Holistic VM Placement for Distributed Parallel Applications in Heterogeneous Clusters

Seontae Kim, Nguyen Pham, Woongki Baek, *Member, IEEE*, and Young-ri Choi, *Member, IEEE*

Abstract—In a heterogeneous cluster, virtual machine (VM) placement for a distributed parallel application is challenging due to numerous possible ways of placing the application and complexity of estimating the performance of the application. This study investigates a holistic VM placement technique for distributed parallel applications in a heterogeneous cluster, aiming to maximize the efficiency of the cluster and consequently reduce the costs for service providers and users. The proposed technique accommodates various factors that have an impact on performance in a combined manner. First, we analyze the effects of the heterogeneity of resources, different VM configurations, and interference between VMs on the performance of distributed parallel applications with a wide diversity of characteristics, including scientific and big data analytics applications. We then propose a placement technique that uses a machine learning algorithm to estimate the runtime of a distributed parallel application. To train a performance estimation model, a distributed parallel application is profiled against synthetic workloads that mostly utilize the dominant resource of the application, which strongly affects the application performance, reducing the profiling space dramatically. Through experimental and simulation studies, we show that the proposed placement technique can find good VM placement configurations for various workloads.

Index Terms—Heterogeneous clusters, distributed parallel applications, VM placement algorithm, machine learning based performance model

1 Introduction

Heterogeneity of hardware configurations for physical nodes exists in a cluster, as physical machines are continuously purchased over time [1], [2], [3]. In most cases, a cluster consists of physical nodes of several different types. Each type of physical nodes is configured differently in terms of the CPU microarchitecture and clock speed, the number of cores, the amount of memory, and network and storage settings, providing different performance capabilities. In heterogeneous clusters, various applications can be deployed and executed together on the same node, due to advances in multicore and virtualization technologies.

A heterogeneous cluster is commonly used to run multiple distributed parallel applications. For a distributed parallel application, multiple virtual machines (VMs) form a *virtual cluster* (VC) to run the application in parallel and coordinate their execution by exchanging messages across the VMs. Distributed parallel applications are popularly employed to solve large scale complex scientific problems such as those in molecular dynamics [4], [5] and computational fluid dynamics [6]. They are also used to process huge amounts of data, as in Hadoop [7] and Spark [8].

In a heterogeneous cluster, for all VMs which execute a distributed parallel application together, it may not be possible to allocate homogeneous resources on which the application shows the best performance. This occurs especially with private cloud clusters on small and medium scales, and even with clusters on a public cloud, where some types of resources have been reserved for the cluster over a long term to reduce the cost significantly (as in Reserved Instances in Amazon EC2 [9]). To improve the resource utilization and performance of the cluster, we can configure the VMs of the application with heterogeneous nodes.

When different resources are used to execute a distributed parallel application, there are numerous possible ways to place the VMs of the application depending on factors such as the number of different node types used for the VMs, the number of VMs running on each type of (physical) nodes, and the number of the nodes used for each type. Moreover, even for the same VC configuration, the performance of the application widely varies depending on co-running VMs or applications, i.e., *co-runners*, which are executed on the same nodes [2], [10], [11], [12].

When running a distributed parallel application, the heterogeneous hardware configuration and/or different levels of interference on each of the nodes can slow down some of the VMs. Depending on how parallelism and synchronization are implemented for a distributed parallel application, the outcome can differ. For distributed parallel applications which are *loosely coupled* or have a load balancing feature such as big data analytics applications [7], [8] and many-task computing applications [13], a few slow VMs may not affect the performance noticeably and using favored resources in part may improve the performance of the application. However, for *tightly coupled* applications such as scientific

• *The authors are with the School of Electrical and Computer Engineering, UNIST, Ulsan 44919, Republic of Korea.*
E-mail: {stkim, nguyenpham, wbaek, ychoi}@unist.ac.kr.

Manuscript received 8 Apr. 2018; revised 24 Oct. 2018; accepted 20 Dec. 2018. Date of publication 1 Jan. 2019; date of current version 8 Oct. 2021.
(Corresponding author: Young-ri Choi.)
Digital Object Identifier no. 10.1109/TSC.2018.2890668

MPI-based applications, one slow VM may cause the different execution progress rates over VMs, degrading the final performance significantly. Moreover, if the application has poor parallel efficiency, adding more VMs does not improve the performance, unlike loosely coupled applications. Thus, estimating the application performance is not trivial, as the effect of each factor such as the heterogeneity of resources, interference, VM configuration, and parallelism pattern on the performance is not clear.

Maximizing the efficiency of a cluster is crucial for service providers such as cloud providers, as doing so reduces the overall costs with an eventual price reduction for users [3]. Moreover, a strategy for efficient VM placement is necessary for providers to provide high quality services to users and to enhance user satisfaction. However, in a heterogeneous cluster, finding the best VM placement for distributed parallel applications, which maximizes the overall performance, is challenging due to the intractably large search space and complexity of estimating their performance.

Earlier studies investigate scheduling techniques which take into account the heterogeneity of hardware configurations and/or interference among applications [2], [3], [11], [12], [14], [15]. However, some prior works mainly consider single node applications [2], [12], [14], and a homogeneous cluster is assumed for an interference modeling technique for distributed parallel applications [11]. An interference and heterogeneity aware scheduling technique focuses on supporting applications popularly used in large scale clouds or datacenters such as distributed analytics frameworks, latency critical services, and web services [15]. The technique employs a greedy approach to allocate and assign the least amount of resources while still satisfying quality of service (QoS) constraints of an application, thus reducing the search space for placement.

This paper investigates a holistic VM placement technique for various types of distributed parallel applications in a heterogeneous cluster, aiming to maximize the overall performance for the benefit of service providers and users. The proposed technique accommodates various factors that have an impact on performance in a combined manner, while reducing the search space and cost for the best VM placement. First, we analyze the effects of the heterogeneity of resources, different VM configurations, and interference between VMs on the performances of various distributed parallel applications. We then propose a placement technique that uses a machine learning algorithm to estimate the runtime of a distributed parallel application for various VM placement configurations.

For a heterogeneous cluster, the total search space for the VM placement of distributed parallel applications is intractable. Thus, we limit the candidate placements for the applications to a treatable subset of all possible placements in the cluster. To find the best VM placement from the huge search space, we also devise a VM placement algorithm based on simulated annealing. To generate training samples for a performance estimation model, a distributed parallel application is profiled against synthetic workloads that mostly utilize the *dominant resource* of the application, which strongly affects the application performance, reducing the profiling space dramatically.

TABLE 1
Specifications of our Heterogeneous Cluster

Type	T1	T2
CPU	Intel quad-core I7-3770 (IvyBridge) 3.40 GHz	Intel hexa-core E5-2620 (SandyBridge) 2.0 GHz
# sockets	1	2
L3	8 MB/socket	15 MB/socket
Memory	16 GB	64 GB (32 GB/socket)
# nodes	8	4
Network (Thr.)	1GE (∼ 70 MB/s)	1GE (∼ 110 MB/s)
Storage (Thr.)	7,200 RPM HDD (∼ 137 MB/s)	7,200 RPM HDD (∼ 109 MB/s)

The main contributions of this paper are as follows:

- We analyze the performance of various MPI-based and big data analytics applications from SpecMPI 2007, NAS Parallel Benchmarks (NPB), Spark, Hadoop, and molecular dynamics simulators in a heterogeneous cluster.
- We explore the correlation between the dominant resource usage and performance of parallel applications ultimately to lower the profiling cost.
- To estimate the performance, we apply a machine learning algorithm in order to deal with the complex performance modeling of a parallel application, which must consider many the relevant factors discussed above in a comprehensive manner.
- We show that it is feasible to build a general model which can estimate the performance of various Hadoop and Spark applications. The model, which is essentially built based on off-line profiling runs of some number of big data analytics applications, can estimate the runtime of a target application with any size of input.
- Our extensive experiment and simulation results show that the proposed placement technique can improve the performance of a heterogeneous cluster by placing VMs of multiple applications based on a heterogeneity and interference aware performance model and the simulated annealing approach.

2 Methodology

Heterogeneous Virtualized Cluster. In our default experiments, we use a heterogeneous cluster which consists of 12 nodes connected via 1 GE switch. Table 1 shows the specifications of our heterogeneous cluster. In the cluster, there are two different types of physical nodes. For type T2, each node has 12 cores, but we use only 8 cores, 4 cores per socket, to simplify the experiments. (Note that when all 12 cores are used, there will be more possible placements, and our proposed technique has no limitation on using all of them.) Thus, 64 cores in total (i.e., 32 cores from each node type) are used to run parallel applications. For the network configuration, both T1 and T2 nodes are configured with Gigabit Ethernet, but they are configured with different networking devices, i.e., T1 nodes with a low-end device and T2 nodes with a high-end device. Therefore, T2 nodes show

TABLE 2
Parallel Applications Used in our Experiments

Type	Name	Size	Abbrev.
SpecMPI 2007	132.Zeusmp2	mtrain	Zeus
NPB	CG	Class C	CG
MPI	LAMMPS	5dhfr	LAMMPS
	NAMD	5dhfr	NAMD
Spark	GrepSpark	12.6 GB	GS
	WordCount	9.5 GB	WCS
	TeraGen	11.0 GB	TG
Hadoop	GrepHadoop	9.5 GB	GH

TABLE 3
VC Configurations Used in our Experiments

Config	# of VMs		# of nodes (# VMs per node)		List notation
	T1	T2	T1	T2	
C1	8	0	8 (1)	0	T1 (1,1,1,1,1,1,1,1)
C2	8	0	4 (2)	0	T1 (2,2,2,2)
C3	4	4	4 (1)	4 (1)	T1 (1,1,1,1), T2 (1,1,1,1)
C4	4	4	2 (2)	2 (2)	T1 (2,2), T2 (2,2)
C5	0	8	0	4 (2)	T2 (2,2,2,2)
C6	0	8	0	2 (4)	T2 (4,4)

higher network throughput than T1 nodes. For the storage configuration, both nodes are configured with the same disk device, but due to differences in other hardware configurations, T1 nodes show higher storage throughput than T2 nodes. Note that the network and storage performances of different types of VMs on clouds vary [16].

