Application Scheduling in Cloud Computing Environment with the Consideration of Performance Interference

Lei Yang[1] and Yu Dai[2]
[1] College of Information Science and Engineering, Northeastern University, Shenyang 110004, P.R.China
[2] College of Software, Northeastern University, Shenyang 110004, P.R.China
Email: {yanglei, daiyu}@mail.neu.edu.cn

Abstract —Virtualization technology which can make multiple virtual machines run in a shared physical host has received much attention. However, the consolidation can result in the contention in the shared resources and lead to a degradation of the performance deployed on the virtual machines. To avoid such degradation while respect the utility of the physical host, it needs to study the performance interference effects of the virtualized environment and schedule the application with the consideration of this interference. Then, we develop an application scheduling framework to improve the application performance when co-located with others in the virtualized environment. The experiments show the better performance of our methods.

Index Terms—Performance interference, virtualized environment, cloud computing, application scheduling

I. INTRODUCTION

Recently, cloud computing has received much attention. Virtualization technology [1], [2] is one of the important techniques in the cloud computing, which allows multiple applications to run on the same hardware simultaneously.

Ideally, the performance of an application should be independent of the others co-located on the same machine. However, modern virtual machine technologies do not provide effective performance isolation [3]. Although extensive works [4]-[6] have been done to achieve performance isolation among VMs consolidated on the same physical host, performance interference still remains especially for some I/O-intensive or mixed type of applications [7]. Then, it needs to predict the performance interference among VMs when you want to deploy an application on a VM which will be consolidated with other VMs on the physical host.

For modeling and predicting the performance interference among VMs, a set of research works [9]-[16] have been done. However, current works usually assume a single performance bottle neck of the applications and only deal with the CPU-intensive application or the I/O intensive applications. However, different mixed application may have very different usage pattern of the CPU resource and I/O resource. Then, using a uniform model to predict this kind of applications may not be very appropriate. The current model training method cannot be used for predicting the performance interference.

In this paper, we develop an application scheduling framework that can improve the application performance when co-located with others in the virtualized environment. The proposed framework leverages performance interference by predicting techniques which acts as the core for application scheduling in the virtualized environment. The experiments show the better performance of the resource allocation method. The main contributions of our work are as follows:

The rest of paper is organized as follows. The next section introduces the related works in the field of application scheduling in the virtualized environment. Section 3 overviews the proposed application scheduling framework. Section 4 and 5 present the methods of modeling and scheduling techniques irrespectively in the proposed framework. The evaluation results are presented in Section 6. We conclude the paper in Section 7.

II. RELATED WORKS

Currently, there have been a number of noticeable efforts putting into the performance modeling of the applications [8]-[15]. Most of these efforts, aim to streamline resource management, i.e., maximization of resource utilization and application performance. A recent study on VM consolidation [17] has revealed that frequent live VM migration may even lead to significant performance degradation, and thus resource usage characteristics of the application should be considered in resource allocation. Reference [18] also reports similar results and develop a workload predicting technique incorporated in the consolidation algorithm. Reference [14] utilizes online feedback to build a multi-input multi-output model to capture the performance interference and to tune resource allocations to mitigate the performance interference.

In the field of modeling the performance interference among applications, resource contention between processes in a single OS is well-researched. Reference [19], [20] introduces a hardware activity vector to

Manuscript received April 13, 2015; revised August 19, 2015.
This work was supported in part by the National Key Technology R&D Program of the Ministry of Science and Technology (2015BAH09F02, 2015BAH47F03), National Natural Science Foundation of China (60903008, 61073062) and the Fundamental Research Funds for the Central Universities (N130417002, N130404011).
Corresponding author email: daiyu@mail.neu.edu.cn.
doi:10.12720/jcm.10.8.603-609.

monitor the access patterns on the cache. Reference [8] uses application characteristics to model the virtualization overheads. References [5], [6] and [12], [13] study the network I/O interference in virtualized cloud environments. References [14], [15] use online feedback to model the performance interference among applications. However, it mainly focuses on CPU-intensive application. Reference [10] proposes a method for modeling the relation between the system-level workloads and the performance interference degree, while the model can only be used for predicting the performance interference among disk I/O-intensive applications. Reference [11] proposes a model for modeling the relation between the system-level workloads and the performance interference degree.

