Journal of Systems Architecture 72 (2017) 42–50

Contents lists available at ScienceDirect

Journal of Systems Architecture

journal homepage: www.elsevier.com/locate/sysarc

Precise contention-aware performance prediction on virtualized multicore system

Yuxia Cheng[a,*], Wenzhi Chen[a], Zonghui Wang[a], Yang Xiang[a,b]

[a] Zhejiang University, Zheda Road 38, Xihu District, Hangzhou, China
[b] Deakin University, 221 Burwood Highway, Burwood, VIC, 3125, Australia

A R T I C L E I N F O

Article history:
Received 11 January 2016
Revised 20 February 2016
Accepted 22 June 2016
Available online 23 June 2016

Keywords:
Multicore
Virtual machine
Resource contention
Performance prediction

A B S T R A C T

Multicore systems are widely deployed in both the embedded and the high end computing infrastructures. However, traditional virtualization systems can not effectively isolate shared micro architectural resources among virtual machines (VMs) running on multicore systems. CPU and memory intensive VMs contending for these resources will lead to serious performance interference, which makes virtualization systems less efficient and VM performance less stable. In this paper, we propose a contention-aware performance prediction model on the virtualized multicore systems to quantify the performance degradation of VMs. First, we identify the performance interference factors and design synthetic micro-benchmarks to obtain VM's contention sensitivity and intensity features that are correlated with VM performance degradation. Second, based on the contention features, we build VM performance prediction model using machine learning techniques to quantify the precise levels of performance degradation. The proposed model can be used to optimize VM performance on multicore systems. Our experimental results show that the performance prediction model achieves high accuracy and the mean absolute error is 2.83%.

1. Introduction

System virtualization enables multiple virtual machines (VMs) to share underlying physical machines. Typically, multiple VMs running on the same physical multicore system, which is called VM consolidation, can improve resource utilization. As multicore systems are pervasive from the embedded to the high end computing infrastructures, it becomes increasingly important to exploit performance opportunities and improve the efficiency of virtualized multicore systems.

The performance of bare metal multicore system is growing rapidly. It becomes more and more important to efficiently utilize the multicore resources in the virtualization environment. However, VMs sharing the same physical multicore processor will contend for shared micro architectural resources as Fig. 1 shows. Due to traditional virtualization systems can not effectively isolate shared micro-architectural resources among VMs [1,2], such as shared Last Level Cache (LLC), memory controller, prefetcher and bus bandwidth et al., VMs will encounter different levels of performance degradation [3]. Shared resource contention among VMs not only decreases the overall system efficiency but also hurts the performance stability in each VM [4].

To address these problems, previous researchers proposed cache partitioning [5] and page coloring techniques [6] to prevent contention problems. While these techniques guarantee fairness among different tasks, they lack flexibility and reduce overall cache utilization [7]. Contention-aware scheduling techniques [1,8–10] are proposed to more flexibly address resource contention by co-scheduling cooperative tasks to share resources. Through effectively co-locating tasks to reduce performance bottlenecks, these scheduling techniques can improve system efficiency. But some tasks may still encounter unstable performance [11] due to the inaccurate performance prediction in the scheduling.

To more precisely manage contention problems, researchers proposed performance modeling techniques [12–15] to infer application behaviours when they are sharing physical resources. Typically, based on the performance monitoring events, the model predicts the application's performance behaviours. Then, performance optimization solutions can be applied according to the predictions. However, these modeling techniques were proposed in the non-virtualized environment that mainly focused on the shared cache utilization. Contrary to the non-virtualized environment, the virtualized environment is more complex due to applications running in different VMs are ignorant about each other. The modeling techniques [16–18] previously proposed in the virtualized environment

* Corresponding author.
E-mail address: rainytech@zju.edu.cn (Y. Cheng).

http://dx.doi.org/10.1016/j.sysarc.2016.06.006
1383-7621/© 2016 Elsevier B.V. All rights reserved.

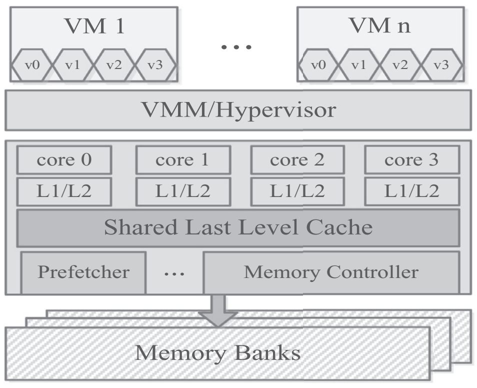


Fig. 1. The simplified architecture of multicore systems.

considered coarse grained factors, such as CPU, memory, storage and network factors. The fine grained micro-architectural level resource contention problems in the virtualized environment needs further investigation.

In this paper, we propose a contention-aware performance prediction model on the virtualized multicore systems. The main contributions of this paper are described as follows:

(1) We investigate the online performance metrics of VMs and identify the performance interference factors that contributes to the VM performance degradation. We find that the data access patterns and working set sizes are the dominant dimensions of resource contention, and then correspondingly design synthetic micro-benchmarks to stress the micro-architectural shared resources. By co-locating micro-benchmarks with real application VMs, we can quantitatively obtain the VMs contention sensitivity and intensity features. These features are correlated to the eventual performance degradation when VMs are co-located on the same multicore system.
(2) Using the obtained contention features, we build the contention-aware VM performance prediction model using machine learning algorithms. We collect the sensitivity and intensity features of applications in VMs, and record the performance degradation results of VMs with different co-location combinations. Using the collected VM's contention features as input variables and the performance degradation as output results, we train the model and build the relationship between contention features and the performance degradation. After the prediction model is built, we can use the model to quantify performance degradation of new VMs with their contention features. The experimental results show that the contention aware prediction model achieves high accuracy and the mean absolute error is 2.83%.

The rest of this paper is organised as follows: Section 2 discusses related work. Section 3 analyses VM runtime performance metrics and contention features. Section 4 presents the performance prediction model. Section 5 shows the experimental results. Section 6 concludes this paper.

2. Related work

In multicore systems, the shared last level cache is the performance critical resource. Previous researchers [12,13,19,20] proposed methods to analyse the shared cache usage in multicore systems. Hardware performance monitoring techniques [21–23] are used to capture program cache behaviours. Tam et al. [24] proposed a technique to estimate shared cache occupation for each core using online cache miss rate curve detection. West et al. [14] proposed hardware performance monitoring based method to predict cache usage for each thread. Based on the cache usage estimation, researchers proposed hardware partitioning [5,25] and page colouring [6,26] techniques to optimize shared cache contention problems in multicore systems.