Xen hypervisor version 4.1.4 is installed on each of the physical nodes, and for dom0, Linux kernel version 3.1.0 is used. For a VM, it is configured with two virtual CPUs and 5 GB of memory. In all experiments, dom0 is pinned to all cores allocated to the active VMs, which execute the workloads, as we assume virtualized cluster systems where no dedicated cores are exclusively assigned to dom0 in order to maximize the utilization of physical CPU cores and flexibility with regard to resource use. Each parallel application is configured with 8 VMs; therefore, four applications in total can be placed on our cluster concurrently.

Note that it is possible to have a larger VM, i.e., one VM per node. However, we observed that for resource intensive applications, the performance can be improved when the VMs are spread out over multiple nodes and executed with an application which has different resource requirements. We can also use the resources more flexibly with smaller VMs.

Distributed Parallel Applications. Table 2 presents parallel applications and their sizes as used in the experiments. We use different parallel workloads from SpecMPI 2007 [17], NPB [18], Spark [8], Hadoop [7], and the two molecular dynamics simulators of LAMMPS [4] and NAMD [5]. Thus, there are four MPI-based applications, which are tightly coupled, and four big data analytic applications, which are loosely coupled. In this work, we focus on scientific and big data analytics applications, as they have different characteristics on communication and synchronization patterns and can therefore show the different effects of resource heterogeneity and interference on the performance.

VC Configurations. In this experimental setup, the total number of different VC configurations even for a single parallel application without considering the placement of co-runners (i.e., VMs or applications running on the same node) is 80. When placing a set of four parallel applications in the default setting, the total number of possible placements exceeds one million. Therefore, searching for the optimal placement of a parallel application in a heterogeneous cluster against all possible placements is impossible. To make the search process tractable, we need to limit the candidate VC configurations of a parallel application.

In this setup, for each parallel application, we consider two types of VC configurations with six configurations in total, as shown in Table 3. In the table, a notation for a VC configuration, which specifies the number of VMs in a node used in the configuration for each node type, is also given. For example, T2(4,4) describes a VC configuration in which four VMs are placed in each of two T2 nodes.

We initially consider *homogeneous VC configurations* in which the application only uses one type of nodes for its VMs, with the number of VMs in each node used for the application equal. For each type of nodes, two homogeneous VC configurations, the *most scaled out* type, where the maximum number of nodes for the same type is used, and the *most consolidated* type, where the minimum number of nodes is used, are utilized. These configurations provide hardware symmetry for the application. An application which requires heavy communication among its VMs may prefer the most consolidated type, while an application that can undergo resource contention among its VMs may prefer the most scaled out type, with a chance to run with other applications with different resource usage characteristics.

Second, we consider *symmetric heterogeneous VC configurations*, in which an application uses two different types of nodes, but the total numbers of VMs in each node type are equal to each other. For a pair of node types, we also have two configurations, the most scaled out and the consolidated types, where the numbers of the VMs in each node are equal. By including these heterogeneous VC configurations which have hardware asymmetry as candidates, we can allow a parallel application to be assigned favored resources partially, with the VMs of the application distributed across even different types of nodes. This may improve the performance of the system.

3 Performance Analysis

3.1 Effects of Different VC Configurations

Recall that each VC configuration given in Table 3 varies depending on the amount of resources used for each node type (i.e., the number of VMs used for each type), and the deployment of the VMs over the cluster (i.e., the number of physical nodes used for the VMs per node type). Fig. 1 shows the runtimes of parallel applications without any co-runners (i.e., *solo runs*) over the six VC configurations. For the configurations, we can make three pairs, (C1, C2), (C3, C4), and (C5, C6). The two configurations in each pair are configured with the same amount of resources, i.e., the

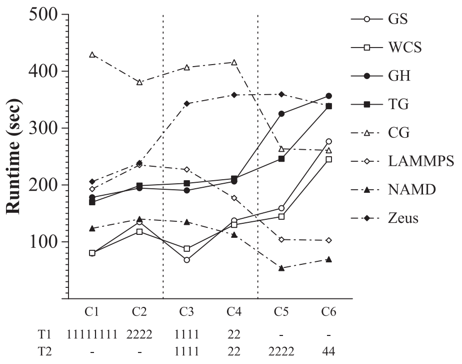


Fig. 1. Runtimes over different VC configurations.

same number of VMs, for each node type. From these results, we can analyze the following.

Analysis 1: Each application has a different preferred node type. As shown in Fig. 1, each application prefers to be executed in either the T1 or T2 nodes depending on its resource requirements. For the two different VC configurations of C2 and C5, which are identical except that C2 uses four T1 nodes, whereas C5 uses four T2 nodes, the resource usage patterns of each parallel application are also analyzed in Table 4. For the results in the table, we execute each application without any co-runners and compute the average resource usage over the VMs running the application during the execution.

In these results, storage-intensive applications have better performance in C2 compared to C5 because their storage I/O throughputs are higher in C2 (i.e., T1 nodes). On the other hand, network-intensive applications have better performance in C5, except for 132.Zeusmp2, as they can exploit higher network I/O throughput in C5 (i.e., T2 nodes). The performance of 132.Zeusmp2 is mainly affected by the CPU performance; therefore, it has a shorter runtime in C2. Regarding CPU utilization, network-intensive applications are also compute-intensive, almost fully utilizing the CPU resources, while storage-intensive applications utilize fewer CPU resources. Thus, in our heterogeneous cluster, big data analytics applications prefer the T1 type, while MPI-based applications (except 132.Zeusmp2) prefer the T2 type.

Analysis 2: The performance of an application is not always improved proportionally to the amount of its preferred resource. We can *naively* expect that the runtime of an application is proportionally reduced as the number of VMs with the favored type increases. However, there are applications that perform quite differently from naive estimation. When we compare runtimes in the three configurations of C2, C4 and C5, which use a total of four (physical) nodes but the number of nodes per type is different, for 132.Zeusmp2 and CG, which require tight synchronization among VMs, the performance is not improved at all unless all of the VMs are executed on the preferred nodes. On the other hand, for all of the big data analytics applications, which are based on a simple communication pattern with a built-in load-balancing feature, until half of the VMs are executed on the favored T1 nodes, there is almost no performance degradation. The effect of heterogeneity on performance of parallel applications is quite dissimilar depending on the synchronization and parallelism patterns.

Analysis 3: Even with the same amount of resources for each type, the effect of how VMs are spread out or consolidated on the performance depends on the application's characteristic and used resource type(s). When the VMs of an application are spread out, they do not contend for the same storage or network resource, whereas when the VMs are consolidated, VMs running on the same physical node can have fast inter-VM communication. Therefore, for storage-intensive applications, where communication among the VMs is not important, they generally prefer the scaled out configuration with less contention over storage I/O. However, for tightly coupled parallel applications, the effects of contention on the network I/O and inter-VM communication jointly affect the performance outcomes. Thus, the performance trend with a different deployment pattern is not always the same with a different resource composition. For 132.Zeusmp2, its performance on the most scaled out configuration (i.e., C1) is 15.91 percent higher than that on the most consolidated configuration (i.e., C2) when the T1 node type is used, while the performance on the most consolidated configuration (i.e., C6) is 5.95 percent higher than that on the most scaled out configuration (i.e., C5) when the T2 node type is used. This

TABLE 4
Runtimes and Resource Usages of Parallel Applications in C2 and C5

Storage App	C2					C5				
	CPU Util (%)			I/O	Time	CPU Util (%)			I/O	Time
	user	sys	wait	(MB/s)	(sec)	user	sys	wait	(MB/s)	(sec)
GS	17.20	1.06	30.99	14.60	134.82	22.59	1.81	30.37	11.49	159.14
WC	30.81	1.42	22.76	11.89	117.76	44.41	2.48	18.13	9.70	144.34
GH	37.20	2.65	7.63	6.20	194.47	41.26	3.16	6.83	3.75	325.23
TG	38.26	5.82	8.45	19.95	198.76	51.98	7.71	15.53	16.10	246.09

Network App	C2					C5				
	CPU Util (%)			I/O	Time	CPU Util (%)			I/O	Time
	user	sys	wait	(MB/s)	(sec)	user	sys	wait	(MB/s)	(sec)
CG	71.96	27.16	0.17	16.74	380.85	72.32	26.33	0.23	24.04	263.52
LAMMPS	64.41	34.32	0.38	11.05	235.08	65.90	31.20	0.91	25.71	103.97
NAMD	45.44	52.58	0.73	8.49	140.05	56.16	38.40	1.81	22.18	53.97
Zeus	39.21	23.95	1.77	9.31	238.57	41.04	22.30	0.63	6.08	359.37

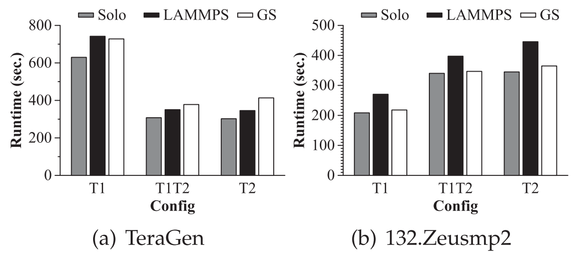


Fig. 2. Runtimes of TeraGen and 132.Zeusmp2 with co-runners.

