International Journal on Advanced Science Engineering Information Technology

Vol.7 (2017) No. 5
ISSN: 2088-5334

A Dynamic Resource Manager with Effective Resource Isolation Based on Workload Types in Virtualized Cloud Computing Environments

Chung Geon Song[1], Na Yoon Hwang[1], Heon Chang Yu[1], Jong Beom Lim[2, *]

[1] *Department of Computer Science and Engineering, Korea University, Seoul 02841, Korea*
E-mail: security0730@korea.ac.kr, nayoo38@korea.ac.kr, yuhc@korea.ac.kr

[2] *Department of Game & Multimedia Engineering, Korea Polytechnic University,*
Siheung-si, Gyeonggi-do 15073, Korea
E-mail: jblim@kpu.ac.kr

***Abstract*— To use computing resources for processing parallel algorithms on demand, cloud computing has been widely used since it is able to scale in response to load increases and decreases. Typically, cloud computing providers offer virtual machines to cloud users with static configurations, and these configurations are not changed until virtual machines are shutting down. To accelerate parallel processing computations in cloud computing environments, we design and implement a dynamic resource manager by isolating resources based on workload types. To avoid unnecessary context switching and increase CPUs affinity, our dynamic resource manager determines whether vCPU to physical CPU core pinning is required. If so, the VM's vCPUs are pinned by our dynamic resource manager, which can guarantee the resource and performance isolation. With our proposed resource manager for virtual machines, we can achieve a performance boost and load balancing at the same time. Performance results show that our proposed method outperforms the default scheduler of Xen about 36.2% by reducing the number of context switching for VMs.**

***Keywords*— Resource manager; hypervisor; virtualization; resource isolation; cloud computing**

I. INTRODUCTION

Scientific applications often require a massive number of computing resources for performing large-scale parallel computations. Traditionally, these needs have been addressed by using high-performance computing (HPC) hardware on physical server machines [1-3]. Cloud computing providers offer virtual machines (VMs) as computing resources to cloud users with user-specified configurations [4, 5] and VMs can be dynamically provisioned on a pay as you go basis for many applications [6-9].

Although requirements of cloud users for cloud computing services are varied, cloud computing ensures that it can supply computing resources on demand by taking advantage of virtualization technology. The advances in virtualization technology have attracted HPC users to cloud computing and enhanced the performance of the hypervisor or virtual machine monitor (VMM). The typical examples of hypervisors include Xen and kernel-based virtual machine (KVM) [10-12].

However, the specific demands of HPC applications in cloud computing environments often mismatch the assumptions and mechanisms provided by default hypervisor settings for various workloads. Figure 1 shows an example of VMs' workloads. For vm1, the application running on the VM is CPU-intensive, in other words, it consumes the CPU resource as much as possible.

On the other hand, the workload of vm2's application is like to be a background service. When the two VMs are running on the same host and the configurations of the VMs are the same, the resource allocated to vm2 has more idle time than vm1's case.

In this paper, we propose a dynamic resource manager with effective resource isolation based on workload types in virtualized cloud computing environments. The proposed resource manager monitors VMs and detects workload types of applications for VMs running on the hypervisor. Then, it dynamically allocates computing resources to VMs on runtime, which can resolve the mismatch and improve utilization of resources effectively.

The rest of this paper is organized as follows. Section2 describes our research motivation and our intuition for designing and implementing the resource manager for VMs. Section 3 provides our proposed dynamic resource manager with resource isolation. The experiments and performance analysis are given in Section 4. Finally, Section 5 concludes the paper.

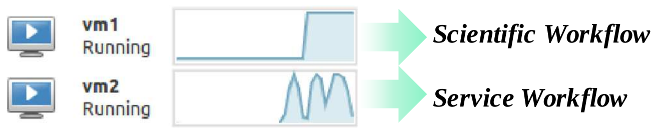

Fig. 1 VMs' workloads.

II. Material And Method

While cloud computing has been considered as an efficient solution for processing parallel applications and CPU intensive workloads, developing a dynamic resource manager with workload types in mind is not fully undertook. We found that most cloud applications could be categorized into two different groups: scientific workflow and service workflow. In this section, we provide research motivation and related work in the area.

In this section, we present our dynamic resource manager with resource isolation based on workload types. The proposed dynamic resource manager is able to recognize the workload types (scientific and service) without prior knowledge and history. With the workload type information the dynamic resource manager allocates recourses to VMs while the VMs are running.

