Configuration Guidance Framework for Molecular Dynamics Simulations in Virtualized Clusters

Jaeung Han, Changdae Kim, Jaehyuk Huh, Gil-Jin Jang, and Young-ri Choi

Abstract—With the advancement of cloud computing, there has been a growing interest in exploiting demand-based cloud resources for parallel scientific applications. To satisfy different needs for computing resources, cloud providers provide many different types of virtual machines (VMs) with various numbers of computing cores and amounts of memory. The cost and execution time of a scientific application vary depending on the types of VMs, number of VMs, and current status of the cloud due to interference among VMs. However, currently, cloud users are solely responsible for selecting the most effective VM configuration for their needs, but often end up with sub-optimal selections. In this paper, using molecular dynamics simulations as a case study, we propose a framework to guide users to select the optimal VM configurations that satisfy their requirements for scientific parallel computing in virtualized clusters. For molecular dynamics computation on a cluster of VMs, the guidance framework uses artificial neural networks which are trained to predict its execution times for various inputs, VM configurations, and status of interference among VMs. Using our performance prediction mechanisms, the guidance framework helps users choose an optimal or near-optimal VM cluster configuration under cost and runtime constraints.

Index Terms—Cloud computing, scientific applications, virtual cluster configuration, molecular dynamics simulations, performance modeling and prediction,

1 Introduction

Cloud computing provides users with elastic computing resources. Such utility-based computing eliminates costs for purchasing and operating physical machines, which are often over-provisioned to meet peak demands. Due to such an advantage of elastic resource provisioning, cloud computing is getting popular for scientific computing, which traditionally has been relying on private massive computing clusters [11], [22], [45].

To satisfy various needs for computing resources, cloud providers provide many different types of machines with various numbers of computing cores and different amounts of memory. Virtualization enables such a variety of machine types, even from the same set of physical machines. Using virtualization, cloud providers support different virtual machines (VMs) out of the same physical machine. The costs for using computing resources primarily depend on the types of VMs, and the number of VMs (or VM counts). The costs will be proportional to the amount of resources, and the time to use the resources.

In the current cloud computing environments which support many different types of VMs with various cores and amounts of memory, cloud users are solely responsible for selecting the optimal VM types and number of VMs for their needs. However, choosing a right configuration of a virtual cluster of VMs for a scientific application can be non-trivial work. Even the same scientific applications require different amounts of computing resources, depending on inputs, and the memory requirements may vary by the data size of applications and their inputs. Also, for a scientific parallel application, adding more cores or more VMs does not reduce the execution time if the parallel efficiency of the application is low, while the cost is proportional to the number of cores or VMs.

For scientific applications, many users commonly use an off-the-shelf software package with user-specific input sets to solve a problem, instead of writing in-house codes for each problem. For example, a generic molecular dynamics simulator can be configured to solve the users' problem with a given input set. These users, who are mostly domain scientists, spend most of their times to correctly run experiments, and analyze data. Leaving the responsibility of choosing right cloud resources on such users can lead to the sub-optimal selections of VM types and counts. Without proper guidance, users often pay for the less cost-effective resources for their needs.

Furthermore, optimal VM configuration may change depending on interference from other VMs, and users may prioritize different requirements such as cost, runtime, and parallel efficiency. Careful consideration of VM configurations can improve the efficiency of virtualized clusters; especially important in the case of shared institutional resources.

In this paper, we propose a framework to guide users to select the optimal VM types and counts that satisfy their requirements in virtualized clusters. The guidance framework searches the best VM types and counts for given input parameters and constraints such as minimizing execution time or cost from users. The framework requires a prediction model for execution times, i.e. runtimes, and memory requirements. We use artificial neural network (ANN) models to predict execution times and memory

- *J. Han, C. Kim and J. Huh are with the Department of Computer science, KAIST, E-mail: juhan@calab.kaist.ac.kr, cdkim@calab.kaist.ac.kr, jhhuh@kaist.ac.kr*
- *G. Jang is with the the School of Electronics Engineering Kyungpook National University, E-mail: gjang@knu.ac.kr*
- *Y. Choi is with the School of Electrical and Computer Engineering, UNIST, E-mail: ychoi@unist.ac.kr*

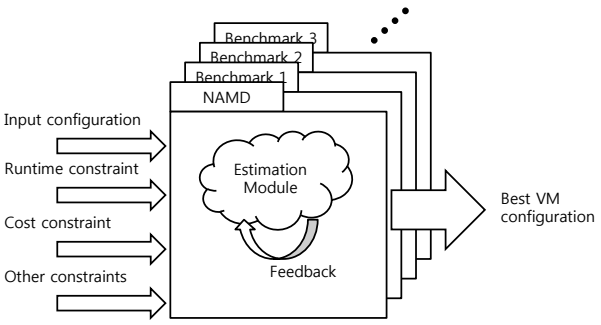


Fig. 1. System overview

requirements for a software package with various inputs under cost and runtime constraints. Figure 1 describes the guidance system overview. The framework has a performance estimation model for a software package. Using the estimation model, the guidance system finds the best VM configuration for input parameters, and constraints given by a user.

As a case study, we use two molecular dynamics simulators, NAMD [4] and LAMMPS [2] for our target software, which are commonly used by many scientists, although the proposed framework is not limited to a specific software package. It has been shown that for many scientific applications, reasonable performance estimation is possible with various models [46], [32], [23]. However, virtualized clusters open a new challenge, since with the same physical cluster, different VM configurations are possible, and interferences among VMs exist as other workloads may share the cluster. Unlike dedicated private physical clusters, in virtualized clusters, interference among user VMs running on the same host can affect the performance of scientific applications. This paper also investigates the effect of VM interference by shared caches and memory. We incorporate the interference effect on the performance prediction model to find the best VM configuration, considering the current cloud status.

Our results demonstrate that such a guidance system to find the optimal VM configuration is feasible in virtualized clusters for scientific parallel workloads. For NAMD and LAMMPS, our guidance system can find an optimal or near-optimal configuration under some constraint specified by users, considering cost, runtime, and parallel efficiency of running scientific applications.

Compared to the prior work on performance estimation for scientific workloads [46], [32], [23], the new contributions of this paper are as follows. First, this paper reveals the importance of selecting an optimal VM configuration provided by cloud-based computing for scientific workloads. The prior work addresses the performance estimation only with a fixed physical node configuration, without considering flexible virtual nodes. Second, the paper quantitatively shows that an ANN-based model is feasible for modeling the performance of a virtual cluster consisting of various different types of virtual machines. Third, this work addresses the effect of interference among VMs running in the same physical node, although this paper uses a limited interference model.

The rest of the paper is organized as follows. Section 2 discusses our methodology including molecular dynamics simulations and the cost model for clusters of VMs. Section 3 analyzes the importance of virtual cluster configurations. Section 4 presents the architecture of our framework, and Section 5 discusses our results using NAMD and LAMMPS on a private cloud system and Amazon EC2. Section 6 discusses the prior work, and finally, Section 7 concludes the paper.

2 Methodology

2.1 Molecular Dynamics Simulators

As a case study, we examine two popular molecular dynamics (MD) simulators, NAMD [4] and LAMMPS [2], for scientific applications. They run on a single processor as well as on parallel computers, communicating via either MPI (Message Passing Interface) or common UDP stack. A molecular dynamics simulator has large parameter spaces for particle types, force fields, constraints, boundary conditions, and simulation parameters. Note that we select NAMD and LAMMPS for our study, but our framework is not limited to these software packages.

As discussed in [36], MD simulations are compute-intensive with respect to the number of atoms and number of time steps to simulate. Usually, thousands or millions of atoms are simulated in three dimensions, and tens or hundreds of thousands of time steps, which have the femtosecond scale, are simulated for even picoseconds of real time.

For each time step, the forces of atoms are computed based on the interaction with other atoms, and their motions are computed by integrating Newton's equation [37]. For parallel computation, each processor owns a set of atoms, which can change dynamically, as a simulation proceeds. For a time step, each processor computes velocities, positions, and forces for its owned atoms. It also communicates the computed positions (before force computation), and the computed forces (after force computation) with its neighboring processors.

As with many other scientific applications, the running platform, and input data and parameters affect the performance and scalability of the simulator. Thus, an optimal number of processes to run a given problem becomes different, depending on the simulator, platform and input parameters.

	The number of atoms
5dhfr	23,558
ApoA1	92,224
ATPase	327,506
STMV	1,066,628

TABLE 1
The number of atoms for MD simulation benchmarks

For the study, we use four MD simulation benchmarks, 5dhfr, ApoA1, ATPase, and STMV. For LAMMPS and NAMD, all the benchmarks are configured to use the all-atom molecular model including bond, angle, dihedral, improper, pair, and coulomb potentials with long-range treatment using PPPM (Particle-Particle-Particle-Mesh) [17], and PME (Particle-Mesh-Ewald) [10], respectively. In our simulations with NAMD and LAMMPS, we use the default value, 1.0 femtosecond, for the size of time steps. Note that the time step of 1 femtosecond is used to accurately compute bonded forces under our all-atom model including flexible bonds to hydrogen [35], [20], while longer time steps such as 4 femtoseconds can be used to compute slow forces [19].

Each benchmark has a different number of atoms and molecular topology. Table 1 shows the number of atoms in each of

the four benchmarks. These four benchmarks were selected, since the four protein systems cover different problem sizes, allowing the validation of our system with a wide range of problem size. For the benchmarks, we vary application parameters for force fields, constraints, and simulation parameters, which have effects on the performance of the applications. For force fields, we vary the values of the cut-off distance between pairs of atoms, and grid sizes of three-dimensional FFT for long-range coulomb potential treatment in PPPM and PME. We also vary the values of a parameter, neighbor skin, which defines an extra distance beyond the cut-off distance.

