* with the single-VM model is very close to other algorithms in a low contentious environment. Figure 3, part b, and c show only a 0.3% and 0.1% increase in the error for *DiHi* with Single-VM model, compared to the best achievable accuracy.

In the private cloud, most of the applications will experience similar intensity of contention in different resources during their execution time, resulting in comparable accuracy of the models for different applications. However, in public clouds, there is a probability of a high contention level for a long duration of time in a specific set of resources, increasing the prediction error for specific applications heavily utilizing the resources. This can be considered as the reason for lower accuracy for the Spark in-memory analytic applications (CloudSuite *graph* and *inMem*). However, the prediction error is in an acceptable range with around 5% error.

C. Evaluation of DiHi with 4 VMs:

Due to resource limitations, experiments on a group of 4 VMs were conducted only on Chameleon and GCE, as presented in Figure 4. *DiHi* predictive models show accurate performance prediction with all clustering algorithms and incorporated single-VM model with less than 5% and 4% average error on Chameleon and GCE, respectively (Figure 4 a and b). This demonstrates accurate contention level estimation of each collaborating VM and final performance prediction. Moreover, the performance of all the models are very close to each other in public clouds for each specific application (Figure 4), however, *DiHi* with KMeans clustering algorithm shows more robust performance and higher accuracy compared to the centralized NN model.

Comparing the accuracy of the results for the execution of the considered applications on 2 and 4 VM group sizes on both Chameleon and GCE, Figures 3 and 4, show that there is less than a 2% increase in the models' average error for larger cluster size. This represents that, with increasing the number of VMs in the group the models are still very accurate with a maximum of 7% prediction error and an average of less than 5% for the considered applications and algorithms.

Based on the results, as presented in Figure 3 and 4 for Chameleon and GCE, the performance of HiBench applications running on Hadoop with disk-intensive operations are more difficult to model in larger cluster sizes, and show more unpredictable behaviours. However, other benchmark applications show very similar accuracy on different clouds and cluster sizes. This can stem from very high and sporadic disk utilization of Chameleon, which has been resulted in limiting their main disk size to 40GB for all instances in recent updates to reduce the disproportionate disk contention.

VI. Conclusions

We proposed *DiHi*, a distributed and hierarchical performance modeling and prediction framework, for multi-VM applications running on contentious cloud environment. In *DiHi*, to probe the resource contention in a wide range of system resources, a comprehensive set of micro-benchmarks is considered. Our evaluation results show that the proposed *DiHi* framework can accurately estimate the contention level experienced by collaborating VMs for hierarchical performance modeling with different clustering algorithms and existing single-VM models. Although the traditional approach to collect all performance data in a central location for model tuning seems more accurate in a high contentious cloud environment, *DiHi* framework provides comparable performance in the same environment and better accuracy in low levels of contention. Finally, *DiHi* has acceptable prediction accuracy with different cluster sizes, and an increase in the size of the cluster won't considerably affect the performance.

References

[1] M. Abadi et al. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] O. Alipourfard, H. H. Liu, J. Chen, S. Venkataraman, M. Yu, and M. Zhang. Cherrypick: Adaptively unearthing the best cloud configurations for big data analytics. In *USENIX Symp. on Networked Systems Design and Implementation (NSDI)*, pages 469–482, March 2017.

[3] D. H. Bailey. Nas parallel benchmarks. In *Encyclopedia of Parallel Comput.*, pages 1254–1259. Springer, 2011.

[4] M. Baughman, R. Chard, L. Ward, J. Pitt, K. Chard, and I. Foster. Profiling and predicting application performance on the cloud. In *IEEE/ACM 11th International Conference on Utility and Cloud Computing (UCC)*, 2018.

[5] R. Begam, H. Moradi, W. Wang, and D. Zhu. Flexible vm provisioning for time-sensitive applications with multiple execution options. In *Proc. of the IEEE Int'l Conf. on Cloud Comput. (CLOUD)*, 2018.

[6] A. Castiglione, M. Gribaudo, M. Iacono, and F. Palmieri. Modeling performances of concurrent big data applications. *Software: Practice and Experience*, 45(8):1127–1144, 2015.

[7] Chameleon Cloud. https://https://chameleoncloud.org.

