I/O Behavior Characterizing and Predicting of Virtualization Workloads

Yanyan Hu, Xiang Long, Jiong Zhang
Department of Computer Science, Beihang University, Beijing, China
Email: huyanyan84@gmail.com, long@buaa.edu.cn, zhangjiong@buaa.edu.cn

Abstract— **In virtual machine system, different workloads are consolidated into a single platform to fully utilize the hardware resources. However, the diversity and strong variation of applications always make it difficult to optimize the resource allocation and thus reduce the system performance and efficiency. Therefore, how to accurately analyze and predict the runtime behavior of applications has become an important basement for virtual machine system optimization. In order to study the characteristic and predictability of virtualization applications, this paper proposes a dynamic behavior characterizing and predicting methodology under Xen virtual machine. We analyze the characteristics of several typical virtualization workloads with fine temporal granularity and apply several online predictors to predict application's runtime I/O behavior. Experiment results demonstrate that the I/O behavior of virtualization workloads can be efficiently predicted by using proper predicting model and configuration. With this result, we further investigate the possibility of virtual machine scheduler optimizing based on I/O behavior characterizing. Several important issues are discussed including I/O computing jobs isolation through asymmetric scheduling, VM dynamic migration based on execution phase tracking and co-scheduling of multiple cooperative virtual machines. Preliminary test results demonstrate that this approach could efficiently reduce the performance degradation caused by scheduling competition in virtual machine system.**

Index Terms— **Virtualization, I/O, behavior analysis, predict, scheduler**

I. INTRODUCTION

Virtualization technology offers many advantages to modern computing environment. By running multiple virtual machines (VMs) in a shared physical platform, virtualization enables high utilization of hardware resources. Many features such as live migration and easy restart of VMs also significantly improve manageability of large-scale computing system. In virtualization environment, different traditional server applications are consolidated into a single physical platform to fully utilize the hardware resources. These workloads always have different runtime behavior and resource requirement. However, current virtual machine schedulers usually employ global-symmetric algorithms to force different workloads to compete for the CPU resources disorderly and thus cause serious performance interference [1], [2] which significantly reduce the system performance and efficiency.

In order to address this problem, previously proposed works have tried to classify applications into different categories(such as I/O-intensive and CPU-intensive ones) and encapsuled them into individual virtual machines [3], [4]. This approach attempts to employ the isolated computing environment provided by VMM to reduce the resource competition between different applications. However, most virtualization workloads contain I/O and computing operations simultaneously and their behavior always vary over time while exhibit complex characteristics. In this case, traditional workload classification at virtual machine granularity would not be accurate and flexible enough to reflect the actual resource requirement of different applications promptly. It could make the resource allocation strategy become either over-provisioned or overloaded and thus cause performance regression. Therefore, it would be challenging to properly optimize the resource allocation at virtualization environment.

A possible solution for this issue is classifying applications at a finer temporal granularity (e.g. couples of scheduling periods) and dynamically allocate computing resources to specific workloads based on their current running states. In this case, the scheduler must be able to perceive the state variation of applications and adjust its resource allocation strategy promptly. Therefore, accurately characterizing and predicting the runtime behavior of applications would become a pivotal problem for intelligent and self-adaptive virtual machine scheduler design. Most of current research works about virtualization workloads concentrate on the performance analysis and scheduling algorithm optimization [5], [6], [7], [8]. The study of fine-granularity application behavior analysis has not been carried out thoroughly. At the same time, although application behavior analysis and prediction has been a common and efficient way to improve the efficiency of hardware system [9], [10] or to optimize the performance of traditional programs [11], [12], [13], none of these presented works specifically dealt with virtualization environment.

Therefore, in this paper we propose a runtime behavior characterizing and predicting methodology for virtualization environment. We first build a test bed based on Xen virtual machine and collect the runtime information of several typical virtualization applications. Then we characterize their time-varying behavior and explore the predictability from several aspects including periodicity, state-transition trait and cooperative relationship cross

This work was supported by the National High Technology Research and Development Program (”863”Program) of China (2007AA01Z118).

different virtual machines. After that, we describe several online predictors and compare their precision when they are applied into different cases. Test results demonstrate that the I/O state variation of applications can be efficiently captured and predicted by using proper predicting model and configuration.

Based on these results, we further exploit the possibility of virtual machine scheduling optimization using I/O behavior predicting and execution phase tracking. We first construct a asymmetric scheduling framework under Xen-4.1.0 virtual machine and deploy two individual cpu subsets to undertake I/O and computing jobs respectively. Then we try to track different execution phases of applications based on their I/O behavior prediction results and migrate VMs cross different cpu subsets if we decide the system performance could benefit from this scheduling adjustment. Several important issues are discussed including the method of execution phase tracking, topology optimization of cpupool allocation and co-scheduling of cooperative virtual machines. Preliminary test results demonstrate that this approach could efficiently efface the competition between I/O and computing jobs and then promote the system performance, especially for some communication-intensive applications.

The contributions of this paper are mainly listed as follows. First, we propose a runtime behavior characterizing and predicting framework for virtualization environment. Second, we discuss several important issues that affect application behavior analysis and predicting. Finally, we exploit a possible methodology of virtual machine scheduler optimization based on I/O behavior predicting and asymmetric scheduling.

The remainder of this paper is organized as follows. We first describe our test environment and method of runtime information collecting in section 2. Then we characterize the I/O behavior of several representative virtualization applications and discuss their predictability in section 3. After that, we compare several online predictors and discuss the principal of predictor selecting in section 4. We further investigate several important issues of virtual machine scheduler optimization based on I/O behavior predicting and provide some preliminary test results in section 5. We introduce some related work in section 6 and finally conclude with a discussion in section 7.

II. Runtime I/O Information Collecting

Our application behavior analysis work is based on Xen-4.1.0 virtual machine which applies an IDD(Isolated Device Domain) device model to manage its I/O devices [14]. In this model, a virtual *frontend* driver in a domainU communicates with a corresponding virtual *backend* driver residing in the isolated driver domain(always Dom0) and forwards I/O requests to a native device driver. The frontend driver and the backend driver notify each other by an I/O event through *event channel* mechanism which is a virtualization of hardware interrupt. Each time an I/O request or response is sent, an I/O event is pent in the corresponding event channel and then delivered into

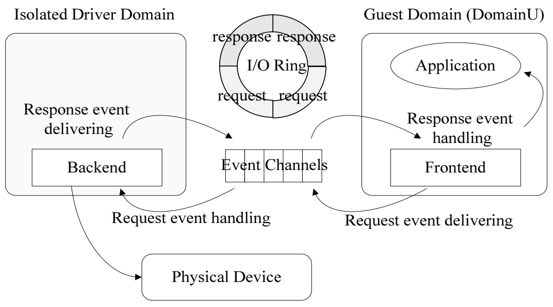


Figure 1. IDD Model of Xen Virtual Machine

the target domain when this domain is scheduled next time.