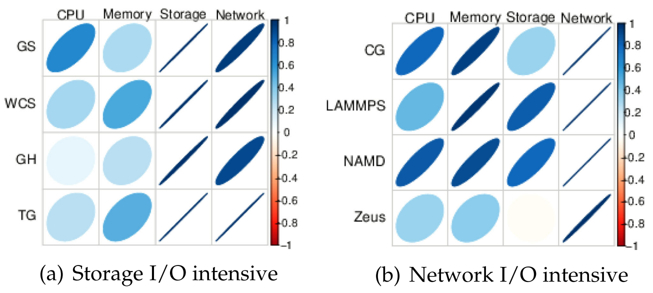


Fig. 4. Correlation between the performance and resource usage patterns.

arises because it utilizes higher network bandwidth in C1 and C6 compared to C2 and C5, respectively. Similarly, LAMMPS and NAMD show different trends when the T1 node type is used (i.e., C1 and C2), and when both the T1 and T2 types are used (i.e., C3 and C4).

3.2 Effects of Interference

To investigate the effects of interference on the performance of a parallel application, we initially use a small virtual cluster composed of 4 VMs running on 4 physical nodes (i.e., in the most scaled out setting) so that we can have the same setting using each of the T1 and T2 node types in our cluster. Fig. 2 shows the execution times of TeraGen and 132. Zeusmp2 over the three different VC configurations of T1 (1,1,1,1), T2(1,1,1,1) and T1(1,1), T2(1,1). In the figure, LAMMPS and GrepSpark are used as co-runners.

In the results, *the interference effect caused by the same co-runner can vary from one VC configuration to another*. Unlike 132. Zeusmp2 which exhibits a similar performance trend when it co-runs with LAMMPS and GrepSpark over all three VC configurations, TeraGen's behavior with LAMMPS and GrepSpark differs. In T1(1,1,1,1), the effect of LAMMPS on the performance is similar to that of GrepSpark, but in the other configurations, the performance degradation by GrepSpark is 8.94∼22.38% higher than that by LAMMPS. Also, *the degree of the interference effect differs depending on the node type*. When TeraGen runs with GrepSpark, the performance degradation of TeraGen is 15.61 percent in T1(1,1,1,1), while it is 36.79 percent in T2(1,1,1,1) compared to solo runs on each configuration. This occurs because when TeraGen and GrepSpark, which are storage-intensive, run on T2 nodes together, the disk I/O becomes a severe performance bottleneck. Therefore, to estimate the final performance of a parallel application accurately, the resource type used and the state of the co-runners for each VM must be considered together.

We next study the effects of co-runners in our default cluster setup (as given in Table 1) using workloads composed of four application instances from Table 2. In order to understand the performance trend with co-runners, we measure the runtime of each of the parallel applications (or application instances) with various co-running applications over the four VC configurations of C1, C3, C4, and C5. (Note that for C2 and C6, each application is executed without any co-running application.) For all of the experiments running multiple applications concurrently, we repeatedly ran applications until the last one finished, as the runtime of each application is different.

Fig. 3 shows the speedup of each application over the four configurations in each experimental run. For each application, we have 148∼388 experimental runs with different co-running applications. In the figure, circles represent the runtime with co-runner(s) in each VC configuration. For the parallel applications, there is no single VC configuration in which every application has the best or worst performance. Therefore, for each of the results, the *speedup* of an application is computed as the runtime of the application in the run over the worst runtime among its solo runs in the six VC configurations.

As shown in the figure, the performances of parallel applications are significantly affected by co-running applications. For the big data analytics applications, they show similar trends over the four configurations, but their speedup values are quite different. For the MPI-based applications, they have quite different performance patterns over the configurations. For 132.Zeusmp2, the range of the speedup over various configurations is only from 0.68 to 1.65, but for NAMD, the range is from 0.67 to 5.04. Moreover, for LAMMPS and NAMD, co-runners can strongly affect their performance in C5, given the widely varying runtimes, unlike the other configurations of C1, C3, and C4 where the co-runner effects are relatively small.

To summarize the above results, in a heterogeneous cluster, the performance of a parallel application is determined by various factors, such as the heterogeneity, interference, and VM deployment across the nodes, but how each factor affects the final performance remains unclear.

3.3 Dominant Resource

Most parallel applications demand multiple resources, such as CPU, memory, network I/O, and storage I/O, and the

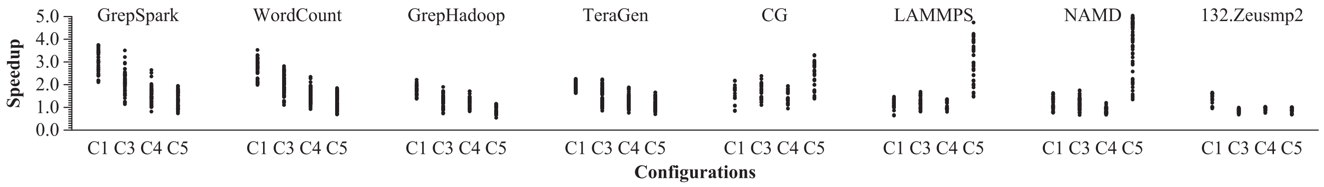


Fig. 3. Performance of parallel applications with co-runners on various VC configurations.

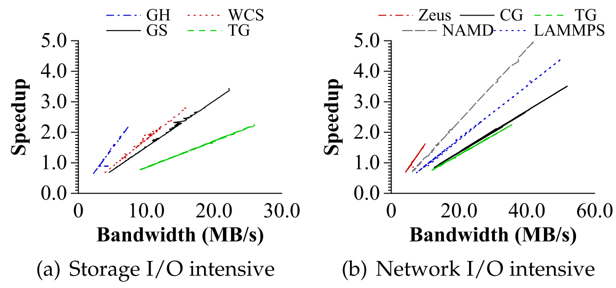


Fig. 5. Dominant resource usage versus performance.

demand for each type of resource varies depending on the characteristics of the application. For each application, we analyze the correlation between the performance and resource usage patterns of CPU utilization, the number of LLC misses per kilo-instruction, and the network and storage bandwidths. The correlation coefficients from the analysis are presented in Fig. 4. For all of the applications used in the experiments except TeraGen, we observe that the performance is closely correlated with the usage of one resource type, either the network I/O or the storage I/O. For TeraGen, there is strong correlation between its speedup and both network I/O and storage I/O. We call a resource (type) which mainly determines the performance of an application the *dominant resource* of the application.

Fig. 5 shows the correlation between the dominant resource usage and the application speedup for two different types of parallel applications, storage-intensive and network-intensive applications. In the result, the speedup of a parallel application increases almost linearly as its dominant resource usage increases. Note that in our experimental setting, cases in which the speedup is bounded even when the bandwidth continues to increase because other resources become a bottleneck do not occur.

4 Placement Based on Machine Learning

Recall that it is not possible to search for the optimal placement of a parallel application in a heterogeneous cluster against all possible placements. Therefore, we need to limit the candidate VC configurations of a parallel application and the VC placements of multiple applications, thus reducing the complexity of performance profiling and VC placements. In this work, when placing a distributed application, we consider only homogeneous VC configurations and symmetric heterogeneous VC configurations, as discussed in Section 2 for a cluster with two different node types.

Consider a heterogeneous cluster composed of T types of physical nodes. To limit the number of the candidate VC configurations, for each application, we can select k types, where k=$2 \times \log_2 T$, out of T types. When selecting k types, we can consider the resource preferences of parallel applications but also select several types randomly. For each selected type, we have two homogeneous VC configurations (i.e., the most scaled out and consolidated ones).

For heterogeneous VC configurations, we select k types out of T types again, and we use a maximum of 2^s types out of k per VC configuration, where $s > 0$ and $2^s \leq k$. Therefore, a candidate VC configuration has 2, ..., or 2^i node types, where $0 < i \leq s$, and the numbers of VMs running on each node type are equal to each other. We assume that each VC

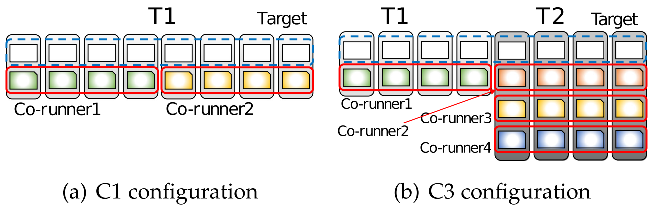


Fig. 6. Examples of candidate VC configurations.

configuration composed of R VMs, where $R \geq 2^s$ and $R \bmod 2^s = 0$, is divided into 2^s *blocks*, and each of the VMs in the same block is executed on a (physical) node with the same type. For each i where $0 < i \leq s$, if $2^i < k$, we generate $\log_2 T$ tuples from the selected k types such that each tuple has 2^i node types, and for $\log_2 T$ tuples, each selected type appears 2^{i-1} times in total. If $2^i = k$, we generate one tuple composed of k types (as we cannot create more than one because of $\binom{k}{2^i}$=1). For each generated tuple, we have two heterogeneous VC configurations discussed above. In this way, the total number of the candidate VC configurations we consider is computed as $2 \times (k + \sum_{i=1}^{s} min(\binom{k}{2^i}, \log_2 T))$, which is scalable as the value of T increases. (Above, a rounded value of $\log_2 T$ is used.)

