Online Optimization of VM Deployment in IaaS Cloud

Pei Fan, Zhenbang Chen, Ji Wang
School of Computer Science
National University of Defense Technology
Changsha, 410073, P.R.China
{peifan,zbchen}@nudt.edu.cn, jiwang@ios.ac.cn

Zibin Zheng
Shenzhen Research Institute
The Chinese University of Hong Kong
Hong Kong, P.R.China
zbzheng@cse.cuhk.edu.hk

***Abstract*—Infrastructure-as-a-Service (IaaS) clouds provide on-demand virtual machines (VMs) to users. How to improve the quality of IaaS cloud services is important for service providers. Currently, the VMs in an IaaS cloud are usually deployed with respect to the maximum utilization of resources. In this paper, we propose an online VM optimization method for IaaS clouds. Our method mainly optimizes the VM deployment in IaaS clouds according to the traffics among VMs. VMs are allocated with respect to cabinet capacities at the beginning. At runtime, we monitor the traffics among VMs to get the traffic topology, based on which related VMs are migrated to neighbors to improve performance and reduce the traffics across cabinets. Preliminary simulation experiments are conducted on a well-know simulator, and the experimental results indicate that our method is effective and promising.**

***Keywords*-IaaS cloud; deployment; online optimization; traffic topology; virtual machine;**

I. Introduction

Nowadays, cloud computing [1] becomes the mainstream internet computing paradigm, which provides a pay-as-you-go style service to cloud users. Basically, there are three types of services in cloud computing [2]: IaaS, PaaS and SaaS. IaaS is the abbreviation of Infrastructure-as-a-Service, where resources are usually provided to users in the form of Virtual Machines (VMs). In an IaaS cloud, users can apply VMs on-demand to deploy and run their applications and provide services to their clients. From the perspective of users, this way of applying and using resources can not only save the cost of providing services, but also improve the reliability. In this background, how to improve the quality of IaaS services is very important for service providers.

By using IaaS services, more and more users are migrating their applications and services from traditional infrastructures to IaaS clouds [3]. With this trend, the bandwidth usage of the VMs in IaaS clouds is rapidly growing. However, the hierarchical nature of a data center supporting an IaaS service places a limit on the backbone bandwidth, which is shared by all the VMs in the IaaS cloud. During the VM deployments in an IaaS cloud, the service provider takes resources utilization as the first factor, but usually ignoring the traffics among VMs. Therefore, the backbone bandwidth may become the bottleneck when the number of the allocated VMs is huge. This will increase the time of the data transfer of VMs caused by the applications running on the VMs. Thus, the performance of the applications may degrade.

In order to improve the quality of IaaS cloud services, cloud providers usually use static schedule solutions. For example, ranking is usually used for computing-intensive applications [4]. A number of literatures, such as [5] and [6], optimize VM deployment by considering physical node resource constraints, *e.g.*, CPU, physical memory and power consumption, without considering the bandwidth of VMs. Some clustering based methods [7][8] are proposed to optimally deploy communication-intensive applications, where the communication performance of physical nodes is used to select the nodes for deployment. Furthermore, an optimal deployment method using the topology information of deployed applications is proposed in [9], in which it needs an application specific method to get the topology information. In an IaaS cloud, the applications running on VMs vary in many aspects, including implementation language, runtime platform, *etc*. For the IaaS provider, it is not practical or even possible to provide an application specific topology information collector for each type of applications.

In this paper, we focus on traffic-aware VM deployment in IaaS clouds and propose novel solutions to address all above issues. Our VM deployment method has two stages: build-time and runtime. In the build-time stage, we use a ranking-based method to sort all cabinets in the decreasing order with respect to their current VM capacities. Based on the order, we deploy VMs into top cabinets. At the runtime stage, we monitor the traffics among VMs dynamically, and use the multi-scale algorithm in our previous work [9] to obtain traffic topology automatically. Different from previous work, we obtain traffic topology from a unified level, *i.e.*, VM level, without requiring application specific topology collectors or users to provide the traffic information. Based on traffic topology, the VMs that have a lot of traffics will be placed in a neighborhood (*e.g.*, VMs migrated to a same cabinet) by live migration. Therefore, the performance of applications can be guaranteed to improve user experience, and the traffics across cabinets can be reduced.