Therefore, in Xen virtual machine, the I/O event behavior could exhibit the characteristic of I/O jobs intuitively. Some other virtual machine systems such as KVM [15] and VMWare ESX [16] also have similar I/O structure although their device models may have a little difference. For example, VMWare ESX server does not have an individual driver domain and manages native device drivers by VMM itself. However, they also apply similar soft interrupt mechanism to notify the GuestOS about the I/O operation state. Therefore, characterizing the I/O behavior of applications through I/O event information could be an universal approach for most virtual machine systems. In this paper, we applied a monitoring tool [17] to collect the I/O event information of virtual machines dynamically. This information is later used to characterize and predict the runtime I/O behavior of applications.

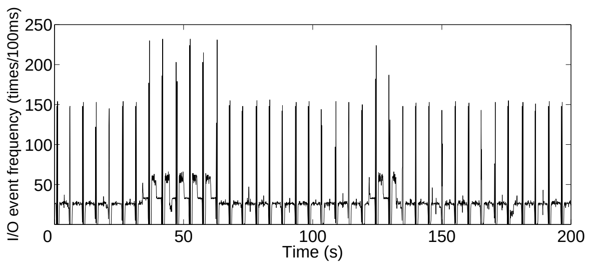
III. Virtualization Workloads Behavior Characterizing

A. Test Environment

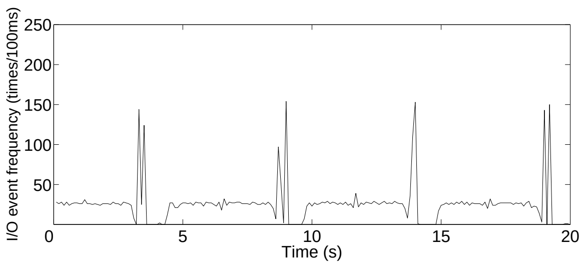
In this section, we will characterize the I/O behavior of several representative virtualization applications to set the stage for I/O state predicting. Our test covers two typical industry-standard benchmarks including `TPC-W` and `NPB-MPI`. We ran these two benchmarks under Xen virtual machine and recorded the I/O behavior of applications by collecting their I/O event information with a time granularity of 100ms. All of our experiments were performed in a x86 server which has an Intel Q9550 quad-core processor, 4GB DDRII-800 memory, double RTL8169 1000Mbps Ethernet NIC and Seagate 1TB SATA hard disk. We ran Xen-4.1.0 virtual machine and used Domain0 as driver domain. Both Domain0 and DomainU ran Debian6.0 Linux distribution with Linux-2.6.32 kernel.

B. I/O Behavior of Typical Server Consolidation Workloads

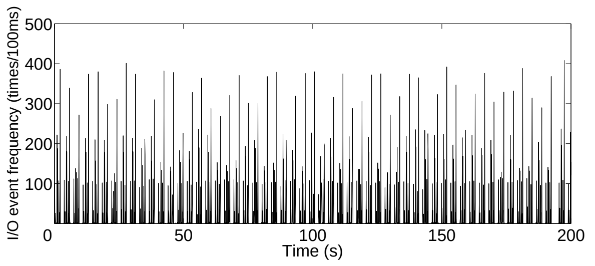
NPB-MPI: `NPB-MPI` is an MPI implementation of the NASA Advanced Supercomputing (NAS) Parallel Benchmark (NPB) [18]. It is a small set of programs designed to help evaluate the performance of parallel supercomputers. The benchmarks, which are derived from