A. Background and Motivation

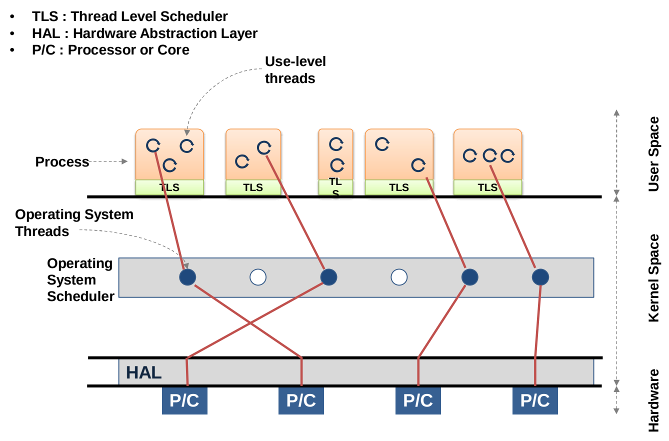

Fig. 2 Basic process scheduling in operating systems

Figure 2 shows the basic process scheduling in operating systems. The scheduler of operating systems is in charge of scheduling processes or threads. Therefore, the scheduler determines the mapping between threads and processors.

As cloud computing emerges, software system has changed from standalone applications into service-based systems based on the virtualization technology. Typically, the major goal of resource management of computer systems was to provide fairness between processes.

However, in cloud computing environments, there is another mediator, that is, hypervisor. With this in mind, a process (threads) can be considered as a VM and, therefore, the cost of context switching for VMs is higher than that for typical processes.

Since cloud computing environments are different from the traditional distributed systems, resource management schemes for processes is not suitable for VMs. In other words, it generates unnecessary context switching between VMs.

Hence, we introduce a novel dynamic resource manager with effective resource isolation for VMs considering performance and load balancing by reducing the number of context switching.

B. Related Work

In [13], Li et al. proposed an affinity-aware dynamic vCPU pinning scheduling for VMs, the mechanism aims at supporting only symmetric multi-processors (SMPs). However, the affinity information should be known before the scheduling process.

In [14], Caglar et al. proposed a log based machine learning approach for optimizing the hypervisor's system parameters by performing three phases, that is, discover, optimize, and observe steps.

While it can optimize the scheduler of the hypervisor, it requires history information for the system and involves computational cost for performing the k-means and simulated annealing algorithms. In [15], Zhou et al. proposed a dynamic VM allocation policy for a cluster by migrating VMs for load balancing.

Our work differs from previous work in that our dynamic resource manager can be applied to both SMPs and asymmetric multi-processors (AMPs) systems since our daemon service implementation is not dependent on a specific system.

Furthermore, the monitoring scheme of our dynamic resource manager periodically checks the recent system information on a real time basis, and does not require history information and complex computation for the resource allocation algorithm.

Moreover, with our dynamic resource manager, both performance gain and load balancing can be achieved by vCPU pinning. In other words, once vCPUs are pinned, other VMs cannot interfere the pinned vCPUs.

C. System Model

Figure 3 shows the architecture of our dynamic resource manager. The dynamic resource manager resides in the host OS and it periodically monitors resources of the system for the host OS and the guest OSes to check whether resource re-allocation is required. The monitor module in the dynamic resource manager automatically categorizes the VMs in the system according to the algorithm.

Then, the allocation module manages the resource allocation to optimize the performance of the system. Our dynamic resource manager has two policies to allocate resources to VMs:

- *allocate_pr*: it allocates resources to VMs evenly by considering load balancing;

- *allocate_sla*: it allocates resources to VMs restrictively by considering SLA.

The *allocate_pr* policy is for the scientific workflow. When the *allocate_pr* policy is used, the dynamic resource manager decides the amount of resources for the scientific workflow and allocates the resources to the VM accordingly. Therefore, the scientific workflow can benefit from the dynamic resource manager.

The *allocate_sla* policy is for the service workflow. When the *allocate_sla* policy is used, the dynamic resource manager allocates the minimum amount of resources to VM provided that SLA violation is not detected.