The main contributions of this paper are three-folds: first, we propose the problem of traffic-aware VM placement in the context of IaaS cloud; second, we present a two-stage traffic-aware deployment method to optimize the VM deployment in IaaS clouds; third, we have carried out preliminary experiments on a simulation environment, and the experimental results indicate that our method is promising.

The rest of this paper is organized as follows: Section II introduces a motivation example and the architecture of our framework; Section III presents our VM deployment method; Section IV describes experiments; Section V discusses the related work and Section VI concludes the paper.

II. Motivation and Architecture

A. Motivation Example

Since more and more scientific applications are moving to IaaS clouds, we use an MPI application to motivate our method. Usually, an MPI application will be run multi-times and has a communication topology [9]. In MPI applications, more than 40% time is spent in collective communication operators [10].

Suppose a user wants to apply 8 nodes, *i.e.*, VMs, from a IaaS cloud to run an MPI application, which uses MPI_Scatter (a collective communication operator in MPI) to distribute a huge size of data. Figure 1 shows the detailed communication steps. At the beginning, node 0 contains all the data sets, labeled by $0 \sim 7$. The application needs to distribute each data set to the corresponding node, *e.g.*, data set 1 to node 1. In the first step, node 0 transfers half of the data sets ($4 \sim 7$) to node 4. In the later steps, each node transfers half of its data to a related node, *e.g.*, node 4 transfers the data sets 6 and 7 to node 6. This process continues until the data distribution is finished.

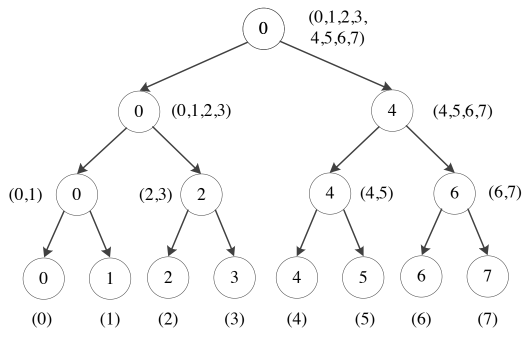


Figure 1: Communication steps of MPI_Scatter

This example shows that there is a traffic topology in this application, *e.g.*, the nodes $0 \sim 3$ communicate a lot. Therefore, the locations of VMs greatly impact the communication performance of the application. For example, if the nodes $0 \sim 3$ are deployed in different cabinets or datacenters, the MPI_Scatter operation may spend more time to transfer data if there is a network congestion, since the bandwidth among cabinets is shared by all the VMs in the IaaS cloud. In principle, the bandwidth between two VMs across cabinets is less than that in a same cabinet. Therefore, if we can detect the communication information of VMs, we can place the VMs that have a lot of traffics to a same cabinet, *e.g.*, placing the $0 \sim 3$ nodes in Figure 1 into a same cabinet.

However, it is hard to get the traffic topology of the MPI application when allocating VMs at the beginning. It is not practical or even impossible to require the user to provide this information. Though there is an approach [9] that uses pre-execution to get this information, *e.g.*, we can use MPI slog2sdk to record the message exchanges during the pre-execution, this method is MPI specific and not general. If the user runs another application, such as a portal system to provide a content service, the pre-execution needs to be adapted. Therefore, if we can monitor the traffics among VMs, we can have a general method, which is also reasonable from the perspective of IaaS service providers. For example, in Figure 1, if the start-up VM deployment of the application places the nodes $0 \sim 3$ into different cabinets, by monitoring the traffics among these VMs, we can optimize the deployment via migrating them into a same cabinet. Furthermore, the whole optimization procedure is transparent to the user.