However, current works always assume perfect performance isolation among VMs hosted on the same physical machine. Some research works [10] considers the performance interference among VMs when considering the problem of VM consolidations, these works always assume the availability of the historical data about the performance interference of the VM to be predicted. And current works always focus on one kind of application and seldom considers the performance interference prediction of mixed application.

III. Framework Overview

In the following, we present the basic framework for application scheduling with the consideration of the performance interference among VMs which is shown in Fig. 1.

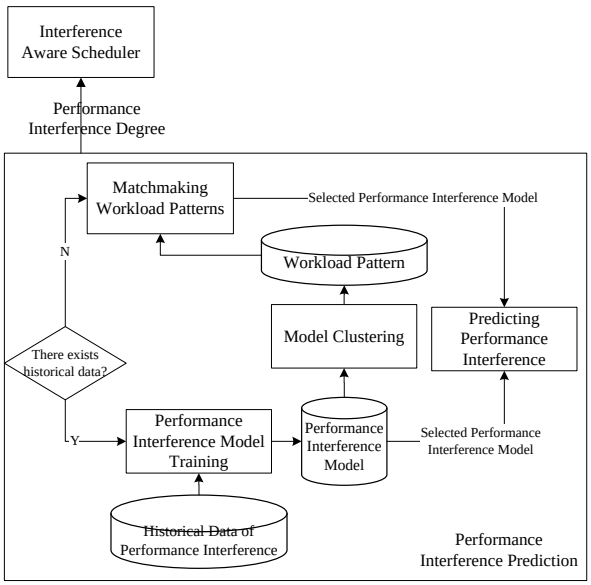

Fig. 1. Basic framework for application scheduling with the consideration of the performance interference.

The proposed application scheduling framework consists of 5 major components: *Interference Aware Scheduling*, *Performance Interference Model Training*, *Predicting Performance Interference*, *Model Clustering and Workload Pattern Matchmaking*. Here, *Interference Aware Scheduling* is used to utilize the performance interference prediction to infer the application performance under interference and generate optimized placement of tasks and physical hosts. *Performance Interference Model Training* is used for modeling the performance interference among VMs based on historical data of the application co-located on the same physical host. *Model Clustering* is used for clustering the available performance interference models by the workload pattern. *Workload Pattern Matchmaking* is used for match making between the workload of the application and the workload patterns of the available models to get the performance interference model for the application whose performance interference will to be predicted. *Predicting Performance Interference* is to get the performance interference degree based on the performance interference model by feeding the corresponding parameters.

In the following, we will discuss the implementation of the major components in the framework.

IV. Methods for Predicting Performance Interference

A. Modeling the Performance Interference Using Linear Regression

As for the problem of application scheduling, we need to know the degree of the performance interference among VMs to determine where to place a "new" VM. We will measure whether the "new" VM will affect the performance of the VMs already deployed on the same physical host. For simplicity, we call the "new" VM as "foreground" VM, while all the other VMs co-located on the same physical host is called as a whole of "background" VM. The aim of predicting the performance interference degree is to measure the performance interference degree between the foreground and the background VMs.

Definition 1. Performance Interference Degree. Performance interference degree reflects the extent to which the performance of the foreground VM (*FW*) will be affected by the background VMs (*BW*). Equation (1) shows how to compute it.

$$PID(FW@BW) = \frac{Perf(FW@BW) - Perf(FW@Idle)}{Perf(FW@Idle)} \quad (1)$$

where $Perf(FW@BW)$ is the performance of the application on the foreground running against the background; $Perf(FW@Idle)$ is the performance of the application on the foreground VM when it runs alone.

To capture VM behaviors that generate performance interference, we collect system-level workloads to find the indicator of the performance interference. We collect the following system-level workload of the VM, such as average CPU utilization (*cpuutil*), average memory utilization (*memutil*), average number of read operations per seconds (*rps*), average number of write operations per

seconds (*wps*), average waiting time of the I/O operations (*await*) and average time spent for the request in the disk device (*svctm*). We can use the following equation to show the relationship between the system-level workload and the performance interference degree.

$$PID(FW@BW) = a_0 + a_1 \times cpuutil_{BW} + a_2 \times memutil_{BW} + a_3 \times rps_{BW} + a_4 \times wps_{BW} + a_5 \times await_{BW} + a_6 \times svctm_{BW} \quad (2)$$

where a_0, a_1, a_2, a_3, a_4, a_5, a_6 are coefficients. The system-level workloads of the background are $cpuutil_{BW}$, $memutil_{BW}$, rps_{BW}, wps_{BW}, $await_{BW}$ and $svctm_{BW}$.