[8] F. J. Clemente-Castello, B. Nicolae, R. Mayo, and J. C. Fernandez. Performance Model of MapReduce Iterative Applications for Hybrid

Cloud Bursting. *IEEE Trans. on Parallel and Distrib. Syst. (TPDS)*, 2018.
[9] Jeffrey Dean and Luiz André Barroso. The tail at scale. *Communications of the ACM*, 56(2):74–80, 2013.
[10] J. Dejun, G. Pierre, and C. Chi. Resource provisioning of web applications in heterogeneous clouds. In *USENIX conf. on Web Appl. Develop.* USENIX Association, 2011.
[11] C. Delimitrou and C. Kozyrakis. iBench: Quantifying interference for datacenter applications. In *IEEE Int'l Symp. on Workload Characterization (IISWC)*, 2013.
[12] C. Delimitrou and C. Kozyrakis. Paragon: QoS-aware Scheduling for Heterogeneous Datacenters. In *Proc. of the Int'l Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2013.
[13] C. Delimitrou and C. Kozyrakis. Quasar: Resource-efficient and QoS-aware Cluster Management. In *Proc. of Int'l Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2014.
[14] N. Ebadi, B. Lwowski, M. Jaloli, and P. Rad. Implicit life event discovery from call transcripts using temporal input transformation network. *IEEE Access*, 7:172178–172189, 2019.
[15] B. Farley, A. Juels, V. Varadarajan, T. Ristenpart, K. Bowers, and M. Swift. More for Your Money: Exploiting Performance Heterogeneity in Public Clouds. In *Proc. of the ACM Symp. on Cloud Comput. (SoCC)*, 2012.
[16] M. Ferdman et al. Clearing the clouds: A study of emerging scale-out workloads on modern hardware. *Proc. of Int'l Conf. on Architectural Support for Programming Languages and Operating Syst. (ASPLOS)*, 2012.
[17] Armando Fox et al. Above the clouds: A berkeley view of cloud computing. Technical report, Dept. Electrical Eng. and Comput. Sciences, University of California, Berkeley, Rep. UCB/EECS, 2009.
[18] Brendan J. Frey and Delbert Dueck. Clustering by passing messages between data points. *Science*, 315(5814):972–976, 2007.
[19] GNU Wget. https://www.gnu.org/software/wget/.
[20] Google Compute Engine. https://cloud.google.com/.
[21] S. Govindan, J. Liu, A. Kansal, and A. Sivasubramaniam. Cuanta: Quantifying Effects of Shared On-chip Resource Interference for Consolidated Virtual Machines. In *Proc. of ACM Symp. on Cloud Comput. (SoCC)*, 2011.
[22] J. Hartigan and M. Wong. Algorithm as 136: A k-means clustering algorithm. *Journal of the royal statistical society. series c (applied statistics)*, 28(1):100–108, 1979.
[23] R. Hecht-Nielsen. Theory of the backpropagation neural network. In *Neural Networks for Perception*, pages 65–93. Elsevier, 1992.
[24] S. Huang, J. Huang, J. Dai, T. Xie, and B. Huang. The HiBench benchmark suite: Characterization of the mapreduce-based data analysis. In *IEEE Int'l Conf. on Data Engineering Workshops (ICDEW)*, 2010.
[25] HyperOpt. https://github.com/hyperopt/hyperopt.
[26] iPerf. http://https://iperf.fr/.
[27] J. Axboe. https://github.com/axboe/fio.
[28] S. A. Jyothi, C. Curino, I. Menache, S. M. Narayanamurthy, A. Tumanov, J. Yaniv, R. Mavlyutov, I. Goiri, S. Krishnan, J. Kulkarni, and S. Rao. Morpheus: Towards Automated SLOs for Enterprise Clusters. In *Symp. on Operating Systems Design and Implementation (OSDI)*, 2016.
[29] J. Kumar and A. K. Singh. Workload prediction in cloud using artificial neural network and adaptive differential evolution. *Future Generation Computer Systems*, 81:41 – 52, 2018.
[30] LMbench. http://www.bitmover.com/lmbench/.
[31] G. Mariani, A. Anghel, R. Jongerius, and G. Dittmann. Predicting cloud performance for hpc applications: A user-oriented approach. In *2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, pages 524–533, 2017.
[32] J. Mars, L. Tang, R. Hundt, K. Skadron, and M. L. Soffa. Bubble-Up: Increasing Utilization in Modern Warehouse Scale Computers via Sensible Co-locations. In *Annual IEEE/ACM Int'l Symposium on Microarchitecture (MICRO)*, 2011.
[34] H. Moradi, W. Wang, and D. Zhu. Adaptive performance modeling and prediction of applications in multi-tenant clouds. In *Proc. of the IEEE*