B. Architecture

Figure 2 shows the architecture of our proposed VM deployment framework. The workflows of our framework are as follows:

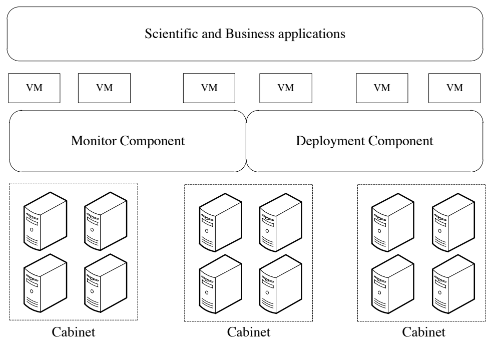


Figure 2: VM deployment framework

- A cloud user submits a request for applying VMs to run an application on the cloud. In the build-time stage, the deployment component uses the Random or Build-time method to select VMs. The detail of Build-time method will be introduced in Section III-A.
- The monitor component takes charge of monitoring the traffic among VMs. Based on the traffic information, in the runtime stage, the deployment component migrates VMs using online optimization, the details of which will be given in Section III-B.
- Some hosts may be in an idle status after migration. In order to reduce the power consumption, the deployment

component switches the hosts off or to low power modes.

III. Online Optimization Method

There are two stages of our VM deployment method: first, at the build-time stage, we rank cabinets with respect to capacity, and allocate VMs based on the cabinet capacities; second, at runtime, we monitor the traffics among VMs to obtain the traffic topology, and then use live migration to do optimization.

A. Build time Stage

Generally, the VM deployment problem is a variant of the class constrained multiple-knapsack problem that is known to be NP-hard [11].

In the build time stage, we use a ranking-based method to allocate VMs based on the cabinet capacity order. This ranking method has two steps: first, sort all cabinets in the decreasing order of their remained VM capacities; second, select the cabinet with the largest VMs capacity, and allocate the VMs to this cabinet until the maximum capacity is reached, then select the next cabinet. The ranking algorithm is presented in Algorithm 1.

Algorithm 1: Build time stage algorithm

```
Input: Cabinet list $cabinetList$, VMs list $vmList$
sortDecreasingCapacity($cabinetList$);
foreach $cabinet \in cabinetList$ do
    foreach $vm \in vmList$ do
        if $cabinet$ has enough resources for $vm$ then
            $cabinet$.addtovmList($vm$)
        end
        $vmList$.delete($vm$);
    end
    if $vmList$ is empty then Break
end
```

B. Runtime Stage

At runtime, we monitor the traffics among VMs, and use the multi-scale algorithm in our previous work [9] to automatically obtain the traffic topology, based on which VMs are migrated.

As introduced in Section II-B, we use the monitor component to monitor the traffic among VMs. The result of monitoring can be modeled by an undirected weight graph, and it is assumed that two adjacent nodes in the graph have data exchanged. We use a simple example of monitor result in Figure 3 to describe the runtime stage algorithm. In Figure 3, we use 4 VMs to run an application, and the nodes and the edges represent the VMs and the traffic among VMs, respectively. The weights on the edges are the amount of data exchanged among VMs.

From Figure 3, we can observe that there are a lot of traffic in the VM pairs (1, 2) and (3, 4), and less traffic in (2, 3). Discovering the traffic topology is a key step of the runtime stage. In this paper, we formulate the traffic topology discovery problem as a graph partitioning problem: we want to find the structure of adjacency, in which nodes are joined together in a tightly knit structure, which means that the nodes within a same structure have more traffic with each other. And, there is less traffic among structures. Though the problem of graph partitioning is well studied, the algorithms for graph partitioning, such as k-means and spectral clustering [7], are not ideally suited to our framework. The reason is these algorithms require users to specify the sizes of clusters. However, this assumption is not practical for cloud computing, since a cloud user or provider may not be the developer of the applications to be deployed.