Tang et al. [11] analysed the performance impact of co-locating large scale datacenter applications due to resource sharing, and showed that there exists both positive and negative impacts of co-locating as application behaviour changes. Mars et al. [27,28] proposed a mechanism named Bubble-up to infer the performance degradation due to co-locating multiple applications on a single multicore server. Zhang et al. [15] and Eyerman and Eeckhout [29] proposed performance models in the multi-thread multicore systems to more precisely co-schedule appropriate tasks. These techniques were proposed in the non-virtualised environment. Contrary to the non-virtualised environment, the virtualised environment has two major differences. One is that there are various kinds of applications running in the virtualised platform, which needs a more comprehensive method to quantify the performance interference. The other is that the applications running in the VMs are ignorant about each other, which needs a smarter scheduler to detect the contention problem.

In virtualised environments, Rao et al. [1] proposed a VCPU migration algorithm to optimize resource contention problems in NUMA (Non-Uniform Memory Access) multicore systems. Liu and Li [2] proposed a NUMA overhead aware hypervisor memory management policy. They introduced a method to estimate the memory zone access overhead using hardware performance counters. Based on the estimation, they proposed two optimization techniques: a NUMA overhead aware buddy allocator and a P2M swap FIFO. Lee and Schwan [30] proposed a region-based scheduling algorithm to manage shared cache resources in multicore platform. Their approach put emphasis on cache/memory-centric scheduling rather than CPU-centric load balancing due to the increasing importance of cache and memory structures to the system performance. However, these techniques only alleviate the interference problems, but can not predict the precise performance degradation due to resource contention.

The VM performance modeling techniques [3,16–18] were proposed to more precisely manage resources in the system. Govindan et al. [3] proposed a method of estimating the cache usage of applications through active probing with the synthetic cache loader benchmark, and uses the cache usage estimation to predict performance degradation of applications upon consolidation with other applications. Kundu et al. [16,17] proposed a method of modeling the performance of VM-hosted applications as a function of resources allocated to the VM and the I/O resource contention it experiences. Chiang and Huang [18] proposed the I/O interference aware scheduling for data-intensive applications in virtualized environment. They used the non-linear models to capture the bursty I/O patterns in data-intensive applications and incorporated the model into the VM scheduling systems. Unlike these modeling techniques, our proposed contention-aware prediction model considers more comprehensive and fine grained micro architectural resources of multicore virtualized systems. In our proposed method, the application contention features only needs to be collected once, then the model can easily utilise these features to quantify the performance degradation of various application combinations with machine learning algorithms.

3. VM runtime analysis

In this section, we describe the method of collecting VM runtime performance metrics and analyse resource contention factors that contribute to VM performance degradation.

3.1. Performance metrics

VMs simultaneously running on the same multicore systems will contend for shared resources, and a single VM running alone has no resource contention from other VMs. Therefore, we use Eq. (1) to measure the level of performance degradation caused by resource contention when two VMs are co-located on the same multicore processor.

$$PD^{A}_{co-run/B} = \frac{P^{A}_{co-run/B} - P^{A}_{alone}}{P^{A}_{alone}} \tag{1}$$

where $PD^{A}_{co-run/B}$ represents the performance degradation of VM A when it is co-running with VM B, $P^{A}_{co-run/B}$ represents the actual performance of VM A when it is co-running with VM B, P^{A}_{alone}represents the performance of VM A when it is running alone.

The shared resource contention in multicore systems are mainly caused by CPU and memory intensive workloads. The performance of these workloads can be measured online using the Cycles Per Instruction (CPI) metric [31,32]. Through hardware performance monitoring counters (PMC) [22], we can monitor each VM's CPI with low overhead. Therefore, Eq. (1) can be written as Eq. (2).

$$PD^{A}_{co-run/B} = \frac{CPI^{A}_{co-run/B} - CPI^{A}_{alone}}{CPI^{A}_{alone}} \tag{2}$$

where $CPI^{A}_{co-run/B}$ is the CPI of VM A when it is co-running with VM B, CPI^{A}_{alone} is the CPI of VM A when it is running alone. The CPI of a VM is calculated by the average CPI of all vCPUs in the VM.

3.2. Resource contention features

To characterise the VM's contention features that cause the performance degradation, we design several synthetic micro-benchmarks to co-run with VMs. Previous research [33] has shown that data access patterns and working set sizes are the dominant factors that contribute to the performance degradation caused by resource contention in multicore systems. Therefore, the micro-benchmarks are designed according to the data access patterns and working set sizes. Random and sequential accesses have large difference in the usage of cache and prefetch units. Read and write data accesses are through separate ports and channels in the multicore systems. The application's performance differs when its working set fits in the shared cache or resides in the main memory.

We design micro-benchmarks with three major dimensions of Random/Sequen-tial, Read/Write, and working set sizes. Then, we use the following micro-benchmarks to characterise different resource contention features. (1) Cache Sequential Read (CSR); (2) Cache Random Read (CRR); (3) Cache Sequential Write (CSW); (4) Cache Random Write (CRW); (5) Memory Sequential Read (MSR); (6) Memory Random Read (MRR); (7) Memory Sequential Write (MSW); (8) Memory Random Write (MRW).

The eight micro-benchmarks are carefully designed to maximise the resource contention in terms of their respective resource utilization dimensions. These micro-benchmarks generate approximately linear interference to the corresponding resource dimensions with the increase of intensity. Based on this property, previous studies [15,27,34] have demonstrated that we can sample only two intensity points to reduce the feature profiling overhead. Cache/Memory represents the micro-benchmark's working set size that fits in the cache or resides in the main memory. As the experiment section shows, the eight micro-benchmarks are sufficient to capture the required contention features that help the model to obtain precise prediction.

For the observed VM A, the performance interference caused by contention can be two folds. One is contention sensitivity, the other is contention intensity. The VM A's contention sensitivity is the performance degradation of VM A that is caused by other VMs' resource contention. Conversely, the VM A's contention intensity is the performance degradation of other VMs that is caused by the resource contention of VM A. We use Eq. (3) and (4) to quantify the contention sensitivity and intensity features of the VM when it is co-running with different micro-benchmarks.