2.2 Cost Model for Virtual Clusters

In this section, based on the pricing model used by a commercial cloud provider, we construct a cost model to evaluate our guidance system. Although the cost model is based on the commercial cloud services, the model provides a reasonable basis for how much it can cost to support each virtual machine type. However, the cost model discussed in this section is to present a possible concrete cost structure of a cloud service as an example, which may not be generalized.

Many cloud providers such as Amazon EC2 [1], Microsoft Windows Azure [3], and Rackspace [5] offer pricing models where users are charged by the time (in hours) executed for each virtual machine. These providers also offer a set of predefined virtual machines types. Each type is different from one another with respect to their hardware specification like the number of virtual CPUs (vCPUs), the amount of memory and storage, and its cost charged per hour. The variety of VM types will further increase, as the number of available cores on physical machines increases with scaling multi-core architectures.

Table 2 shows 23 virtual machine types and their specifications provided by Amazon EC2. In the table, the CPU performance of each VM type is specified as the product of the number of vCPUs and clock speed of a physical CPU, and the networking performance is classified into *moderate*, *high*, and *10 Gigabit*, whose values are assigned 1, 2, and 3, respectively. For storage, each of VM types is configured with one of three different storage types, SSD, HDD, and Amazon Elastic Store (EBS). However, for a VM type with EBS, the amount of storage is specified as 0, since a user should configure the storage of a VM separately with EBS which is charged based on an uncombined pricing model.

In this work, to compare the prices of different VM configurations, we use a simple pricing model, which is consistent with the Amazon EC2 prices. The cost of a VM can be represented by a simple function, where four factors, CPU, memory, and storage, and network, determine the price. The cost function of a virtual machine can be computed as follows:

$$c_{cpu} * C * F + c_m * M + c_s * S + c_{net} * NET$$

where C is the number of vCPUs, F is the clock speed of a physical CPU, M is the amount of memory in GB, S is the amount of storage in GB, and NET is the networking performance. c_{cpu} is the unit cost, dollars per hour, of CPU performance, c_m is the unit cost for 1 GB memory, c_s is the unit cost for 1 GB storage, and c_{net} is the unit price of networking performance. For c_s, there are two different types, c_{ssd} for SSD storage, and c_{hdd} for HDD storage.

Based on the above price function, the coefficients are computed by a multi-variable linear regression technique to fit the function to the EC2 price model. The computed values of c_{cpu}, c_m, c_{ssd}, c_{hdd} and c_{net} as follows:

$$c_{cpu} = 1.44 \times 10^{-2}$$
$$c_m = 7.11 \times 10^{-3}$$
$$c_{ssd} = 1.91 \times 10^{-4}$$
$$c_{hdd} = 5.58 \times 10^{-5}$$
$$c_{net} = 2.85 \times 10^{-4}$$

For this analysis, we use the costs for US East (Virginia) region and Linux usage in June 2015. Using those coefficients, we compute the estimated costs for virtual machine types of Amazon EC2, and compare the estimated costs with the actual costs in the last two columns of Table 2.

Based on the cost analysis of Amazon EC2, we define a new cost model to estimate the cost in our own cloud environment, where physical machines are configured with 2.0GHz CPU and HDD, and connected via a 10 Gigabit Ethernet switch. Moreover, we configure each VM to have 10 GB HDD so that it has sufficient storage to install all needed software for NAMD and LAMMPS and execute them. Thus, in our model, F, S, and NET are assigned to 2.0, 10, and 3, respectively, and c_{hdd} is used for storage. Our cost model is defined as follows:

$$2.88 \times 10^{-2} * C + 7.11 \times 10^{-3} * M + 1.41 \times 10^{-3}$$

Type	CPU	Mem (GB)	Storage (GB)	Network	Act. cost ($/hour)	Est. cost ($/hour)
m4.large	4.8	8	0[3]	1	0.126	0.126
m4.xlarge	9.6	16	0[3]	2	0.252	0.252
m4.2xlarge	19.2	32	0[3]	2	0.504	0.504
m4.4xlarge	38.4	64	0[3]	2	1.008	1.008
m4.8xlarge	38.4	160	0[3]	3	2.52	2.519
m3.medium	2.5	3.75	4[1]	1	0.067	0.064
m3.large	5	7.5	32[1]	1	0.133	0.132
m3.xlarge	10	15	80[1]	2	0.266	0.266
m3.2xlarge	20	30	160[1]	2	0.532	0.532
c4.large	5.8	3.75	0[3]	1	0.11	0.110
c4.xlarge	11.6	7.5	0[3]	2	0.22	0.220
c4.2xlarge	23.2	15	0[3]	2	0.441	0.441
c4.4xlarge	46.4	30	0[3]	2	0.882	0.882
c4.8xlarge	92.8	60	0[3]	3	1.763	1.763
c3.large	5.6	3.74	32[1]	1	0.105	0.113
c3.xlarge	11.2	7.5	80[1]	1	0.21	0.230
c3.2xlarge	22.4	15	160[1]	2	0.42	0.460
c3.4xlarge	44.8	30	320[1]	2	0.84	0.919
c3.8xlarge	89.6	60	640[1]	3	1.68	1.838
d2.xlarge	9.6	30.5	6000[2]	1	0.69	0.69
d2.2xlarge	19.2	61	12000[2]	2	1.38	1.38
d2.4xlarge	38.4	122	24000[2]	2	2.76	2.76
d2.8xlarge	86.4	244	4800[2]	3	5.52	5.52

[1] These instances use SSD storage.
[2] These instances use HDD storage.
[3] These instances are only available with Amazon Elastic Block Store (EBS).

TABLE 2
Amazon EC2 virtual machine types

2.3 Virtual Machine Types

As shown in the previous section, in our price model, two factors, computing cores (CPU resource) and memory, determine the cost per unit time. In this work, we offer 4 types of CPU resource, 1 vCPU, 2 vCPUs, 4 vCPUs, and 8 vCPUs per VM.

# of hosts	1 host	2 hosts	4 hosts	8 hosts	10 hosts
# of vCPUs	total # of VMs in virtual cluster				
1	8	16	32	64	80
2	4	8	16	32	40
4	2	4	8	16	20
8	1	2	4	8	10

TABLE 3
Virtual cluster configurations

Unlike the predefined memory size for each VM type in Amazon EC2, we use a flexible memory model for VMs. For each vCPU/VM type, the memory of a VM can be configured between 1 GB and 14 GB, but the smallest unit of the memory resource is 1 GB. Thus, with the combinations of CPU and memory resource types, it is possible that the model provides total 56 types of VMs. We use such a flexible memory model, as it can be potentially more cost-effective for cloud users.

In this paper, the memory size of a VM is determined by the minimum memory required to execute an application with given input parameters properly. The minimum amount of memory is computed based on the maximum memory usage to run the application with the input parameters.

2.4 Setups for Virtual Clusters

Our private cloud system consists of a cluster of ten physical machines connected via a 10 Gigabit Ethernet switch. Each machine has two Intel Xeon E5-2650 processors and 48 GB memory, and each processor has eight cores. We use Xen hypervisor version 4.1 for virtualization. The guest operating system for virtual machines is a para-virtualized Linux of kernel version 3.1.0. For inter-VM communication in NAMD and LAMMPS, the typical UDP stack is used.

On executing MD simulators, we explore 20 configurations of virtual clusters (VCs) as given in Table 3. We use virtual machines with 1 vCPU, 2 vCPUs, 4 vCPUs or 8 vCPUs over 1, 2, 4, 8, and 10 hosts. Among 16 physical cores on each host, only 8 cores are used for the scientific workload, while the remaining 8 cores are reserved for co-running VMs that execute different workloads. For example, if we use a virtual cluster configuration with 2 vCPUs, then only 4 VMs will be allocated on each host to run a MD simulator (either NAMD or LAMMPS). In real cloud computing environment, it is likely that VMs running a parallel application (like a MD simulator) are deployed over separate hosts, and each of the VMs is executed with other co-runner VMs on the same host. Thus, the above setup makes more realistic cloud computing environment. We do not over-commit CPU resources for VMs, and thus the total number of vCPUs of VMs running on a host is always equal to or less than the number of physical cores on the host.

To reduce the number of VC configurations that need to be evaluated, we use the maximum memory available in a host, instead of varying the memory size for each VM type. However, for cost calculation, we use the maximum memory size used by a VM as the memory size of the VM. For example, we may run a MD simulator on a VM with 1 vCPU and 14 GB memory, and the simulator may use only the maximum 1.5 GB memory during the run. Assuming 2 GB is enough to run the simulator, the cost is inferred with 2 GB memory size and 1 vCPU. For memory configuration of a VM, we distribute 40 GB evenly to each of all the VMs running on the same host, regardless of whether it runs a MD simulator or other workloads.

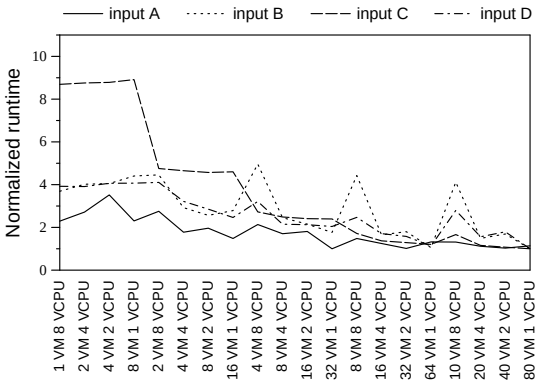


Fig. 2. Normalized runtimes of NAMD on 4 STMV inputs

For a benchmark application with a given input set, we run it for NAMD and LAMMPS over 20 configurations, and measure three metrics, *runtime*, *looptime*, and *memory usage*. Runtime is the total time spent for executing the problem, and looptime is the time spent for executing the main loop of the problem, excluding the initial and final phase of the execution. The looptime is proportional to the number of simulated time steps, which is configurable in a MD simulator. To reduce the time for running each benchmark application over different configurations, we use a relatively small number of time steps. However, as shown in Section 4, the looptime can be estimated almost accurately by multiplying the time per time step with the number of time steps.