Then, if the coefficients a_0, a_1, a_2, a_3, a_4, a_5, a_6 in Equation (2) is known, we can predict the performance interference degree of the foreground VM when the background VM's system-level workloads is known. Then, in the following, we will present how to use linear regression to estimate the coefficients a_0, a_1, a_2, a_3, a_4, a_5, a_6.

To estimate the coefficients is to find a combination of a_0', a_1', a_2', a_3', a_4', a_5', a_6' which can make the observed value much more closet to the predicted one, as shown in equation (3).

$$Q = \sum_i \left(PID(FW@BW_i) - PID\left(\overline{FW@BW}_i\right) \right)^2 \quad (3)$$

where BW_i is the i^{th} observations; $PID(FW@BW)$ is the observed performance interference degree in the i^{th} observations; $PID\left(\overline{FW@BW}_i\right)$ is the predicted performance interference degree, which can be calculated as Equation (4).

$$PID\left(\overline{FW@BW}\right) = a_0' + a_1' \times cpuutil_{BW} + a_2' \times memutil_{BW} + a_3' \times rps_{BW} + a_4' \times wps_{BW} + a_5' \times await_{BW} + a_6' \times svctm_{BW} \quad (4)$$

We can use least square approach [24] to estimate the coefficients a_0', a_1', a_2', a_3', a_4', a_5', a_6'. And then the performance interference degree of the foreground VM can be predicted if the system-level workloads of the background VM are known. As for the limitations of the paper, we will not present the concrete algorithm in details.

If the historical data of the performance interference between VMs for training the model is available, we can use the above method to find the coefficients. However, as for the problem of VM placement, we may not know the historical data. Then, in this situation, we will use the past experience to infer the performance interference between VMs. Then, in the following, we will discuss how to make a prediction of the performance interference under this situation.

B. Generating Combined Performance Models

Performance interference has a relation with the system-level workloads of the background and foreground VMs. When the system-level workloads of two applications are similar, we can infer one application's performance interference from the performance of the other.

Imagine we have a set of performance interference degree models, signified as H={$PID(FW_1@)$, $PID(FW_2@)$, …, $PID(FWn@)$}. Here, we call FW_i the workload pattern.

Since the historical data about the performance interference of *vm* (the virtual machine to be placed) with other VMs is not enough for the model training, we will find the performance interference degree models with the highest similarity of the foreground workload with the VM *vm*. If we can find a set of performance interference degree models whose workload pattern is similar with *vm*, then we can combine these models together to generate the model of *vm*.

In the following, we will show how to compute the similarity degree between two workload patterns and how to generate the combined model for *vm*.

Imagine we have 2 workload patterns wp_i and wp_j, each of which is a vector of the system-level workloads as described above. We will use Euclidean distance to compute the similarity between workload patterns as shown in Equation (5).

$$d(wp_i, wp_j) = \frac{1}{\sqrt{\begin{array}{l} w_{cpu} \times \left(cpuutil_i - cpuutil_j\right)^2 \\ + w_{mem} \times \left(memutil_i - memutil_j\right)^2 \\ + w_{rps} \times \left(rps_i - rps_j\right)^2 + w_{wps} \times \left(wps_i - wps_j\right)^2 \\ + w_{await} \times \left(await_i - await_j\right)^2 \\ + w_{svctm} \times \left(svctm_i - svctm_j\right)^2 \end{array}}} \quad (5)$$

where w_{cp}, w_{mem}, w_{prs}, w_{wps}, w_{await} and w_{svtm} are the weights for adjusting since the value ranges of the system-level workloads are different.

Then, we can find the workload pattern similar to *vm*. Here, we can use a threshold, and when the similarity degree between the workload pattern and *vm* is beyond the threshold, it means that this workload pattern's corresponding performance interference degree model can be used for predicting the interference degree of *vm*. If we can find more than one workload patterns whose similarity degree with *vm* is beyond the threshold, we can use the following equation to generate the performance interference model of *vm*.

$$PID(FW@BW) = \sum_i \left(\frac{d_i}{sumd} \right) \times PID(FW_i@BW) \quad (6)$$

where FW is the workload pattern of VM *vm*. FW_i is the workload pattern whose similarity degree with *vm* is beyond the threshold. $PID(FW_i@BW)$ is the performance interference degree model corresponding to FW_i. d_i is the similarity degree between FW and FW_i; *sum* is the sum of the similarity degree between FW and each FW_i, that is, $sum=\sum d_i$.