For example, when we use a maximum of two node types per VC configuration (i.e., s = 1), for homogeneous VC configurations, we have k different types; thus, we have $2 \times k$ configurations. For heterogeneous VC configurations with two node types, we generate $\log_2 T$ pairs from k types, while for each pair, we have two configurations. Thus, the total number of candidate VC configurations is $6 \times \log_2 T$.

Moreover, we restrict the placement of VC configurations for multiple applications such that for a VC configuration composed of 2^s blocks, each of the VMs in the same block has the same set of co-runner(s). Fig. 6 shows two examples of candidate VC configurations for a parallel application in our default cluster setup with s=1.

Fig. 7 shows an overview of our placement technique. In a heterogeneous cluster, a set of the candidate VC configurations with homogeneous and heterogeneous resources is computed for a parallel application. A model based on a machine learning algorithm which can consider various relevant factors conjointly is built to estimate the runtime of a target application on a certain VC configuration. For MPI-based applications with a diversity of synchronization patterns, a target application that will be executed in the cluster is profiled against a synthetic workload that mainly consumes the dominant resources of the application to generate training samples. For big data analytics applications based on the same programming model, it is possible to generate training samples by exhaustively profiling a small set of big data analytics applications. The performance metrics measured during profiling are used as inputs to the model.

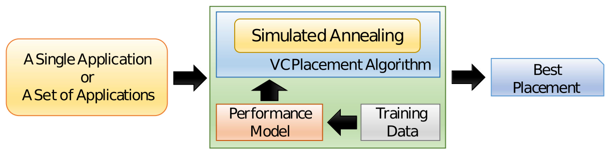
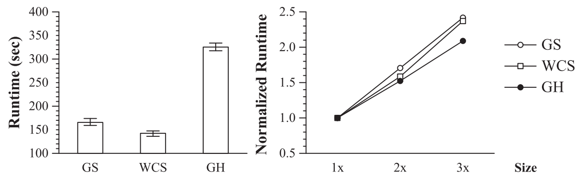


Fig. 7. System overview.



(a) Runs of the same size (b) Runs with different sizes

Fig. 8. Performances of GrepSpark, WordCount, and GrepHadoop.

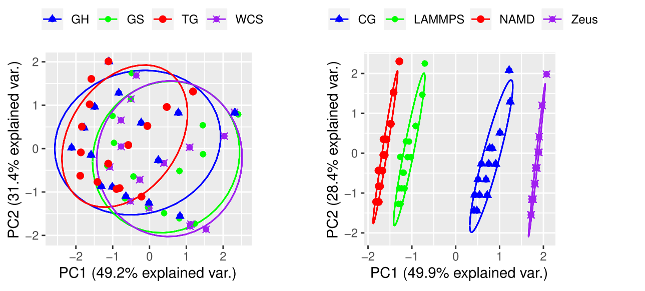
Our placement algorithm can place a single parallel application as the application is submitted to the cluster (i.e., in the on-line mode), and it can make placement decisions for multiple applications simultaneously (i.e., in the batch mode). Because it is still infeasible to search for all possible placements in most heterogeneous clusters, even when we restrict the candidate VC placements for parallel applications, we present a VC placement algorithm based on simulated annealing which approximates the global optimal solution [19]. The algorithm estimates the performance of a hypothetical placement of applications using the performance model, and explores the large search space for the best placement. Similar placement approaches based on simulated annealing and stochastic hill climbing have been used for parallel applications in homogeneous clusters [11] and for web-service workloads [3].

4.1 Performance Model

Big data analytics applications are based on the same programming model of "map", "shuffle", and "reduce" to process a large amount of data [20]. In addition, the size of the input data has a significant impact on the runtimes of the applications. Fig. 8a shows the average runtimes of GS, WCS, and GH for five different input data of the same size, on the C5 configuration. The standard deviation of the runtimes is also presented in the figure. Fig. 8b shows their runtimes over various input sizes on the C5 configuration, where the sizes of "2x" and "3x" are two and three times the size of "1x". In the figure, the runtimes of the applications tend to increase in proportion to the input data size. However, if the sizes of the input data are identical, the runtimes of the application are similar.

However, MPI-based applications have various communication and synchronization patterns [21], [22]. Moreover, their performances are affected by various parameters which are usually specific to each application. (For example, in molecular dynamics simulators, the performance is affected by the number of atoms, the molecular topology, the cut-off distance between the atoms, etc. [10].)

Figs. 9a and 9b present the results of a principal component analysis (PCA) with 95 percent confidence ellipses, which present the regions including 95 percent of samples, for the big data analytics and MPI-based applications, respectively. The analysis in each case is performed on the measured runtimes of the parallel applications with various co-runner states in the C5 configuration to understand the performance trend of the applications under various interference settings. For each type of applications, an identical set of co-running synthetic workloads (which mainly utilize the dominant resource of the corresponding type as discussed in Section 4.1.1) is used to generate interference. In the figures, the similarity of the big data analytics applications and the



(a) Big data analytics (b) MPI-based

Fig. 9. Principal component analysis.

dissimilarity of MPI-based applications are apparent, as the regions of big data analytics applications overlap but those of MPI-based applications are separated. Therefore, for big data analytics applications, we build one *general* performance model based on the off-line profiling of a few big data analytics applications. For a target big data analytics application that will run in the cluster, our modeling technique requires a limited number of profiling runs with a certain size of input data, and the built model can estimate the runtime of the target application for different input data of any size.

For MPI-based applications, we build a model for each of the applications. In this work, we do not associate the model with different inputs (for example, different proteins and cut-off distances for molecular dynamics simulators). However, our model can be combined with other modeling techniques [10], which consider different inputs when modeling MPI-based applications such as NAMD and LAMMPS in a homogeneous cluster, in order to eliminate the need for profiling with each different input.

4.1.1 Generating Training Samples

We generate training samples by the off-line profiling of a parallel application over various setups. Based on the observation that the performance of a parallel application tends to be directly correlated with the usage of its dominant resource in Section 3.3, we reduce the profiling runs of the application significantly through the use of a synthetic workload that mainly consumes its dominant resource. During the profiling step, synthetic workload mixes which utilize all types of resources are not used.

We implemented two types of synthetic workload generators with different intensity levels. For a synthetic workload using the network resource, for a pair of VMs, each VM sends and receives some number of messages to/from the other VM every second. For a synthetic workload using the storage resource, each VM reads and writes some amount of data from/to its local file system every second. For each of the network and storage synthetic workloads, as the intensity of each workload increases, the network and storage bandwidths of a VM running the workload also increases. When profiling an application in each VC configuration, interference is generated in a block as a unit by running synthetic workloads with the same intensity in all of the remaining VMs on each node of the block, and each block can have a different interference level.

While profiling a parallel application, we measure various performance metrics for each VM of the application, and then compute the average or some aggregated value of each metric over the VMs. For the performance model, the metrics of CPU idle % and user %, send and receive for network I/O (in MB/s), and read and write for storage I/O (in MB/s) are selected as inputs, as they are highly correlated with the performance of the application. Note that the profiling process below needs to be done only once, unless the physical hardware configurations of the cluster are changed.

Profiling of Big Data Analytics Applications. We perform exhaustive off-line profiling for a small set of big data analytics applications such that various behaviors of big data analytics applications are contained in a set of training data. For an application in the set, we run it without any co-runners and also with synthetic workloads as co-runners over all of the VC configurations. We also profile the runtimes of the application with other parallel applications in the set. To make the model estimate the runtime of an application with different input sizes, for each of the parallel applications in the set, we also profile it with several input files of different sizes. Moreover, we generate training samples that show the correlation between the runtimes and input sizes from profiled data with different input sizes.

In addition, for target Hadoop and Spark applications that will run in the cluster, we need to profile each of the applications for certain input data (whose size is reasonably small) without any co-runners and with synthetic workloads of its dominant resource in all VC configurations. Exhaustive profiling for the target applications is not necessary.

Profiling of MPI-Based Applications. To generate training samples for each MPI-based application, we profile its runtimes without any co-runners, and only with synthetic workloads in each candidate VC configuration. To build an individual model for each application, additional profiling runs with real applications are not necessary.

4.1.2 Building Machine-Learning Based Models

In a performance model for a target application A_{target}, we basically provide the following as inputs.

- A target VC configuration VC_{target}.
- The solo run runtime of A_{target} and the average performance metrics over all of the VMs of A_{target} on VC_{target} measured during the solo run.
- In each of blocks on VC_{target}, for each co-running application C_i on the block, the average performance metrics over the VMs of C_i running on the block measured during the solo run.

For the models of MPI-based applications, their solo run runtimes (with the target input data) in all of the candidate VC configurations are profiled off-line. Therefore, the measured information is used as the input. On the other hand, for the general model of the Hadoop and Spark applications, the model uses previously profiled information pertaining to the runtime and metrics of the solo run of a target application for certain input data as inputs in order to predict the runtime with target input data whose size can differ from the profiled size. In consequence, this model additionally takes the sizes of the current target input and (estimated) output data along with those of the previously profiled run for the same application as inputs.

We attempted various machine-learning techniques using *Weka* [23], and concluded that Reduced Error Pruning Tree (REPTree) shows the best performance for our performance estimation. Machine-learning techniques such as ANN [24], SVM [25], and Gaussian Process Regression [24] do not work well if a given training data set cannot be fitted in any kernel functions. For our modeling problem, the function was not built properly from the inputs and outputs in our training set, resulting in huge error rates. Both RandomTree and REPTree are decision tree learning based algorithms [23]. In a decision tree, every non-leaf node splits the data space into subspaces based on an input attribute and a threshold, and every leaf node is assigned a target value [26]. In our decision tree, attributes which represent A_{target}, VC_{target} and applications co-running on VC_{target} discussed above are used. As the data spaces are partitioned based on important attributes that determine the performance, a decision tree is suitable for our problem. By following the root to a leaf node in the tree with the given input values, the estimated runtime of A_{target} is presented at the leaf node.