Eq. (3) shows the VM Contention Sensitivity (VCS).

$$VCS^{A}_{bench_i} = \frac{CPI^{A}_{co-run/bench_i} - CPI^{A}_{alone}}{CPI^{A}_{alone}} \tag{3}$$

where $VCS^{A}_{bench_i}$ is the contention sensitivity of VM A when it is co-running with $bench_i$ (the $bench_i$ is one of the eight micro-benchmarks), $CPI^{A}_{co-run/bench_i}$ is the CPI of VM A when it is co-running with $bench_i$, CPI^{A}_{alone} is the CPI of VM A when it is running alone.

Eq. (4) shows the VM Contention Intensity (VCI).

$$VCI^{A}_{bench_i} = \frac{CPI^{bench_i}_{co-run/A} - CPI^{bench_i}_{alone}}{CPI^{bench_i}_{alone}} \tag{4}$$

where $VCI^{A}_{bench_i}$ is the contention intensity of VM A when it is co-running with $bench_i$, $CPI^{bench_i}_{co-run/A}$ is the CPI of $bench_i$ when it is co-running with VM A, $CPI^{bench_i}_{alone}$ is the CPI of $bench_i$ when it is running alone.

By co-running with the micro-benchmarks, we collect a set of contention features of the VM. These features are the dominant factors that contribute to the performance degradation due to resource contention. The following section describes how to leverage these features in the model to predict performance degradation.

4. Contention-aware performance prediction model

We propose the contention-aware performance prediction model based on the VM runtime features of contention sensitivity and intensity described in the previous section. In our deployment scenario, we assume that applications deployed in the VMs are typically long running, CPU and memory intensive workloads. The contention-aware performance prediction model mainly consists of two parts:

(1) VM performance training module is responsible for collecting new VMs performance features. The training features include the VCS and VCI of VMs collected in the training multicore system. These features are used in the prediction model to predict performance degradation of co-located VMs.
(2) VM performance online prediction module is used to predict the performance degradation of co-located VMs and to indicate which co-location combinations can be applied. The model regards VCS&VCI features as inputs and outputs the predicted performance degradation. The prediction results can be used for contention-aware VM scheduling. The VM online performance events are periodically monitored to update the prediction model when the prediction error is larger than a predefined threshold.

4.1. VM training

Applications deployed in the VMs are typically long running services. Before new VMs are deployed onto the physical production system, we collect the VM's runtime features on the training system. The runtime features are the VM's VCS and VCI metrics described in Section 3.2.

During the model training phase, we collect both VM's contention features and the actual performance degradation when two VMs are co-located onto the same multicore processor. The VM's contention features are obtained via co-locating the new VM with different micro-benchmarks respectively as Eq. (3) and 4 show. The actual performance degradation of VMs are obtained via co-locating two application VMs onto the same multicore processor as Eq. (2) shows.

By running the VM alone, running the VM with micro-benchmarks, and running the VM with other VMs, we can collect the contention features and performance degradation results as training data to build our prediction model.

$$[VCS^{cg}_{micro_1}, VCI^{sp}_{micro_1}, \ldots, VCS^{cg}_{micro_8}, VCI^{sp}_{micro_8}, PD^{cg}_{co-run/sp}] \tag{5}$$

$$[VCS^{sp}_{micro_1}, VCI^{cg}_{micro_1}, \ldots, VCS^{sp}_{micro_8}, VCI^{cg}_{micro_8}, PD^{sp}_{co-run/cg}] \tag{6}$$

Eq. (5) and (6) present two examples of training data records that are used to build the prediction model. Eq. (5) shows the training data of using VM contention sensitivity of *cg* and VM contention intensity of *sp* to predict the performance degradation of VM *cg* when it is co-running with VM *sp*. Similarly, Eq. (6) shows the training data of performance degradation of VM *sp* when it is co-running with VM *cg*. *cg* and *sp* are two benchmarks in the NPB benchmark suite. After the prediction model is built, the model can use the contention features of VM A and VM B as input to predict the performance degradation of VM A ($PD^A_{co-run/B}$) and VM B ($PD^B_{co-run/A}$) respectively.

4.2. Online prediction model

In the prediction model, we use machine learning algorithms to build the relationships between VMs' contention features and the performance degradation. Eq. (7) shows the input variables and the output result in the prediction model. When VM A is co-running with VM B in the same multicore system, the equation outputs the predicted performance degradation of VM A ($\widetilde{PD}^A_{co-run/B}$), using the contention sensitivity features of VM A and the contention intensity features of VM B as inputs.

$$\widetilde{PD}^A_{co-run/B} = ML(VCS^A_{micro_1}, VCI^B_{micro_1}, \ldots, VCS^A_{micro_n}, VCI^B_{micro_n}) \tag{7}$$

The ML in Eq. (7) denotes machine learning algorithms. In our prediction model, we can utilize multiple machine learning algorithms once the input variables and output results are properly defined. For example, Eq. (8) shows the linear regression model.

$$\widetilde{PD}^A_{co-run/B} = \sum_{i=1}^{N} \left(a_i VCS^A_{micro_i} + b_i VCI^B_{micro_i}\right) + c \tag{8}$$

In the linear model, the performance degradation of VM A is proportional to each dimension of VM A's contention sensitivity and VM B's contention intensity. The weights that each dimension of A's sensitivity and B's intensity contribute to the overall performance degradation are determined by the coefficient a_i and b_i. The linear model assumes that VM A's performance degradation from each dimension is additive. However, different dimensions of contention features may be overlapped and interact with each other in the real system, we rely on the regression algorithm to alleviate this effect.

In the next Section 5, we will use other advanced non-linear algorithms and the experiment results show the improved prediction accuracy. For example, in the regression tree model, the relationships between VM's contention features and the performance degradation are learned through information gain theory to address the complex interaction problems.

Table 1
Multicore systems configurations.

System models	Dell R710	Dell R910
Processor type	Intel Xeon E5620	Intel Xeon E7520
Num. of cores	4 cores (2 sockets)	4 cores (4 sockets)
Shared cache	12 MB	18 MB
Clock frequency	2.4 GHz	1.87 GHz
Memory	32 GB	64 GB

4.3. Discussion

In the previous section, we present the prediction model based on the two VMs contention for the ease of description. In the multi-VM scenario, we can extend the model by regarding VM B as the sum of the rest VMs (denoted as VM_{mix}) in the system. To obtain the VCS and VCI features of VM_{mix}, we use the micro-benchmarks to co-run with VM_{mix} online and monitor their CPIs respectively. We calculate the micro-benchmark's performance degradation as the VCI of VM_{mix}. For the VCS of VM_{mix}, we select the VM among VM_{mix} that has the largest performance degradation as the VCS of VM_{mix}, so that we can guarantee the new VM will not violate the QoS of all VMs in the VM_{mix}. In this way, we can obtain the VCS and VCI features of VM_{mix}, and use them as input variables of the model to predict performance degradation as described in Section 4.2.