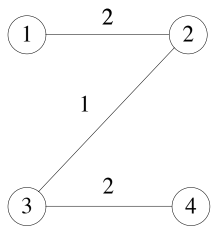


Figure 3: An example of monitor result

In our previous work [9], we use a multi-scale clustering algorithm to discover the topology of an undirected weight graph. The multi-scale clustering algorithm does not assume any particular number of clusters. However, the multi-scale clustering algorithm is proposed for static schedule, and not suitable for the scenario in this paper. Figure 4 demonstrates the problem. In this example, there are 4 VMs, and the numbers on the lines show the traffics among VMs. In the build-time stage, suppose we deploy the VMs $1 \sim 3$ in a same cabinet, and the 4th VM in another. Using the multi-scale clustering algorithm in [9], we have the traffic topology with two groups: (1, 2) and (3, 4). Therefore, at runtime, the 3rd VM is migrated to the cabinet at where the 4th VM is located. However, this will result in a poor performance. After migration, the traffic value between two cabinets is 3, but the before value is 2. The performance will be decreased since more traffic among cabinets will increase the time for transferring data. To tackle this problem, we modify the multi-scale clustering algorithm by adding a cost function for a VM placement. Based on the cost function, whether to migrate VMs can be determined.

The cost function of a VM placement is the total traffic that is needed for crossing the cabinets, and is defined as

$$C = \sum \omega(v_i, v_j) \tag{1}$$

where v_i and v_j are the two VMs that deployed in different cabinets. $\omega(v_i, v_j)$ is the traffic value between v_i and v_j. A

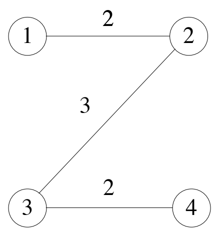


Figure 4: Another example of monitor result

VM can be migrated to another cabinet only if:

$$C_{after} < C_{before} \tag{2}$$

where C_{after} is the total traffic among the cabinets after migration, and C_{before} is the value before migration.

Algorithm 2: The runtime stage algorithm

```
  Input: Cabinet list $cabinetList$, VM partition
         $vmCluster$, Host list $hostList$
1 $vmCluster$=discoverTopology();
2 if $C_{after} < C_{before}$ then
3     foreach $v \in vmCluster$ do
4         foreach $vm \in v$ do
5             $vm$.mirgate();
6         end
7     end
8 end
```

Based on the cost function, our runtime stage optimization can ensure the decrease of the total traffic among cabinets. Algorithm 2 shows the runtime stage algorithm.

- Step 1 (1st line): using the multi-scale algorithm to discover the traffic topology.
- Step 2 (2nd line): calculate the total traffic among cabinets before and after migrate via (1), and migrate VMs based on the traffic topology only if $C_{after} < C_{before}$.
- Step 3 (3rd-8th lines): do the migration, where v is the set of the VMs that in a same cluster, and the VMs in v are migrated into a same cabinet.

IV. Experiments and Evaluation

In this section, we evaluate our traffic-aware VM deployment method by the experiments carried on a simulation environment. We first describe the setup of our experiments, and then give the evaluation results.

A. Experiment Setup

We carried out our simulation experiments on CloudSim-3.0 [12][13], which is a framework for modeling and simulation of cloud computing infrastructures and services.

Our simulated experimental environment has one datacenter consisting of 6 cabinets, and each cabinet has 4 hosts. The topology and configuration of the infrastructure are given in Figure 5. The hosts insides a cabinet are connected by an 1Gb/s edgeswitch, and cabinets are connected by a 500Mb/s aggswitch. CloudSim can simulate edgeswitches and aggswitches. We use an MPI benchmark called NPB (NAS Parallel Benchmarks) [9] as the application running on VMs. NPB is a widely used MPI Benchmark, which consists of the programs designed to help evaluate the performance of supercomputers. In order to simulate real applications on cloud, we obtained the traces of a NPB program on a small scale real infrastructure, and modeled the workload in CloudSim based on the traces. Two NPB applications are used in the preliminary experiments:

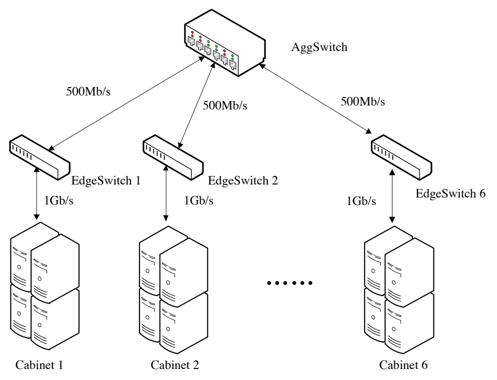


Figure 5: Experimental Datacenter Infrastructure

- App_4: This application is deployed on 4 VMs, and uses a balance binary tree as the traffic topology. The 1st VM communicates with the 2nd VM, and the 3rd VM communicates with the 4th one. At last, the 3rd VM sends data to the 1st VM.
- App_8: This application is deployed on 8 VMs, and the traffic topology is the same with that in Figure 1.

To evaluate our deployment method, we simulate the scenarios of running each application on the VMs deployed by 4 different deployment methods:

Random: the hosts to allocate VMs are selected randomly, and no optimization happens at runtime.

Build-time: optimization happens at build-time, by using the build-time optimization method in Section III-A. During the runtime stage we do not optimize.

Runtime: the VMs are selected randomly at build-time, and the optimization is carried out at runtime by using our runtime optimization method in Section III-B.

Build & Run: optimization happens both in build-time and runtime stages, which shows the overall advantage of our deployment method.

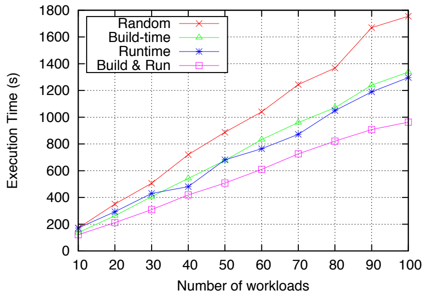
B. Experiment Results and Evaluation

We use the following metrics in our experiments.

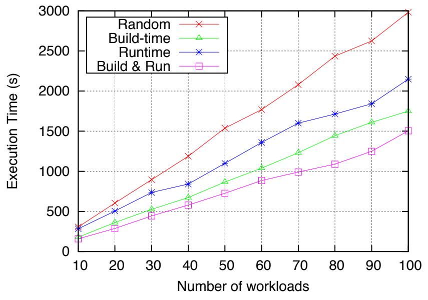
- Execution time: the execution time of a job is defined as the duration between sending out a job and receiving the correct result.

- Data exchanged: the data exchanged is the amount of the data exchanged among cabinets.

In each experiment, we change the number of workloads form 10 to 100 with a step value of 10. We obtain the traffic topology via monitoring the first 5 workloads, and migrate VMs based on the traffic topology. Figures 6a and 6b show the results of running App_4 on 4 VMs and App_8 on 8 VMs, respectively. Each execution time in Figure 6 is the average execution time of running 20 times with a workload.



(a) Results of App_4



(b) Results of App_8

Figure 6: Execution time of different deployment methods

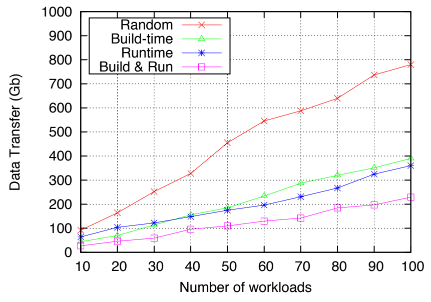
From Figure 6, we can draw following conclusions:

(1) In all cases, Built & Run method performs better than the rest methods.