3 Importance of Virtual Cluster Configurations

In this section, we apply the cost model presented in the previous section to the MD simulations with NAMD and LAMMPS on various VM types and the number of VMs. By evaluating the molecular dynamics simulators with various benchmark applications and VC configurations, we present how the selection of VC configurations affects the execution times and costs for virtual clusters. In addition, we investigate the effect of interference on the performance of the target parallel applications.

3.1 Runtimes and Costs with Various VC Configurations

Runtime and cost to run an application in cloud systems vary significantly, depending on used VC configurations, and given input parameters of the application. Different VC configurations can provide different amount of resources, and also affect the communication pattern of the application. Moreover, the input parameters can change the resource usage noticeably. In this section, we show that there is no single best VC configuration for NAMD and LAMMPS.

Figures 2 and 3 present normalized runtimes and costs for completing the NAMD application with four different sets of input parameters over 20 VC configurations. For x-axis, m VM n VCPU denotes m VMs with n vCPUs per VM. In these figures, for a given input, each runtime is normalized to the minimum runtime, and each cost is normalized to the minimum cost over all the configurations for the same input. For these results, only VMs that run the NAMD application are executed on hosts without any

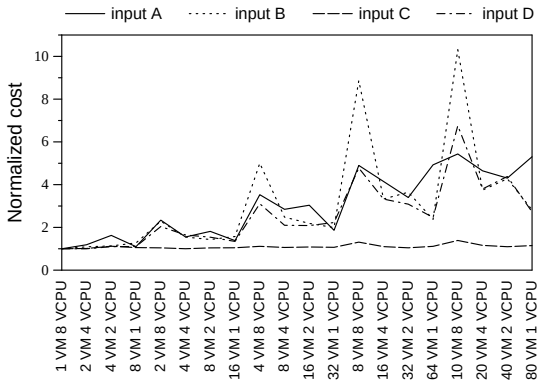


Fig. 3. Normalized costs of NAMD on 4 STMV inputs

co-runner VMs. The runtimes and costs change considerably, as the numbers of vCPUs and VMs change, and the four input sets exhibit quite different patterns in runtimes and costs. In case of input C, using the small number of total vCPUs cannot reduce the overall cost (as shown in Figure 3), unlike the other inputs, because its runtime decreases significantly when the large number of total vCPUs is used (as shown in Figure 2). Moreover, even when the same number of total vCPUs is used, runtimes and specially costs are quite different depending on used VC configurations.

Using the same number of total vCPUs, VC configurations with 8 vCPUs tend to increase runtimes and incur much larger cost than the other configurations. This is because processing all network packets for 8 MPI processes within a VM can become a performance bottleneck. Note that as grid sizes of three-dimensional FFT get larger, the amount of data to be exchanged among the processes increases, and so runtimes for inputs with large grid sizes such as input B increase noticeably, while those for inputs with small grid sizes such as input C are not affected, when VMs with 8 vCPUs are used.

3.2 Optimal VC Configurations

To show that optimal VC configuration depends on user requirements such as minimizing runtime or cost, we ran two MD simulators, NAMD and LAMMPS, with 108 and 97 different input parameters discussed in Section 2.1, respectively. For the results in this section, we executed only VMs running either NAMD or LAMMPS on hosts without any co-runner VMs. Figures 4 and 5 present the distributions of the best VC configurations for NAMD and LAMMPS, respectively, over various criteria. In these figures, the x-axis lists 20 possible VC configurations, where m:n denotes m VMs with n vCPUs per VM, and y-axis shows the percentage of input sets, which have the corresponding VC configuration as the best one.

Figures 4a and 5a show the distributions of the best VC configurations for minimizing runtime to complete the execution for NAMD and LAMMPS, respectively, while Figures 4b and 5b show the distributions of the best VC configurations for minimizing cost. Note that this cost includes the effect of different memory requirements. For both of NAMD and LAMMPS, the distribution of the best VC configurations for minimizing cost is quite different from that for minimizing runtime. Using the large number of total vCPUs likely reduces runtime, while using the small number of total vCPUs is likely less expensive.

The best VC configurations optimizing for runtime only, and specially for cost only may be predictable. However, users may impose some additional constraint, along with a target criterion or goal, or consider two or more criteria together to find the best configuration for their needs. In this case, finding the best VC configuration gets more complicated. Figures 4c and 5c present the distributions of the best VC configurations for minimizing both cost and runtime for NAMD and LAMMPS, respectively. We use a new metric which combines the cost and runtime aspects, called *cost-runtime-product*. The cost-runtime-product is defined as the multiplication of cost and runtime for running an application with an input set. This metric can provide a single metric considering both cost and runtime. Unlike the best configuration for runtime or for cost, the configurations with large or small numbers of total vCPUs become less efficient for cost-runtime-product.

Figures 4d and 5d present the distributions of the best VC configurations for minimizing runtime with a constraint that its cost does not exceed 1.5 times the minimum cost, and Figures 4e and 5e present the best VC configuration distribution for minimizing cost with a constraint that its runtime does not exceed 1.5 times the minimum runtime. In these figures, the best VC configurations are widely distributed over various configurations, compared to the results for optimizing runtime and/or cost without any constraint.

Parallel efficiency, which is the ratio of ideal to actual runtime, can be also provided as constraints. Figures 4f and 5f present the distribution of the best VC distributions for minimizing runtime with a constraint that its parallel efficiency is at least 50%. (For example, if the runtime for 1 VM with 8 vCPUs is 20 seconds, and 2 VMs with 8 vCPUs are used, the ideal runtime is 10 seconds, since the number of vCPUs used is doubled. If the runtime for 2 VMs with 8 vCPUs is 16 seconds, then the parallel efficiency is 10/16*100, i.e. 62.5%. Thus, this satisfies the constraint.) Constraints on parallel efficiency can be used to filter expensive VC configurations with the large number of total vCPUs, which reduce runtime only slightly, compared to the configurations with the less number of total vCPUs.

Based on the above results, we can conclude that the optimal VC configuration for an application with a given input can be arbitrarily different depending on the criteria or goals to optimize, and constraints imposed by users. Thus, finding the optimal VC configuration is hard for users, and inaccurate prediction may lead to sub-optimal results.

3.3 Interference with Co-runner Virtual Machines

The optimal VC configurations in the previous section have been selected by evaluating the results without executing co-runner VMs which run other workloads. However, in practical cloud computing systems, resources are unlikely dedicated to a particular workload, and their performance may suffer from contention on shared resources. Some resources like CPU and memory may be easier to partition and allocate to each VM. However, other resources like network and disk I/O subsystems, and architectural resources like caches, and memory controllers are not easy to partition and allocate to each VM.

There have been earlier efforts to reduce performance interference due to cache sharing and/or non-uniform memory access (NUMA) effects [47], [9], [7], [38]. The effects of shared caches and/or NUMA can be mitigated by scheduling-based techniques that schedule threads in a multi-core system, or VMs on multiple virtualized hosts to minimize the overall last-level cache (LLC) miss rate as studied in [47], [9], [7].