The heterogeneous multicore systems are commonly seen. We build the prediction model on the same type of multicore systems. When it comes to the heterogeneous multicore systems, we have to build the model for each type of the multicore system using the same procedure described in our framework. As the contention aware performance model runs for a long period of time, we can accumulate more and more new VMs' contention statistics and periodically update the model to refine the prediction accuracy.

In our deployment scenario, we assume that applications in the VMs are long running services and their workloads change infrequently. To address the workload phase change problem, we continuously monitor the CPI of VMs. When the performance monitor observes the CPI of a VM has drastic changes for a predefined period, the model will update the contention features of the corresponding VM. The online performance monitoring and phase change detection methods can be found in [28].

As we focused on the multicore resource contention problems, the I/O and network contention problems are beyond the scope of this paper.

5. Performance evaluation

We evaluate the proposed contention-aware performance prediction model on two types of multicore systems summarised in Table 1. VMs run on the qemu-kvm-1.0 virtualized platform. Both the host and guest operating systems used in the experiments are ubuntu 12.04 with the Linux kernel version 3.8.0–35. Each VM is configured with 4 VCPUs and 8 GB memory. We use the following benchmarks to run in VMs.

(1) NPB. The NAS Parallel Benchmark (NPB) suite [35] is a set of benchmarks developed for evaluating the performance of parallel systems. The NPB benchmark suite consists of 5 parallel kernels and 3 simulated application benchmarks.
(2) SPEC CPU 2006. The SPEC CPU 2006 [36] is an industry-standardized, CPU and memory intensive benchmark suite. The benchmarks mainly test a system's processor and memory subsystem resources.

We collect training and testing data sets to train the prediction model. In the experiment, we use NPB-OMP and NPB-MPI

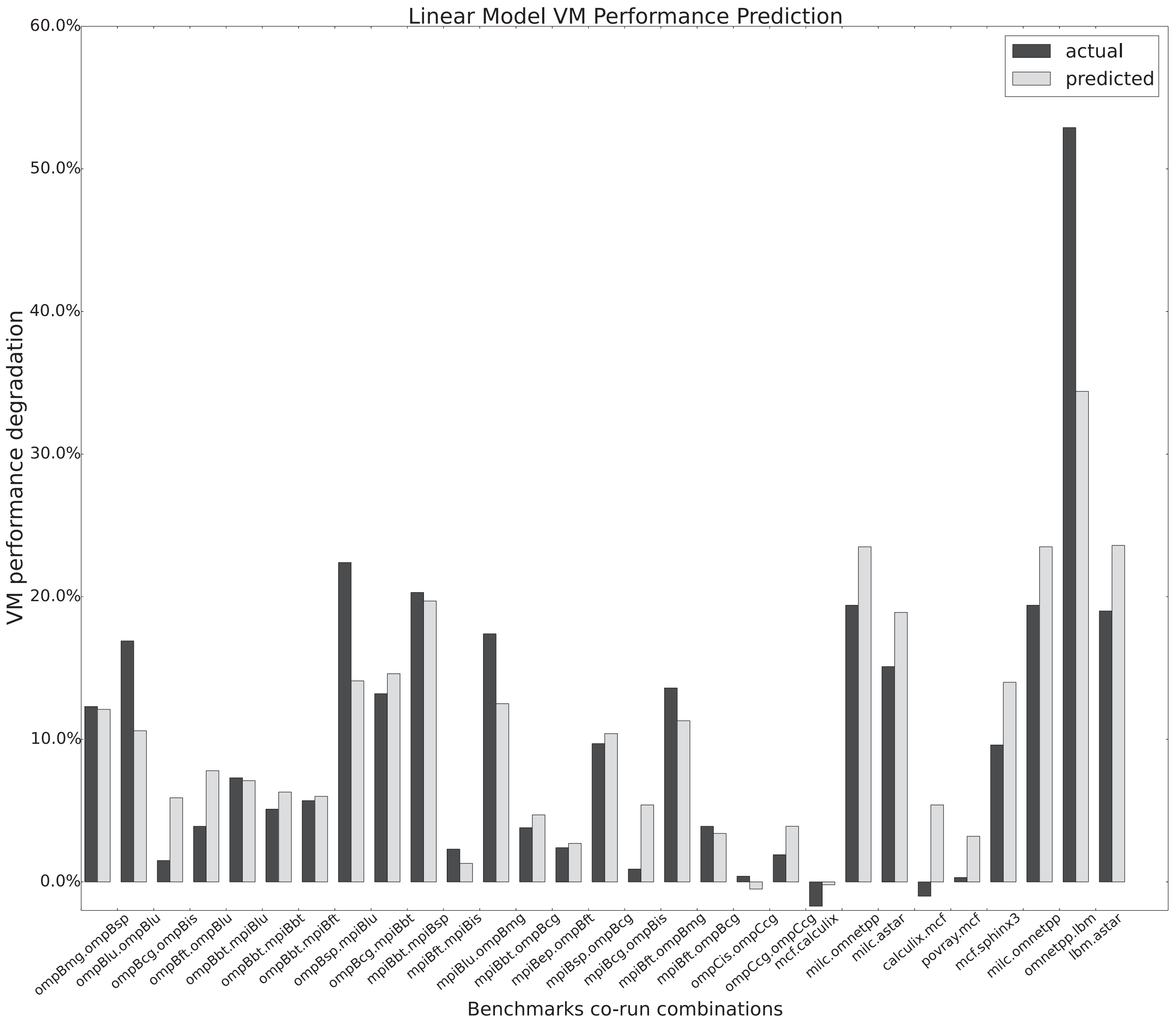


Fig. 2. The comparisons between the actual performance degradation and the predicted performance degradation under the linear regression model.

benchmarks with B and C classes, and use SPEC CPU 2006 benchmark with *ref* inputs as our workloads. Workloads are running inside VMs. VMs are pair-wise co-located onto the multicore processors. During the offline training phase, we co-run 600 combinations of different workloads contending for shared resources and collect the corresponding performance data. The collected data format is regularised according to the weka *arff* file [37]. Each data record contains VM's VCS and VCI features as well as the actual performance degradation results. We use the 10-fold cross-validation method [38] to evaluate the prediction model.