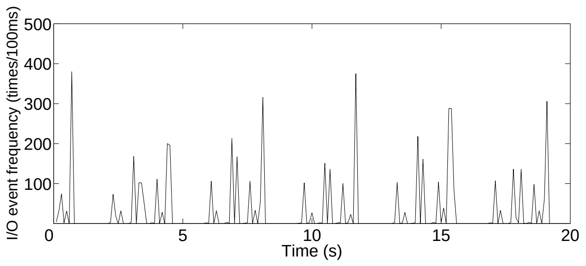
(a) I/O behavior of LU program



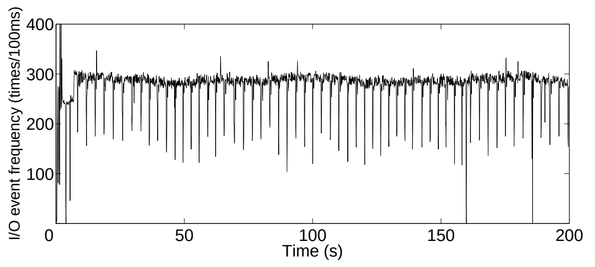
(b) Result zoomed into 20s



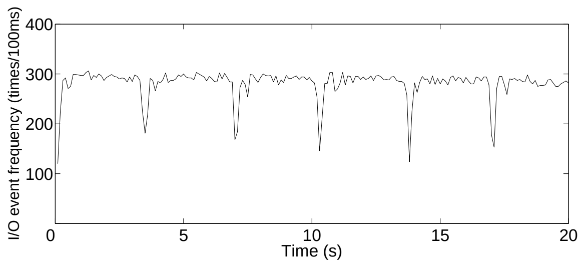
(c) I/O behavior of SP program



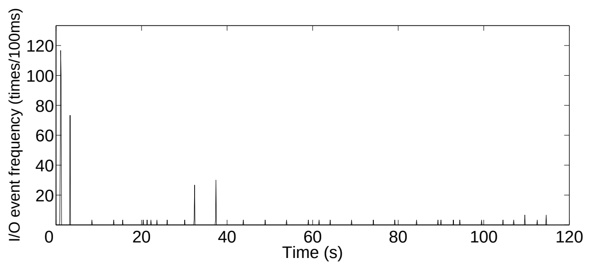
(d) Result zoomed into 20s



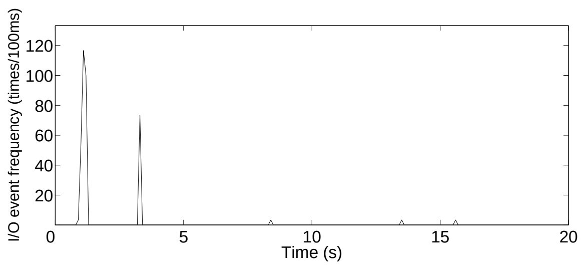
(e) I/O behavior of CG program



(f) Result zoomed into 20s



(g) I/O behavior of EP program



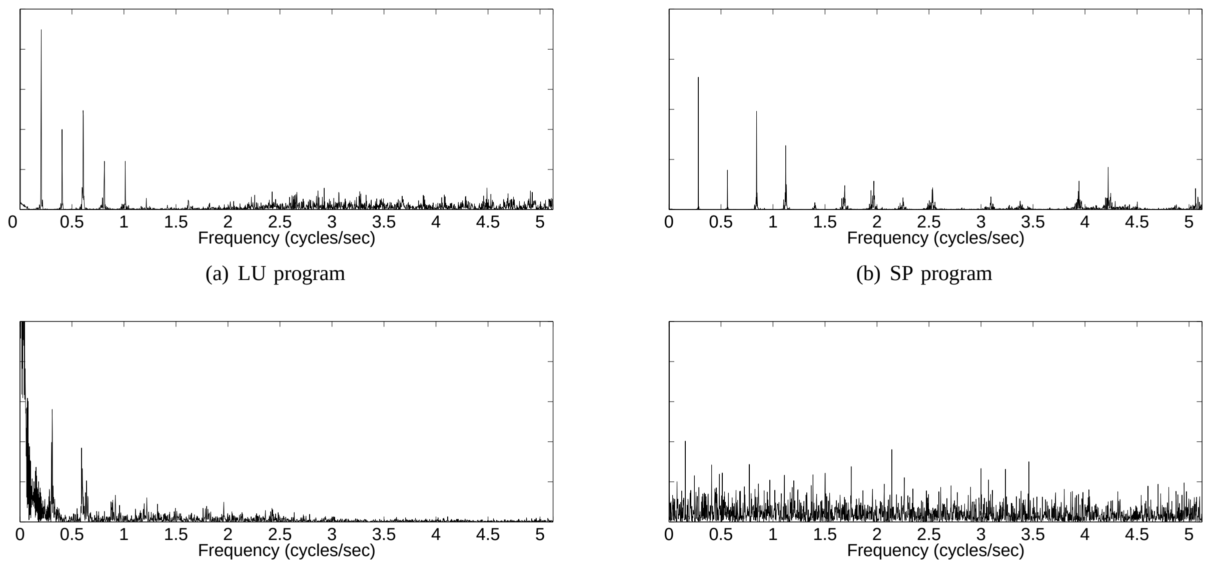
(h) Result zoomed into 20s

Figure 2. I/O behavior of NPB-MPI benchmark

computational fluid dynamics (CFD) applications, consist of five kernels and three pseudo-applications. In this test, we selected four benchmarks including `LU, CG, SP` and `EP`. Test results of other three benchmarks (`MG, BT` and `FT`) were excluded here because their have similar behavior. We used the Class C problem sizes to achieve a proper data scale and total running time and tested each benchmark using 4 parallel computing nodes with each of them had one VCPU and 512MB memory. The I/O event frequency variation of each individual VM was recorded during the execution. We only demonstrate the result of primary node(DomainU_1) here because other computing nodes have very similar I/O behavior. We will discuss the similarity and correlation between the I/O behaviors cross different computing nodes in detail in section III.C.

As illustrated in Figure 2, these programs exhibit significantly different I/O behaviors during their execution. For `LU` benchmark, I/O intensity keeps in lower level in most time while high-intensity I/O operations appear periodically. Conversely, the I/O intensity of `CG` benchmark always keeps in high level and only companied by transient low-intensity I/O load states. For `SP` benchmark, high and low intensity I/O load states appear alternately and the variation does not exhibit noticeable periodicity. The `EP` benchmark has little I/O operations during its entire execution.

This behavior distinction actually reflects the communication activity difference between these parallel benchmarks. `LU` benchmark applies SSOR algorithm to solve regular-sparse lower and upper triangular systems. Computing nodes perform fine-granularity point-to-point communication during each iteration and thus cause con-



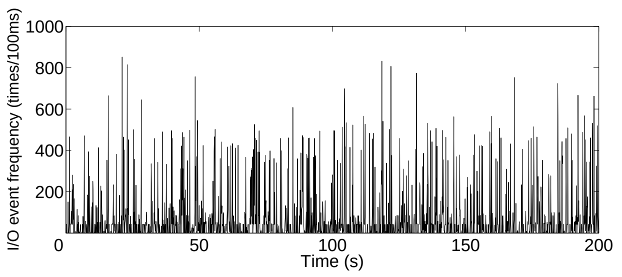
(a) LU program (b) SP program (c) CG program (d) TPC-W front-tier

Figure 3. Periodograms of NPB-MPI and TPC-W benchmarks

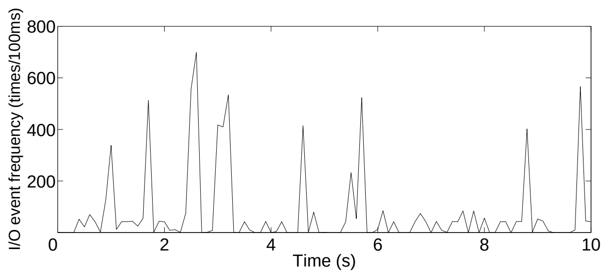
tinuous but low-intensity I/O operations. The periodic high-intensity I/O states are caused by data transmission at the joint of two iterations since result data need to be synchronized before starting the next step calculation. CG benchmark uses a conjugate gradient method to compute an approximation to the smallest eigenvalue of a large, sparse, symmetric positive definite matrix. Different from LU, it performs intensive irregular collective communication cross different nodes when executing sparse-matrix vector multiplication during each loop. Therefore, continuous high-intensive I/O operations are observed during the execution and periodic low-intensive I/O states always appear at the end of each iteration. SP benchmark solves multiple independent systems of non diagonally dominant scalar pentadiagonal equations. This program is used to test the balance cross communication and computing. Its communication is mainly composed of irregular point-to-point long message transmission and each process can perform computing work as soon as it receives data from other processes. Therefore, low-intensity I/O load states are always followed by irregular high-intensity I/O operations which demonstrate the transmission of messages during the execution. EP benchmark, as the acronym suggests, is an ”embarrassingly parallel” kernel. It requires virtually no inter-processor communication and only coordination of pseudo-random number generation at the beginning and collection of results at the end. Therefore, it produces little I/O operations during its entire execution.

We further apply fourier transform to analyze the periodicity of different programs. Using 100ms sampling period, the range of frequency discernible is 0 to 5 cycles/sec according to the Nyquist rate. As illustrated in Figure 3, LU and CG programs exhibit strong periodicity just as our intuitively observation. This periodicity can be attributed to the execution iteration loops of these two programs. For example, the frequency spectrum peak of 0.2 cycle/sec for LU and 0.29 cycle/sec for CG exactly match the iteration periods illustrated in figure 3(b)(about 5s, total 250 iterations during 1283s execution) and figure 3(f)(about 3s, total 75 iterations during 247s execution). This strong structural regularity could significantly facilitate the following I/O state predicting. By contrast, SP's periodogram is a little unexpected. There is a peak value that reflects an implicit periodicity of 0.32 cycles/sec which is difficult to perceive from temporal behavior intuitively while could match the iteration period of SP program (total 400 iterations during 1473s execution). This implicit periodicity could also promote the precision of history predictors. We will demonstrate the effect of application's periodicity for I/O state predicting through several experiments in section IV.

