Future Generation Computer Systems ▮ (▮▮▮▮) ▮▮▮–▮▮▮

Contents lists available at ScienceDirect

Future Generation Computer Systems

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

Smart VM co-scheduling with the precise prediction of performance characteristics

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

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

HIGHLIGHTS

- Identify the performance interference factors between contention VMs.
- Build VM performance prediction model to quantify the precise levels of performance degradation.
- Design contention-aware VM scheduling algorithms to improve system efficiency and guarantee the QoS of VMs.

ARTICLE INFO

Article history:
Received 9 August 2016
Received in revised form
1 November 2016
Accepted 21 November 2016
Available online xxxx

Keywords:
Virtual machine
Shared resource contention
Performance prediction
VM co-location

ABSTRACT

Traditional virtualization systems cannot effectively isolate the shared micro-architectural resources among VMs. Different types of CPU and memory-intensive VMs contending for these shared resources will lead to different levels of performance degradation, which decreases the system efficiency and Quality of Service (QoS) in the cloud. To address these problems, we design and implement a smart VM co-scheduling system with precise prediction of performance characteristics. First, we identify the performance interference factors and design synthetic micro-benchmarks. By co-running these micro-benchmarks with VMs, we decouple two kinds of VM performance characteristics: VM contention sensitivity and contention intensity. Second, based on the characteristics, we build VM performance prediction model using machine learning techniques to quantify the precise levels of performance degradation. By co-running large numbers of different VMs and collecting their performance scores, we train a robust performance prediction model. Finally, based on the prediction model, we design contention aware VM scheduling algorithms to improve system efficiency and guarantee the QoS of VMs in the cloud. Our experimental results show that the performance prediction model achieves high accuracy and the smart VM scheduling algorithms based on the prediction improves system efficiency and VM performance stability.

1. Introduction

System virtualization is the fundamental technique in the Infrastructure as a Service (IaaS) cloud computing paradigm. The virtualization software enables multiple virtual machines (VMs) to share underlying physical machines. Typically, multiple VMs running on the same physical server, which is called VM consolidation, can improve resource utilization in cloud data centers. As cloud computing is pervasive, it becomes increasingly important to exploit performance opportunities and improve the efficiency of cloud platform.

However, the widely used commodity hypervisors (Xen, KVM, and VMware ESX etc.) cannot effectively provide performance isolation between VMs [1,2]. In the typical SMP (Symmetric Multi-Processor) server, VMs will contend for the underlying shared micro-architectural resources as shown in Fig. 1. The shared micro-architectural resources, including shared Last Level Cache (LLC), integrated memory controller, prefetcher, and bus bandwidth etc., are critical for the overall system performance [3]. Therefore, different types of CPU and memory-intensive VMs contending for these resources will lead to different levels of performance degradation, which decreases the system efficiency and Quality of Service (QoS) in the cloud [4]. As the micro-architectural resources play an increasingly important role in the system performance, it becomes more and more important to efficiently address the shared resource contention problem in the cloud.

* Corresponding author.
E-mail address: yuxia.cheng@deakin.edu.au (Y. Cheng).

http://dx.doi.org/10.1016/j.future.2016.11.022
0167-739X/© 2016 Elsevier B.V. All rights reserved.

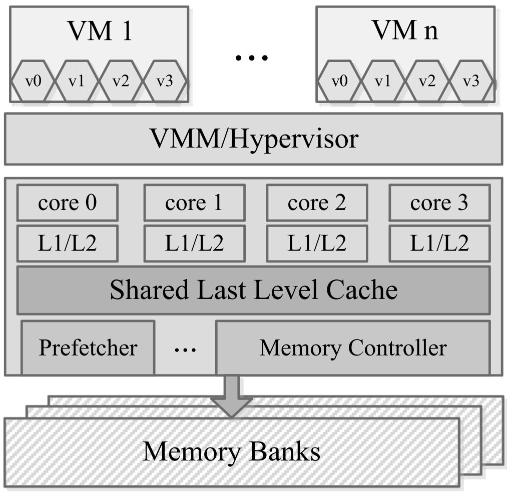


Fig. 1. The simplified architecture of multicore systems.

To alleviate the performance degradation caused by shared resource contention in the multicore systems, 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 [8–10,1] 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 due to the inaccurate performance prediction in the scheduling [11].

To more precisely manage contention problems, researchers proposed performance modeling techniques [12–15] to infer application behaviors when they are sharing physical resources. Typically, based on the performance monitoring events, the model predicts the application's performance behaviors. Then, performance optimization solutions can be applied according to the predictions. However, these modeling techniques were proposed in the non-virtualized and small-scale environment that mainly focused on the shared cache utilization. Other modeling techniques [16–18] proposed in the virtualized environment considered coarse grained factors, such as CPU, memory, storage and network factors. The fine grained micro-architectural level resource contention problems in the large-scale virtualized cloud environment needs further investigation.

In the large-scale cloud data center, addressing the shared resource contention problem to improve system efficiency and performance stability faces three major challenges. (1) First of all, different types of VMs have complex interactions with the underlying micro-architectural resources. How to effectively and efficiently measure the performance of VMs online and capture their performance characteristics are non-trivial in the large-scale data center. (2) Secondly, the overhead of migration VMs between physical servers is high, which requires extra compute and network resources. How to precisely predict the performance of co-scheduled VMs in advance to reduce re-scheduling migration overhead remains the open problem. (3) Thirdly, how to build a smart co-scheduling system, which integrates the process of VM performance monitoring, characterizing, predicting, and scheduling, requires intricate system design.