When constructing a tree, REPTree considers all of the input attributes to make a branching decision, and it prunes the tree using reduced-error pruning. On the other hand, RandomTree considers a set of K randomly selected attributes at each node to build a tree, and performs no pruning. For our runtime estimation, we selected attributes that can reflect the performance of various parallel applications carefully and provided them to the models as inputs. Hence, by considering all of the attributes, REPTree works better than RandomTree in our study.

With modeling based on REPTree, we use a regression tree mode which predicts the output in the form of a real number to estimate the runtime of A_{target} on VC_{target}. To improve the model accuracy and avoid over-fitting, we used bagging, also called bootstrap aggregating [27]. Similarly, bagging has been used to lower the error rates of performance models for HPC workloads on a multi-core system [28]. For bagging, we generate 100 training sets by sampling, from all of the training samples we have, uniformly and with replacement. We then build 100 models using the 100 training sets. Finally, we compute the average of the estimated runtimes from the models as the final estimation.

The accuracy of a model based on REPTree is affected by the quality of a training data set, as the model is built by recursively partitioning a training data set to subsets based on input attributes. To have an accurate model, training samples in the set need to have similarity to instances of real workloads running on the cluster. If the values of input attributes for an application (i.e., performance metrics) are very different from those in the training set, the model is unlikely to estimate the runtime accurately. In such a case, the model can be re-trained by adding this application's samples to the training set, as discussed in earlier work [28].

4.2 Placement Based on Simulated Annealing

Our simulated annealing based placement algorithm can make placement decisions in both the batch and on-line modes. To place VMs for a given set of new applications (or a single application), we initially hypothetically distribute the VMs of the applications randomly over available nodes in a heterogeneous cluster, and estimate the performance of the applications with this random placement. To find the best placement, we change the hypothetic placement by randomly selecting one application from the new applications, placing it in a different VC configuration, and then adjusting the other new applications affected by this new placement. We then compute the performance again for a new placement state. If the estimated performance in the new state is better than that in the previous state, we can conclude that the new placement is better. In such a case, we always move to the new state. If the move results in a worse state, we probabilistically move to the new state. The above process is repeated for a predefined number of iterations. In the algorithm, we basically consider candidate VC configurations for applications and the placement of applications that satisfies our restriction on co-runners as discussed above. Note that if none of the candidate VC configurations of a parallel application is available in a cluster, then a VC configuration which is closest to one of the candidate VC configurations can be used.

In this work, when placing VMs in a heterogeneous cluster, we aim to maximize the overall speedup for parallel applications running in the cluster. To estimate the overall speedup in a certain placement, the algorithm initially estimates the runtime of each application based on the performance model, and subsequently computes the speedup over the worst runtime among its solo runs in the candidate VC configurations. It then uses the geometric mean of the speedups of all the applications for the performance, as the range of speedups under different placements differ for each application.

5 RESULTS

5.1 Performance Estimation Accuracy

We evaluate our performance models using a heterogeneous cluster composed of 12 nodes with two node types, as described in Section 2. For each Hadoop and Spark application in Table 2, we use two additional sizes, i.e., sizes which are two and three times larger (i.e., 2x and 3x) than the specified size (i.e., 1x) in the table to evaluate the model.

For synthetic workloads used on profiling, there are five levels of intensity for each type of synthetic workloads. For each VC configuration composed of two blocks, we run a target application with a synthetic workload on only one of the blocks and also both of the blocks (if possible). This is done for each intensity level. For example, in C3, we have five profiling runs only by running synthetic workloads on the T1 block, five runs only by running them on the T2 block, and five runs by running them on both of the blocks. In a virtualized system, dom0 is required to handle I/O requests from VMs. Thus, if a VM in a node does not use the CPU intensively, the performance of dom0, which handles I/O requests of other VMs running in the same node, can be noticeably improved. With regard to network-intensive applications used in our experiments, most of them fully utilize the CPU, unlike the storage intensive applications; thus, if co-running VMs do not fully use the CPU, their runtimes can be reduced. To model these cases, we make a network synthetic workload which can use the CPU at the two levels of ∼50% and ∼100% of a VM. For storage synthetic workloads, only one CPU level of ∼50% is used.

TABLE 5
Validation Results of Applications

App	Error(%)	App	Error(%)
Zeus	12.71	GS	21.36
CG	21.42	WCS	16.22
LAMMPS	18.24	TG	21.72
NAMD	20.73	GH	34.27
Average	21.84	Average	23.39

For each of the Hadoop and Spark applications in Table 2, we profile it with small size input (i.e., half the size of 1x) for six solo runs, and 40 co-runs with synthetic workloads, except for TeraGen. For a collection of applications for exhaustive profiling, we use the five additional applications of WordCountHadoop, JoinHadoop, ScanHadoop, SortHadoop and TeraSortHadoop. We also perform exhaustive profiling of the target applications. However, to demonstrate that the model can estimate the runtime of the Hadoop or Spark application $A_{BigData}$ for any input size without heavy profiling, we remove all training samples that contain any data of $A_{BigData}$ from the set of training samples obtained by exhaustive profiling. Subsequently, we build the model using a subset of training samples which does not include any samples of $A_{BigData}$ along with previously profiled samples of $A_{BigData}$ with a small input size. Finally, we predict the runtime of $A_{BigData}$ with the given target input data, similar to the evaluation method in earlier work [28]. Note that for TeraGen whose performance is correlated with both the network I/O and storage I/O, we built three models with storage synthetic workloads, network synthetic workloads, and both types of synthetic workloads. There were no major differences among the error rates of the three models. We used the model with network synthetic workloads (of 80 profiling runs) in our study, as it has the lowest error rate.

To profile each MPI-based application, we have six solo runs, and 80 co-runs with network synthetic workloads. Note that many MPI-based scientific applications, including the MPI-based applications used in this experiment, iterate a given number of time steps for the execution; in many cases, the runtime of each time step (i.e., each iteration) is uniform, as demonstrated in the literature [10], [29]. Thus, for such an MPI-based application, we can have each profiling run only with a small number of time steps instead of running all of the iterations specified for the application and predict the actual runtime proportionally to the number of time steps. By exploring this iteration-based pattern, the profiling overhead can be reduced significantly.

Table 5 shows the average error rates of the REPTree models with bagging for the parallel applications, where 100∼180 and 166∼406 test cases were used for MPI-based and big data analytics applications, respectively. The error rate in each test case is computed as $|r_{est} - r_{act}|/r_{act} \times 100$, where r_{est} and r_{act} are the estimated and actual runtimes, respectively. Each of the built models is validated using the experimental results of the application running together with other parallel applications in our cluster. To evaluate the accuracy of the big data analytics model, which can estimate the runtime of an application with different input sizes, we used a set of testing data which includes the experimental runs using three different input sizes for each application.

In the REPTree models, for nodes in higher levels of a tree, attributes representing a target VC configuration are used the most (especially for root nodes), followed by attributes related to the dominant resource usage of a target application, as they are critical features to determine the runtime of a target application.

When ANN, SVM, and Gaussian Process Regression are used, the average error rate is very high, especially for MPI-based applications. For these algorithms, the average error rates for the MPI-based and big data analytics applications exceed 100 and 60 percent, respectively. These rates are not acceptable for reliable performance models. With RandomTree and REPTree (without bagging), the respective average error rates are 45.76 and 26.22 percent for big data analytics applications, while the corresponding error rates are 24.68 and 22.53 percent for MPI-based applications. REPTree with bagging provides greater accuracy than the other algorithms.

Note that we have attempted to have one model of the MPI-based applications based on exhaustive off-line profiling of a few MPI-based applications, similar to the general model of the big data analytics applications. For target MPI-based applications, their profiling runs only for solo runs are used in a training data set. However, the error rates are quite high as up to 45 percent, given that MPI-based applications show quite different characteristics, as analyzed in Section 4.1.

5.2 Experimental Results on a Cluster with Two Types

5.2.1 Methodology

We experimentally evaluate the performance of our ML-based placement technique using a real cluster with two node types as described in Section 2. When placing parallel applications on the cluster, our simulated annealing based placement algorithm uses the performance model built in Section 5.1 to estimate the runtimes of applications for a hypothetical placement. We compare our technique (`ML-G`) with the following five heuristics:

- Random placement (`Random`): This heuristic randomly places parallel applications among the candidate VC configurations.
- Greedy placement (`Greedy`): This algorithm places a parallel application in one of the most consolidated homogeneous VC configurations, where the application has the smallest solo run runtime, if it is available. This is inspired by the greedy algorithm in Quasar [15], where for an application, the scheduler initially sorts server types according to their performance and then selects servers which is capable of higher performance in the sorted order while attempting to pack the application within a few servers.
- Heterogeneity-aware interference-ignorant placement (`HA`): This algorithm considers the effect of different VC configurations with heterogeneous resources on the performance of a parallel application. It places a parallel application in one of its candidate VC configurations based on their solo run runtimes in a greedy manner. It assigns the application an available VC configuration with the smallest runtime, without reflecting possible interference effects caused by co-runners.
- Interference-aware heterogeneity-ignorant placement (`IA`): If two parallel applications which have the same type of dominant resource are placed together on a set of the same nodes, their performance will be degraded, as they compete for the same type of resources in the nodes. To reduce the interference effect, this algorithm creates groups of two for given applications such that two applications with different dominant resource types are paired, if possible, and then places each pair randomly (on C1, C3 or C5 in our setup), being oblivious to the heterogeneity of resources. Note that this approach can be used only in the batch mode.
- ML-based placement with individual models (`ML-I`): This algorithm places a parallel application in the same manner as `ML-G` except that the individual modeling approach is also used for Hadoop and Spark applications. For target Hadoop and Spark applications, the profiling overhead for `ML-I` can exceed that for `ML-G`, which can use small input size for profiling.