Then, we can use the equation (6) to predicting the performance degree of *vm*.

V. Methods for Scheduling Applications in Virtualized Environment

With the help of performance interference prediction, the proposed application scheduling system can now schedule the incoming tasks to different virtual machines in a way that minimizes the interference effects from co-located applications. Generally speaking, optimally assigning tasks to physical machines in parallel and distributed computing environments has been shown to be an NP-complete problem [21]. In this work, we explore a number of heuristic techniques to find a good solution for the scheduling problem. In the following, we will present the aim of the scheduling problem and then give the solutions to this problem.

Specifically, the aim of the application scheduling in the virtualized environment is to reduce the total performance interference degree while respect to the utilization of the physical resource. Given a set of application deployment requests T and for each task $t \in T$, it has the requirement towards the resource which is denoted as R=<*cpu*, *memory*, *disk*>. Here, *cpu* is the amount of CPU resource the task *t* needed, *memory* is the amount of memory resource the task *t* needed and *disk* is the amount of disk resource the task *t* needed. Given a set of physical hosts $PM=\{pm_1, pm_2, \ldots, pm_n\}$, then, the aim of the scheduling problem is to find an optimal mapping from the set of tasks to the set of physical hosts, to satisfy the resource requirement of the tasks and achieve the highest utilization of the physical resource while considering the interference effects among VMs. In this work, we explore the following two scheduling algorithms.

Online scheduling algorithm is to make a quick scheduling decision that becomes necessary when the tasks arrive at rapid speed. In such a scenario, the tasks will arrive at the queue at arbitrary times and the scheduler will dispatch an incoming task immediately without waiting for later tasks. With the goal of minimizing the number of physical machines and the overall performance interference degree of all the tasks, the online scheduling algorithm maps each incoming task to the physical machine with the minimal interference degree and with the minimal rest of available resources. The online scheduling algorithm is presented in Algorithm 1.

Algorithm 1. Algorithm for Online Scheduling the Application
Input: Task *t*; set of Physical hosts *PM*; the workload pattern (*FW*) of task *t*
Output: placement plan (*candidatePM*) of task *t*
Begin
1. For each pm_i in the *PM* do
2. Begin
3. If *t.cpu*<=pm_i.CPUAvailable and *t.memory*<=pm_i.memoryAvailable and *t.disk*<=pm_i.diskAvailable then
4. begin
5. *BW*=getBackgroundWorkloadPattern(pm_i);
6. *pid*=GetPID(*FW*, *BW*);
7. *utility*=GetUtiltity(*t*, pm_i);
8. If *min*> $\frac{c \times pid}{utility}$ then //*c* is a constant
9. begin
10. *min*= $\frac{c \times pid}{utility}$;
11. *candidatePM*= pm_i;
12. end
13. end
14. end
End

In a batch scheduling scenario, the scheduling process takes place when the queue that holds the incoming tasks is full. Imagine we have a queue of incoming tasks, and the length of the queue is *m*, *n* is the number of physical hosts. In the batch scheduling algorithm, the first step is to take the first task t_1 in the queue as the input to run the online scheduling algorithm, and secondly, another task t_2 from the rest of the queue which has the least interference with t_1 will be picked out and t_2 will be taken as the input to run the online scheduling algorithm. The batch scheduling algorithm is shown in Algorithm 2.

Algorithm 2. Algorithm for Batch Scheduling the Application
Input: Queue of Tasks *Q*; set of Physical hosts *PM*;
Output: placement plan (*candidatePMs*) of *T*.
Begin
1. while *Q* is not empty do
2. begin
3. *candidatePMs*[t_1]=OnlineScheduling(t_1, *PM*, t_1.*workloadPattern*); // run the online scheduling algorithm
4. For each task t_i in *Q* and *i*<>1 do
5. begin
6. *PID*=GetPID(t_1.*workloadPattern*, t_i.*workloadPattern*);
7. If *min*>*PID* then
8. *t*=t_i;
9. End
10. *candidatePMs*[*t*]=OnlineScheduling(*t*, *PM*, *t*.workloadPattern);
11. RemoveFromQueue(t_1, *t*);
12. end
End

VI. Experiments

In order to verify the effectiveness of our framework, we have done a set of experiments. In the first and second parts of the experiments, we will evaluate the performance interference prediction methods proposed in this paper. And in the second parts of the experiments, we will evaluate the effectiveness of the proposed application scheduling method.