In this paper, we design and implement a smart VM co-scheduling system with the precise prediction of different VMs' performance characteristics in the cloud data center. The main contributions of this paper are described as follows:

(1) We identify the main resource contention factors between co-located VMs, and design the corresponding micro-benchmarks to quantify the levels of contention. We find that the data access patterns and working set sizes are the dominant factors for micro-architectural resource contentions. The synthetic micro-benchmarks are used to stress the micro-architectural shared resources. By co-running micro-benchmarks with real application VMs, we decouple two kinds of VM performance characteristics: the VM contention sensitivity and contention intensity. The sensitivity and intensity features are well correlated with the eventual performance degradation of VMs contending for shared resources.
(2) Using the obtained contention features, we build the performance prediction model with machine learning algorithms to quantify the precise levels of performance degradation. We collect the sensitivity and intensity features of applications in VMs, and record the performance degradation results of VMs with a large number of different co-scheduling combinations. The collected features and the performance degradation results are used as training set. The more numbers of VMs trained in the model, the more precise the online prediction results. The contention features collection time and the model training time are reduced using the decoupled sensitivity and intensity features.
(3) Based on the performance prediction model, we design the contention-aware VM scheduling algorithms in cloud data center according to different scheduling objectives: to improve overall system throughput or to guarantee the QoS of each VM. Before a new VM is deployed into the cloud, its contention features are first collected in a training server. Then, the model predicts performance degradation of the new VM when co-scheduling with other VMs, which provides the precise guidance for contention-aware VM scheduling.

The experimental results show that the performance prediction model achieves high accuracy and the mean absolute error is 2.83%. The contention aware VM scheduling algorithms based on the prediction model can effectively improve system throughput and guarantee the QoS of VMs compared with the traditional scheduling algorithms.

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

2. Related work

In cloud data center, performance optimization for the virtualized system is an important research area. How to exploit the performance opportunities in multicore systems, and how to improve resource utilization in cloud data centers as well as guarantee the QoS of VMs face many challenges.

In multicore systems, the shared last level cache is the performance critical resource. Previous researchers [12,19,20,13] proposed methods to analyze the shared cache usage in multicore systems. Hardware performance monitoring techniques [21–23] are used to capture program cache behaviors. 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 coloring [6,26] techniques to optimize shared cache contention problems in multicore systems.

Efficient co-scheduling of different threads to more constructively use shared on chip resources is another promising technique to alleviate contention problems [7]. Jiang et al. [27] and Radojković et al. [9] theoretically analyzed the optimal thread co-scheduling combinations on multicore processors. Tang et al. [11] analyzed the performance impact of co-scheduling large scale datacenter applications due to resource sharing, and showed that there exists both positive and negative impacts of co-scheduling as application behavior changes. Fedorova et al. proposed DI [10] and DINO [8] thread scheduling algorithms to reduce cache contention by spreading cache intensive threads apart and co-scheduling them with less cache intensive threads. Mars et al. [28] proposed a mechanism named Bubble-up to infer the performance degradation due to co-locating multiple applications on a single multicore server. Based on the Bubble-up mechanism, Yang et al. [29] further proposed the Bubble-Flux mechanism to effectively co-locate latency sensitive applications with batch jobs. Zhang et al. [15] and Eyerman et al. [30] proposed performance models in the multithread multicore systems to more precisely co-schedule appropriate tasks. These techniques were proposed in the non-virtualized environment.

In virtualized environments, Rao et al. [1] proposed a VCPU migration algorithm to optimize resource contention problems in NUMA (Non-Uniform Memory Access) multicore systems. The algorithm uses the uncore penalty metric to infer VCPU performance in the system and adaptively migrates VCPUs to minimize the system-wide uncore penalty. Liu et al. [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 et al. [31] 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.

The VM performance modeling techniques [16,17,4,18,32,33] were proposed to more precisely manage resources in the system. Govindan et al. [4] 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. [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 et al. [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.

In cloud data centers, many solutions [34–38] were proposed to increase server utilization and reduce resource conflicts. Nathuji et al. [34] designed a virtual machine QoS aware control framework named Q-Clouds. The Q-Clouds framework reserves suitable resources and tunes resource allocations to mitigate performance interference effects. Delimitrou et al. [35] proposed a heterogeneity and interference aware cluster management that uses collaborative filtering techniques to classify and deploy different applications in datacenter. Vasic et al. [36] proposed the DeepDive system to identify and manage performance interference among VMs co-located on the same physical server in cloud environments. Their approach to identify interference is based on VM performance classification and exhaustive interference analysis. Once identified the interference, the VM is migrated to a less loaded machine.

3. VM performance characteristics analysis

In this section, we describe the method of collecting VM runtime performance metrics and analyze 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.

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

where $PD^{A}_{co\text{-}run/B}$ represents the performance degradation of VM A when it is co-running with VM B, $P^{A}_{co\text{-}run/B}$ 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 [39,40]. 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).

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

where $CPI^{A}_{co\text{-}run/B}$ 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.

Fig. 2 shows the NPB benchmark [41] performance results reported inside VM compared with the CPI results monitored from outside VM using PMC. The x-axis represents different cases that two benchmarks are running inside two co-located VMs respectively. The y-axis represents performance degradation of benchmarks. The performance degradation is calculated using both benchmark reported runtime and the monitored CPI. Two metrics Pearson correlation coefficient is 0.995, which demonstrates that CPI is a suitable metric to measure VM performance for CPU and memory intensive workloads. The CPI metric of each VM can be easily obtained using PMC in the hypervisor, which does not need to enter into VMs to get the application's reported performance results that will violate the user's confidentiality in the cloud.

3.2. Resource contention features

In multicore systems, each core has its private processing unit and L1/L2 caches. Data requests that miss in the private caches are sent to the 'uncore' subsystem shared by multiple cores [1], which contains the shared LLC, interconnect, prefetcher, memory controller and other micro-architectural units. However, system software (hypervisor or operating system) [22] cannot directly manage the usage of these shared micro-architectural resources. Applications simultaneously contending for these resources will lead to performance degradation [11].