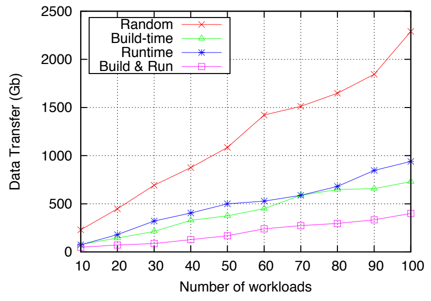
(2) Random method performs worst in all cases. The reason is the VMs are distributed in different cabinets, and much traffic is needed among cabinets.

(3) In Figure 6a, Runtime method is better than Build-time method. However, in Figure 6b, Built-time method is better than Runtime method. The reason is: when an application only needs few VMs, such as App_4, the VMs in a same topology group can be migrated to a same cabinet; however, when an application needs many VMs, such as App_8, a cabinet may not be able to store the VMs in a same topology group, which makes the performance after migration worse.

The bandwidth among cabinets is shared by all VMs. If we can restrict the communications among VMs only among neighbor nodes, *i.e.*, in the same cabinet, we can reduce the time of data transfer and have a better utilization of bandwidth. Figure 7 shows the size of the traffics across cabinets resulted from different methods. We can observe that in all case, Built & Run method has the least traffic among cabinets, and Random method has the largest. The result of the comparison between Build-time and Runtime methods is similar to that of Figure 6.



(a) Results of App_4



(b) Results of App_8

Figure 7: Data exchanged among cabinets

We also analyze the impact of the start time of online migration. In this experiment, we use 100 workloads on 4 VMs, and the start time of the migration range from the 5th workloads to the 85th workloads with a step value of 5. Figure 8 shows the execution time of different start time. We can observe that the execution time of the application is gradually increased, which implies that migrating early achieves a better performance.

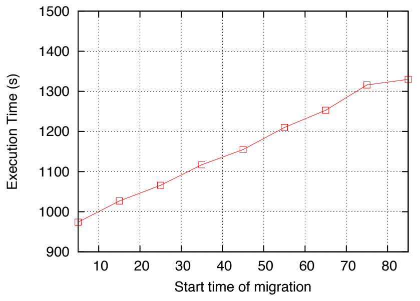


Figure 8: Impact of the start time of migration

V. Related Work

To improve the quality of IaaS services, a number of practices use VM placement to improve the performance. Existing work focuses on using VMs placement methods to optimize SLA requirement or the utilization of resources. Chaisiri *et al.* [14] propose an optimal virtual machine placement algorithm, which minimizes the cost for hosting virtual machines in a multiple cloud provider environment. In [15], a dynamic SLA-aware VM placement algorithm is proposed for cloud computing. Elmroth and Larsson [16] propose the interfaces for placement and migration of the VMs in federated clouds. Different from previous work, our method focuses on the traffic among VMs, and is a hybrid method by combing static and dynamic optimizations. In addition, our method gets traffic information from a unified level, and provides a transparent optimization for IaaS cloud services.

VI. Conclusion and Future Work

To improve the quality of IaaS cloud services, we propose a traffic-aware VM deployment in this paper. Our hybrid deployment method has a built-time optimization to allocate VMs with respect to the resource utilization at the beginning, and a runtime optimization to reallocate VMs by using the traffic information among VMs. According to the experiments carried out on a simulation environment, our method can not only improve the performance of the applications running on VMs, but also the utilization of the bandwidth in IaaS clouds.

The work in this paper is still in progress. There are two directions for the next step: first, we want to consider the cost saving of IaaS clouds in terms of our method, both from user and service provider perspectives; second, we plan to carry out larger experiments in the simulation environment, and also the experiments in a real experimental environment.

Acknowledgment

This research is support by the National Basic Research Program (973) of china under the Grant No. 2011CB302603, the National Natural Science Foundation of China under the Grant No. 61100078, and SRFDP 20114307120015.

References

[1] B. Hayes, “Cloud computing,” *Commun. ACM*, vol. 51, no. 7, pp. 9–11, 2008.