[33] H. Moradi, W. Wang, A. Fernandez, and D. Zhu. uPredict: A user-level profiler-based predictive framework for single vm applications in multi-tenant clouds. In *Proc. of the IEEE Int'l Conf. on Cloud Engineering (IC2E)*, 2020.
Int'l Conf. on High Performance Comput. and Communications (HPCC), 2019.
[35] R. Murphy. On the effects of memory latency and bandwidth on supercomputer application performance. In *IEEE Int'l Symp. on Workload Characterization*, 2007.
[36] D. Novaković, N. Vasić, S. Novaković, D. Kostić, and R. Bianchini. DeepDive: Transparently Identifying and Managing Performance Interference in Virtualized Environments. In *Annual Technical Conf. (USENIX ATC)*, 2013.
[37] F. Romero and C. Delimitrou. Mage: Online and Interference-Aware Scheduling for Multi-Scale Heterogeneous Systems. In *Proc. of Int'l Conf. on Parallel Architectures and Compilation Techniques (PACT)*, 2018.
[38] M. Safaei Pour, A. Mangino, K. Friday, M. Rathbun, E. Bou-Harb, F. Iqbal, S. Samtani, J. Crichigno, and N. Ghani. On data-driven curation, learning, and analysis for inferring evolving internet-of-things (IoT) botnets in the wild. *Computers & Security*, page 101707, 2019.
[39] A. Sahba and J. J. Prevost. Hypercube based clusters in cloud computing. In *World Automation Congress (WAC)*, 2016.
[40] J. Scheuner and P. Leitner. Estimating Cloud Application Performance Based on Micro-Benchmark Profiling. In *Int'l Conf. on Cloud Comput. (CLOUD)*, 2018.
[41] Scikit-learn. http://scikit-learn.org/stable/.
[42] Y. Sfakianakis, C. Kozanitis, C. Kozyrakis, and A. Bilas. QuMan: Profile-based Improvement of Cluster Utilization. *ACM Trans. on Architecture and Code Optimization (TACO)*, 15(3):27:1–27:25, 2018.
[43] J. Snoek, H. Larochelle, and R. P. Adams. Practical Bayesian Optimization of Machine Learning Algorithms. In *Adv. in NIPS.*, pages 2951–2959, 2012.
[44] B. C. Tak, C. Tang, H. Huang, and L. Wang. Pseudoapp: Performance prediction for application migration to cloud. In *2013 IFIP/IEEE International Symposium on Integrated Network Management (IM 2013)*, pages 303–310, 2013.
[45] L. Tang, J. Mars, W. Wang, T. Dey, and M. L. Soffa. ReQoS: Reactive Static/Dynamic Compilation for QoS in Warehouse Scale Computers. In *Proc. of Int'l Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2013.
[46] B. Varghese, O. Akgun, I. Miguel, L. Thai, and A. Barker. Cloud benchmarking for maximising performance of scientific applications. *IEEE Transactions on Cloud Computing*, 7(1):170–182, 2019.
[47] K. Wang and M. M. H. Khan. Performance prediction for apache spark platform. In *2015 IEEE International Conference on High Performance Computing and Communications (HPCC)*, 2015.
[48] F. Xu, H. Zheng, H. Jiang, W. Shao, H. Liu, and Z. Zhou. Cost-effective cloud server provisioning for predictable performance of big data analytics. *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, 30(5):1036–1051, May 2019.
[49] Y. Cheng. Mean shift, mode seeking, and clustering. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17(8):790–799, 1995.
[50] N. J. Yadwadkar, B. Hariharan, J. E. Gonzalez, B. Smith, and R. H. Katz. Selecting the best vm across multiple public clouds: A data-driven performance modeling approach. In *Proc. of the Symp. on Cloud Computing (SoCC)*, 2017.
[51] L. Yao and Z. Ge. Distributed parallel deep learning of hierarchical extreme learning machine for multimode quality prediction with big process data. *Engineering Applications of Artificial Intelligence*, 81:450 – 465, 2019.
[52] Y. Yu, V. Jindal, F. Bastani, F. Li, and I. Yen. Improving the smartness of cloud management via machine learning based workload prediction. In *IEEE Annual Computer Software and Applications Conference (COMPSAC)*, 2018.
[53] T. Zhang, R. Ramakrishnan, and M. Livny. Birch: An efficient data clustering method for very large databases. In *Proc. of the ACM SIGMOD Int'l Conf. on Management of Data*, 1996.