In the first and second parts of the experiments, two VMs are created in a physical host with a Xen hypervisor. Each VM domain only runs one application. All the configuration of the physical hosts are the same and as the followings. The CPU is Intel Core i3 3.3G with 4G memory and 250G disk and the version of the operating system is Ubuntu 12.04. The configuration of the virtual

machine is as the following. The VMs are created using the Xen hypervisor with 4VCPU, 1G memory and 8G disk and the operating system is Ubuntu 12.04.

In the first experiment, we will test the effectiveness of the performance interference degree model based on linear regression. Take the applications in our experiment as the foreground VM and train the performance interference degree model. Fig. 2 shows the average error and the maximum error as well as the minimum error between the predicted performance degree and the observed one.

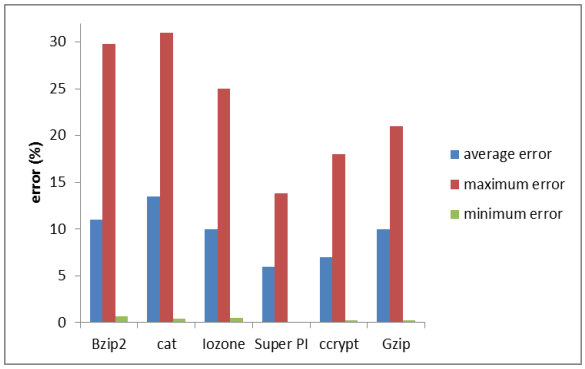

Fig. 2. Result of average, maximum and minimum error.

From Fig. 2, the I/O-intensive application's errors (such as cat) are bigger than the other type of application (such as Super PI). The prediction result can be accepted since the average error ranges from 6% to 13%.

We also test the predicted performance interference degree of applications *cp*, *dd* and *spinlock* which has no historical data about the performance interference in our experiments. Take the average predicted performance interference degree and the observed one of the applications. Fig. 3. shows the result.

From Fig. 3, the prediction result can be accepted since the average error ranges from 7% to 14%.

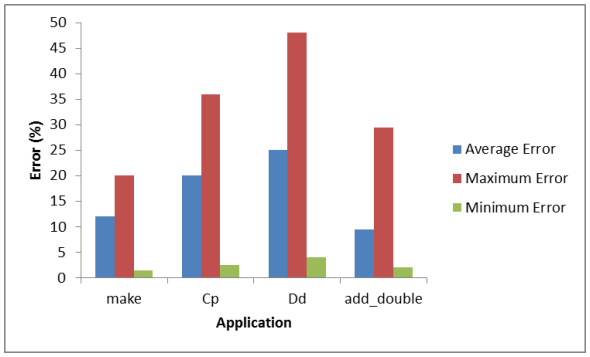

Fig. 3. Result of average, maximum and minimum error.

The above experiment results show the predicted performance interference degree is close to the observed one.

We evaluate the application scheduling framework in a 36-node Xen-based private virtual cluster, which consists of 12 physical servers, each of which is configured with Intel Core i3 CPU, 4GB memory and one 250GB disk.

We evaluate the proposed application scheduling framework using the applications in the above experiments. In the experiments, we compare the performance of the proposed application scheduler framework with 2 other main competitors in practical use: the one (we call *NonInterferenceScheduler*) which do not consider the performance interference among VMs and uses a min-min heuristic algorithm [22] for finding the assignments of the task; the one [7] (we call *InterferenceScheduler*) which considers the performance interference among VMs. The constant *c* in the proposed algorithm is set to 4. The result is shown in Fig. 4 and Fig. 5. In Fig. 4, the normalized runtime is a ratio of the response time of the application after scheduled in the virtualized environment to the response time of the application in the non-sharing resource environment.