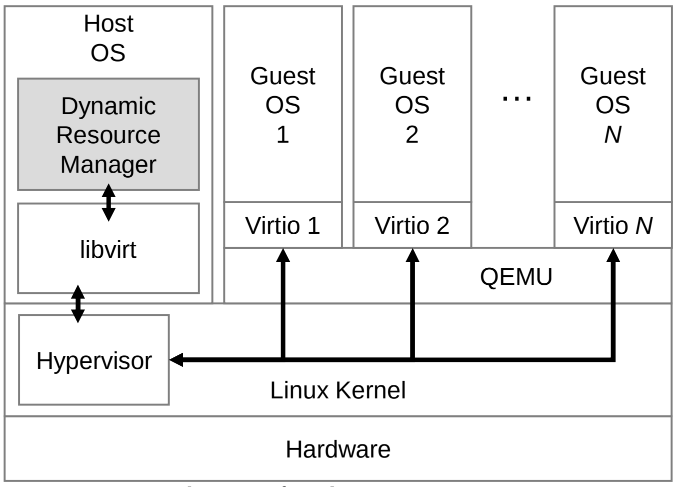

Fig. 3 Architecture of our dynamic resource manager

D. Basic Idea

The hypervisor regards vCPU as a logical computation unit for VMs. Unlike physical machines, by taking advantage of virtualization technology, the hypervisor is capable of adjusting the number of vCPUs of VMs regardless of the number of physical CPU cores while VMs are running.

When the number of vCPU is greater than the number of physical CPU cores (overcommit), context switching is inevitable. On the other hand, when the number of vCPU is less than the number of physical CPU cores, VMs are underutilizing the host resources. Based on applications' workload types of VMs, our proposed dynamic resource manager adds additional vCPUs considering the host's available physical CPU cores.

Then, to avoid unnecessary context switching and increase CPUs affinity, our dynamic resource manager determines whether vCPU to physical CPU core pinning is required. If so, the VM's vCPUs are pinned by our dynamic resource manager, which can guarantee the resource and performance isolation.

Figure 4 illustrates an example of the basic idea behind our vCPU pinning algorithm for the dynamic resource manager. Suppose the workload type of VM1's application is scientific workflow, our dynamic resource manager is able to detect the type of the application by monitoring performance metrics. The T_{check} parameter is used for monitoring period and the administrator can adjust the parameter according to the resource management policy.

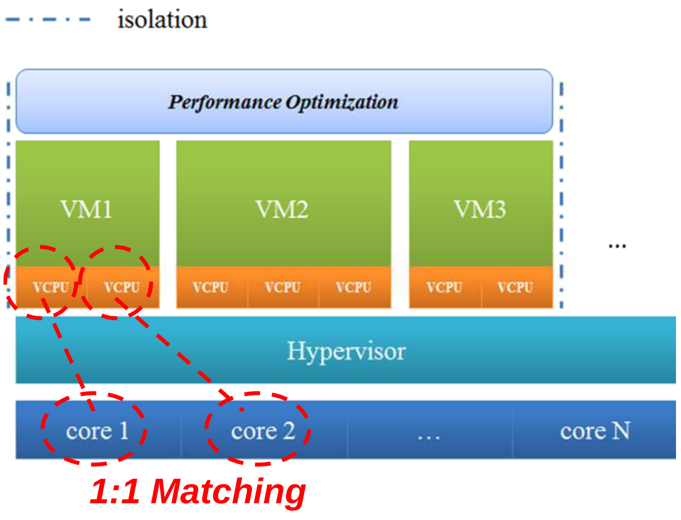

Fig. 4 An example of vCPU to physical CPU core pinning.

In our design, when the performance metric exceeds the threshold value, our dynamic resource manager regards it as the scientific workflow. Whereas, when the performance metric is below the threshold value, it is considered as the service workflow.

Note that the workload type can be changed from scientific to service or vice versa on runtime. With this in mind, our dynamic resource manager can achieve the performance gain while maintaining SLA by isolating resources. The next subsection describes the algorithmic details for our dynamic resource manager with resource isolation, and the symbols used in the algorithms are listed in Table 1.

TABLE I
SYMBOLS USED IN THE ALGORITHMS

Symbol	Description
VM_i^{sc}	i-th VM of VM^{sc}
VM_i^{se}	i-th VM of VM^{se}
$List_{sc}$	A list of VM^{sc}
$List_{se}$	A list of VM^{se}
T_{check}	Monitoring period
α	Threshold of utilization
β	Threshold of count
N_{sc}	The number of VM^{sc}
N_{se}	The number of VM^{se}
add_vm()	A function to add a VM to a list
delete_vm()	A function to delete a VM from a list
allocate_pr()	A function to perform the allocate_pr policy with a VM list
allocate_sla()	A function to perform the allocate_sla policy with a VM list

E. Resource Monitoring Algorithm

The resource monitoring step is essential for the dynamic resource manager. Based on the resource monitoring information, our dynamic resource manager can categorize the workflow types of VMs and allocate the specified amount of resources to VMs.