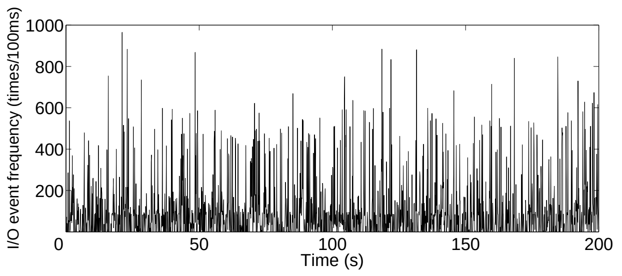
TPC-W: TPC-W benchmark is a double-tier application based on the TPC-W standard [19] for an online bookstore. It is usually used to evaluate the performance of commercial servers. A TPC-W implementation always requires two software modules to support the test system and drive the benchmark: an frontend application server handling user sessions and a backend database server keeping merchandize information. In our test, we applied a TPC-W-NYU [20] test suit which is a fully J2EE compliant application to setup our test environment. We used JBoss 3.2.7 [21] as the frontend application server and MySQL 4.1 [22] as the backend database server. We deployed these two tiers into individual virtual machines which both have two VCPUs and 1GB memory allocation. Then we used the workload generator provided by TPC-W-UVA [23] to simulate multiple concurrent browser clients accessing the application from another physical client in the same local network. We ran **shopping mix** test for an hour with 100 parallel sessions and recorded the I/O event frequency variation of



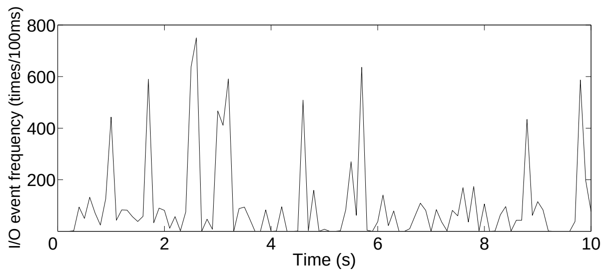
(a) I/O event frequency of front-tier



(b) Result zoomed into 10s



(c) I/O event frequency of back-tier



(d) Result zoomed into 10s

Figure 4. I/O behavior of TPC-W benchmark

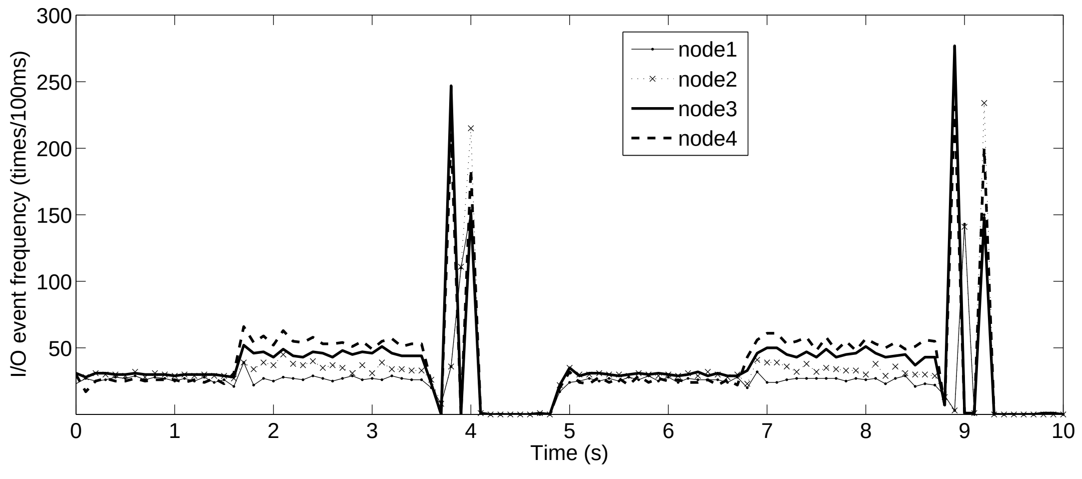
both frontend and backend virtual machines with 100ms sample period. Test result is illustrated in Figure 4.

Different from NPB benchmarks, the I/O event frequency of TPC-W test servers exhibit strong variation during the test. We analyze the periodograms of its I/O behavior and there are a cluster of peaks at high frequencies depicted in figure 3(d) which demonstrate a strong aperiodicity. This phenomenon is mainly caused by the diversity of online transaction server's I/O operations which are composed of customer network access for the web server, disk operations for the database server and inter-VMs communication between frontend and backend tiers. On the other hand, the access requests from parallel browser clients also have several different types including browsing, ordering and shopping and the behavior of customers always have strong randomness. Therefore, intuitively it would be difficult to distinguish the regularity of I/O behavior of TPC-W severs. However, when we zoom our test result into a 10 second duration which includes 100 samples, we find the I/O load intensity will always stay in a stable state for several continuous sample periods. Therefore, although we may not be able to get as much benefit from the periodicity of TPC-W application as we did from NPB programs, we could still predict the I/O load states of TPC-W severs correctly and differentiate different execution phases if using a proper configuration.

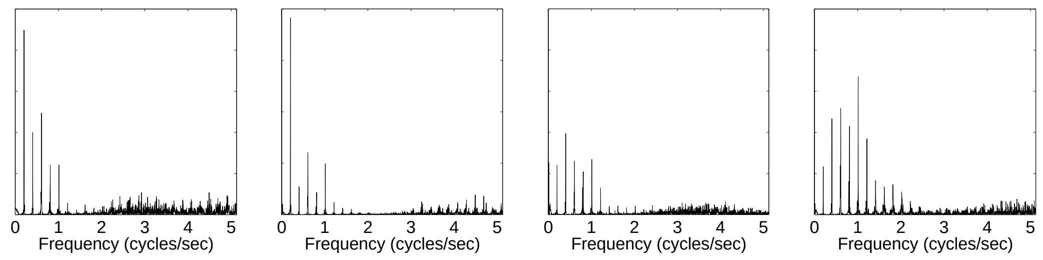
C. I/O behavior correlation cross virtual machines

Besides the variation and periodicity distinction of different benchmarks, we also observed some noticeable relevance characteristic between multiple cooperative virtual machines. For instance, we zoom the test result of LU program into 10 seconds which contains 100 samples and compare the variation detail of I/O behaviors of 4 computing nodes. As illustrated in Figure 5a, there are strong similarity between them. The only noticeable difference is the peak value of node1 at 3.7 second is not as high as three other ones and the intensity peak of node1 at 9th second appears about one sample period(100ms) later than three others. Similar phenomenon can be observed from other `NPB-MPI` benchmarks and front-tier and back-tier of `TPC-W` benchmark. In order to provide a more objective measurement of this similarity, we compare the I/O behavior periodograms of these cooperating VMs and try to match the peak frequencies of each virtual machine to indicate how strongly the periodicity is related across cooperating virtual machines. As exemplified from figure 5b to figure 5e, although the relative amplitude of the peaks vary across different VMs, the approximate match rate across all cooperative VMs is on average of more than 80%. We also calculate the cross-correlation coefficient and get the similar result. This high matching rate indicates that the periodicity of I/O behavior is shared across all VMs which cooperate together during their execution.