5.1. Prediction accuracy

In the evaluation, we randomly choose 90% of the collected data as the training data set and 10% of the collected data as the testing data set. The prediction model is built using the training data. Then, using VM's VCS and VCI features in the testing data, the model outputs the predicted performance degradation. We compare the model predicted results with the actual performance degradation recorded in the testing data.

Fig. 2 presents the comparison of the actual performance degradation results and the predicted results under the linear regression model. The x axis shows different benchmarks co-run combinations. The y axis is the VM's performance degradation. For example, the first combination in the x axis ompBmg.ompBsp represents the OMP B class benchmark *mg* is co-running with the OMP B class benchmark *sp* in the same multicore system. The corresponding y axis shows that the predicted and actual performance degradation of the benchmark *mg* are 12.1% and 12.3% respectively. From the comparisons, we observe that in most cases the linear model predicts the performance degradation very close to the actual results. However, in some cases, there exist a relatively large gap between the predicted and actual results. The reason is that the linear model is insufficient to explore the non-linear parts in the complex interactions among micro-architectural resources.

In our proposed framework, we can replace the linear regression algorithm with other machine learning algorithms. Fig. 3

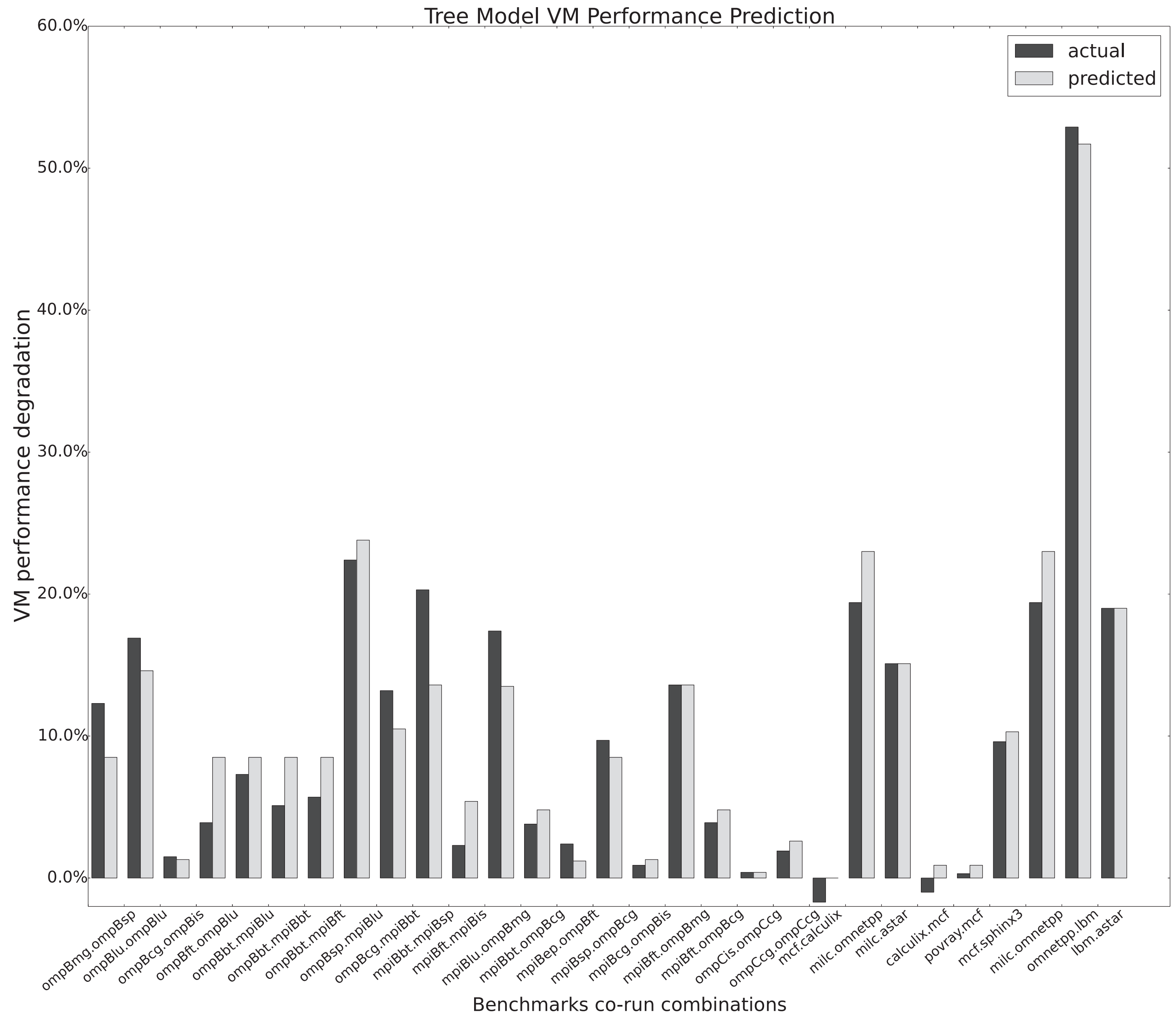


Fig. 3. The comparisons between the actual performance degradation and the predicted performance degradation under the REPTree model.

presents the actual and predicted results under the REPTree model. The REPTree algorithm is one of the decision tree implementations in the Weka tool [37]. The decision tree uses the divide and conquer strategy to learn a set of rules to build the relationships between the input variables and the output result. Unlike the linear model, the tree model can more effectively capture the complex non-linear correlations. As shown in Fig. 3, the gap between the predicted and actual results under the REPTree model is relatively smaller than that under the linear model.

To compare the prediction accuracy between two models, we use the mean absolute error metric (MAE) to get a more comprehensive overview about the prediction results. Eq. (9) shows the method of calculating the mean absolute error, where p_i represents the predicted performance degradation of VM_i, and a_i represents the actual performance degradation of VM_i.

$$\mathrm{MAE} = \frac{|p_1 - a_1| + \ldots + |p_n - a_n|}{n} \tag{9}$$

Fig. 4 shows the MAE comparisons between the linear model and the REPTree model. The x axis shows the benchmark that co-runs with other benchmarks. The y axis shows the prediction mean absolute error. For example, ompBmg in the x axis represents the OMP class B benchmark *mg* co-runs with other benchmarks. We record the predicted and actual performance degradation of *mg* in each run and calculate the MAE using Eq. (9).