The resource monitoring algorithm is periodically performed as follows (cf. Algorithm 1).

1) When a VM is created: Since our dynamic resource manager does not rely on history information of VMs, the default policy is used when a VM is created. In other words, the type of workflow of the newly created VM is unspecified and the default resource scheduling policy is used. Note that when the default resource scheduling policy is used, the VM's resource is static and does not changed during the VM is running in the system.

2) *Checking the resource utilization of VMs*: For a newly created VM, the resource monitoring algorithm checks the resource utilization. When the utilization of the VM exceeds the threshold *α*, it increases *count_scientific* parameter. Similarly, when the utilization of the VM below the threshold *α*, it increases *count_service* parameter. The *count_scientific* and *count_service* parameters are used when determining the types of workflow.

In this procedure, the *count_scientific* and *count_service* parameters cannot have the positive values simultaneously when determining the workflow types. Either *count_scientific* or *count_service* parameter can have the positive value in order to become a workflow type.

Algorithm 1. Dynamic Resource Monitoring Algorithm

```
1:  /* The algorithm is performed every $T_{check}$ */
2:  /* Resource monitoring for $VM^{se}$ */
3:  for $i$ =1 to $N_{se}$ do
4:      if ($VM_i^{se}.cpuUtilization$ > α) then
5:          increase $VM_i^{se}.count$ by 1;
6:      else
7:          decrease $VM_i^{se}.count$ by 1;
8:      end if
9:  end for
10: /* Resource monitoring for $VM^{sc}$ */
11: for $i$ =1 to $N_{sc}$ do
12:     if ($VM_i^{sc}.cpuUtilization$ < α) then
13:         increase $VM_i^{se}.count$ by 1;
6:      else
7:          decrease $VM_i^{se}.count$ by 1;
16:     end if
17: end for
```

More specifically, when the resource monitoring algorithm increases the *count_scientific* parameter, it initializes the *count_service* parameter to 0. Similarly, when the resource monitoring algorithm increases the *count_service* parameter, it initializes the *count_scientific* parameter to 0. By doing so, the dynamic resource manager maintains the workflow types of VMs in the system.

Algorithm 2 Dynamic Resource Management Algorithm

```
1:  /* The algorithm is performed every $T_{check}$ */
2:  /* Dynamic resource management for $VM^{se}$ */
3:  for $i$ = 1 to $N_{se}$ do
4:      if ($VM_i^{se}.count > \beta$) then
5:          $VM_i^{se}.count \leftarrow 0$;
6:          delete_vm($List_{se}$, $VM_i^{se}$);
7:          add_vm($List_{sc}$, $VM_i^{se}$);
8:          allocate_pr($List_{sc}$);
9:      end if
10: end for
11: /* Dynamic resource management for $VM^{sc}$ */
12: for $i$ = 1 to $N_{sc}$ do
13:     if ($VM_i^{sc}.count > \beta$) then
14:         $VM_i^{sc}.count \leftarrow 0$;
15:         delete_vm($List_{sc}$, $VM_i^{sc}$);
16:         add_vm($List_{se}$, $VM_i^{sc}$);
17:         allocate_sla($List_{se}$);
18:     end if
19: end for
```

3) *Determining the workflow types*: Based on the resource monitoring information, our dynamic resource manager is able to determine the types of workflow. When the *count_scientific* parameter reaches *β*, the dynamic resource manager considers the workflow as the scientific type. Similarly, when the *count_service* parameter reaches *β*, the dynamic resource manager considers the workflow as the service type.

Note that a VM cannot become both scientific and service workflow types since the resource monitoring algorithm guarantees the mutual exclusion for the *count_scientific* and *count_service* parameters.

F. Dynamic Resource Management Algorithm

With the workflow type information, the dynamic resource manager allocates resources to VMs as follows (cf. Algorithm 2).

1) *Resource maintenance*: The dynamic resource manager maintains the available resources of the system as Equation 1 for performance and load balancing.

$$Total\_Res = \gamma \cdot Res_{scientific} + (1-\gamma) \cdot Res_{service}, \tag{1}$$

where γ is the system parameter and $0 < \gamma < 1$.

The γ parameter can be tuned according to the number of scientific workflows and the number of service workflows in the system. With this resource maintenance method, the resource isolation is effectively achieved by taking advantage of the libvirt library.