[2] Q. Zhang, L. Cheng, and R. Boutaba, “Cloud computing: state-of-the-art and research challenges,” *Journal of Internet Services and Applications*, vol. 1, no. 1, pp. 7–18, 2010.

[3] J. Li, M. Qiu, Z. Ming, G. Quan, X. Qin, and Z. Gu, “Online optimization for scheduling preemptable tasks on iaas cloud systems,” *Journal of Parallel and Distributed Computing*, vol. 72, no. 5, pp. 666 – 677, 2012.

[4] Z. Zheng, Y. Zhang, and M. R. Lyu, “Cloudrank: A qos-driven component ranking framework for cloud computing,” in *Proc. 29th Int’l Symp. Reliable Distributed Systems (SRDS’10)*, 2010, pp. 184–193.

[5] K. H. Kim, A. Beloglazov, and R. Buyya, “Power-aware provisioning of virtual machines for real-time cloud services,” *Concurrency and Computation: Practice and Experience*, vol. 23, no. 13, pp. 1491–1505, 2011.

[6] A. Beloglazov, J. Abawajy, and R. Buyya, “Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing,” *Future Generation Computer Systems*, vol. 28, no. 5, pp. 755 – 768, 2012.

[7] P. Fan, J. Wang, Z. Chen, Z. Zheng, and M. Lyu, “A spectral clustering–based optimal deployment method for scientific application in cloud computing,” *International Journal of Web and Grid Services*, vol. 8, no. 1, pp. 31–55, 2012.

[8] P. Fan, J. Wang, Z. Zheng, and M. Lyu, “Toward optimal deployment of communication-intensive cloud applications,” in *Proc. 4th Int’l Conf. on Cloud Computing (CLOUD’11)*. IEEE, 2011, pp. 460–467.

[9] P. Fan, Z. Chen, J. Wang, Z. Zheng, and M. Lyu, “Topology-aware deployment of scientific applications in cloud computing,” in *Proc. 5th Int.l Conf on Cloud Computing (CLOUD’12)*. IEEE, 2012, pp. 319–326.

[10] R. Thakur and W. Gropp, “Improving the performance of collective operations in mpich,” in *Recent Advances in Parallel Virtual Machine and Message Passing Interface*, ser. Lecture Notes in Computer Science, J. Dongarra, D. Laforenza, and S. Orlando, Eds. Springer Berlin / Heidelberg, 2003, vol. 2840, pp. 257–267.

[11] H. Shachnai and T. Tamir, “On two class-constrained versions of the multiple knapsack problem,” *Algorithmica*, vol. 29, no. 3, pp. 442–467, 2001.

[12] R. Calheiros, R. Ranjan, A. Beloglazov, C. De Rose, and R. Buyya, “Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms,” *Software: Practice and Experience*, vol. 41, no. 1, pp. 23–50, 2011.

[13] S. Garg and R. Buyya, “Networkcloudsim: Modelling parallel applications in cloud simulations,” in *Proc. 4th Int’l Conf. on Utility and Cloud Computing (UCC’11)*. IEEE, 2011, pp. 105–113.

[14] S. Chaisiri, B. Lee, and D. Niyato, “Optimal virtual machine placement across multiple cloud providers,” in *Proc. 4th Int’l Conf. on Services Computing Conference (APSCC’09)*. IEEE, 2009, pp. 103–110.

[15] H. Van, F. Tran, and J. Menaud, “Sla-aware virtual resource management for cloud infrastructures,” in *Proc. 9th Int’l Conf. on Computer and Information Technology (CIT’09)*. Ieee, 2009, pp. 357–362.

[16] E. Elmroth and L. Larsson, “Interfaces for placement, migration, and monitoring of virtual machines in federated clouds,” in *Proc. 8th Int’l Conf. on Grid and Cooperative Computing (GCC’09)*. IEEE, 2009, pp. 253–260.