As shown in Fig. 4, most benchmarks have smaller MAE when using the REPTree model than using the linear model. In general, the REPTree model has higher prediction accuracy than the linear model. The reason is that the REPTree model can more effectively dig out the non-linear portions of the relationships between shared resource contention and the performance degradation of VMs. While the linear model is based on the assumption that the contention interference factors and the performance degradation have linear correlations, other non-linear interactions that contribute to the performance is not captured in the linear model.

Table 2 shows five classic machine learning algorithms that are used in our proposed model framework. To compare the performance of these algorithms, we present four metrics that

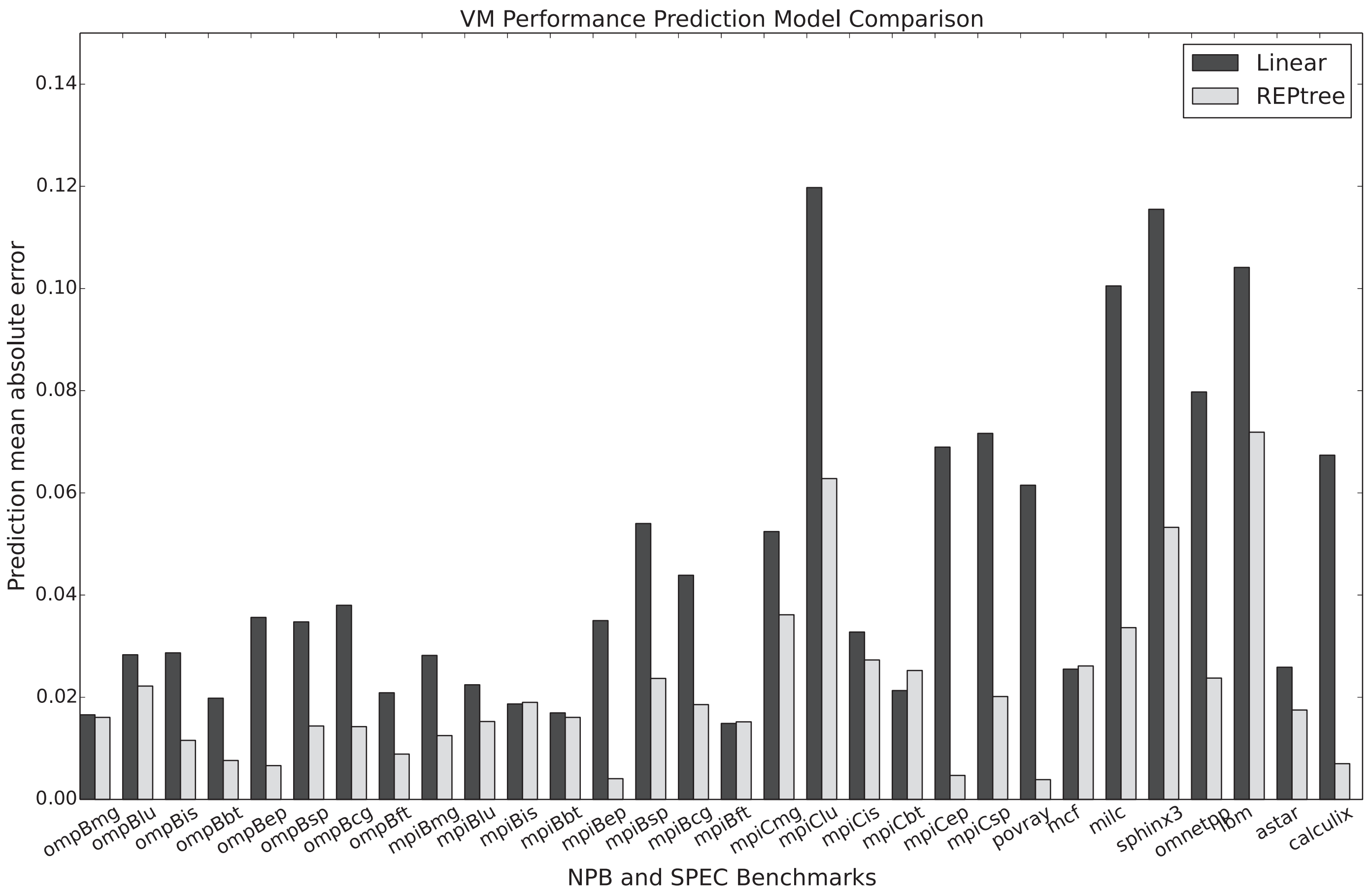


Fig. 4. The prediction mean absolute error (MAE) comparisons of the Linear regression model and the REPTree model.

Table 2
Summary of machine learning algorithms.

Algorithms	Correlation coefficient	MAE	RMSE	RRSE
Linear	0.7939	0.0480	0.0708	0.6054
REPTree	0.8133	0.0357	0.0679	0.5806
M5P	0.8526	0.0344	0.0609	0.5204
Bagging	0.8799	0.0283	0.0557	0.4760
Neural network	0.8309	0.0382	0.0724	0.6189

are commonly used in evaluation: Correlation coefficient, Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), and Root Relative Squared Error (RRSE). The Linear and REPTree algorithms have discussed in previous section. The rest three algorithms are the M5P, Bagging, and Neural Networking.

The M5P algorithm is a classic decision tree model, and is typically used in the numeric prediction scenarios. The Bagging algorithm is a strengthen version of the REPTree algorithm. The Bagging algorithm consists of multiple REPTrees, and the prediction result is the mean value of multiple REPTree output results. The Neural Networking algorithm is a popular machine learning algorithm in processing big data, and it can be used in the non-linear relationships of numeric prediction.

As shown in Table 2, all five algorithms obtain acceptable prediction accuracy. This demonstrates that the VM's contention sensitivity (VCS) and intensity (VCI) features are the dominant factors that contribute to the VM's performance degradation. Therefore, using advanced machine learning algorithms with VM's VCS and VCI features as inputs, we can achieve desirable prediction accuracy. The Bagging algorithm shows the best prediction results due to its synthetic property.