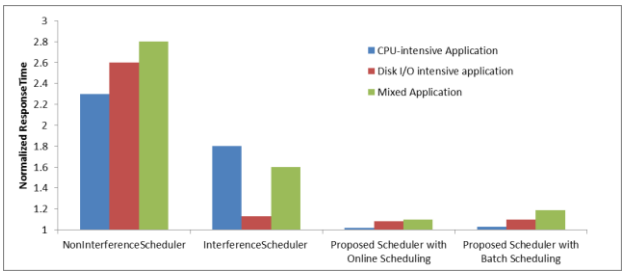

Fig. 4. Comparison result of normalized response time of different scheduling algorithms.

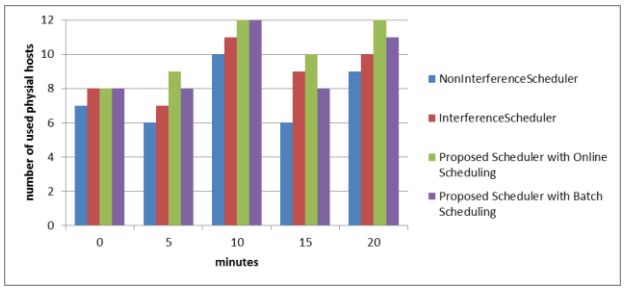

Fig. 5. Comparison result of the number of used physical hosts of different scheduling algorithms.

From Fig. 4, the normalized response time of the proposed batch scheduling algorithm is always the lowest among all the algorithms. And the *NonInterferenceScheduler* always has the highest normalized response time. This is the *NonInterferenceScheduler* does not considers the performance interference among VMs when scheduling the application. As for the *InterferenceScheduler* can achieve almost the same normalized response time as the proposed algorithms when the application is Disk I/O intensive application while if the application is of other types the normalized response time is high. This is because the prediction model in *InterferenceScheduler* is only disk I/O intensive application. Then, when the application is of other types, the accuracy of the prediction result cannot be insured which may result in a bad performance in the scheduling. The proposed batch scheduling algorithm can get a little bigger normalized

response time than the proposed online scheduling algorithm.

From Fig. 5, the *NonInterferenceScheduler* always has the lowest number of the used physical hosts than others. This is because that in *NonInterferenceScheduler*, it does not consider the performance interference and only considers the utility of the physical hosts. As for the proposed online scheduling and batch scheduling algorithms, the latter one can always use less physical hosts than the former one while only at a sacrifice of a little response time degradation (from Fig. 5 we can find it).

The above experiment results verify the effectiveness of the proposed method for predicting the performance interference and also the proposed application scheduling framework for mitigating the performance interference while respecting the utility of the physical hosts.

VII. Conclusions

In this work, we present an application scheduling framework with the consideration of the performance interference among VMs. We propose a method for predicting the performance interference of applications in the virtualized environment. We develop 2 scheduling algorithms that work with the performance interference prediction to manage VM placement in virtualized environment.

In the future wok, we will use other non-linear method to analyze the relationship between the performance interference degree and the system-level workloads in order to improve the accuracy of the prediction of the performance interference. And the system-level workloads related to network I/O will be considered.

Acknowledgment

The authors would like to thank the reviewers for their detailed reviews and constructive comments, which have helped improve the quality of this paper. This work was supported in part by the National Key Technology R&D Program of the Ministry of Science and Technology (2015BAH09F02, 2015BAH47F03) and the Fundamental Research Funds for the Central Universities (N130417002, N130404011)).

References

[1] Xen Virtual Machine Moniter. (Jan. 12, 2013). [Online]. Available: http://www.xen.org

[2] P. Barham, B. Dragovic, and K. Fraser, "Xen and the art of virtualization," in *Proc. ACM Symposium on Operating Systems Principles*, New York, 2003, pp. 164-177.

[3] D. Gupta, L. Cherkasova, R. Gardner, and A Vahdat, "Enforcing Performance Isolation across virtual machines in xen," in *Proc. International Conference on Middleware*, Melbourne, 2006, pp. 342-362.

[4] Y. Koh, R. Knauerhase, and P. Brett, "An analysis of performance interference effects in virtual environments," in *Proc. International Symposium on Performance Analysis of Systems and Software*, California, 2007, 200-209.

[5] Y. D. Mei, L. Liu, and X. Pu, "Performance measurements and analysis of network i/o applications in virtualized cloud," in *Proc. International Conference on Cloud Computing*, Miami, 2010, pp. 59-66.

[6] X. Pu, L. Liu, and Y. D. Mei, "Who is your neighbor: Net I/O performance interference in virtualized clouds," *IEEE Trans on Services Computing*, vol. 6, no. 3, 314-329, 2013.