TABLE 6
Workloads Used in our Experiments

	App1	App2	App3	App4	S:N:B
WL1	WordCount	WordCount	TeraGen	Zeus	2:1:1
WL2	GrepSpark	WordCount	CG	NAMD	2:2:0
WL3	WordCount	TeraGen	LAMMPS	Zeus	1:2:1
WL4	GrepSpark	TeraGen	NAMD	Zeus	1:2:1
WL5	WordCount	GrepHadoop	TeraGen	CG	2:1:1
WL6	GrepSpark	GrepSpark	NAMD	LAMMPS	2:2:0
WL7	WordCount	GrepHadoop	NAMD	LAMMPS	2:2:0
WL8	GrepSpark	GrepSpark	WordCount	GrepHadoop	4:0:0
WL9	CG	LAMMPS	NAMD	Zeus	0:4:0

For the experiments in this section, we provide a set of four applications as input to the placement algorithms (i.e., batch mode). In `Greedy` and `HA`, we need to order multiple applications in the set. For each application, we compute the maximum speedup as r_{worst}/r_{best}, where r_{worst} and r_{best} are the runtimes of its worst (i.e., longest) and best (i.e., shortest) solo runs between the C2 and C6 configurations for `Greedy` and among the six candidate VC configurations for `HA`. Then, we sort parallel applications in a workload in a decreasing order of their computed maximum speedups. For each application in the sorted order, `Greedy` and `HA` make a placement decision. For `Random` and `IA`, the average performance of five random placements is shown in the results.

Table 6 presents the nine workloads used in our experiments. The ratio of the number of storage-intensive applications (denoted as "S"), that of network-intensive applications (denoted as "N"), and that of applications utilizing both network and storage resources (denoted as "B") in each workload is also given in the table. For the big data analytics applications, the input sizes (i.e., 1x) described in

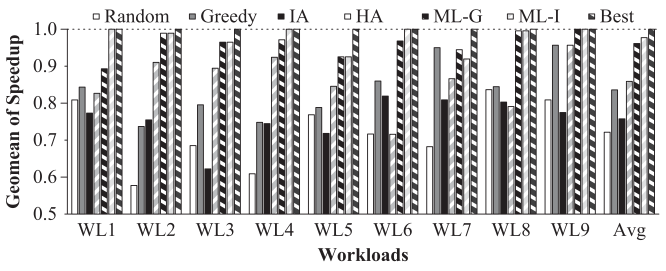


Fig. 10. Speedups of workloads, normalized to the best placement.

Table 2 are used in these workloads. The workloads are selected such that there are various ratios of the numbers of storage-intensive, network-intensive, and storage-network-intensive applications, including two homogeneous workloads that are composed of only one type of applications. In our experimental setup, the total number of candidate placements for four applications is 74. For each workload, we explore all of the candidate placements to find the best placement of the four applications (Best), which has the maximum geometric mean of the speedups.

5.2.2 Experimental Results

Fig. 10 shows the speedup of each of the nine workloads (on the geometric mean), normalized to that of the best placement. Our ML-based placement technique ML-G achieves 96.13 percent of the performance of Best on average. The performance improvements of ML-G compared to Random, Greedy, HA, and IA are 24.80, 12.95, 10.61, and 21.11 percent on average, respectively. The performance of ML-G is fairly comparable to that of ML-I, which achieves 97.72 percent of Best on average. With regard to homogeneous workloads, they are less sensitive to the placement configuration, as they consist of similar applications in terms of preferred resources and interference effects. Thus, the performance difference between good and poor configurations becomes smaller. Our performance models have modest error rates, but the models can compute the relative performance gap between different placements and distinguish between good and poor placements reasonably well.

For various algorithms, we analyze the complexity of profiling overhead for a target application with the corresponding target input in a cluster which has T node types. With Greedy, HA, IA, and ML-I, the profiling complexity is $O(\log T)$, while with ML-G, no additional profiling is not required, because we use the previously profiled information for the application (possibly with a different input size). In our setup of a cluster with two types, for Greedy, HA, and IA, two, six, and three solo runs are profiled, respectively. For ML-I, 46 and 86 runs are profiled for storage and network intensive applications.

We subsequently analyze the best placements of the workloads in our heterogeneous cluster. Table 7 indicates that there is no single best placement that works for all workloads, despite the fact that there is a tendency of network-intensive applications to favor T2 nodes that provide a higher network I/O bandwidth, while storage-intensive applications prefer to be placed on T1 nodes where they have only one co-runner VM and may use more of the storage I/O resource. Five different sets of VC configurations are used for the best placements. Even when the same set of VC configurations is used to place four applications, depending on the combination of applications, the same application can be assigned a different VC configuration.

TABLE 7
Best Configuration for Each Workload

	App1	App2	App3	App4
WL1	WCS: C1	WCS: C3	TG: C3	Zeus: C5
WL2	GS: C3	WCS: C2	CG: C3	NAMD: C5
WL3	WCS: C3	TG: C3	LAMMPS: C5	Zeus: C2
WL4	GS: C3	TG: C3	NAMD: C5	Zeus: C1
WL5	WCS: C1	GS: C5	TG: C1	CG: C5
WL6	GS: C3	GS: C3	NAMD: C5	LAMMPS: C1
WL7	WCS: C1	GH: C1	NAMD: C6	LAMMPS: C6
WL8	GS: C1	GS: C3	WCS: C3	GH: C5
WL9	CG: C5	LAMMPS: C5	NAMD: C2	Zeus: C2

Recall that we include two heterogeneous VC configurations as the candidate VC configurations for a parallel application. All of the best placements except for WL5, WL7 and WL9 use the heterogeneous VC configuration of C3, and all of the applications placed in C3, except for CG in WL2, are storage-intensive big data applications. For big data analytics applications, the performance can be improved by allocating the favored nodes partially, and the performance can be enhanced if their VMs are executed with other applications over different nodes, with less contention over the same types of resources. This result shows that it is beneficial to exploit heterogeneous VC configurations with hardware asymmetry.

5.3 Large Scale Simulations

To evaluate our placement technique in a large-scale heterogeneous cluster, we simulated the placement algorithms using runtime traces collected from real experiments in our 12 node clusters. In our simulations, a heterogeneous cluster consists of 80 T1 nodes and 40 T2 nodes, 120 nodes in total, with the same configuration of VMs and candidate VC configurations used as in Sections 5.1 and 5.2.

The simulator, which was implemented in C++, computes the placement of 40 parallel applications. In our ML-based algorithm, for a hypothetical placement, it estimates the speedup for a parallel application based on the performance model built in Section 5.1. For a final placement determined by the algorithm, the simulator computes the geometric mean of the speedups of 40 applications using their actual runtimes as measured in our 12 node cluster with the same VC and co-runner configurations. Note that due to the limited cluster setup used here, in some cases we were not able to generate precisely the same placement of a parallel application with co-runners computed by the algorithm. In such cases, we used the runtime of an application measured in the configuration closest to the computed case. We also implemented the simulator of HA in a similar manner, while for Greedy, we were able to evaluate the performance based on measured solo run runtimes.

Fig. 11 shows the speedups of the four workloads of WL1, WL3, WL5 and WL6 (in Table 6) on a large cluster in the batch and on-line scheduling modes. In a workload, ten instances of each application are submitted. In the figures, the speedups are normalized to those of ML-I, as searching

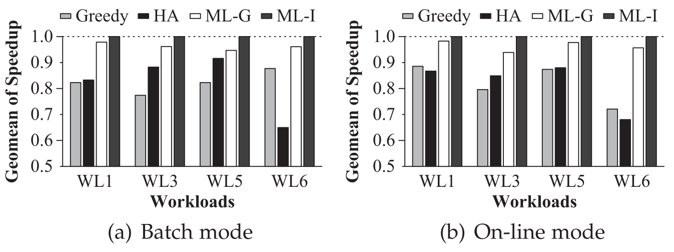


Fig. 11. Speedups in a large scale cluster.

for the best placement is infeasible with a large cluster. In the on-line mode, we assume that 40 applications are submitted at the same time, but there is an order among them. Each application is placed in sequence in the submission order. For a workload, five submission orders are randomly selected. The average value of the five runs is given in Fig. 11b.

In the batch mode, compared to `Greedy` and `HA`, our `ML-G` improves the performance by 14.32 and 14.70 percent on average, respectively. In the on-line mode, the performance improvements with `ML-G` are 15.09 and 15.05 percent on average, compared to `Greedy` and `HA`, respectively. With individual models in `ML-I`, the performances improve correspondingly by 3.79 and 3.60 percent in the batch and on-line modes, compared to `ML-G`.

Because the on-line algorithm only considers the placement of one application at a time, and it cannot change the placements of existing applications in the cluster, the numbers of possible VC configurations and placements for the application are reduced during on-line scheduling. A submission order of applications in a workload affects the performance of each placement technique. However, in our simulation runs, in no case do `Greedy` and `HA` show better performance than `ML-G`. The minimum performance improvements of `ML-G`, compared to `Greedy` and `HA`, over all of the runs of the four workloads, are 4.54 and 1.34 percent, respectively.