To characterize the VM's contention features that cause the performance degradation, we design several synthetic micro-benchmarks to co-run with VMs. Previous research [42] 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

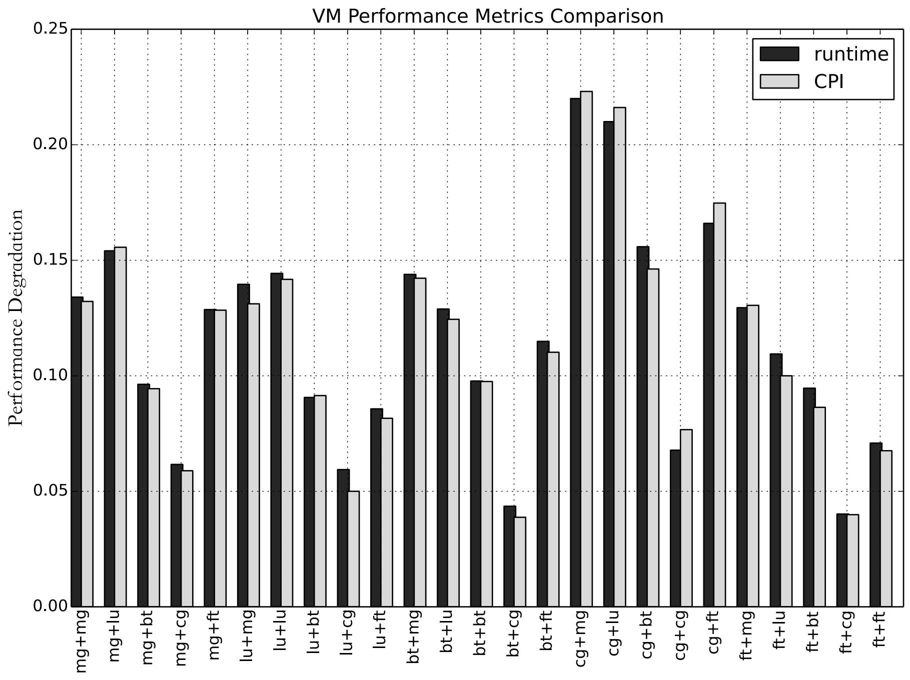


Fig. 2. The VM performance degradation metrics comparison.

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/Sequential, Read/Write, and working set sizes. Then, we use the following micro-benchmarks to characterize 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 maximize 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 [28,15,35] have demonstrated that we can sample only two intensity points to reduce the feature profiling overhead. Cache/Memory represent 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 the Eqs. (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).

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

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), $CPI^{A}_{co\text{-}run/bench_i}$ 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).

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

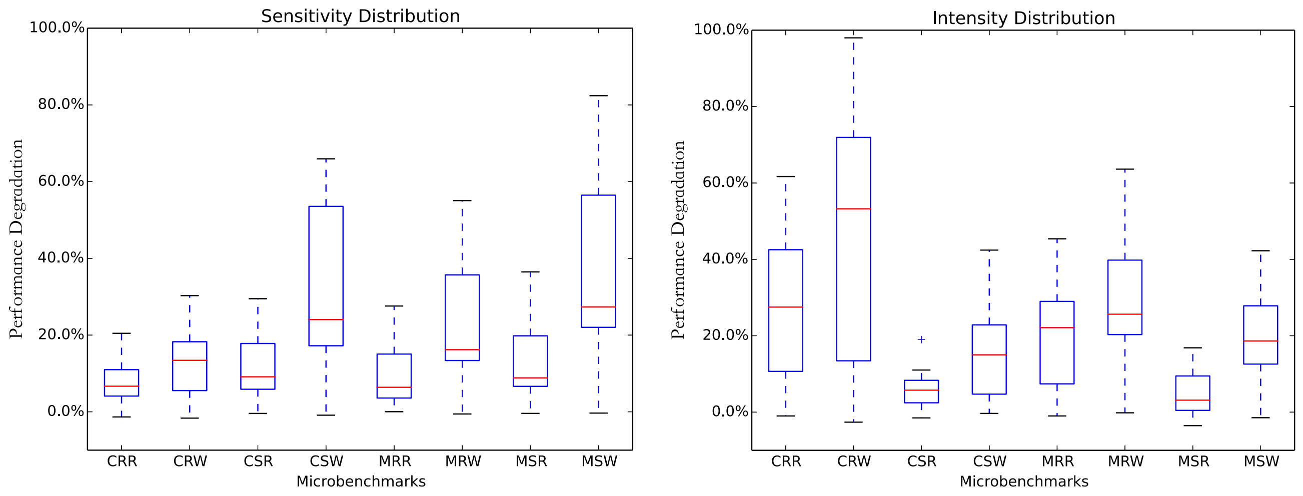
where $VCI^{A}_{bench_i}$ is the contention intensity of VM A when it is co-running with $bench_i$, $CPI^{bench_i}_{co\text{-}run/A}$ is the CPI of $bench_i$ when it is co-running with VM A, $CPI^{bench_i}_{alone}$ is the CPI of $bench_i$ when it is running alone.

Fig. 3 plots the contention sensitivity and intensity distributions of NPB [41] and SPEC CPU 2006 [43] benchmarks when they are co-running with different micro-benchmarks. The results show that different applications have a wide range of contention sensitivity and intensity. For example, the VCS under the contention of CRR micro-benchmark ranges from 0% to 20%, while the VCI of CRR micro-benchmark ranges from 0% to 60%. Each micro-benchmark co-running with the VM profiles a representative contention feature. As Fig. 3 shows, the contention sensitivity and intensity values have low correlations with each other, which helps the model capture more effective features to generate prediction results.

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 VM scheduling framework

In this section, we present the shared resource contention aware VM scheduling system. The system mainly consists of three parts: the VM performance training module, the VM performance prediction model, and the contention-aware VM scheduler.



(a) VM contention sensitivity (VCS). (b) VM contention intensity (VCI).