[7] R. C. Chiang and H. H. Huang, "TRACON: Interference-aware scheduling for data-intensive applications in virtualized environments," in *Proc. International Conference for High Performance Computing, Networking, Storage and Analysis*, Seattle, 2011.

[8] T. Wood, L. Cherkasova, and K. Ozonat, "Profiling and modeling resource usage of virtualized applications," in *Proc. International Conference on Middleware*, Leuven, 2008, pp. 366-387.

[9] B. R. Nikzad, T. Javid, and M. Reza, "On modelling and prediction of total CPU usage for applications in mapreduce environments," in *Proc. International Conference on Algorithms and Architectures for Parallel Processing*, Fukuoka, 2012, pp. 414-427.

[10] S. Kundu, R. Rangaswami, and K. Dutta, "Application performance modeling in a virtualized environment," in *Proc. International Conference on High Performance Computer Architecture*, Bangalore, 2010, pp. 1-10.

[11] A. A. Bankole and S. A. Ajila, "Cloud client prediction models for cloud resource provisioning in a multitier web application environment," in *Proc. Service Oriented System Engineering*, San Francisco Bay, 2013, pp. 414-427.

[12] X. Pu, L. Liu, and Y. D. Mei, "Understanding performance interference of I/O workload in virtualized cloud environments," in *Proc. International Conference on Cloud Computing*, Miami, 2010, pp. 51-58.

[13] Y. D. Mei, L. Liu, and X. Pu, "Performance analysis of network I/O workloads in virtualized data centers," *IEEE Transactions on Services Computing*, vol. 6, no. 1, pp. 48-63, 2013.

[14] N. Ripal, K. Aman, and G. Alireza, "Q-clouds: Managing performance interference effects for QoS-aware clouds," in *Proc. European Conference on Computer Systems*, Paris, 2010, pp. 237-250.

[15] T. Chen and R. Bahsoon, "Self-adaptive and sensitivity-aware QoS modeling for the cloud," in *Proc. International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, San Francisco, 2013, pp. 43-52.

[16] Y. Koh, R. Knauerhase, and P. Brett, "An analysis of performance interference effects in virtual environments," in *Proc. International Symposium on Performance Analysis of Systems & Software*, California, 2007, pp. 200-209.

[17] G. Jung, M. A. Hiltunen, and K. R. Joshi, "Mistral: Dynamic managing power, performance, and adaptation cost in cloud infrastructures," in *Proc. International Conference on Distributed Computing Systems*, Genova, 2010, pp. 62-73.

[18] N. Bobroff, A. Kochut, and K. Beaty, "Dynamic placement of virtual machines for managing SLA violations," in *Proc. International Symposium on Integrated Network Management*. Munich, 2007, pp. 119-128.

[19] A. Settle, J. Kihm, A. Janiszewski, and D. Connors, "Architectural support for enhanced SMT job scheduling," in *Proc. International Conference on Parallel Architectures and Compilation Techniques*, Alberta, 2004, pp. 63-73.

[20] M. Ghosh, R. Nathuji, and M. Lee, "Symbiotic scheduling for shared caches in multi-core systems using memory footprint signature," in *Proc. International Conference on Parallel Processing*, Kyoto, 2011, pp. 11-20

[21] P. Fortemps, "Jobshop scheduling with imprecise durations: A fuzzy approach," *IEEE Transactions on Fuzzy Systems*, vol. 5, no. 4, pp. 557-569, 1997.

[22] H. Oscar and E. K. Chul, "Heuristic algorithms for scheduling independent tasks on nonidentical processors," *Journal of the ACM*, vol. 24, no. 2, pp. 280-289, 1977.

Lei Yang, received the M.S. and Ph.D. degrees in computing application technology from Northeastern University, Shenyang, Liaoning, China, in 2004 and 2007 respectively. He has been a faculty member of college of information science and technology at Northeastern University since 2004, where he is currently an associate professor. His major research interests include cloud computing, service computing and quality management of service. Recent years, as a project leader and main researchers he has undertaken many national and province projects.

Yu Dai, received her M.S. and Ph.D degree in computing application technology from Northeastern University, Shenyang, Liaoning, China, in 2006 and 2008 respectively. She is currently an associate professor in the college of software, Northeastern University. Her research interests include cloud computing, service computing and quality management of service.