When the number of scientific workflows is greater than the number of service workflows, the dynamic resource manager increases the γ parameter. On the other hand, when the number of service workflows is greater than the number of scientific workflows, the dynamic resource manager decreases the γ parameter.

2) For scientific workflow: The dynamic resource manager allocates $\gamma \cdot Res_{scientific}/n_{scientific}$ to a VM, where $n_{scientific}$ is the number of scientific workflows.

3) *For service workflow*: The dynamic resource manager allocates the minimum amount of resources to a VM if SLA violation is not detected.

Note that like the resource monitoring algorithm, the resource allocation procedure is also performed periodically. Therefore, the dynamic resource manager can achieve both performance and load balancing by reducing the number of context switching for VMs.

III. Results And Discussion

In this section, we provide the experimental results to show the effectiveness of the proposed dynamic resource manager. The NAS Parallel Benchmarks (NPBs) are used for scientific workflow applications. Table 2 shows the experimental settings for performance evaluation.

Although the experimental environment is configured with the Xen hypervisor, our dynamic resource manager can be used with other hypervisors like KVM since the resource monitoring module and the dynamic resource allocation module are developed without dependency of hypervisors.

TABLE II
Experimental Settings

Parameter	Value
Host OS	Ubuntu 14.04
Hypervisor	Xen
Host CPU	Intel(R) i5-4590T
The number of host CPU cores	Quad core without hyper-threading
Host Memory	8 GB

Figure 5 shows the performance results of NPBs for the default scheduling method and the proposed method with our dynamic resource manager. The results show that with our dynamic resource manager, the execution time is reduced by 36.2% for the benchmarks on average. This demonstrates that our dynamic resource manager effectively controls the vCPU to physical CPU core mapping and monitors the VMs' resource status.

Because our dynamic resource manager is running as a background service at dom0, the overhead of controlling VMs is almost zero without requiring history information or complex computation. It is interesting to note that the proposed solution can achieve almost 2 times faster than the default scheduling policy for some of the benchmark results (e.g., BT and LU).

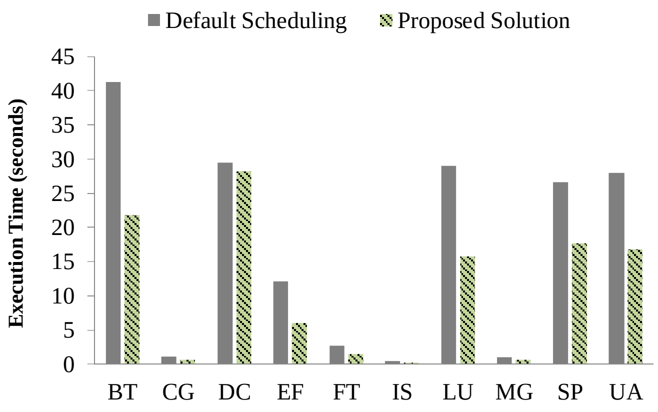

Fig. 5 Performance comparison for benchmark runtime

This signifies that when we apply the proposed dynamic resource manager to the scalable clusters, the performance gain will become large. Since we focus on implementing the prototype of the dynamic resource manager by pinning vCPUs to physical CPU cores and detecting the workload types of VMs' applications with our novel algorithm, we leave the deployment our dynamic resource manager to a large-scale cluster as future work.

For some of the benchmark results (e.g., CG), the difference of execution time is comparable. The reason why this result is introduced is that some applications do not affected by CPU performance.

Nevertheless, the proposed solution always results in the better execution time in comparison with the default scheduling policy. With our dynamic resource monitoring and management algorithm, the number of context switching can be reduced since our algorithm does not allow vCPU overcommits.

The downside of our approach is as follows. Since the resources are allocated on runtime, some vCPUs mappings to physical CPU cores should be altered as the number of VMs is increasing or decreasing or the workload types of the applications are changing.

To overcome this challenge, we can use the epoch parameter to determine the re-allocation of vCPUs. After epoch, our dynamic resource manager checks whether the vCPUs mapping has to be altered. If this is the case, the algorithm adds or subtracts vCPUs according to the monitoring information without violating SLA and allowing overcommits.

TABLE III
Performance Results for the Number of Context Switching

	Default Scheduling	Proposed Solution
BT	200,932	110,576

Table 3 shows performance results for the number of context switching. For the BT benchmark, the number of context switching is 200,932 when the default scheduling method is used. When our dynamic resource manager is used, the number of context switching is 110,576, which is about 55% of reduction in comparison with the default scheduling method.