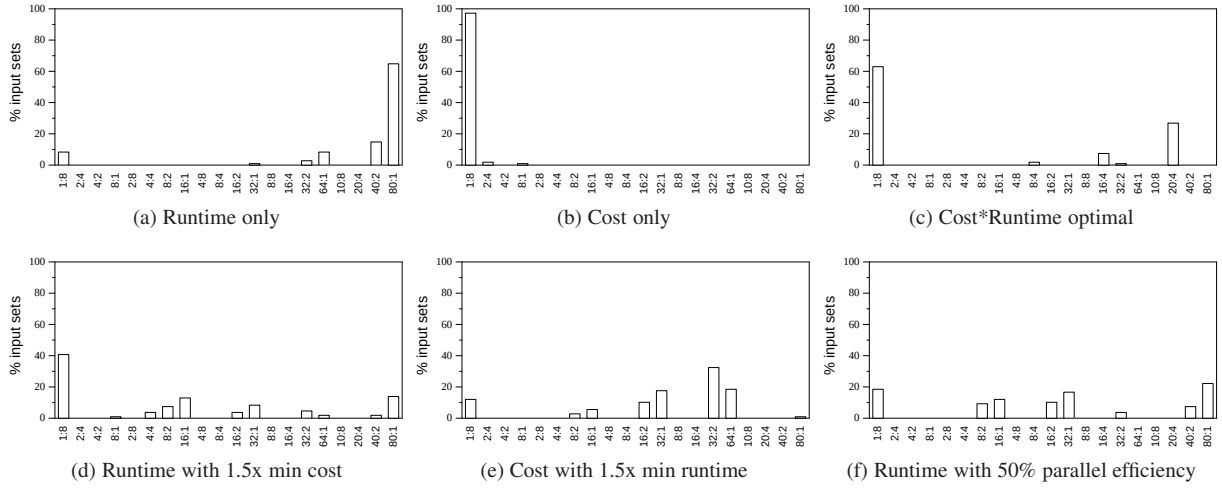


Fig. 4. The best VC configuration distributions with various criteria on NAMD

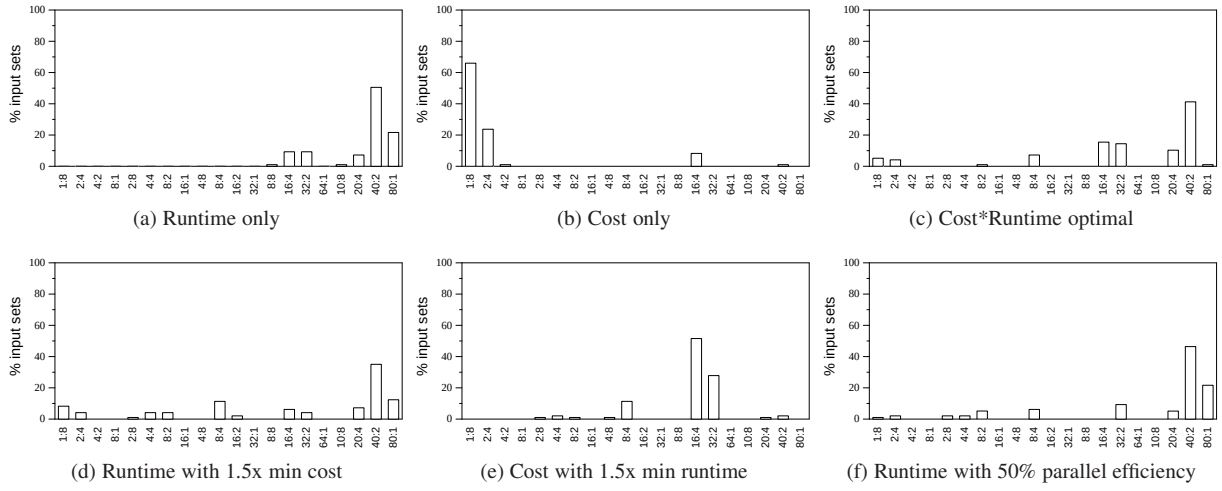


Fig. 5. The best VC configuration distributions with various criteria on LAMMPS

In [38], Rao et al. proposed a VM scheduling algorithm in a NUMA multi-core system, which uses a new performance metric, considering remote access penalty, LLC contention, and cross-chip data sharing overhead. In [33], Nathuji et al. investigated a technique that guarantees the application level performance of consolidated virtual applications running on a multi-core server, by allocating additional resources to VMs if the performance of the VMs is degraded due to shared cache interference. In [26], a memory model that characterizes the resource usage of an application at all memory levels was investigated on resource shared environments such as virtualized datacenters and clouds.

To reduce interference effects, VM performance can be isolated by controlling resources for I/O processing as in [14], [13]. In [14], a technique to isolate the network I/O resource for each VM running on the same host was proposed, and in [13], a technique to divide the storage I/O resource for each VM in a similar way to divide CPU or memory resources was studied.

Even though many techniques were investigated in literature, the interference effects due to resource sharing are not easily avoidable in cloud computing systems. Thus, we should reflect interference effects on estimating the performance. In this section, we study how interference influences the performance of the scientific workload.

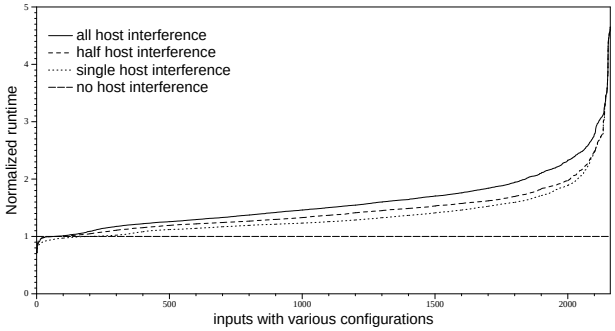


Fig. 6. Normalized runtimes of NAMD sorting each line in ascending order

In this work, we focus on the interference effects due to shared caches and memory among VMs. We generate the interference by executing a synthetic workload, random memory write, along with our scientific workload. Recall that in our setup, only 8 cores out of 16 cores in each host are used for the scientific workload, while the remaining 8 cores are reserved for running other workloads. Thus, the remaining 8 cores can be used to run the random memory

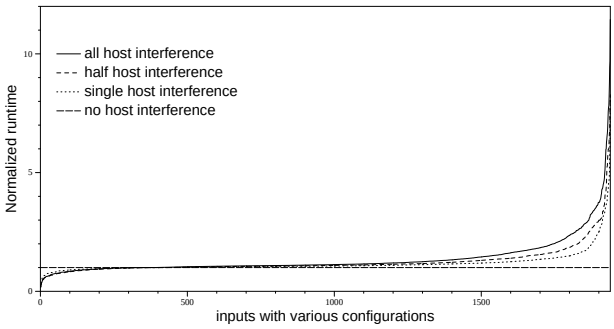


Fig. 7. Normalized runtimes of LAMMPS sorting each line in ascending order

write workload. We configure co-runner VMs to have the same number of vCPUs as the VMs for NAMD or LAMMPS have, and fully utilize all the co-runner VMs by running the random write workload.

Most HPC applications running on parallel computers are tightly coupled. Thus, the overall performance of the applications will degrade if one or more processes slow down due to interference. Interference that the applications experience depends on several factors. In this work, we control the intensity of interference by varying the number of hosts, in which the random write workload is executed, as a single host, half hosts, and all hosts. For the *single host interference*, we run co-runner VMs on only one of the hosts used in a VC configuration, for the *half host interference*, we run them on the half of the hosts, and for the *all host interference*, we run them on the all hosts. Note that for VC configurations with one host, the above three cases are all the same.

Figure 6 presents the runtimes when NAMD runs concurrently with the random write workload, normalized to those when NAMD runs alone without any co-runner VMs. For the x-axis, total 2,160 runs of NAMD for 108 different inputs over 20 VC configurations are listed in ascending order of their normalized runtimes under each of four different interference levels. Executing the scientific workload with the random write workload aggravates performance 1.47 times on average and 4.65 times at maximum, depending on the intensity of interference. Similarly, Figure 7 presents the normalized runtimes of LAMMPS running with the random write workload. For the x-axis, total 1,940 runs of LAMMPS for 97 different inputs over 20 VC configurations are listed in ascending order of the normalized runtimes. The performance of LAMMPS degrades 1.26 times on average, and 11.45 times at maximum.

In summary, we showed that the performance of scientific workloads is affected considerably by the interference with co-runner VMs, and its effect is different depending on an application as well as the intensity of interference. To accurately estimate the performance of scientific workloads in cloud computing systems, the interference and its intensity must be considered. It is important to note that in this work, we include the interference effects due to shared caches and memory in our performance estimation model, but the interference effects due to contention for other resources can be also included in the estimation model, if proper parameters to reflect the interference effects are selected.

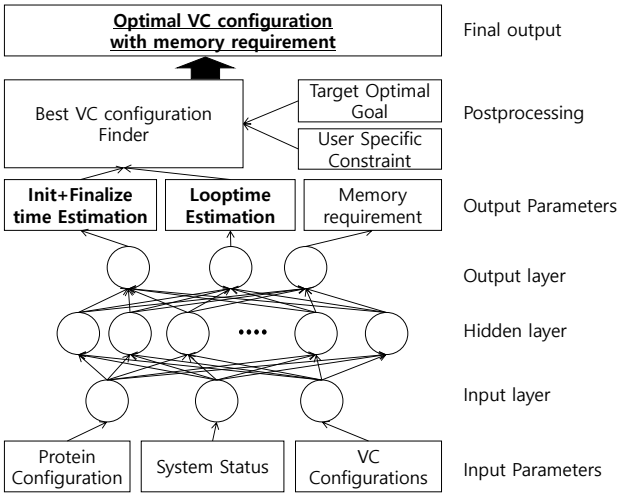


Fig. 8. Architecture of ANN estimation model

Category	Parameter
Protein Configuration	atoms, bonds, angles dihedrals, impropers, rcut fftx,ffty,fftz neighbor
System status	Interference intensity
VC configurations	Total # of VMs vCPUs per VM

TABLE 4
Input parameters of the ANN model

4 ANN-based Guidance Framework

In this section, we present the designs of the proposed guidance framework for virtual clusters. The guidance system employs artificial neural networks (ANNs) to find the best VC configuration with respect to a target optimal goal under some constraint specified by a user. In the framework, the ANN model predicts runtime and memory requirements for each VC configuration, and a post-ANN processing picks the best VC configuration based on the predicted times and memory requirements under the constraint. Figure 8 shows our ANN-based design.

Tables 4 and 5 present the inputs and outputs of the ANN model. The protein configurations are input parameters for a molecular dynamics simulator. The system status for interference and VC configuration are also used as inputs. Note that we limit our search space for the best VC configuration to all possible combinations of 20 configurations given in Table 3 and possible memory configurations.

Interference intensity can be measured only through direct accesses to physical machines, and thus only the cloud provider can measure interference. The proposed framework can be part of a service from the cloud provider to help its users select optimal VM configurations given available machines and interference. For private clouds where users can collect the status of physical machines, the interference can be directly obtained. The proposed framework can still be used without direct accesses to the physical system, ignoring the effect of interference. Although the accuracy is reduced without the consideration for interference, the guidance system can still provide a good configuration, as shown in the Amazon EC2 results in Section 5.3, where we could not use the interference intensity as an input for ANNs.

Parameter Category	Parameter Name
Execution time estimations	Init+Finalize time
	Loop time
Memory requirements	Master node
	Compute node

TABLE 5
Output parameters of the ANN model

Steps	Runtime	Looptime	Looptime/step	Initial+Final
50	90.02	54.59	1.0918	35.43
100	146.31	111.25	1.1125	35.06
500	583.41	547.87	1.0957	35.54
1,000	1,134.70	1,098.85	1.0988	35.85
3,000	3,382.04	3,347.04	1.1156	35.00
5,000	5,794.56	5,759.06	1.1518	35.59
10,000	11,610.05	11,574.9	1.1574	35.15

TABLE 6
Runtime analysis of STMV over various time steps for NAMD

4.1 Artificial Neural Networks

Artificial neural networks (ANNs) are popularly used for solving machine learning problems. Among various types of ANNs, we use a radial basis function network (RBFS) [34] that can predict a target value from a set of input values, which is often referred to as a regression in statistics [41]. Our model is configured as a *fully connected feedforward* network as shown in Figure 8.