5.2. Overhead analysis

The extra overhead of the contention aware VM performance prediction model mainly consists of 4 parts: (1) the VM training overhead; (2) the model building overhead; (3) the online performance monitoring overhead; (4) the prediction calculation overhead. Once the training data is collected, the model building and the prediction calculation overhead is negligible. For example, in the REPTree model, the algorithm uses the divide and conquer techniques to learn the rules. The time complexity of building the REPTree model is $O(n|D|log(|D|))$, where n is the number of features in the training set D, $|D|$ is the number of training samples. The models used in the experiment can be built within tens of milliseconds on our experimental platforms. After the model is built, the prediction calculation only needs small constant CPU time.

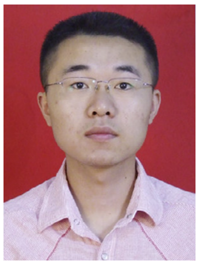
Due to the performance monitor is periodically running online, we keep the monitoring overhead under 0.5% CPU consumption by using the low overhead hardware performance monitoring counters. The VM training requires co-running the VM with micro-benchmarks for tens of seconds. We use the micro-benchmarks to capture the contention features of the VM which reduces the exhaustive co-running combinations with other VMs. In terms of the VM's long running service time, the training overhead is worthwhile to improve the system efficiency. To further reduce the training overhead, we save the VM's contention features. When the same type of new VMs need to be deployed, the contention features can be used without repeated training process.

6. Conclusions and future work

In this paper, we present the contention-aware VM performance prediction model on the virtualized multicore systems. Based on the analysis of VM performance degradation caused by shared resource contention, we find that VM's data access patterns and working set sizes are dominant factors that interfere with the behaviour of VM performance. Therefore, we design synthetic micro-benchmarks to obtain VM's contention sensitivity and intensity features. These features are well captured the cause of performance degradation due to shared resource contention. Then, based on these features, we build the contention-aware VM performance prediction model using machine learning techniques. The precise performance prediction capability makes possible of the more effective contention-aware VM scheduling algorithm design. The experimental results show that the proposed contention aware VM performance prediction model achieves high accuracy and the mean absolute error is 2.83%.