Fig. 3. The VM performance degradation distribution under different types of micro-architectural resource contention.

4.1. Overview

We propose the performance prediction based contention-aware VM scheduling system to alleviate resource contention problems among VMs and to improve system efficiency or to guarantee the QoS of VMs. In our deployment scenario, we assume that applications deployed in the VMs are typically long running, CPU and memory intensive workloads.

The overview of the contention-aware VM scheduling system is presented in Fig. 4, which includes three major 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 server. These features are used in the prediction model to predict performance degradation of co-located VMs.

(2) VM performance prediction model 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 are used for contention-aware VM scheduler. The VM online performance events are periodically monitored to update the prediction model when the prediction error is larger than a predefined threshold.

(3) Contention-aware VM scheduler is responsible for scheduling VMs onto proper physical servers based on the prediction result. The scheduler executes the proper scheduling algorithms to guarantee QoS of VMs, or to improve overall system throughput according to predefined resource management strategies.

4.2. VM Training

Applications deployed in the VMs are typically long running services. Before new VMs are deployed onto the physical production server, we collect the VM's runtime features on the training server. 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 server. The VM's contention features are obtained via co-locating the new VM with different micro-benchmarks respectively as Eqs. (3) and (4) show. The actual performance degradation of VMs are obtained via co-locating two application VMs onto the same multicore server 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.

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

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

Eqs. (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 ($PD^{A}_{co\text{-}run/B}$) and VM B ($PD^{B}_{co\text{-}run/A}$) respectively.

4.3. 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 ($\widetilde{PD}^{A}_{co\text{-}run/B}$), using the contention sensitivity features of VM A and the contention intensity features of VM B as inputs.

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

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.

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

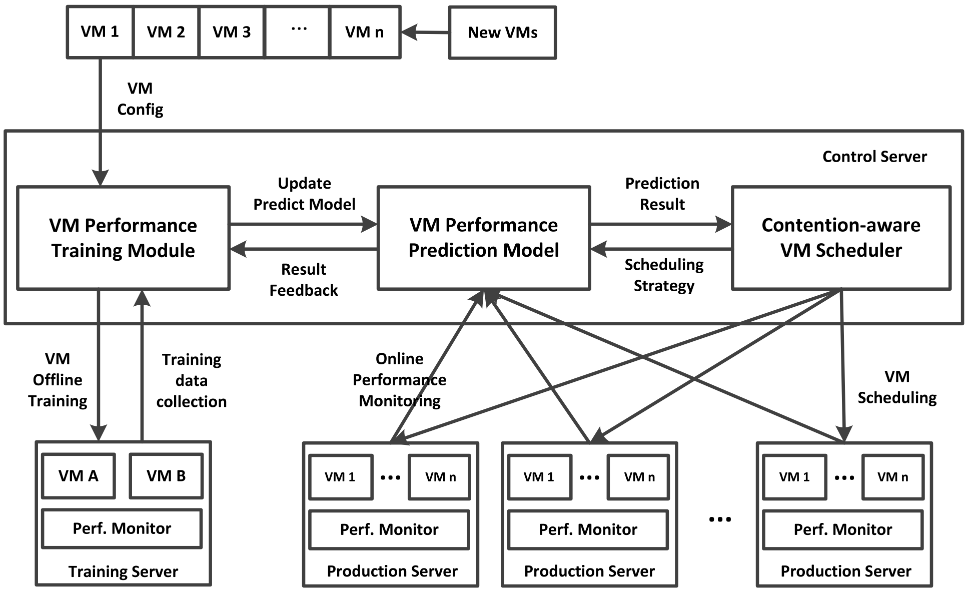


Fig. 4. Overview of the smart VM co-scheduling system with the precise prediction of performance characteristics.

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 Section 5, we 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.

4.4. Contention aware VM scheduling

Based on the prediction model, we can design VM scheduling algorithms to determine VM co-location strategies according to different resource management goals. For example, in order to improve overall system throughput and guarantee VM quality of service, we propose the following two algorithms as the use cases of the prediction model.

(1) Contention-Aware Best Fit (CABF) algorithm aims to improve overall system throughput by searching the best location for each VM that reduces the performance degradation caused by resource contention. The CABF algorithm is a greedy algorithm. First, CABF sorts the physical servers in the system by their available physical CPU and memory capacities in descending order. Second, CABF searches the sorted server list SL to find the suitable server S_j for the new VM_i so that S_j's CPU and memory capacities meet the requirement of VM_i. At the same time, the algorithm uses the prediction model to predict the performance degradation (PD_i^j) of VM_i when it is running on S_j. Finally, CABF returns the best server S_k that meets VM_i's resource requirement and has the least performance degradation.

(2) Contention-Aware QoS Assurance (CAQA) algorithm aims to guarantee VM's QoS by searching the proper location for each VM that makes sure the performance degradation is within some given threshold. The CAQA algorithm is an greedy algorithm similar to CABF. The difference is that CAQA adds QoS guarantee when searching proper server S_j (line 6). CAQA makes sure that the performance degradation of VM_i is within the predefined threshold PD_{QoS}. However, with the QoS requirement, the algorithm may fail to find a suitable server, then extra physical servers are needed in the system.

Algorithm 1 : Contention Aware Best Fit (CABF)

```
Input: $VM_i$; Server List $SL = \{S_j | j = 1, \ldots, n\}$
Output: $VM_i$ to $S_k$ placement
 1: sort $SL$ by each $S_j$'s CPU and memory capacities in descending
    order;
 2: $PD_i^{min}$ = IntMax;
 3: for $j = 1 \rightarrow n$ do
 4:    if $S_j$'s CPU and memory capacities meet the requirement of
       $VM_i$ then
 5:       $PD_i^j \leftarrow model(VM_i, VM_{S_j})$;
 6:       if $PD_i^j < PD_i^{min}$ then
 7:          $PD_i^{min} \leftarrow PD_i^j$; $S_k \leftarrow S_j$;
 8:       end if
 9:    end if
10: end for
11: return $S_k$;
```

The two scheduling algorithms give examples of how to leverage the prediction model. With the ability of precise performance prediction, we can deploy more advanced algorithms to further improve system efficiency in the future.