IV. Conclusion

In this paper, we proposed a dynamic resource manager for parallel processing in virtualized cloud computing environments. Our proposed resource manager for virtual machines classifies workloads into two categories (service workload and scientific workload). Then, it effectively isolates computing resources for virtual machines during runtime. Performance results show that our proposed method outperforms the default scheduler of Xen about 36.2% by reducing the number of context switching for VMs. Future work includes finding the optimal resource monitoring period and applying machine learning algorithms for the tasks of resource isolation and scheduling.

Acknowledgment

This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (NRF-2015R1D1A1A01061373). The author to whom correspondence should be addressed is JongBeom Lim.

References

[1] D. Zhao *et al.*, "FusionFS: Toward supporting data-intensive scientific applications on extreme-scale high-performance computing systems," in *2014 IEEE International Conference on Big Data (Big Data)*, 2014, pp. 61-70, 2014.

[2] I. A. T. Hashem, I. Yaqoob, N. B. Anuar, S. Mokhtar, A. Gani, and S. Ullah Khan, "The rise of “big data” on cloud computing: Review and open research issues," *Information Systems*, vol. 47, pp. 98-115, 1// 2015.

[3] M. Ben Belgacem and B. Chopard, "A hybrid HPC/cloud distributed infrastructure: Coupling EC2 cloud resources with HPC clusters to run large tightly coupled multiscale applications," *Future Generation Computer Systems*, vol. 42, pp. 11-21, 1// 2015.

[4] J. Lim, H. Yu, and J. M. Gil, "Detecting Sybil Attacks in Cloud Computing Environments Based on Fail‐Stop Signature," Symmetry, vol. 9, no. 3, p. 35, 2017.

[5] J. Lim, T. Suh, J. Gil, and H. Yu, "Scalable and leaderless Byzantine consensus in cloud computing environments," *Information Systems Frontiers*, journal article vol. 16, no. 1, pp. 19-34, 2014.

[6] S. S. Manvi and G. Krishna Shyam, "Resource management for Infrastructure as a Service (IaaS) in cloud computing: A survey," *Journal of Network and Computer Applications*, vol. 41, pp. 424-440, 5// 2014.

[7] N. Grozev and R. Buyya, "Performance Modelling and Simulation of Three-Tier Applications in Cloud and Multi-Cloud Environments," *The Computer Journal*, vol. 58, no. 1, pp. 1-22, 2015.

[8] A. M. Kadhum and M. K. Hasan, "Assessing the Determinants of Cloud Computing Services for Utilizing Health Information Systems: A Case Study," *International Journal on Advanced Science, Engineering and Information Technology*, Health information system; cloud computing; cloud services; healthcare; health informatics; preliminary study; qualitative interview; Iraq vol. 7, no. 2, 2017.

[9] M. A. Rodriguez and R. Buyya, "A taxonomy and survey on scheduling algorithms for scientific workflows in IaaS cloud computing environments," *Concurrency and Computation: Practice and Experience*, vol. 29, no. 8, pp. 1-32, 2017, Art. no. e4041.

[10] P. Barham *et al.*, "Xen and the art of virtualization," *SIGOPS Oper. Syst. Rev.*, vol. 37, no. 5, pp. 164-177, 2003.

[11] I. Habib, "Virtualization with KVM," *Linux J.*, vol. 2008, no. 166, p. 8, 2008.

[12] S. Varrette, M. Guzek, V. Plugaru, X. Besseron, and P. Bouvry, "HPC Performance and Energy-Efficiency of Xen, KVM and VMware Hypervisors," in *2013 25th International Symposium on Computer Architecture and High Performance Computing*, 2013, pp. 89-96.

[13] Z. Li, Y. Bai, H. Zhang, and Y. Ma, "Affinity-Aware Dynamic Pinning Scheduling for Virtual Machines," in *2010 IEEE Second International Conference on Cloud Computing Technology and Science*, 2010, pp. 242-249.

[14] F. Caglar, S. Shekhar, and A. Gokhale, "iTune: Engineering the Performance of Xen Hypervisor via Autonomous and Dynamic Scheduler Reconfiguration," *IEEE Transactions on Services Computing*, vol. PP, no. 99, pp. 1-1, 2016.

[15] W. Zhou, S. Yang, J. Fang, X. Niu, and H. Song, "VMCTune: A Load Balancing Scheme for Virtual Machine Cluster Using Dynamic Resource Allocation," in *2010 Ninth International Conference on Grid and Cloud Computing*, 2010, pp. 81-86.