5.4 Experimental Results on a Cluster with Four Types

We investigate the performance of our placement technique in a heterogeneous cluster composed of 20 nodes with four different node types. In addition to T1 and T2, each node in T3 is configured with two Intel octa-core E5-2640 v3 (Haswell) 2.60 GHz processors, 20 MB L3 per socket, 32 GB memory, and Gigabit Ethernet, and there are four T3 nodes in the cluster. Each of T3 nodes has two sockets, but we only use one socket with 8 cores. To have another node type T4, we change the CPU frequency of four nodes that have the same specification as T3 nodes to 1.60 GHz. The network throughput of T3 and T4 is similar to that of T2.

In the experiment, we use WL1 shown in Table 6. We have two instances of each application in WL1 and the workload is composed of a total of eight application instances. For the on-line mode experiments, five submission orders of eight applications are randomly selected, and the average value of the five runs is presented below. Recall that in a cluster with T node types, we select $k = 2 \times \log_2 T$ types out of T, and we use up to 2^s types per heterogeneous VC configuration, where $s > 0$ and $2^s \leq k$. In the cluster with four node types (i.e., $T = 4$), we evaluate the performance of our placement technique for two cases where a candidate VC configuration uses a maximum of two node types (i.e., $s = 1$) and four node types (i.e., $s = 2$).

There is a trade-off between the potential performance enhancement using more node types per VC configuration and the overhead to profile applications and search for the best placement. As T and s increase, the numbers of candidate VC configurations for an application and profiling runs of the application increase. For each parallel application, the total numbers of the candidate VC configurations with $s = 1$ and $s = 2$ are 12 ($= 6 \times \log_2 T$) and 14 ($= 6 \times \log_2 T + 2$), respectively, as discussed in Section 4. (Note that due to the limited cluster setup used in the experiments, we have 13 candidate VC configuration with $s = 2$.) When building models for the applications in WL1, the number of profiled runs of each application against synthetic workloads with $s = 2$ is 2.5 times larger than that with $s = 1$.

When up to two types are used (i.e., $s = 1$) for a VC configuration, for the batch mode, the performance of `ML-I` is 11.66 percent, and 13.29 percent better than those of `Greedy` and `HA`, respectively. For the on-line mode, the performance of `ML-I` is 14.45 percent, and 13.93 percent better than those of `Greedy` and `HA`, respectively. For the batch and on-line modes, the respective performance improvements with using a maximum of four types (i.e., $s = 2$) are 6.38 and 3.64 percent, compared to using a maximum of two types.

In our experimental results, when using more node types per VC configuration, much higher profiling overhead is required, but the performance gain tends to be relatively small with considering the overhead. For 132.Zeusmp2, it tends to have no performance gain by utilizing heterogeneous resources and it is less affected by interference as shown in Section 3. Thus, 132.Zeusmp2 still prefers to be placed on nodes with the favored type, i.e., T1 nodes, regardless of which application co-runs on T1 nodes, and consequently, a heterogeneous VC configuration with four node types is not selected for 132.Zeusmp2. With regard to WordCount, the performance can be better on a VC configuration with all of T1, T2, T3 and T4 node types, compared to a VC configuration with T3 and T4 node types. Therefore, WordCount is placed on a VC configuration with all of the four node types to improve the performance. However, a VC configuration with T3 and T4 types is also used for WordCount, when one of the four node types is exhausted by other applications in the workload. Depending on the combination and submission order of applications in a workload, it is possible that heterogeneous VC configurations is not well utilized, having small performance improvement. Even when a cluster has a large number of heterogeneous node types, it may be effective enough to use a small number of node types to run a parallel application (instead of using all the node types per VC configuration) while keeping the profiling overhead of the application within reasonable bounds.

5.5 Discussion

Effects of Profiling Against Mixed Synthetic Workloads. In our 12 node cluster described in Section 2, the storage and network intensive applications have 40 and 80 profiling runs, respectively, with synthetic workloads with five intensity levels, which mainly use the dominant resource. However, when a synthetic workload which utilizes the CPU, network and storage resources in a mixed manner is used, both types of the applications end up having 400 profiling runs. For WordCount and 132.Zeusmp2, we build the performance model based on mixed synthetic workloads. The accuracy of these models is decreased by less than 1.1 percent, compared to that of the models using only the dominant resource, demonstrating the effectiveness of using the dominant resource on the profiling process.

A Different Number of VMs for Applications. In the above results, all of the applications are configured with 8 VMs. However, applications have different resource requirements; thus, they use different numbers of VMs for their virtual clusters (i.e., different VC sizes). We evaluate our placement method for a workload composed of applications with 4, 8, 16 VMs using the 12-node cluster with two node types. With a given VC size, each VC configuration has four homogeneous and two heterogeneous candidate VC configurations as discussed in Section 2. (Note that due to the limited size of the cluster used in our experiments, there are only three candidate VC configurations for 16 VMs.) To use 4 or 16 VMs for an application, additional profiling runs of the application are done using synthetic workloads, and these runs are added to a training data set to build a model.

Each application has two blocks regardless of a used VC size, but the assumption that each of the VMs in the same block has the same set of co-runner(s) is relaxed. This assumption is to restrict possible VC configurations, reducing the overhead of profiling and placement, but it needs to be relaxed if it restricts too many VC configurations, preventing a resource provider from utilizing the cluster resource effectively. In general, heterogeneous co-runners can exist for a block of a VC configuration, if there are two candidate VC configurations that use the same type of physical nodes, but the numbers of used nodes differ. For example, in the configuration of T1(1,1,1,1),T2(1,1,1,1) for 8 VMs, a block in a T1 node or a T2 node has heterogeneous co-runners if there are two co-runners using 4 VMs each with the T1(1,1),T2(1,1) configuration. When a block has heterogeneous co-runners, we provide the average performance metrics of the co-runners to the performance model as inputs.

In our experiments, we use the two workloads of WL3 and WL5 in Table 6, which consist of two applications with 4 VMs each, one application with 8 VMs and one application with 16 VMs, and use individual models for all applications in WL3 and WL5. For WL3 and WL5, our technique shows 91.48 and 100 percent of the performance with `Best`, respectively. Adding a different VC size to the model requires more profiling of an application, but when we use 50 percent of profiling runs which include ones with intensities of three and five, the accuracy of the model is degraded only by around 1 percent on average. With the model of 50 percent samples, the performance of WL5 becomes 96.19 percent of `Best`, while that of WL3 remains the same.

*VM Placement with VM Live Migration.*For the on-line scheduling mode in Section 5.3, we do not change the placement of existing applications to place a new application. It is possible to migrate VMs of an existing application to ensure better placement of applications. However, if the VM migration overhead is too high, the efficiency of a cluster will be decreased. In the on-line scheduling for the four workloads in Section 5.3, if we allow existing VMs to be migrated whenever a placement with VM migration increases the efficiency, the performance will be similar to that with the batch model, showing around 7 percent higher performance than the on-line mode of `ML-G` and `ML-I` on average, under the assumption that the VM migration overhead is none. To study the performance of `ML-G` over various VM migration overheads, we run simulations in the on-line mode with VM migration for WL3 (similar to Section 5.3). We modify our simulated annealing algorithm to consider migrating already placed VMs to improve the efficiency. Note that for a parallel application, VMs in the same block must be migrated together as a unit. For WL3, the possible maximum improvement by employing VM migrations is 10.15 percent. As the overhead to migrate an application to a new placement increases to 2 percent of the average runtime of all the applications used in Table 2, the performance of WL3 cannot match that in the on-line mode without VM migration. The VM migration overhead must be considered when re-arranging existing VMs in an effort to improve the performance.

6 Related Work

Several techniques to schedule applications in heterogeneous and consolidated clusters have been studied. Paragon is a QoS-aware scheduler that considers heterogeneous resources and interference in a large scale datacenter for single node applications [2]. A fair placement technique based on game theory is proposed to perform pairwise task collocations [30]. A QoS-aware management system called Quasar is proposed to handle resource allocation and assignment of incoming workloads, including distributed applications, in a heterogeneous cluster [15]. The above three techniques use collaborative filtering to estimate the performance of applications. In Quasar [15], for an application, it performs four classification techniques for estimating the effects of scale-up, scale-out, heterogeneity and interference separately. It also employs a greedy scheduler that exams and selects nodes, one by one, from the highest performance platform for the application to find a placement that satisfies a QoS constraint. This greedy scheduler is well suited for applications commonly used in large scale datacenters such as web server, latency critical and stateful services, and distributed analytics frameworks. Quasar mostly considers non-virtualized systems. Mage considers the heterogeneity across and within servers when scheduling latency-critical and batch applications, improving the performance while satisfying QoS requirements [31]. For long running web-service applications, the impacts of microarchitectural heterogeneity and interference are analyzed and a mapping technique based on a stochastic hill climbing is presented [3].

Interference-aware management techniques that normalize a different level of interference to a score have been studied [11], [12], [14]. An off-line profiling based model [12] and a runtime profiling model [14] are presented for single node applications. An interference modeling technique for distributed parallel applications [11] is proposed for a homogeneous cluster, not considering hardware heterogeneity.

Techniques to find the best virtual cluster configuration for distributed parallel applications have been investigated [10], [32]. For MapReduce applications, a system to automatically find a good cluster configuration regarding its size and resource types is presented for clouds [32]. A configuration guidance framework for scientific MPI-based applications with various inputs is investigated to find the optimal VM configuration that satisfies the cost and runtime requirements on clouds [10]. The optimal or near-optimal cloud configuration for big data analytics applications can be found efficiently based on Bayesian Optimization [33]. A technique to estimate the performance of advanced analytics such as machine learning by using a small size input is proposed [34]. Machine learning algorithms have been used to predict the performance of virtualized applications [35] and HPC workloads on a single multicore machine [28], and also predict the effects of GPGPU hardware configurations [36].