4.5. 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

Algorithm 2 : Contention Aware QoS Assurance (CAQA)

```
Input: $VM_i$; Server List $SL = \{S_j | j = 1, \dots, n\}$
Output: $VM_i$ to $S_k$ placement
1: sort $SL$ by each $S_j$'s CPU and memory capacities in descending order;
2: $PD_i^{min}$ = IntMax;
3: for $j = 1 \rightarrow n$ do
4:     if $S_j$'s CPU and memory capacities meet the requirement of $VM_i$ then
5:         $PD_i^j \leftarrow model(VM_i, VM_{S_j})$;
6:         if $PD_i^j < PD_{QoS}$ and $PD_i^j < PD_i^{min}$ then
7:             $PD_i^{min} \leftarrow PD_i^j$; $S_k \leftarrow S_j$;
8:         end if
9:     end if
10: end for
11: return $S_k$;
```

Table 1
Physical server configurations.

Server 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

VCI features of VM_{mix}, and use them as input variables of the model to predict performance degradation as described in Section 4.3.

The heterogeneous multicore servers are commonly seen in cloud data centers [44,45]. We build the prediction model on the same type of multicore systems. When it comes to the heterogeneous multicore servers, 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 scheduling system runs for a long period of time [35], 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 scheduler will re-schedule the VM to another physical server using the proposed scheduling algorithms.

As we focused on the multicore resource contention problems, the I/O and network contention problems [46,47] are beyond the scope of this paper.

5. Performance evaluation

We evaluate the proposed prediction model and scheduling algorithms on two types of multicore servers summarized 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 [41] 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 [43] 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 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 regularized according to the weka *arff* file [48]. 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 [49] 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. 5 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. 6 presents the actual and predicted results under the REPTree model. The REPTree algorithm is one of the decision tree implementations in the Weka tool [48]. 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. 6, 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.

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

Fig. 7 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).

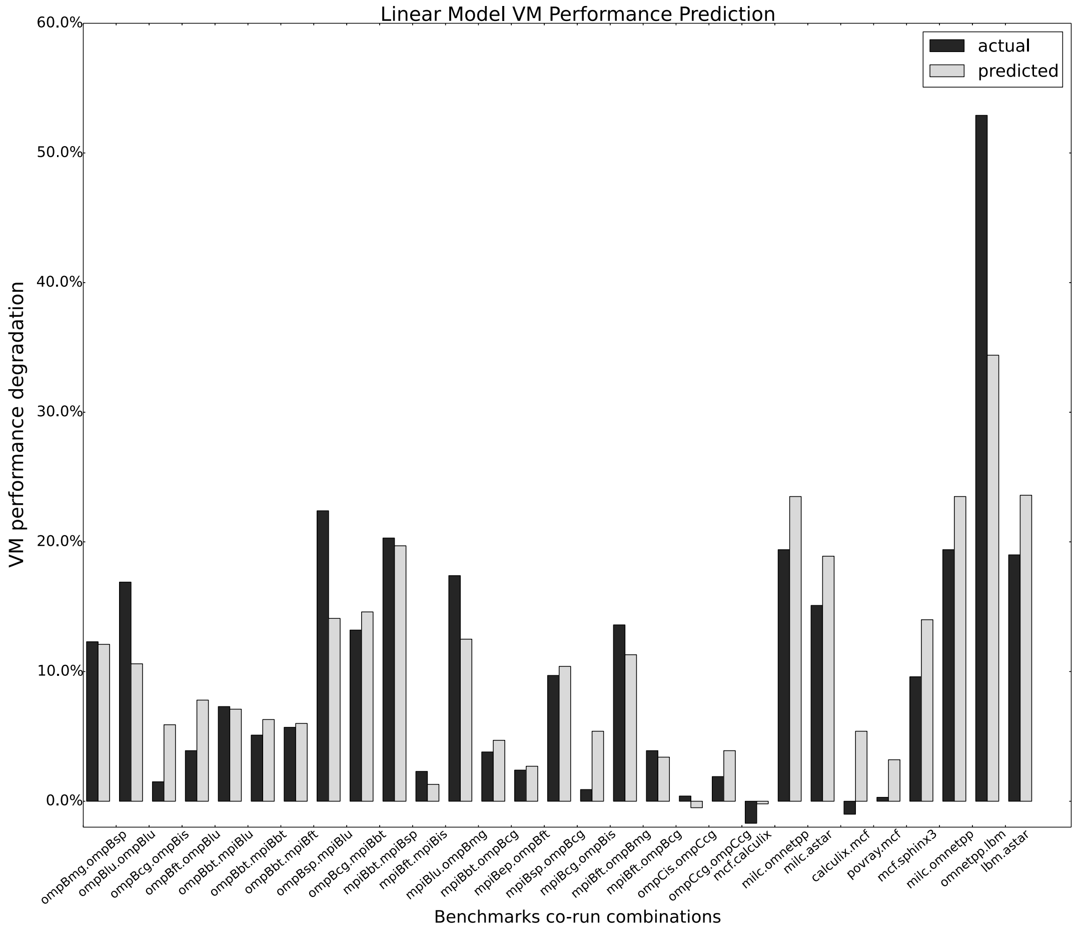


Fig. 5. The comparisons between the actual performance degradation and the predicted performance degradation under the linear regression model.

As shown in Fig. 7, 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 scheduling framework. To compare the performance of these algorithms, we present four metrics that 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. In the following experiments, we use the Bagging algorithm to predict performance degradation of VMs.