A three-layer ANN consists of input, hidden, and output layers. First, given input values are fed into input nodes and forwarded to the hidden nodes through edges. Second, the input and hidden nodes are activated based on the weighted sum of the fed values, and generate signals to feed the nodes connected to them. Finally, the output node generates an output and is used as a predictor for the desired target. The hidden nodes model various latent factors in the system, enabling nonlinear mapping from input values to output targets. This technique is also known to deliver accurate prediction of the complex system, allowing some amount of noises in training input data sets. In training phase, the errors between the generated outputs and the real targets are computed, and the weights in the lower layers are gradually adjusted in a direction of reducing the errors. This kind of ANN training algorithm is called *backpropagation* [25], because the prediction errors are propagated through the network in a reverse direction. Once a model is optimized, predicting an output for an unknown input can be done very quickly if the size of the network is not too large. For both of NAMD and LAMMPS, we use three-layer ANNs with 88 hidden nodes, and experimentally select 0.001 for a learning rate, and 0.05 for a momentum factor.

4.2 Estimating Runtime and Memory Usage

The molecular dynamic simulators are based on the loop of time steps like many other scientific applications. The execution of the simulators is composed of an initialization phase, where parameters are set and atoms are defined, the loop of time steps, which is discussed in Section 2.1, and a final phase which is executed after the loop to finish the simulation. Note that the iteration-based pattern of scientific applications has been analyzed similarly in [46], where the iteration-based pattern is exploited to predict the performance of an application across different platforms based on partial execution.

To understand the iteration-based execution in details, we run experiments of NAMD for an input of STMV over various numbers of time steps on a virtual cluster of 4 VMs with 8 vCPU each (without any co-runner VMs). Table 6 presents the runtime, looptime, looptime per time step, and time for the initialization and final phases of NAMD in seconds. The results show that the number of time steps does not affect the looptime per time step, and the time spent for the initialization and final phases. We also observe the similar behavior for LAMMPS. Thus, in this work, we run NAMD and LAMMPS with a given input for 50 steps, and derive execution times for longer runs based on the those runs.

With the ANNs, we estimate the main looptime, the initial and final phase time, the required memory for the master node, and the required memory for the compute nodes to calculate the cost, based on given inputs of protein configuration, interference intensity, and a VC configuration. The runtime of an application can be estimated easily as the sum of the estimated looptime for given time steps and initial and final phase time. The memory requirements from the master node and compute nodes can be different. Often the master node requires more memory than the compute nodes to gather and summarize the results. Therefore, we estimate two memory requirements separately. The estimated memory requirements are used to find the best memory configurations for master and compute VMs, or used to filter out unsuitable memory configurations for them.

To train ANNs, we adopted several optimizations to improve their accuracy. Firstly, the error of the ANN model can be very high, if the backpropagation algorithm uses the absolute values of runtimes. For example, the ANN model predicts raw runtimes for two cases, 3 seconds and 102 seconds, respectively, but the actual runtimes are 1 second and 100 seconds. Their absolute differences are 2 seconds in both cases, but the error of the former is 200%, while that of the later is only 2%. Therefore, we used the log-scaled runtimes for training ANN models, since log-scaling reduces the difference between the longest and shortest runtimes. In this way, we can get smaller error rates. Additionally, we normalized the log-scaled runtimes to follow the normal distribution with $\sigma = 1, \mu = 0$. Secondly, to provide stable update and ensure the reliability of the ANN model, the coefficients of hidden nodes are updated per 30 samples. Thirdly, to avoid the hazard of a local minimum, the gradient for the previous momentum is added to calculate current coefficients of hidden nodes.

4.3 Finding the Best VC Configurations

To find the best VC configuration under a constraint, the VC configuration generator generates all available VC configurations with various VM types for master and compute nodes and VM counts. As shown in Figure 8, for each VC configuration, the ANN model estimates the looptime, initial and final phase time, memory requirements for the master and compute nodes. Based on the above estimations, the total runtime and cost, cost-runtime-product, and parallel efficiency can be estimated. Using the estimations for all available VC configurations, the VC configuration finder can select the best VC configuration that satisfies the specified optimal goal and constraint (if any) for the given input set. In this ANN-based guidance system, it is also possible to provide more detailed analysis data such as cost and runtime graphs over various VC configurations to users.

To search the best VC configuration under a constraint, we can use an exhaustive search with all possible combinations of VM

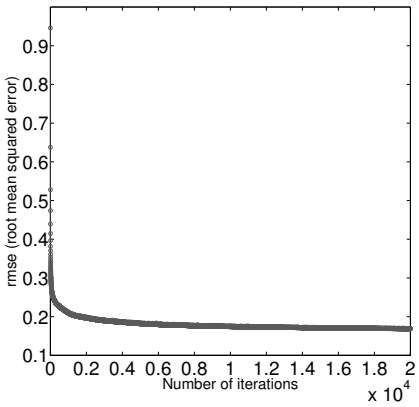


Fig. 9. Root mean squared error over different number of iterations

types and VM counts. If the number of VM types and the number of available machines increase, the exhaustive search may not be efficient for searching. In such cases, more selective searching methods, such binary searching, can be used. However, in this paper, with the relatively moderate number of combinations of VM types and VM counts, searching speed was too short for further effort to improve the searching method. Therefore, we did not investigate the improvement of searching methods.

5 Results

In this section, we present how effectively the proposed guidance system can recommend the best (or optimal) VC configuration to satisfy users' demands. On finding the best VC configuration, provided molecular dynamics input parameters as well as the current status of the cloud due to interference among VMs need to be taken into account. Thus, for 108 and 97 different inputs of NAMD and LAMMPS, respectively, we evaluate 20 different VC configurations described in Section 2.4, under four different levels of VM interference of none, a single host, half hosts, and all hosts. First, we present the accuracy of our ANN-based estimation model for runtimes and memory usages. Second, we present how closely the VC configuration recommended by our system matches the actual best configuration. Third, we discuss the evaluation of the guidance system on a public cloud service, Amazon EC2.

For our ANN model and guidance system, due to the limited number of training samples, we used the *5-fold* cross validation technique [25]. We randomly shuffle the samples, and divide them into 5 different sets. We then use sets 2 - 5, to train the ANN model, and evaluate the performance by using the remaining set (i.e. set 1) in the first round. At the next round, we use set 1 and sets 3 - 5 for training, and evaluate the performance by using set 2. In this way, we finally obtain five ANN models in total, and predict the runtime of every data we have. Every training sample is used up to 20,000 times in adapting the models to ensure convergence. The value of root mean squared error (RMSE) according to the number of iterations for training is shown in Figure 9.

5.1 The Accuracy of Time and Memory Estimation

First, we discuss how accurately the ANNs can estimate the execution times for molecular dynamics simulation. The effectiveness of the proposed guidance system depends on the accuracy of the ANN model, which estimates runtime and memory usage. In the ANN model, we use total 8,640 data samples from

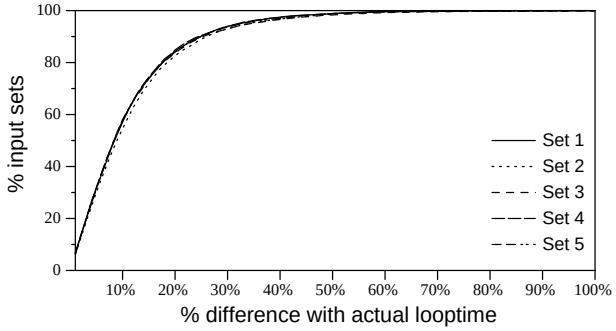


Fig. 10. Cumulative distributions for looptime estimation in NAMD

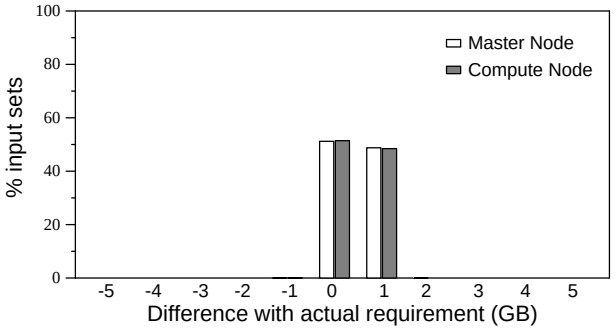


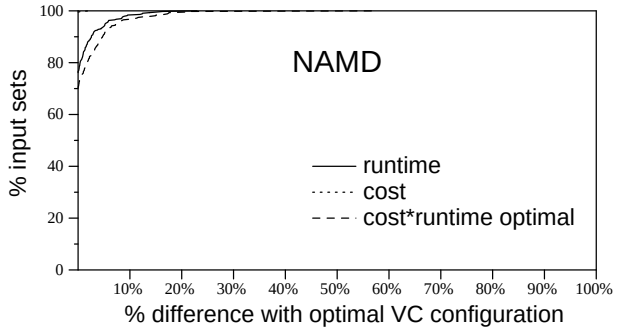
Fig. 11. Memory requirement estimation in NAMD

108 input configurations for NAMD, and 7,760 samples from 97 input configurations for LAMMPS, with 20 VC types, and 4 different interference intensities. To find an accurate ANN model, we tried various values of the learning rate, momentum factor, and number of training samples.

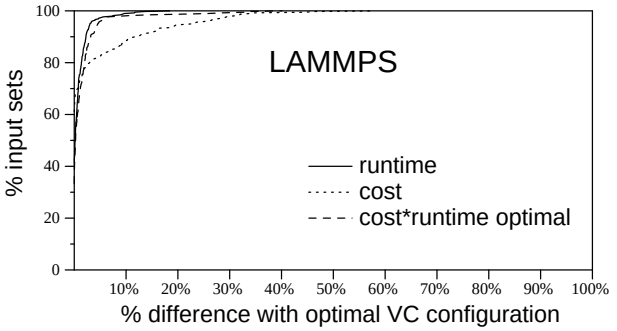
Figures 10 and 11 show the results on looptime and memory size estimation for NAMD. Figure 10 presents the cumulative distributions of the errors for the looptime in NAMD. In realistic executions of NAMD, which simulates many time steps, the looptime dominates the overall execution time. X-axis is the error of the estimated looptime compared to the measured looptime for a given input set of NAMD input parameters, a VC configuration and a level of interference intensity. The figure shows five curves for each of the five ANN modeling sets. Around 80% of input sets have less than 18% error in all the five sets.