This phenomenon can be easily understood. For `NPB-MPI` test, several virtual machines were actually configured as a virtual cluster to undertake different threads of the same parallel program. These VMs communicate with each other to transfer intermediate data during the execution. Therefore, the network I/O behaviors of these VMs always exhibit strong dependency with each other. Similar situation also exists in `TPC-W` test since the web server and database server need to coordinate together to accomplish a customer request. This conformity could not only help us to enhance traditional predictors with precision and efficiency but also provide some possibility to optimize the virtual machine scheduling based



(a) I/O behaviors of LU program cross each computing node



(b) Periodogram of node1 (c) Periodogram of node2 (d) Periodogram of node3 (e) Periodogram of node4

Figure 5. Correlation of I/O behavior of cooperating VMs

on the correlation cross multiple VMs. We will elaborate our *cross-VMs* predicting mechanism in section IV.C and *VM-Grouping* scheduling optimization in section V.C.

D. Disk and network I/O

In above discussion, we did not explicitly distinct disk and network I/O. Actually, for NPB-MPI benchmark, only a little disk I/O operations are performed since reference data need to be loaded at the beginning of program execution. Most I/O operations are caused by network communication which transforms intermediate data cross cooperative virtual machines. This characteristic is very common in most network-based HPC applications. Compared to NPB benchmarks, the component of I/O operations for TPC benchmark is more complex. Since the web sever and database sever are deployed in individual VMs, the I/O operations of front-tier VM are almost completely caused by network communication while the back-tier VM needs to perform some disk I/O operations to load commodity and customer information from database. However, with the test running, the frequency of disk I/O operations of back-tier gradually reduced. This is because only a small quantity of commodity and customers information are deployed in database sever during our test and thus most of them can be cached into the memory. In practical use, the system might have no enough memory to cache all these data and periodic disk I/O operations would be unavoidable. In this case, we should cautiously distinguish these two kinds of I/O operations since they have totally different characteristic.

However, since the I/O intensity of disk access is much lower then the one of network communication in our study cases, we did not treat them separately in this paper. Actually, we think network I/O should have more influence to the performance of current virtualization-based applications since most of them are deployed through network. Most of current research works also proved that network latency is a critical issue that limits the perform of HPC applications under virtualized and cloud environment [24], [25]. On the other hand, compared to network transmission, the performance of disk access actually suffers much less from scheduling competition because of the existence of disk cache. Current research works about disk I/O mostly focused on the disk bandwidth allocation and Qos mechanism providing [7], [26]. Therefore, in this paper, we just concentrate our scheduling optimization on the promotion of network capacity and we will discuss this issue in detail in section V.

E. summary

In summary, our characterization finds that the I/O behavior of virtualization workloads have significant variability even at a coarse granularity. This variation actually reflects the running and communication characteristic of different applications. Some workloads such as NPB benchmarks exhibit strong periodicity because of their structural parallel programming paradigm. Therefore, it would be easy to accurately predict their I/O states and then provide some optimization for virtual machine scheduling. However, some other applications such as TPC-W servers have noticeable randomness for their I/O behavior. This irregularity could induce serious difficulty for our I/O state predicting and thus prevents further scheduling optimization. In the next section ,we will propose our methodology of I/O behavior predicting and introduce several typical predicting models and compare their performance for different applications.

IV. I/O Behavior Predicting

Currently, there have been a large number of studies that attempt to predict program behavior based on their execution information. Most of them classify the execution of the program into different states where each state is defined as a region of a program with relatively stable behavior [11], [27]. In this paper, we also adopt this way to predict I/O behavior of virtualization applications. We try to dynamically identify different I/O load states of workloads based on their I/O event frequency and then provide some guide for execution phase tracking and further scheduling decision. In this section, we first describe some important issues that affect the I/O behavior prediction and then introduce several online predictors which are applied to our study cases. We also discuss the behavior similarity between multiple cooperative virtual machines and propose a cross-VMs mechanism to enhance traditional predictors.

A. Configuration of temporal and spatial sampling

There are two important factors that affect the I/O behavior predicting: 1) the period of temporal sampling 2) the precision of spatial sampling. Ordinarily, fine-granularity temporal sampling could provide more variation details while coarse-granularity temporal sampling could filter noise more efficiently and reduce the overhead of information collecting and analyzing. In order to achieve a optimal configuration, we changed the temporal sampling period from 50ms to 500ms and compared the visibility variation of program's I/O behavior. Test results demonstrate that the detail of I/O state transition could be sufficiently exposed with the sampling period of 100ms while avoid inducing too much noise which could lead to some unnecessary state-transitions. Therefore, in this paper, we select a temporal granularity of 100ms as the sampling period.

Besides the temporal sampling period, the precision of spatial sampling is another pivotal factor that affects the prediction accuracy. In our test, the I/O load intensity is described by I/O event frequency value which is continuous on real number scope. Although some simple statistic predictors(such as *Last Value* and *Average(N)*) can directly work on these value sequences, we need to convert them into discrete state sequences before applying some history-based predictors such as discrete Markov model and run length encode(RLE) predictor. In this case, the converting precision should be sensitive enough to distinguish different I/O load states while avoid over-enlarging the capacity of state space. As illustrated in Figure 2 and Figure 4, the amplitude of fluctuation always keeps under 50times/100ms when the I/O frequency keeps in stable state. Therefore, in this paper, we encode history I/O event frequency values at a precision of 50 times/100ms (i.e., a frequency value of 25times/100ms will be converted to state(`1`) while 125times/100ms will be converted to state(`3`)). We have experimentally verified that this configuration provides the best trade-off between sensitivity of state transition and predictor efficiency.