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

5.2. Contention-aware VM scheduling evaluation

Based on the precise performance prediction, the VM scheduler can take advantage of smart VM co-locations to alleviate performance degradation problems caused by shared resource contention. We compare the following three scheduling algorithms to analyze the benefits of precise performance prediction: (1) Contention Unaware Least Load, short for CULL algorithm; (2) Contention Aware Best Fit, short for CABF algorithm; (3) Contention Aware QoS Assurance, short for CAQA algorithm. The CULL algorithm only considers VM's CPU and memory requirements, but does not utilize the performance prediction capability proposed in this paper. When a new VM needs to be deployed onto physical server, the CULL algorithm searches the least loaded server to hold

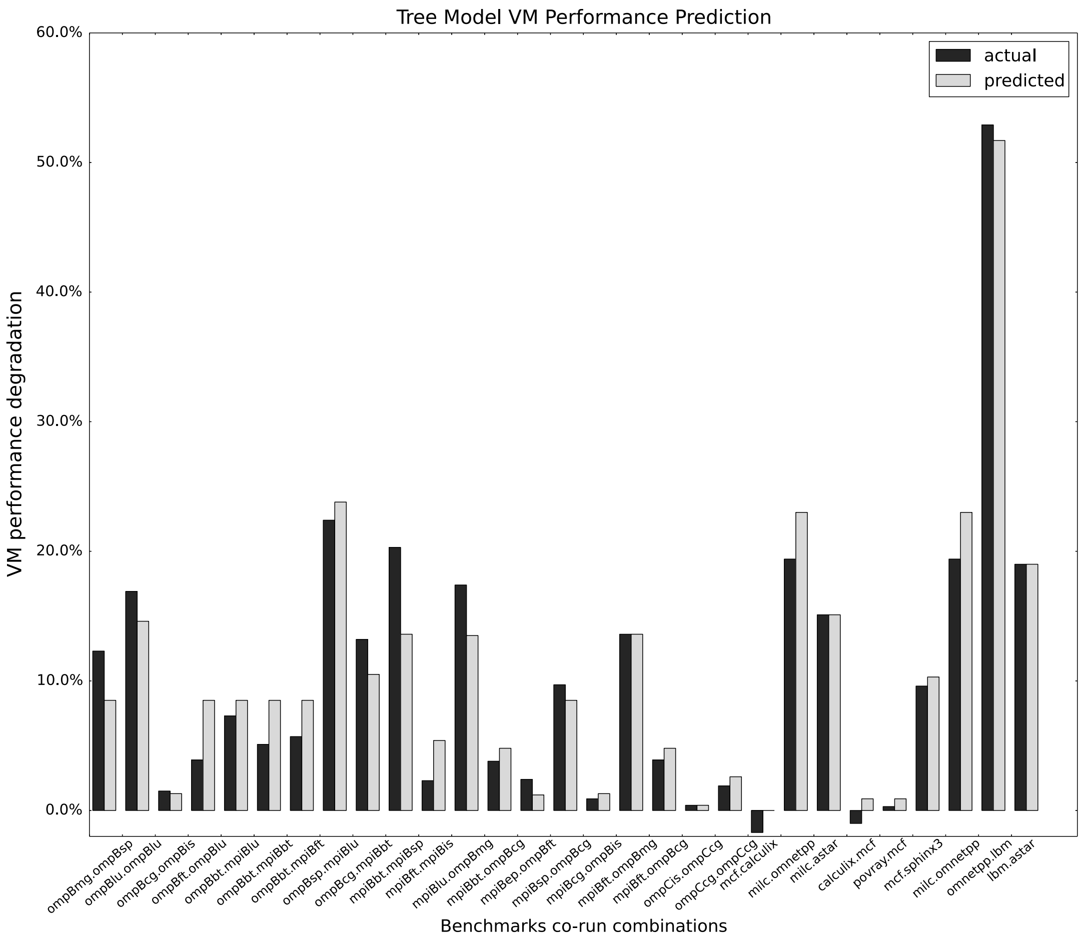


Fig. 6. The comparisons between the actual performance degradation and the predicted performance degradation under the REPTree model.