Figure 11 presents the distribution of memory size estimation in NAMD. If enough memory is not provided, the application will run significantly slowly or crash. Therefore, we estimate the memory requirement conservatively, giving some room by adding extra memory to the estimation. To reduce the negative differences from the actual memory requirements, the training inputs for the actual memory requirements are rounded up. In the figure, more than 50% of input sets in the validation set match the actual memory requirements. For the rest of the input sets, the model overestimates the memory requirement. Only for 0.035% of master node and 0.116% for compute node of total input sets, the model under-estimates the memory requirement, which makes the actual execution crash. If 1 GB extra memory is added to the estimated memory requirement, such negative difference can be eliminated for all the input sets.

As shown in the results, this ANN model can estimate the runtime and memory size for various application inputs and VC configurations with modest errors. We can use this estimation

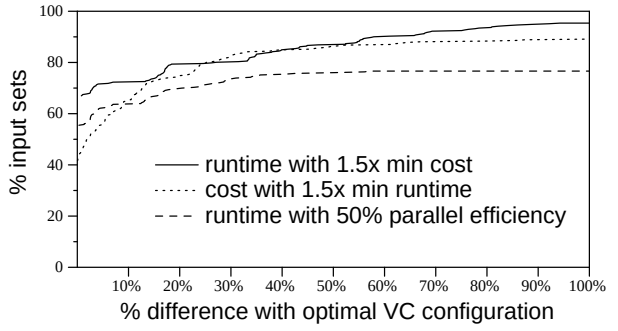


(a) % difference to optimal VC configurations for three goals in NAMD

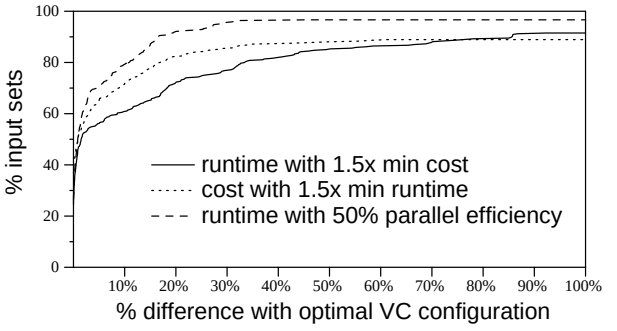


(b) % difference to optimal VC configurations for three goals in LAMMPS

Fig. 12. Cumulative distribution to optimal VC configurations



(a) % difference to optimal VC configurations with constraints in NAMD



(b) % difference to optimal VC configurations with constraints in LAMMPS

Fig. 13. Cumulative distribution to optimal VC configurations with constraints

model to find the best VC configuration. A high degree of accuracy is not necessarily required, as long as the model can estimate relative runtime differences reasonably for different VC configurations. Note that in this section, we presented the results only for NAMD, but the ANNs can estimate the runtime and memory size for LAMMPS with similar accuracy.

5.2 Finding the Best VC Configurations

In this section, we present whether the guidance system can find a VC configuration close to the best VC configuration. For each combination of VM types and VM counts, the required memory is determined by the maximum memory usage with a given input. For running each input under four different levels of interference intensity, 432 and 388 test inputs have to be verified for NAMD and LAMMPS, respectively.

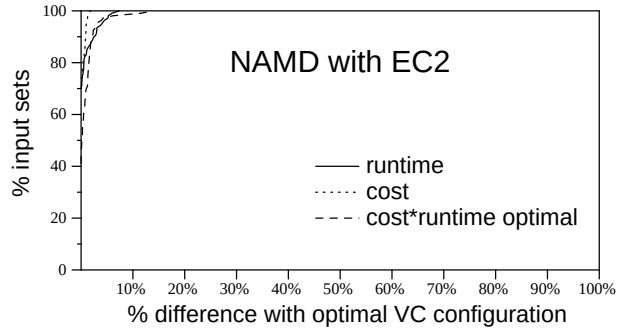
Figure 12 presents the CDFs of the percentage of difference for all possible inputs, for runtime, cost, and cost-runtime minimization goals. The percentage of difference is a ratio of goal difference between the optimal configuration and the one recommended by our system, to that between the optimal one and the worst one. The goal is one of runtime, cost, and cost-runtime product. If the percentage of difference is 0%, then our system recommends the best configuration, but if the percentage of difference is 100%, then the system recommends the worst one to users.

As shown in the Figure 12, the guidance framework can predict the best configuration for the three goals very effectively both with NAMD and LAMMPS. For more than 90% of the test inputs, the runtime and cost-runtime-product with the recommended configuration are within 5% difference with the optimal runtime and cost-runtime-product for both NAMD and LAMMPS. For 99.31% of the test inputs, the cost with the recommended VC configuration is optimal for NAMD, but the accuracy for the cost goal is slightly lower with LAMMPS. Without extra cost, runtime, or parallel efficiency constraints, the framework performs very effective for the three different minimization goals.

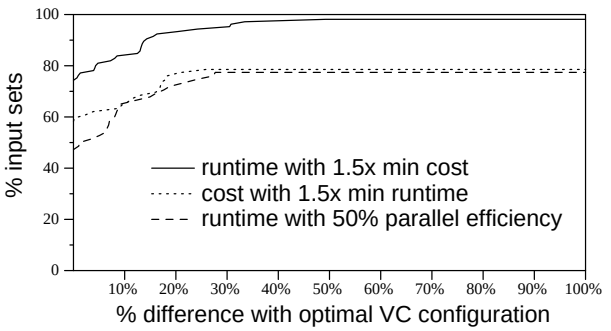
Constraints	% violated configurations	% geomean violation
Runtime with 1.5x min cost in NAMD	4.62%	4.15%
Cost with 1.5x min runtime in NAMD	10.87%	3.29%
Runtime with 50% parallel efficiency in NAMD	23.37%	10.97%
Runtime with 1.5x min cost in LAMMPS	8.50%	4.69%
Cost with 1.5x min runtime in LAMMPS	11.08%	15.70%
Runtime with 50% parallel efficiency in LAMMPS	3.35%	12.24%

TABLE 7
Violated configuration statistics in three constraints

Next, we present how effectively the guidance system can find the best VC configuration for the goals with constraints. When a constraint is given with a goal, the system mostly recommends a configuration that satisfies the constraint, but a subset of recommended configurations may violate the constraints. We use the three constraints as described in Section 3.2. The goals with the constraints are to minimize runtime with 1.5x minimum cost, cost with 1.5x minimum runtime, and runtime with at least 50% parallel efficiency.



(a) % difference to optimal VC configurations for three goals in NAMD



(b) % difference to optimal VC configurations with constraints in NAMD

Fig. 14. Cumulative distribution to optimal VC configurations on Amazon EC2

First, Table 7 shows how often the framework cannot meet the constraints. For each constraint condition, the percentage of recommended configurations violating constraints is given in the second column for the total 432 and 388 test inputs of NAMD and LAMMPS, respectively. The violation ratio is 12.95% and 7.64% on average for NAMD and LAMMPS, respectively. Furthermore, even for the violated cases, the recommended configuration is very close to the optimal one. The configurations with 3-16% error (on geometric mean) are recommended for the three goals with constraints.

Figure 13 presents the CDFs of the percentage of difference with three different constraints, runtime minimization with 1.5x cost limit, cost minimization with 1.5x runtime limit, and runtime minimization with 50% parallel efficiency limit. Note that the CDFs do not converge to 100% of the input sets, since for the remaining inputs, the guidance system could not meet the constraints as discussed above. As expected, the accuracy of the guidance framework becomes lower with given constraints, compared to the simple runtime, cost, cost-runtime minimization scenarios. However, the guidance framework can still recommend reasonable configurations with the constraints satisfied or with relatively minor violation of the constraints. For example, for NAMD with the runtime with 1.5x min cost constraint, about 80% of cases exhibit less than 17% difference from the optimal one, although the other two constraints show slightly lower accuracy. LAMMPS shows better accuracy except for the runtime with 1.5x min cost constraint.

In summary, the guidance framework can recommend very accurate VC configurations, if the goal is a simple cost, runtime, or cost-runtime minimization. With complex constraints, the accuracy is lower than the simple scenarios. However, considering that currently available systems based on virtualized clouds do not provide any cues for expected performance or whether the constraints will be satisfied, the guidance framework improves the quality of cloud services for this type of scientific computing. With increasing numbers of cores in a physical system, the diversity of possible virtual machines will increase in cloud computing. With the increasing diversity, a recommendation system for the best VC configuration will become more important.

5.3 Results in Amazon EC2 clusters

To evaluate the guidance framework on a real public cloud service, the NAMD application was tested on the Amazon EC2 service. Since direct accesses to physical machines are not available to users, this EC2-based setup does not support the interference model. As discussed in Section 4, to fully support the interference model, the framework should be part of the cloud service so that the current interference states of physical machines become available. This EC2 evaluation shows how effectively the framework estimates the performance and cost, even if the interference model is not added in a public cloud service.