B. Predictors comparison and selection

Simple Statistics predictor: The most basic predictor we considered is the *Last Value* predictor. A Last Value predictor assumes that the program executes in a stable phase and that the current behavior will repeat. The prediction for the next interval is simply the last measured value. More sophisticated predictors use histories of metric values to smooth out noise and isolated peaks. A typical and most common-used instance is the exponentially weighted moving average (*EWMA*) predictor. Compared to normal *Average(N)* predictor which chooses the average over the last N values, EWMA places more emphasis on the most recent data. An EWMA prediction for the value is computed as:

$$(1-\alpha)x_n + \alpha\bar{x}_{n-1}$$

where x_n is the *n*-th metric value, $\bar{x}_{n-1}$ is the EWMA over the first *n-1* values, and $\alpha \leq 1$ is the filter constant that determines the relative weight of older values compared to more recent ones. Simple statistics predictors are always used in reactive systems and the time-delay is an inherent defect of them. Therefore, although they can achieve well precision when applied to some little-variation cases, they cannot perceive the state transition promptly.

Markov predictor: Markov model is a classic predictor which has been widely used in computer architecture to predict both prefetch address and branch. The basic idea behind it is the next state of the system is only related to the last set of states. A Markov chain is a sequence of random variables X_1, X_2, X_3, ... with the Markov property, namely that, given the present state, the future and past states are independent. The first step for inferring the Markov model is to establish the state space which in this paper is determined based on the I/O event frequency. The second step is to construct the rate matrix - the matrix containing the rates for transition from one state to another - which is calculated based on history state sequence. This transition probability matrix is finally used to predict the state of coming duration.

An important defect of standard Markov model is that it is not sensitive to abrupt state changing. Its prediction is based on the transition matrix which comes from the specific state sequence of last intervals. For some applications such as *LU* and *CG*, there are many sections of stable behavior interspersed with periodic abrupt state transitions. In this case, the Markov model could make incorrect prediction.

Run Length Encode predictor: Run Length Encoding (*RLE*) predictor [11] is another classic predictor used in computer science field. It is commonly used to compress continuous stable states and predict abrupt but periodic state-transitions. The basic idea behind this predictor is using a run-length encoded version of the history to index into a prediction table. The table reference is a hash of the state value and the number of times that this state has occurred in a row. The lower bits of the hash function provide an index into the table while the higher bits provide a tag. Each predicting period, the tag will be checked. If there is a match, the value stored in the table entry will provide a prediction for the state of coming duration. If the tag match fail, the prior state is assumed to be the prediction result for next execution period which means no state-transition will happen. The hash table will be updated in two cases: (1)state value changes which means state-transition happens; (2)tag match which means a table entry need to be updated. In the first case, we directly insert a new entry into the hash table since we want to predict this state-transition before its next happening. Execution intervals where the same state continuously occurs will not be stored into the table since they will be correctly predicted as ”last phase ID” when the table lookup missed. In the second update case where a tag match happens, we update the value of this table entry because the observed run length may have potentially changed.

Compared to standard Markov predictor, Run-length predictor yields more efficient history encoding for programs with successive intervals of stable behavior. It could efficiently compress the data size of continuous stable state while keeping enough sensitivity to phase transition. Therefore, it would be more appropriate for those applications which always stay in stable state while companied by periodic but abrupt phase changing.

Predictors Comparison In order to evaluate the applicability of different predictors for virtualized applications, we test the precision of four different predictors including: 1)Last Value; 2)EWMA(10) filter ($\alpha = 0.3$); 3)Order(1) Markov predictor; 4)Run-length encoded predictor. For both Markov and RLE predictors, we apply a history size of 100. We test the precision of predictors with both entire state predicting and state-transition predicting. The entire precision is tested through the predicting hit rate which demonstrates the matching degree between prediction results and the actual ones. The precision of state transition depicts the sensitivity and temporal accuracy of predictors for state changing. A prediction of state transition is thought to be correct only when both origin and coming state match the actual ones. We apply all these predictors on five different workloads including `LU, CG, SP, EP` programs from `NPB-MPI` benchmark and `TPC-W` benchmark. For `NPB-MPI` benchmarks, we only test the I/O behavior of primary node because all other nodes have similar results. For `TPC-W` benchmark, we test the I/O state variation of front-tier and back-tier VMs respectively. All results are illustrated in Table I and Table II (standard Markov and RLE predictors are referred to as `Markov-S` and `RLE-S` while corss-VM enhanced ones are referred to as `Markov-C` and `RLE-C` respectively).

TABLE I.
ENTIRE PREDICTING PRECISION OF STATE VALUE

	LU	SP	CG	EP	TPC1	TPC2
LastValue	85.9%	49.1%	89.9%	99.6%	38.0%	40.1%
EWMA	88.0%	65.4%	84.5%	99.6%	72.6%	75.1%
Markov-S	94.0%	84.8%	90.6%	99.6%	79.6%	70.8%
RLE-S	90.1%	75.2%	89.6%	99.6%	68.0%	70.0%
Markov-C	95.2%	87.0%	93.1%	99.6%	86.3%	74.1%
RLE-C	92.4%	80.7%	90.6%	99.6%	69.6%	73.3%

TABLE II.
PREDICTING PRECISION OF STATE-TRANSITION

	LU	SP	CG	EP	TPC1	TPC2
LastValue	N/A	N/A	N/A	N/A	N/A	N/A
EWMA	28.1%	19.0%	34.0%	N/A	18.4%	16.2%
Markov-S	48.0%	48.2%	49.6%	N/A	46.6%	46.7%
RLE-S	63.3%	58.7%	65.9%	N/A	58.9%	57.4%
Markov-C	58.9%	57.1%	62.4%	N/A	54.1%	52.8%
RLE-C	78.7%	68.8%	78.7%	N/A	61.0%	61.2%

As illustrated in Table I and Table II, different predictors exhibit significantly performance distinction. As expected, Last Value predictor has the worst precision for both entire state predicting and state-transition predicting because of its reactive manner. Compared to Last Value predictor, another simple statistic predictor EWMA has better performance. However, its state-transition predicting accuracy is much worse then the ones of Markov and RLE predictors(about 23% and 36% lower respectively). This is because simple statistic predictors always use immediate past behavior as representative of future behavior and thus cause serious time delay. Therefore, this kind of predictors would not be suitable for virtualization environment where programs’ hebavior always exhibit strong variability during their execution.

Standard Markov predictor has better entire-state predicting precision then RLE predictor since the latter one has stronger history persistence which could cause some erroneous prediction of state-transition. However, its sensitivity of state changing also leads to a better accuracy of state-transition predicting. The average precision of standard RLE predictor for state-transition predicting is 60.8%, about 13 percents higher than the one of standard Markov predictor(47.4%). This difference is more noticeable in those cases where programs have stronger periodicity and more long-term stable phases which followed by abrupt I/O-intensity changing such as `LU` and `CG` benchmarks. Since our purpose of I/O behavior predicting is tacking different execution phases of applications and providing direction for virtual machine scheduler, incorrect state-transition prediction could cause false phase tracking result and then leads to irrational scheduling adjustment. Therefore, the precision of state-transition predicting is actually more important than the one of entire state predicting. In this paper, we apply RLE model as our predictor since it has the best precision of state-transition predicting in our study cases.