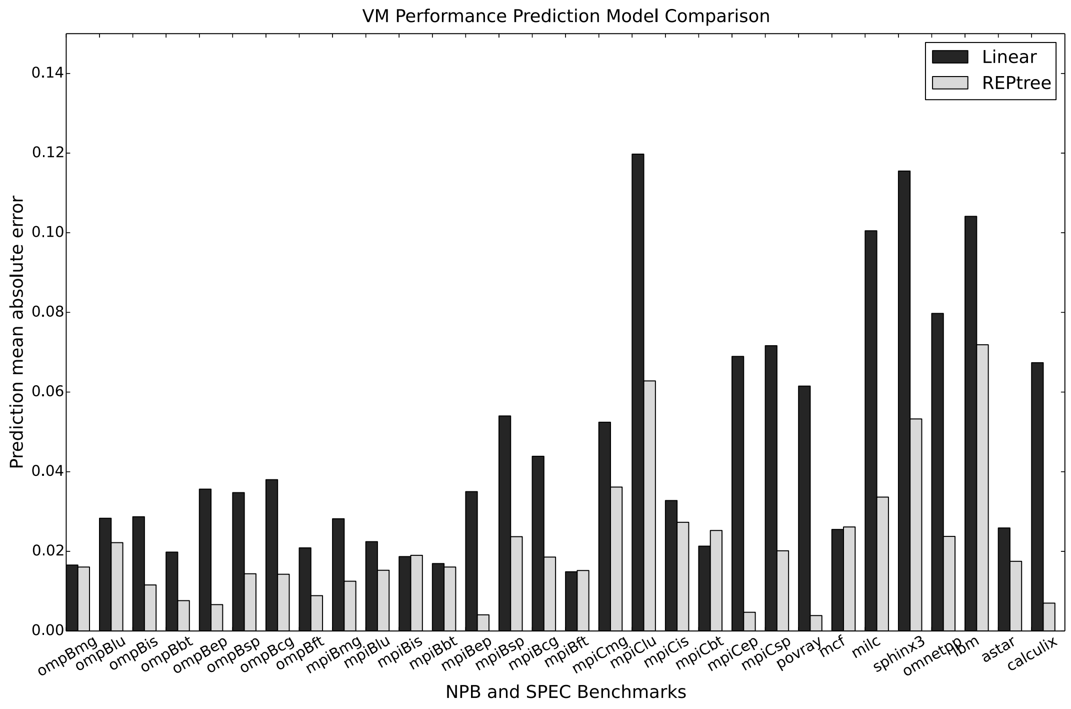


Fig. 7. The prediction mean absolute error (MAE) comparisons of the Linear regression model and the REPTree model.

the VM without considering the resource contention problem. The CABF and CAQA algorithms are presented in Section 4.4.

In the experiment, we simulate a scenario of 1000 VMs to be deployed onto 100 R710 servers and 75 R910 servers (a

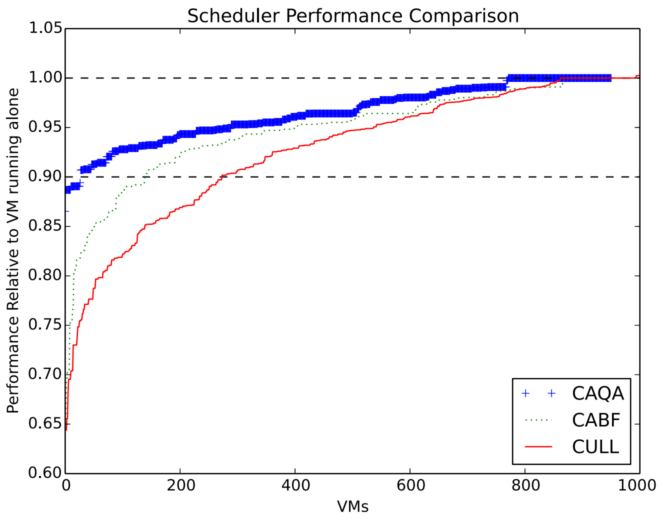


Fig. 8. The relative performance of VMs under three scheduling algorithms.

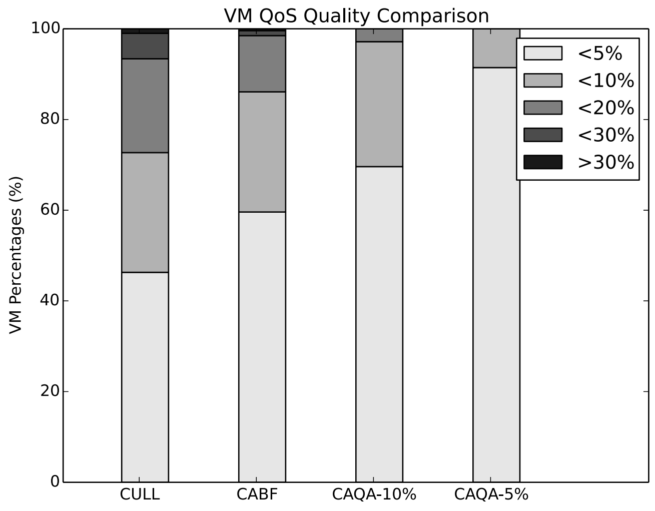


Fig. 9. The VM QoS distribution under four scheduling algorithms.

total of 100*2 + 75*4 = 500 multicore chips, R710 and R910 configurations shown in Table 1). We use three different scheduling algorithms to schedule 1000 VMs onto proper physical servers respectively.

Fig. 8 shows the VMs relative performance under three different scheduling algorithms. The x axis shows the deployed VMs in the system, and the VMs are sorted by their performance degradation in descending order numbered from 1 to 1000. The y axis represents the VM's relative performance compared to when it is running alone. For example, in the CULL algorithm, the highest performance degradation reaches 36%. In the CABF algorithm, the average performance of VMs are improved than in the CULL algorithm where no resource contention factors are taken into consideration. In the CAQA algorithm, more than 95% of VMs meet the predefined QoS requirement that the VM's performance degradation cannot exceed 10%. Due to the QoS requirement, the number of VMs that can be deployed onto the same amount of physical servers is smaller than no QoS requirement scheduling strategies. Therefore, when the CAQA algorithm cannot find suitable servers for new VMs that meet the QoS requirement, the scheduler will report the requirement of more physical resources.

As shown in Fig. 8, the VM performance degradation problem is much worse in the CULL algorithm than in the CABF and CAQA algorithms. This demonstrates that the contention aware VM scheduling algorithms can avoid co-locate VMs that will intensively contend for shared resources and can improve the overall system efficiency in cloud data center.

Fig. 9 shows the QoS of VMs under different VM scheduling algorithms. The x axis shows four scheduling algorithms, where CAQA-10% and CAQA-5% represent the predefined QoS requirement that the VM's performance degradation cannot exceed 10% and 5% respectively. The y axis shows the distribution of VM performance degradation, and the color of histogram from deep to shallow represents the level of performance degradation from high to low. For example, in the CULL algorithm, the number of VMs whose performance degradation is less than 5% only occupies 46.3% of the total number of VMs in the system. While in the CAQA-5% algorithm, the number of VMs whose performance degradation is less than 5% occupies 91.45% of the total number of VMs in the system. Due to the precise performance prediction, the CAQA-5% algorithm guarantees the QoS of most VMs in the system, and the performance degradation of the rest 8.55% VMs is within 5%–10%.

From the experiment, we show the benefits of leveraging performance prediction capabilities in VM scheduling algorithms, and the results demonstrate that contention-aware VM scheduling algorithms not only improves system efficiency but also guarantees the QoS of VMs.