Instances	total # of VMs in virtual cluster					
c4.large	8	16	32	64	80	160
c4.xlarge	4	8	16	32	40	80
c4.2xlarge	2	4	8	16	20	40
c4.4xlarge	1	2	4	8	10	20

TABLE 8
Virtual cluster configurations used in Amazon EC2

	The number of vCPUs
c4.large	2
c4.xlarge	4
c4.2xlarge	8
c4.4xlarge	16

TABLE 9
The number of vCPUs in Amazon EC2 c4 type instances

Table 8 shows the cluster configurations for the EC2 setup. We evaluate four types of instances, and for each instance type, 6 different numbers of VMs are used, examining 24 virtual cluster configurations. The number of vCPUs for each instance type is shown in Table 9. We use the same input parameters as the evaluation using our private cloud system, with total 108 test inputs with NAMD.

Figure 14 presents the CDFs of the percentage of difference for six different goals (without and with constraints) on Amazon EC2. For simple goals of runtime, cost, and cost-runtime minimization, the accuracy of our prediction is very high. For the other more complicated goals with constraints, the accuracy

Constraints	% violated configurations	% geomean violation
Runtime with 1.5x min cost in NAMD	1.87%	11.25%
Cost with 1.5x min runtime in NAMD	22.42%	12.23%
Runtime with 50% parallel efficiency in NAMD	25.23%	12.06%

TABLE 10
Violated configuration statistics in three constraints on Amazon EC2

of prediction slightly decreases, compared to the result with our private cloud evaluation. Table 10 presents the percentage of recommended configurations violating constraints and the percentage of geometric mean error for the violated cases. For runtime with 1.5x minimum cost, the percentage of violation is less than 2%, showing a very high accuracy. However, for cost with 1.5x minimum runtime, the violation percentage is increased to 22.42% from 10.87% in our private cloud evaluation. For runtime with 50% parallel efficiency, the violation percentage is similar to the private cloud evaluation. However, even for the violated cases, the configurations with 11-12% error (on geometric mean) are recommended by the guidance framework.

Compared to the NAMD results on our private cloud system, the results on Amazon EC2 are generally similar, although the percentage of violation for the goal of cost with 1.5x minimum runtime increases. The most critical difference of the EC2 evaluation is the lack of interference model. However, even without the interference model, the guidance framework can provide similar accuracy compared to the private cloud evaluation, except for one goal with modestly worse accuracy.

6 Prior Work

There have been earlier efforts on analyzing the performance of running applications in cloud computing. In [29], Li et al. systematically compared the performance and cost of four different cloud providers to help users find the best cloud provider for their need. Their results show the difficulty of selecting one from various cloud providers with different features and pricing models. In [11], the performance and cost of running a data intensive astronomy application are analyzed using Amazon EC2. The application performance and cost are significantly affected by how to execute the application using computing and storage resources differently. Wang et al. discussed the meaning of pricing for cloud users and providers respectively, using single machine benchmarks [43]. In [12], an automated scaling framework was proposed for enterprise applications in clouds. The framework dynamically allocates resources by adding more resources to VM instances or more VM instances to optimize the total cost of the applications. In [39], Redekopp et al discussed that Bulk Synchronous Parallel (BSP) graph processing can be done on clouds at a reduced cost by exploiting cloud elasticity. Unlike the above work, we show the difficulties of finding the best configuration (i.e. the virtual machine types and counts) of a virtual cluster for scientific parallel applications, whose configurations cannot be changed during execution in general, and then propose a guidance system to find it. We also address the effects of user constraints over runtime, cost, and parallel efficiency.

In the Amazon EC2 pricing model, it provides spot instances, in which users bid the maximum price to pay for unused Amazon EC2 resources, as well as on-demand instances, in which users pay fixed prices for VM instances. Our work only considers to use on-demand instances, but spot instances can be used. In [6], a pricing model for spot instances is analyzed, and in [28], optimal solutions to maximize cloud provider's profit from price auctions are studied. We can extend our guidance system to utilize spot instances, resulting in reducing the cost to run applications on cloud systems.

Finding optimal resource provisioning on clouds has been studied for MapReduce applications. In [44], linear programming is used to choose which cloud service to use, and how to run MapReduce computations on the chosen cloud for minimizing cost or completion time. In their work, several factors, not only VM configurations, but also data transfer and storage operations, are taken into account to optimize for MapReduce computations. In [16], Herodotou et al. proposed a system to find a good cluster configuration in clouds, including MapReduce configuration parameters such as the number of map and reduce tasks for a given MapReduce application. Since MapReduce configuration parameters have significant impacts on the performance of MapReduce applications, the authors in [15] and [21] focused on finding the optimal MapReduce parameters. In addition, techniques to find intelligent deployment of matrix-based big-data analysis programs, in terms of cost and completion time, for clouds was proposed [18]. The above work shares the same general goal as our work on helping cloud users, usually scientists and non-experts, utilize cloud resources efficiently. However, we focused on traditional scientific parallel workloads such as molecular dynamics simulators, which have different characteristics from those of MapReduce (or big data analysis) workloads. MapReduce workloads are based on a simple programming model of map and reduce with a plain communication pattern, compared to scientific parallel workloads. Moreover, the scientific workloads are resource-intensive, and so are highly susceptible to interference with co-runner VMs on the same host. Thus, we take into account the interference effects in our performance estimation models, unlike the above work. Also, our proposed system can find the optimal or near-optimal configuration under various criteria of user constraints, considering runtime, cost, parallel efficiency, and combinations of them, which are common considerations of scientists on running scientific parallel applications.

Scheduling and resource provisioning techniques for scientific workflows [31], data processing workflows [24], and large graph processing tasks [30] have been investigated for clouds. These techniques consider not only deadline (or makespan) but also cost (or budget) on scheduling tasks of workflows. For hybrid clouds that are composed of public and private clouds, the effects of the bandwidth of communication channels between public and private clouds on the performance of workflows (with respect to deadline and cost) were analyzed in [8], and scheduling heuristics that consider data transfer costs and times were presented in [42].

Artificial neural networks (ANNs) have been used to predict the performance of applications on non-virtualized and virtualized platforms [40], [27]. In [40], ANN models are built for scientific parallel applications, SMG 2000 and HPL, which have large parameter spaces, on two large scale platforms. In [27], the authors observed that regression models do not work well to model application performance in virtualized systems, and thus they used ANNs instead. A single dual core machine was used for their experiments. Their results both in [40], [27] showed that ANN models predict the application performance with high accuracy. In our work, we use ANNs to predict the performance of applications on various virtual clusters in clouds, which have different VM types and counts, as well as their memory requirements.

Yang et al. proposed an observation-based performance prediction technique, which translates performance across platforms based on relative performance between two platforms [46]. However, their technique has a limitation to predict the performance of an application accurately when different number of processors and input parameters are used for the

application. In [23], a predictive analytical model for performance and scalability was developed for SAGE. Their model is based on the analysis of the code, inspection of data structures, and analysis of runtime traces for SAGE, and thus it is specific to SAGE application. A semi-automatic toolkit that utilizes the static and dynamic analysis of application binaries was introduced for analyzing the behaviors of single node applications and modeling the performance [32]. In their work, a model to predict the performance for different problem sizes was built, but the performance of parallel applications was not the focus.

7 Conclusions

In this work, we proposed a framework to guide cloud users to find the best virtual cluster configurations for their applications in virtualized clusters. We showed the difficulties to find the best configuration of a virtual cluster of VMs for scientific parallel applications, and developed a guidance system to find the optimal VM types and VM counts for given input parameters of the applications and status due to interference among VMs. Using ANNs which predict execution times and memory requirements, our system can support flexible constraints, with the post-ANN processing to search the best VC configuration satisfying any given constraint. Our results showed that our guidance system can find an optimal or near-optimal configuration under various constraints, for cost, runtime, parallel efficiency, and combinations of them. As cloud computing becomes more popular, and the number of available cores increases on multi-core architecture, the variety of VM types offered by cloud providers will increase further. Proper guidance on selecting a VC configuration will become necessary.

Acknowledgments

This work was partly supported by the National Research Foundation of Korea (NRF) funded by the Ministry of Science, ICT and Future Planning (No. NRF-2012R1A1A1043329 and No. NRF-2013R1A2A2A01015514), and also partly supported by the year of 2012 Research Fund of the UNIST (Ulsan National Institute of Science and Technology).

References

[1] Amazon EC2. http://aws.amazon.com/ec2/.

[2] LAMMPS. http://lammps.sandia.gov/.

[3] Microsoft Windows Azure. http://www.microsoft.com/windowsazure/.

[4] NAMD. http://www.ks.uiuc.edu/Research/namd/.

[5] Rackspace cloud servers. http://www.rackspace.com/cloud/.

[6] O. Agmon Ben-Yehuda, M. Ben-Yehuda, A. Schuster, and D. Tsafrir. Deconstructing amazon ec2 spot instance pricing. In *Cloud Computing Technology and Science (CloudCom), 2011 IEEE Third International Conference on*, pages 304–311. IEEE, 2011.

[7] J. Ahn, C. Kim, J. Han, Y.-R. Choi, and J. Huh. Dynamic virtual machine scheduling in clouds for architectural shared resources. In *Proceedings of the 4th USENIX conference on Hot Topics in Cloud Ccomputing*, HotCloud'12. USENIX Association, 2012.

[8] L. F. Bittencourt, E. R. Madeira, and N. L. Da Fonseca. Scheduling in hybrid clouds. *Communications Magazine, IEEE*, 50(9):42–47, 2012.

[9] S. Blagodurov, S. Zhuravlev, and A. Fedorova. Contention-aware scheduling on multicore systems. *ACM Transactions on Computer Systems (TOCS)*, 28(4):8, 2010.

[10] T. Darden, D. York, and L. Pedersen. Particle mesh Ewald: An N log (N) method for Ewald sums in large systems. *The Journal of chemical physics*, 98(12):10089–10092, 1993.

[11] E. Deelman, G. Singh, M. Livny, B. Berriman, and J. Good. The cost of doing science on the cloud: the montage example. In *Proceedings of the 2008 ACM/IEEE conference on Supercomputing*, SC '08, pages 50:1–50:12, Piscataway, NJ, USA, 2008. IEEE Press.