C. Cross-VMs predicting

In section III.C, we have exploited the fact that periodicity tends to be shared across multiple cooperative virtual machines. These VMs always take I/O state transition simultaneously although their I/O intensities might have some difference. Therefore, in this paper, we construct a cross-VMs mechanism through integrating the history information among multiple virtual machines to promote the precision and efficiency of traditional predictors. In our design, we gather all prediction results from multiple cooperative virtual machines at each predicting period. If most of them predict a coming state-transition, we will revise the prediction results of those VMs which predict to keep in a stable state and update their prediction values to the last state-transition result that actually happened(e.g. a prediction result of state-keeping around current state(2) will be revised to a state-transition to state(3) if last state-transition happened at state(2) is targeting to state(3)). If there are only two cooperative virtual machines, we will trace their predicting accuracy for state-transition and make prediction decision according to the one that has better precision history.

We evaluate this cross-VM mechanism by applying it to Markov and RLE predictors. Test result demonstrates that the precision of state-transition predicting got an obvious promotion. The average precision improvement of all five benchmarks with two predictors is about 9.2%. On the other hand, we also observe some difference between the precision improvements cross different applications. The average precision improvements of Markov-C and RLE-C are 10.9% and 12.8% when they were applied to `NPB-MPI` benchmark, while the same ones of `TPC-W` benchmark are only 6.8% and 3.0%. This is because the cooperative virtual machines running NPB programs almost have no other I/O operations besides their inter-VM communications through network. In this case, virtual machine's I/O behavior significantly depends on each others'. By comparison, the I/O behavior of `TPC-W` virtual machines are much more complex. Besides the network communication between two VMs, the web server also needs to interact with clients while the back-tier sever needs to access database. This I/O operation variety increases the randomness of I/O behavior and thus reduces the consistency between two `TPC-W` virtual machines. In this case, the precision promotion benefits from cross-VM optimization is much less noticeable than the one of `NPB` tests.

We also tried to use the prediction result of one virtual machine to predict the I/O states of other cooperative VMs. However, for some applications, the I/O load intensity could vary over different VMs although they have similar periodicity and state-transition characteristic. In this case, simply using the I/O intensity of one virtual machine to predict other one's could cause serious mean error. Therefore, in this paper, we only apply the cross-VMs mechanism to enhance the predicting of state-transition.

D. summary

In this section, we introduce several typical online predictors and discuss several important issues about I/O behavior predicting. Our tests proved that the I/O load state of applications could be efficiently predicted if using proper predicting model and configuration. With these prediction results, we could perceive coming I/O load variation promptly and then provide some directions for virtual machine scheduling to reduce the performance degradation caused by disordered scheduling competition. In the next section, we will propose our conception of virtual machine scheduling optimization based on asymmetric virtual machine scheduler and I/O behavior predicting and describe several important mechanisms that support our design.

V. Scheduling optimization based on I/O behavior characterizing

Current researches have proved that the scheduling competition cross I/O and computing jobs is an important issue that causes the performance degradation of virtual machine system. In order to address this problem, we have applied a multi-core dynamic partitioning method to realize a asymmetric scheduling framework under Xen-3.1.0 virtual machine. This approach tried to isolate the negative influence to the computing jobs by scheduling I/O-intensive VCPUs on separate processor core which employs frequent context switch and event-prior preemptive scheduling algorithm. Although this method could address the performance interference issue in some degree, it also reduced the efficiency of scheduler because it just applied a simple reactive way to handle the I/O load variation of virtual machines: a VCPU migration is always instantly performed when an I/O event arriving. Since frequent VM migration could cause serious overhead especially when it happens cross different sockets, we should schedule VMs more cautiously to protect the scheduler efficiency while keeping enough sensitivity to the variation of I/O load intensity. In this case, accurate I/O state predicting and execution phase tracking would be very helpful for the scheduler to make correct scheduling decision. In last section, we have described our methodology of I/O state predicting and proposed several principles of predictor evaluation and selecting. In this section, we will further introduce the framework of our asymmetric virtual machine scheduler based on Xen-4.1.0 virtual machine and discuss how to perform an optimal scheduling decision based on the result of I/O behavior predicting.

A. Asymmetric scheduler based on Xen virtual machine

4.0 and earlier versions of Xen virtual machine system employed traditional global-symmetric scheduler to manage VMs. Both the main scheduler framework and specific scheduling algorithm (such as Credit or SEDF) need to be modified to realize the asymmetric scheduling. Currently, the latest Xen-4.1 system provides a `CPUPOOL` feature which allows several individual cpupools coexist in the system. Each cpupool has specific quantity of CPU resources which are managed by its own scheduler and virtual machines can be migrated cross different cpupools. Although only Credit scheduler completely supports the CPUPOOL mechanism currently, most important management functions have been realized in Xen-4.1.0 system, including cpupool creating and destroying, processor resource adding and removing, VM migrating and etc. Based on this framework, asymmetric scheduling can be achieved in a more modular and graceful way.

As illustrated in Figure 10, we increase a event-prior round-robin(RR) scheduler to Xen virtual machine and use it to manage a specific *I/O cpupool*. This scheduler applies frequent context switch to speed up I/O operations and schedules VMs in sequence according to their orders in scheduling queue. Beside I/O cpupool, there is another *computing cpupool* which applies default Xen-Credit scheduler to undertake virtual machines that perform less I/O operations. During system running, we trace the I/O load states of each virtual machine dynamically and consider to migrate them between I/O and computing cpupools if we predict their I/O load intensity will significantly change. Different from our past research work [17], the VM migration will only be performed when we believe the entire system performance could benefit from this scheduling adjustment rather than a single I/O operation could be accelerated. Therefore, this decision will be made based on the history analysis and prediction of application's I/O behavior. We will detail our methodology of VM migration decision making in next section.

B. Execution phase tracking and VM dynamic migration

As illustrated in section III, the I/O load state of most applications always keep on stable state while transient variations only happen occasionally. If we can distinguish different execution phases of applications based on their I/O load intensity and schedule VMs on I/O cpupool only when they are in heaviest I/O load state, we can maximize the benefit from I/O scheduling and reduce the overhead of VM migration efficiently. To achieve this, we need to specify a proper threshold to differentiate heavy and low I/O load intensity and setup a proper condition for VM migration.