5.3. Overhead analysis

The extra overhead of the contention aware VM scheduling system 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 physical server. 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 performance prediction based contention aware VM scheduling system. 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 behavior 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. Based on these features, we build the VM performance prediction model using machine learning techniques. The precise performance prediction capability makes possible of the contention-aware VM scheduling algorithm design. The experimental results show that the proposed solutions can improve the overall efficiency of virtualized systems and also help guarantee the QoS of single VM.

References

[1] J. Rao, K. Wang, X. Zhou, C.-Z. Xu, Optimizing virtual machine scheduling in numa multicore systems, in: 2013 IEEE 19th International Symposium on High Performance Computer Architecture, (HPCA), IEEE, 2013, pp. 306–317.

[2] M. Liu, T. Li, Optimizing virtual machine consolidation performance on NUMA server architecture for cloud workloads, in: 2014 ACM/IEEE 41st International Symposium on Computer Architecture, (ISCA), IEEE, 2014, pp. 325–336.

[3] Y. Cheng, W. Chen, Z. Wang, X. Yu, Performance Monitoring based traffic-aware virtual machine deployment on NUMA systems, IEEE Syst. J. PP (99) (2015) 1–10.

[4] 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.

[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. Archit. 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: 2011 38th Annual International Symposium on Computer Architecture, (ISCA), 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: 2011 IEEE International Symposium on Performance Analysis of Systems and Software, (ISPASS), IEEE, 2011, pp. 76–86.

[13] A. Sandberg, A. Sembrant, E. Hagersten, D. Black-Schaffer, Modeling performance variation due to cache sharing, in: High Performance Computer Architecture (HPCA), 2013 IEEE 19th International Symposium on, 2013, pp. 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: 2014 47th Annual IEEE/ACM International Symposium on Microarchitecture, (MICRO), IEEE, 2014, pp. 406–418.

[16] S. Kundu, R. Rangaswami, K. Dutta, M. Zhao, Application performance modeling in a virtualized environment, in: 2010 IEEE 16th International Symposium on High Performance Computer Architecture, (HPCA), 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, IEEE Trans. Parallel Distrib. Syst. 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: 2014 IEEE International Symposium on Performance Analysis of Systems and Software, (ISPASS), 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] Y. Jiang, X. Shen, J. Chen, R. Tripathi, Analysis and approximation of optimal co-scheduling on chip multiprocessors, in: Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques, ACM, 2008, pp. 220–229.

[28] 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.

[29] 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.

[30] 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.

[31] M. Lee, K. Schwan, Region scheduling: efficiently using the cache architectures via page-level affinity, ACM SIGARCH Comput. Archit. News 40 (1) (2012) 451–462.

[32] S.M. Zahedi, B.C. Lee, REF: Resource elasticity fairness with sharing incentives for multiprocessors, in: Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems, ACM, 2014, pp. 145–160.

[33] Y. Cheng, W. Chen, Z. Wang, Y. Xiang, Precise contention-aware performance prediction on virtualized multicore system, J. Syst. Archit. (2016).

[34] 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.

[35] 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, 2013, pp. 77–88.

[36] N. Vasic, D. Novaković, S. Miučin, D. Kostic, R. Bianchini, Dejavu: accelerating resource allocation in virtualized environments, ACM SIGARCH Comput. Archit. News 40 (1) (2012) 423–436.

[37] R.C. Chiang, J. Hwang, H.H. Huang, T. Wood, Matrix: Achieving predictable virtual machine performance in the clouds, in: USENIX 11th International Conference on Autonomic Computing, USENIX, 2014, pp. 45–56.

[38] J. Moses, R. Iyer, R. Illikkal, S. Srinivasan, K. Aisopos, Shared resource monitoring and throughput optimization in cloud-computing datacenters, in: Parallel & Distributed Processing Symposium (IPDPS), 2011 IEEE International, IEEE, 2011, pp. 1024–1033.

[39] 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.

[40] 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.

[41] The NAS Parallel Benchmarks. http://www.nas.nasa.gov/publications/npb.html [online].

[42] C. Delimitrou, C. Kozyrakis, iBench: Quantifying interference for datacenter applications, in: 2013 IEEE International Symposium on Workload Characterization, (IISWC), IEEE, 2013, pp. 23–33.

[43] Spec cpu 2006. http://www.spec.org/cpu2006/ [online].

[44] C. Reiss, A. Tumanov, G.R. Ganger, R.H. Katz, M.A. Kozuch, Heterogeneity and dynamicity of clouds at scale: Google trace analysis, in: Proceedings of the Third ACM Symposium on Cloud Computing, ACM, 2012, p. 7.

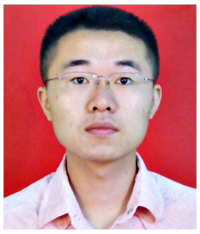
[45] J. Mars, L. Tang, Whare-map: heterogeneity in homogeneous warehouse-scale computers, in: ACM SIGARCH Computer Architecture News, ACM, 2013, pp. 619–630.

[46] H. Li, M. Dong, K. Ota, Radio Access Network Virtualization for the social Internet of things, IEEE Cloud Comput. 2 (6) (2015) 42–50.

[47] G. Luo, Z. Qian, M. Dong, K. Ota, S. Lu, Network-aware re-scheduling: Towards improving network performance of virtual machines in a data center, in: International conference on algorithms and architectures for parallel processing, 2014.

[48] Weka 3. http://www.cs.waikato.ac.nz/ml/weka/ [online].

[49] 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.

Zhongxian Tang received the B.S. degree in computer science and technology from Zhejiang University, Hangzhou, China, in 2014. He is currently pursuing the M.S. degree of computer science and technology in Zhejiang University, Hangzhou, China. His current research interests include operating systems, virtualization technology, and distributed systems.

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.