[12] S. Dutta, S. Gera, A. Verma, and B. Viswanathan. Smartscale: Automatic application scaling in enterprise clouds. In *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*, pages 221–228, June 2012.

[13] A. Gulati, A. Merchant, and P. J. Varman. mclock: handling throughput variability for hypervisor io scheduling. In *Proceedings of the 9th USENIX conference on Operating systems design and implementation*, OSDI'10, pages 1–7. USENIX Association, 2010.

[14] D. Gupta, L. Cherkasova, R. Gardner, and A. Vahdat. Enforcing performance isolation across virtual machines in Xen. In *Proceedings of the ACM/IFIP/USENIX 2006 International Conference on Middleware*, Middleware '06, pages 342–362. Springer-Verlag New York, Inc., 2006.

[15] H. Herodotou and S. Babu. Profiling, what-if analysis, and cost-based optimization of mapreduce programs. In *Proceedings of the 37th international conference on Very Large Data Bases*, PVLDB'11. VLDB Endowment, 2011.

[16] H. Herodotou, F. Dong, and S. Babu. No one (cluster) size fits all: automatic cluster sizing for data-intensive analytics. In *Proceedings of the 2nd ACM Symposium on Cloud Computing*, page 18. ACM, 2011.

[17] R. W. Hockney and J. W. Eastwood. *Computer simulation using particles*. CRC Press, 1988.

[18] B. Huang, S. Babu, and J. Yang. Cumulon: Optimizing statistical data analysis in the cloud. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, pages 1–12. ACM, 2013.

[19] J. A. Izaguirre, S. Reich, and R. D. Skeel. Longer time steps for molecular dynamics. *The Journal of chemical physics*, 110(20):9853–9864, 1999.

[20] J. C. Jo and B. C. Kim. Determination of proper time step for molecular dynamics simulation. *Bulletin of the Korean Chemical Society*, 21(4):419–424, 2000.

[21] K. Kambatla, A. Pathak, and H. Pucha. Towards optimizing hadoop provisioning in the cloud. In *Proceedings of the 2009 conference on Hot topics in cloud computing*, HotCloud'09. USENIX Association, 2009.

[22] K. Keahey and M. Parashar. Enabling on-demand science via cloud computing. *Cloud Computing, IEEE*, 1(1):21–27, May 2014.

[23] D. J. Kerbyson, A. H. J., A. Hoisie, F. Petrini, H. J. Wasserman, and M. Gittings. Predictive performance and scalability modeling of a large-scale application. In *Proceedings of the 2001 ACM/IEEE conference on Supercomputing (SC)*, Supercomputing '01, pages 37–37, 2001.

[24] H. Kllapi, E. Sitaridi, M. M. Tsangaris, and Y. Ioannidis. Schedule optimization for data processing flows on the cloud. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*, pages 289–300. ACM, 2011.

[25] R. Kohavi et al. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *Ijcai*, volume 14, pages 1137–1145, 1995.

[26] R. Koller, A. Verma, and R. Rangaswami. Generalized ERSS tree model: Revisiting working sets. *Performance Evaluation*, 67(11):1139–1154, 2010.

[27] S. Kundu, R. Rangaswami, K. Dutta, and M. Zhao. Application performance modeling in a virtualized environment. In *High Performance Computer Architecture (HPCA), 2010 IEEE 16th International Symposium on*, pages 1 –10, 2010.

[28] U. Lampe, M. Siebenhaar, A. Papageorgiou, D. Schuller, and R. Steinmetz. Maximizing cloud provider profit from equilibrium price auctions. In *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*, pages 83–90. IEEE, 2012.

[29] A. Li, X. Yang, S. Kandula, and M. Zhang. Cloudcmp: comparing public cloud providers. In *Proceedings of the 10th annual conference on Internet measurement*, IMC '10, pages 1–14, 2010.

[30] J. Li, S. Su, X. Cheng, Q. Huang, and Z. Zhang. Cost-conscious scheduling for large graph processing in the cloud. In *High Performance Computing and Communications (HPCC), 2011 IEEE 13th International Conference on*, pages 808–813. IEEE, 2011.

[31] M. Malawski, G. Juve, E. Deelman, and J. Nabrzyski. Cost-and deadline-constrained provisioning for scientific workflow ensembles in iaas clouds. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, page 22. IEEE Computer Society Press, 2012.

[32] G. Marin and J. Mellor-Crummey. Cross-architecture performance predictions for scientific applications using parameterized models. In *Proceedings of the joint international conference on Measurement and modeling of computer systems*, SIGMETRICS '04, pages 2–13, 2004.

[33] R. Nathuji, A. Kansal, and A. Ghaffarkhah. Q-clouds: managing performance interference effects for QoS-aware clouds. In *Proceedings of the 5th European conference on Computer systems*, EuroSys '10, pages 237–250, New York, NY, USA, 2010. ACM.

[34] J. Park and I. W. Sandberg. Universal approximation using radial-basis-function networks. *Neural computation*, 3(2):246–257, 1991.

[35] J. C. Phillips, R. Braun, W. Wang, J. Gumbart, E. Tajkhorshid, E. Villa, C. Chipot, R. D. Skeel, L. Kale, and K. Schulten. Scalable molecular dynamics with NAMD. *Journal of computational chemistry*, 26(16):1781–1802, 2005.

[36] S. Plimpton. Fast parallel algorithms for short-range molecular dynamics. *Journal of computational physics*, 117(1):1–19, 1995.

[37] S. Plimpton, R. Pollock, and M. Stevens. Particle-Mesh Ewald and rRESPA for Parallel Molecular Dynamics Simulations. In *Proceedings of the Eighth SIAM Conference on Parallel Processing for Scientific Computing*, 1997.

[38] J. Rao, K. Wang, X. Zhou, and C.-Z. Xu. Optimizing Virtual Machine Scheduling in NUMA Multicore Systems. In *HPCA '13: Proceedings of the 19th IEEE International Symposium on High Performance Computer Architecture*, 2013.

[39] M. Redekopp, Y. Simmhan, and V. K. Prasanna. Optimizations and analysis of bsp graph processing models on public clouds. In *Parallel & Distributed Processing (IPDPS), 2013 IEEE 27th International Symposium on*, pages 203–214. IEEE, 2013.

[40] K. Singh, E. İpek, S. A. McKee, B. R. de Supinski, M. Schulz, and R. Caruana. Predicting parallel application performance via machine learning approaches. *Concurrency and Computation: Practice and Experience*, 19(17):2219–2235, 2007.

[41] M. T. *Machine Learning*. WCB/McGraw-Hill, New York, New York, 1997.

[42] R. Van den Bossche, K. Vanmechelen, and J. Broeckhove. Cost-efficient scheduling heuristics for deadline constrained workloads on hybrid clouds. In *Cloud Computing Technology and Science (CloudCom), 2011 IEEE Third International Conference on*, pages 320–327. IEEE, 2011.

[43] H. Wang, Q. Jing, R. Chen, B. He, Z. Qian, and L. Zhou. Distributed systems meet economics: pricing in the cloud. In *Proceedings of the 2nd USENIX conference on Hot topics in cloud computing*, HotCloud'10, pages 6–6, Berkeley, CA, USA, 2010. USENIX Association.

[44] A. Wieder, P. Bhatotia, A. Post, and R. Rodrigues. Orchestrating the deployment of computations in the cloud with conductor. In *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*, pages 27–27. USENIX Association, 2012.

[45] A. K. Wong and A. M. Goscinski. A unified framework for the deployment, exposure and access of HPC applications as services in clouds. *Future Generation Computer Systems*, 29(6):1333–1344, 2013.

[46] L. T. Yang, X. Ma, and F. Mueller. Cross-platform performance prediction of parallel applications using partial execution. In *Proceedings of the 2005 ACM/IEEE conference on Supercomputing*, SC '05, pages 40–, Washington, DC, USA, 2005. IEEE Computer Society.

[47] S. Zhuravlev, S. Blagodurov, and A. Fedorova. Addressing shared resource contention in multicore processors via scheduling. In *ACM SIGARCH Computer Architecture News*, volume 38, pages 129–142. ACM, 2010.

Jaeung Han received the B.S. and M.S. degrees in Computer Science from Korea Advanced Institude of Science and Technology (KAIST) in 2009 and 2011, respectively. Currently, he is a Ph.D. candidate in Computer Science at KAIST. His research interests include parallel computing, cloud computing, and virtualization.

Changdae Kim received the BS degree in computer science from KAIST, Korea, in 2010, and the MS degree in computer science from KAIST, Korea, in 2012. He is currently a PhD candidate in computer science at KAIST. His research interests focus on emerging computer architecture, operating systems, and cloud computing.

Jaehyuk Huh is an Associate Professor of Computer Science at Korea Advanced Institute of Science and Technology (KAIST). His research interests are in computer architecture, parallel computing, virtualization and system security. He received a BS in computer science from Seoul National University, and an MS and a PhD in computer science from the University of Texas at Austin.

Giljin Jang is an Assistant Professor at Kyungpook National University, South Korea. He received his B.S. and M.S. degree in computer science from the Korea Advanced Institute of Science and Technology (KAIST), Daejon, South Korea in 1997 and 1999 respectively. He also received his Ph.D. degree in the same department in February 2004. Dr. Jang's research interests include machine learning theory, acoustic signal processing, pattern recognition, speech recognition and enhancement, multimedia data analysis, and biomedical signal engineering.

Young-ri Choi received the BS degree in computer science from Yonsei University, Seoul, Korea, in 1998. She received the MS and PhD degrees in computer sciences from the University of Texas at Austin in 2002 and 2007, respectively. Her research interests include cloud computing, virtualization, scientific computing, and network protocols. She is currently an Assistant Professor in the School of Electrical and Computer Engineering at UNIST.