7 Concluding Remarks

In this work, we investigated a placement technique for distributed parallel applications in a heterogeneous cluster, aiming to maximize the overall performance for the benefits of service providers and users. Using the experiments on heterogeneous clusters and large scale simulations, we demonstrated the feasibility of a heterogeneity and interference aware placement approach for distributed parallel applications, which considers various factors to influence the performance of a distributed parallel application in a combined manner for maximizing the efficiency.

Acknowledgments

This work was partly supported by Institute for Information & Communications Technology Promotion(IITP) grant funded by the Korea government(MSIT) (No. 2015-0-00590, High Performance Big Data Analytics Platform Performance Acceleration Technologies Development) and National Research Foundation of Korea (NRF) funded by the Korea government(MSIT) (NRF-2015R1C1A1A02037400 and NRF-2016M3C4A7952634).

References

[1] L. A. Barroso, J. Clidaras, and U. Hoelzle, *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines*. San Rafael, CA, USA: Morgan & Claypool, 2009.

[2] C. Delimitrou and C. Kozyrakis, “Paragon: QoS-aware scheduling for heterogeneous datacenters,” in *Proc. 18th Int. Conf. Archit. Support Program. Languages Operating Syst.*, 2013, pp. 77–88.

[3] J. Mars and L. Tang, “Whare-map: Heterogeneity in “Homogeneous” warehouse-scale computers,” in *Proc. 40th Annu. Int. Symp. Comput. Archit.*, 2013, pp. 619–630.

[4] LAMMPS. [Online]. Available: http://lammps.sandia.gov/

[5] NAMD. [Online]. Available: http://www.ks.uiuc.edu/Research/namd/

[6] OpenFOAM. [Online]. Available: http://http://www.openfoam.com/

[7] Apache Hadoop. [Online]. Available: http://hadoop.apache.org/

[8] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, “Spark: Cluster computing with working sets,” in *Proc. 2nd USENIX Conf. Hot Topics Cloud Comput.*, 2010, pp. 10–10.

[9] Amazon EC2 Reserved Instances Pricing. [Online]. Available: https://aws.amazon.com/ec2/pricing/reserved-instances/pricing/

[10] J. Han, C. Kim, J. Huh, G. J. Jang, and Y. Choi, “Configuration guidance framework for molecular dynamics simulations in virtualized clusters,” *IEEE Trans. Serv. Comput.*, vol. 10, no. 3, pp. 366–380, May/Jun. 2017.

[11] J. Han, S. Jeon, Y. Choi, and J. Huh, “Interference management for distributed parallel applications in consolidated clusters,” in *Proc. 21st Int. Conf. Archit. Support Program. Languages Operating Syst.*, 2016, pp. 443–456.

[12] 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 *Proc. 44th Annu. IEEE/ACM Int. Symp. Microarchit.*, 2011, pp. 248–259.

[13] I. Raicu, I. T. Foster, and Y. Zhao, “Many-task computing for grids and supercomputers,” in *Proc. 1st Workshop Many-Task Comput. Grids Supercomput.*, 2008, pp. 1–11.

[14] H. Yang, A. Breslow, J. Mars, and L. Tang, “Bubble-flux: Precise online QoS management for increased utilization in warehouse scale computers,” in *Proc. 40th Annu. Int. Symp. Comput. Archit.*, 2013, pp. 607–618.

[15] C. Delimitrou and C. Kozyrakis, “Quasar: Resource-efficient and QoS-aware cluster management,” in *Proc. 19th Int. Conf. Archit. Support Program. Languages Operating Syst.*, 2014, pp. 127–144.

[16] Amazon EC2 Instance Types. [Online]. Available: https://aws.amazon.com/ec2/instance-types/

[17] SPEC MPI 2007. [Online]. Available: https://www.spec.org/mpi2007/

[18] NPB. [Online]. Available: https://www.nas.nasa.gov/publications/npb.html

[19] R. Eglese, “Simulated annealing: A tool for operational research,” *Eur. J. Oper. Res.*, vol. 46, no. 3, pp. 271–281, 1990.

[20] J. Dean and S. Ghemawat, “MapReduce: Simplified data processing on large clusters,” in *Proc. 6th Conf. Symp. Opearting Syst. Des. Implementation*, 2004, pp. 10–10.

[21] M. S. Müller, M. van Waveren, R. Lieberman, B. Whitney, H. Saito, K. Kumaran, J. Baron, W. C. Brantley, C. Parrott, T. Elken, H. Feng, and C. Ponder, “SPEC MPI2007—An application benchmark suite for parallel systems using MPI,” *Concurrency Comput.: Practice Exp.*, vol. 22, no. 2, pp. 191–205, Feb. 2010.

[22] R. Riesen, “Communication patterns,” in *Proc. 20th Int. Conf. Parallel Distrib. Process.*, 2006, Art. no. 8.

[23] E. Frank, M. A. Hall, and I. H. Witten, *Data Mining: Practical Machine Learning Tools and Techniques*, 4th ed. San Mateo, CA, USA: Morgan Kaufmann, 2016.

[24] C. M. Bishop, *Pattern Recognition and Machine Learning*. Berlin, Germany: Springer, 2006.

[25] S. R. Gunn, “Support vector machines for classification and regression,” ISIS Tech. Rep. 14.1, pp. 5–16, 1998.

[26] L. Rokach and O. Maimon, “Top-down induction of decision trees classifiers - a survey,” *IEEE Trans. Syst. Man Cybern. Part C Appl. Rev.*, vol. 35, no. 4, pp. 476–487, Nov. 2005.

[27] C. McCue, *Introduction Data Mining and Predictive Analysis*, 2nd ed. London, U.K.: Butterworth-Heinemann, 2015.

[28] T. Dwyer, A. Fedorova, S. Blagodurov, M. Roth, F. Gaud, and J. Pei, “A practical method for estimating performance degradation on multicore processors, and its application to HPC workloads,” in *Proc. Int. Conf. High Perform. Comput. Netw. Storage Anal.*, 2012, Art. no. 83.

[29] L. T. Yang, X. Ma, and F. Mueller, “Cross-platform performance prediction of parallel applications using partial execution,” in *Proc. ACM/IEEE Conf. Supercomput.*, 2005, pp. 40–40.

[30] Q. Llull, S. Fan, S. M. Zahedi, and B. C. Lee, “Cooper: Task colocation with cooperative games,” in *Proc. IEEE Int. Symp. High Perform. Comput. Archit.*, 2017, pp. 421–432.

[31] F. Romero and C. Delimitrou, “Mage: Online and interference-aware scheduling for multi-scale heterogeneous systems,” in *Proc. 27th Int. Conf. Parallel Archit. Compilation Techn.*, Nov. 2018, Art. no. 19.

[32] H. Herodotou, F. Dong, and S. Babu, “No one (cluster) size fits all: Automatic cluster sizing for data-intensive analytics,” in *Proc. 2nd ACM Symp. Cloud Comput.*, 2011, Art. no. 18.

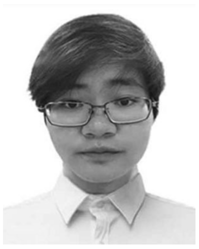
[33] O. Alipourfard, H. H. Liu, J. Chen, S. Venkataraman, M. Yu, and M. Zhang, “Cherrypick: Adaptively unearthing the best cloud configurations for big data analytics,” in *Proc. 14th USENIX Symp. Netw. Syst. Des. Implementation*, 2017, pp. 469–482.

[34] S. Venkataraman, Z. Yang, M. Franklin, B. Recht, and I. Stoica, “Ernest: Efficient performance prediction for large-scale advanced analytics,” in *Proc. 13th Usenix Conf. Netw. Syst. Des. Implementation*, 2016, pp. 363–378.

[35] S. Kundu, R. Rangaswami, A. Gulati, M. Zhao, and K. Dutta, “Modeling virtualized applications using machine learning techniques,” in *Proc. 8th ACM SIGPLAN/SIGOPS Conf. Virtual Execution Environ.*, 2012, pp. 3–14.

[36] G. Wu, J. L. Greathouse, A. Lyashevsky, N. Jayasena, and D. Chiou, “GPGPU performance and power estimation using machine learning,” in *Proc. IEEE 21st Int. Symp. High Perform. Comput. Archit.*, 2015, pp. 564–576.

Seontae Kim received the BS degree in computer science and engineering from the Ulsan National Institute of Science and Technology (UNIST), Ulsan, Republic of Korea, in 2013. His research interests include cloud computing, virtualization, and task scheduling.



Nguyen Pham received the BEE degree in mathematics and computer science, Honor program from the University of Science, VNUHCM, and the master’s degree in electrical and computer engineering from the Ulsan National Institute of Science and Technology. Her research interests include high performance computing, big data processing, resource scheduling, and machine learning.

Woongki Baek received the BS and MS degrees from Seoul National University, and the PhD degree in electrical engineering from Stanford University, in 2011. He is an associate professor with the School of Electrical and Computer Engineering, UNIST. His research interests include architecture, system software, and programming models for parallel computing. Prior to joining UNIST, he worked for Microsoft as a platform software engineer and for ETRI as a senior researcher. He was the recipient of the Stanford Graduate Fellowship and the Samsung Scholarship. He is a member of the ACM and IEEE.

Young-ri Choi received the BS degree in computer science from Yonsei University, and the MS and PhD degrees in computer science from the University of Texas at Austin. Her research interests include cloud computing, scientific computing, big data analytics platforms, and network protocols. She is an associate professor with the School of Electrical and Computer Engineering, Ulsan National Institute of Science and Technology (UNIST). She is a member of the IEEE.