References

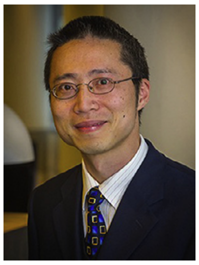
[1] J. Rao, K. Wang, X. Zhou, C.-Z. Xu, Optimizing virtual machine scheduling in numa multicore systems, in: High Performance Computer Architecture (HPCA), 2013 IEEE 19th International Symposium on, IEEE, 2013, pp. 306–317.
[2] M. Liu, T. Li, Optimizing virtual machine consolidation performance on NUMA server architecture for cloud workloads, in: Computer Architecture (ISCA), 2014 ACM/IEEE 41st International Symposium on, IEEE, 2014, pp. 325–336.
[3] S. Govindan, J. Liu, A. Kansal, A. Sivasubramaniam, Cuanta: quantifying effects of shared on-chip resource interference for consolidated virtual machines, in: Proceedings of the 2nd ACM Symposium on Cloud Computing, ACM, 2011, p. 22.
[4] R. Nathuji, A. Kansal, A. Ghaffarkhah, Q-clouds: managing performance interference effects for QoS-aware clouds, in: Proceedings of the 5th European Conference on Computer Systems, ACM, 2010, pp. 237–250.
[5] F. Liu, Y. Solihin, Studying the impact of hardware prefetching and bandwidth partitioning in chip-multiprocessors, in: Proceedings of the ACM SIGMETRICS joint international conference on Measurement and modeling of computer systems, ACM, 2011, pp. 37–48.
[6] X. Zhang, S. Dwarkadas, K. Shen, Towards practical page coloring-based multicore cache management, in: Proceedings of the 4th ACM European Conference on Computer Systems, ACM, 2009, pp. 89–102.
[7] S. Zhuravlev, J.C. Saez, S. Blagodurov, A. Fedorova, M. Prieto, Survey of scheduling techniques for addressing shared resources in multicore processors, ACM Comput. Surv. (CSUR) 45 (1) (2012) 4.
[8] S. Blagodurov, S. Zhuravlev, M. Dashti, A. Fedorova, A Case for NUMA-aware Contention Management on Multicore Systems, in: USENIX Annual Technical Conference, USENIX, 2011.
[9] P. Radojković, V. Čakarević, M. Moretó, J. Verdú, A. Pajuelo, F.J. Cazorla, M. Nemirovsky, M. Valero, Optimal task assignment in multithreaded processors: a statistical approach, ACM SIGARCH Comput. Arch. News 40 (1) (2012) 235–248.
[10] S. Zhuravlev, S. Blagodurov, A. Fedorova, Addressing shared resource contention in multicore processors via scheduling, in: ACM SIGARCH Computer Architecture News, ACM, 2010, pp. 129–142.
[11] L. Tang, J. Mars, N. Vachharajani, R. Hundt, M.L. Soffa, The impact of memory subsystem resource sharing on datacenter applications, in: Computer Architecture (ISCA), 2011 38th Annual International Symposium on, IEEE, 2011, pp. 283–294.
[12] T. Dey, W. Wang, J.W. Davidson, M.L. Soffa, Characterizing multi-threaded applications based on shared-resource contention, in: Performance Analysis of Systems and Software (ISPASS), 2011 IEEE International Symposium on, IEEE, 2011, pp. 76–86.
[13] A. Sandberg, A. Sembrant, E. Hagersten, D. Black-Schaffer, Modeling performance variation due to cache sharing, High Perform. Comput. Arch. (HPCA), 2013 IEEE 19th Int. Symp. 1 (2013) 155–166.
[14] R. West, P. Zaroo, C.A. Waldspurger, X. Zhang, Online cache modeling for commodity multicore processors, ACM SIGOPS Oper. Syst. Rev. 44 (4) (2010) 19–29.
[15] Y. Zhang, M.A. Laurenzano, J. Mars, L. Tang, SMiTe: precise QoS prediction on real-system SMT processors to improve utilization in warehouse scale computers, in: Microarchitecture (MICRO), 2014 47th Annual IEEE/ACM International Symposium on, IEEE, 2014, pp. 406–418.
[16] S. Kundu, R. Rangaswami, K. Dutta, M. Zhao, Application performance modeling in a virtualized environment, in: High Performance Computer Architecture (HPCA), 2010 IEEE 16th International Symposium on, IEEE, 2010, pp. 1–10.
[17] S. Kundu, R. Rangaswami, A. Gulati, M. Zhao, K. Dutta, Modeling virtualized applications using machine learning techniques, in: Proceedings of the 8th ACM SIGPLAN/SIGOPS Conference on Virtual Execution Environments, ACM, 2012, pp. 3–14.
[18] R.C. Chiang, H.H. Huang, TRACON: interference-aware schedulingfor data-intensive applicationsin virtualized environments, Parallel Distributed Syst. IEEE Trans. 25 (5) (2014) 1349–1358.
[19] T. Dwyer, A. Fedorova, S. Blagodurov, M. Roth, F. Gaud, J. Pei, A practical method for estimating performance degradation on multicore processors, and its application to HPC workloads, in: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis, IEEE, 2012, p. 83.
[20] C. Xu, X. Chen, R.P. Dick, Z.M. Mao, Cache contention and application performance prediction for multi-core systems, in: Performance Analysis of Systems & Software (ISPASS), 2010 IEEE International Symposium on, 2010, pp. 76–86.
[21] L. Zhao, R. Iyer, R. Illikkal, J. Moses, S. Makineni, D. Newell, CacheScouts: fine–grain monitoring of shared caches in CMP platforms, in: Proceedings of the 16th International Conference on Parallel Architecture and Compilation Techniques, IEEE, 2007, pp. 339–352.
[22] R. Azimi, D.K. Tam, L. Soares, M. Stumm, Enhancing operating system support for multicore processors by using hardware performance monitoring, ACM SIGOPS Oper. Syst. Rev. 43 (2) (2009) 56–65.
[23] A. Yasin, A top-down method for performance analysis and counters architecture, in: Performance Analysis of Systems and Software (ISPASS), 2014 IEEE International Symposium on, IEEE, 2014, pp. 35–44.
[24] D.K. Tam, R. Azimi, L.B. Soares, M. Stumm, RapidMRC: approximating L2 miss rate curves on commodity systems for online optimizations, in: ACM SIGARCH Computer Architecture News, ACM, 2009, pp. 121–132.
[25] E. Ebrahimi, C.J. Lee, O. Mutlu, Y.N. Patt, Fairness via source throttling: a configurable and high-performance fairness substrate for multicore memory systems, ACM Trans. Comput. Syst. (TOCS) 30 (2) (2012) 7.
[26] X. Ding, K. Wang, X. Zhang, ULCC: a user-level facility for optimizing shared cache performance on multicores, in: Proceedings of the 16th ACM Symposium on Principles and Practice of Parallel Programming, ACM, 2011, pp. 103–112.
[27] J. Mars, L. Tang, K. Skadron, M.L. Soffa, R. Hundt, Increasing utilization in modern warehouse-scale computers using bubble-up, IEEE Micro 32 (3) (2012) 88–99.
[28] H. Yang, A. Breslow, J. Mars, L. Tang, Bubble-flux: precise online QoS management for increased utilization in warehouse scale computers, in: Proceedings of the 40th Annual International Symposium on Computer Architecture, ACM, 2013, pp. 607–618.
[29] S. Eyerman, L. Eeckhout, Probabilistic job symbiosis modeling for SMT processor scheduling, in: Proceedings of the 15th International Conference on Architectural Support for Programming Languages and Operating Systems, ACM, 2010, pp. 91–102.
[30] M. Lee, K. Schwan, Region scheduling: efficiently using the cache architectures via page-level affinity, ACM SIGARCH Comput. Arch. News 40 (1) (2012) 451–462.
[31] G. Ren, E. Tune, T. Moseley, Y. Shi, S. Rus, R. Hundt, Google-wide profiling: a continuous profiling infrastructure for data centers, IEEE micro 30 (4) (2010) 65–79.
[32] X. Zhang, E. Tune, R. Hagmann, R. Jnagal, V. Gokhale, J. Wilkes, CPI2: CPU performance isolation for shared compute clusters, in: Proceedings of the 8th ACM European Conference on Computer Systems, ACM, 2013, pp. 379–391.
[33] C. Delimitrou, C. Kozyrakis, iBench: quantifying interference for datacenter applications, in: Workload Characterization (IISWC), 2013 IEEE International Symposium on, IEEE, 2013a, pp. 23–33.
[34] C. Delimitrou, C. Kozyrakis, Paragon: QoS-aware scheduling for heterogeneous datacenters, in: Proceedings of the 18th International Conference on Architectural Support for Programming Languages and Operating systems, ACM, 2013b, pp. 77–88.
[35] The NAS Parallel Benchmarks. http://www.nas.nasa.gov/publications/npb.html. [online].
[36] Spec cpu 2006. http://www.spec.org/cpu2006/. [online].
[37] Weka 3. http://www.cs.waikato.ac.nz/ml/weka/. [online].
[38] Cross-validation. https://en.wikipedia.org/wiki/cross-validation-(statistics). [online].



Yuxia Cheng received the Ph.D. degree in computer science and technology from Zhejiang University, Hangzhou, China, in 2015. He is currently an associate research fellow at the School of Information Technology, Deakin University. His current research interests include multicore architecture, operating systems, virtualization and security.

Wenzhi Chen was born in 1969. He received the Ph.D. degree from Zhejiang University, Hangzhou, China. He is currently a Professor and a Ph.D. Supervisor with the College of Computer Science and Technology, Zhejiang University. His areas of research include computer graphics, computer architecture, system software, embedded systems, and security.

Zonghui Wang was born in March 1979. He received the Ph.D. degree from the College of Computer Science and Technology, Zhejiang University, Hangzhou, China, in 2007. He is a Lecturer with the College of Computer Science and Engineering, Zhejiang University. His research interests focus on cloud computing, distributed systems, computer architecture, and computer graphics.



Yang Xiang received his Ph.D. in Computer Science from Deakin University, Australia. He is currently a Full Professor at the School of Information Technology, Deakin University. He is the Director of the Network Security and Computing Lab (NSCLab) and the Associate Head of School (Industry Engagement). His research interests include network and system security, distributed systems, and networking. In particular, he is currently leading his team developing active defense systems against largescale distributed network attacks. He is the Chief Investigator of several projects in network and system security, funded by the Australian Research Council (ARC). He has published more than 180 research papers in many international journals and conferences, such as IEEE Transactions on Computers, IEEE Transactions on Parallel and Distributed Systems, IEEE Transactions on Information Security and Forensics, and IEEE Journal on Selected Areas in Communications.