Threshold of I/O load intensity: This threshold is used to decide whether a VM has been in heavy I/O load state and should be scheduled on I/O cpupool. Since the I/O load intensity of different programs have significantly difference and could vary over time, static threshold would be inappropriate. We have introduced that the principle of our execution phase tracking is maximizing the benefit of I/O speeding up while limit the frequency of VM migration, therefore the threshold setting should be able to maximize the I/O load intensity distinction between different execution phases. To achieve this object, we borrowed a idea of segmentation from digital image processing. The purpose of image segmentation is to

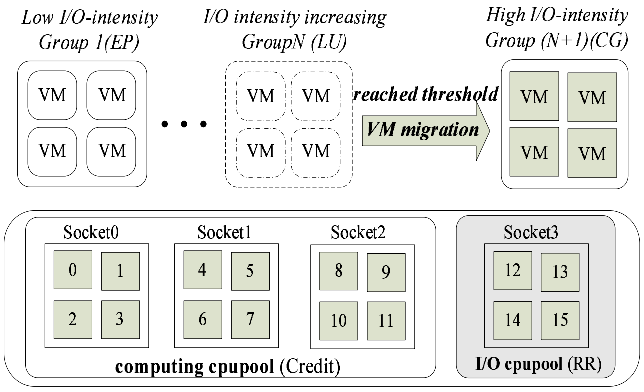

Low I/O-intensity Group 1(EP)
I/O intensity increasing GroupN (LU)
High I/O-intensity Group (N+1)(CG)
VM
reached threshold
VM migration
Socket0
Socket1
Socket2
Socket3
0 1 2 3
4 5 6 7
8 9 10 11
12 13 14 15
computing cpupool (Credit)
I/O cpupool (RR)

Figure 6. Asymmetric scheduling model of Xen virtual machine

distinguish the object from background based on their characteristic difference such as gray scale. In our study, transient I/O state transitions could also be considered as isolated objects that spread cross the whole execution process which could be considered as the background. The difference is the characteristic distinction is reflected by the I/O load intensity rather than gray value in digital image processing.

In our study, we applied OTSU (Maximization of interclass variance) algorithm [28] to calculate the threshold dynamically. Compared to other common-used segmentation algorithms such as adaptive iteration and morphology method, this algorithm has the best adaptability and can get best results for most of our study cases. The original OTSU algorithm tries to select an optimum threshold by maximizing the interclass variance in a digital image based on the histogram and probability of pixel's gray value. In our study, we replace the gray value of pixels by I/O event frequency and calculate the interclass variance based on the histogram of I/O intensity history. The revised OTSU algorithm applied in this paper can be formulated as follow:

Let the samples of a given I/O intensity history be represented in m intensity levels (I/O event frequency value scope) [0, 1, 2... m]. The number of samples at level i can be denoted by n_i.

Total number of samples:

$$N = \sum_{1}^{m} n_i.$$

Probability distribution of different I/O intensity levels:

$$P_i = \frac{n_i}{N}$$

Separate history samples into two classes $C_0 = (1 \sim k)$ and $C_1 = (k + 1 \sim m)$ which represent background and objects respectively (low and heavy I/O execution phases in our study) by a threshold k. In this case, the probabilities of class occurrence of C_0 and C_1 are given by:

$$\omega_0 = \sum_{k}^{i=1} P_i = \omega_k$$
$$\omega_1 = \sum_{m}^{i=k+1} P_i = 1 - \omega_k$$

and the class mean values of C_0 and C_1 are given by:

$$\mu_0 = \sum_{i=1}^{k} \frac{i \times P_i}{\omega_0} = \frac{\mu_k}{\omega_k}$$
$$\mu_1 = \sum_{i=k+1}^{m} \frac{i \times P_i}{\omega_1} = \frac{\mu - \mu_k}{1 - \omega_k}$$

where

$$\mu_k = \sum_{i=1}^{k} i P_i$$

and

$$\omega_k = \sum_{i=1}^{k} P_i$$

The total mean value of I/O intensity history is:

$$\mu = \sum_{i=1}^{m} i P_i = \omega_0 \mu_0 + \omega_1 \mu_1$$

and we can easily verify the following relation for any choice of k:

$$\omega_0 + \omega_1 = 1$$

Finally, the interclass variance σ_k^2 are given by:

$$\sigma_k^2 = \omega_0(\mu_0 - \mu)^2 + \omega_1(\mu_1 - \mu)^2 = \omega_0\omega_1(\mu_1 - \mu_0)^2 = \frac{[\omega_k \times \mu - \mu_k]^2}{\omega_k \times [1 - \omega_k]}$$

We iterate threshold k from 1 to m and find the result which can maximize the interclass variance σ_k^2.

An important factor affecting the threshold calculation precision is the history size. Larger history size could better reflect the entire characteristic of program while smaller one could detect more detail of variation. We compared the effect of different history size and finally set it to 200 in this study. This time length has been able to cover several complete iteration periods in most cases while avoid inducing too much noisy which could lead to unnecessary phase transition. We applied this algorithm to our study cases and got ideal result. For samples of LU, SP and CG programs in Figure 3, the I/O intensity thresholds are 97, 69 and 247 respectively. These results have been able to tell transient execution phases from stable execution process clearly. In order to keep enough sensitiveness to the variation of applications, we recalculate the threshold each 50 sample periods. Test result demonstrates that the time consumption for each threshold update is less than 65 us in our test bed which only causes very slight influence to the scheduler efficiency. Actually for those applications which have well periodicity and stable I/O intensity(such as NPB benchmarks), the threshold result will always keep in a stable scope during the entire execution. For TPC-W benchmark, this threshold could need more frequent update because of the strong variation of application's I/O intensity.

VM migration condition: This condition is used decide whether we should treat the coming period as a new execution phases and perform a VM migration to adapt to the variation of application I/O intensity. In our study, we apply a weaker condition for migrating VM to I/O cpupool and a stricter condition for reverse operation. Each time a VM's prediction result exceeds the I/O intensity threshold, we will migrate it to the I/O cpupool immediately. However, only when a VM's practical I/O intensity is lower than the threshold in three continuous sampling periods, we will check whether its I/O prediction result of coming period is also under the threshold. If so, this VM will be migrated back to the computing cpupool. This setting makes sure that heavy I/O execution phases could always be perceived and optimized in the first place although computing performance could suffer from I/O scheduling. In order to evaluate the influence of scheduling computing jobs on I/O cpupool, we tested the performance variation of EP program when scheduling it with different context switch frequency. Test result demonstrates that the performance degradation is only about 8.7% when the period of scheduling switch reduced from 10ms to 0.5ms. This degradation is much less than the performance reduction caused by I/O latency increasing, especially for those communication-intensive applications(eg. the performance of CG program reduced more than 40% when average network latency increases from 0.5ms to 10ms). Therefore, we think weaker entry condition for I/O-intensive execution phase should be a rational choice for our study. Actually, this selection could be more reasonable for network-based large-scale system such as science cloud where a large number of virtualized computing nodes are integrated through network to perform parallel applications. In this case, the benefit from network capacity improvement would be more notable and more sufficient to offset the negative influence caused by computing performance degradation.

In summary, with the I/O intensity threshold and VM migration condition, we divide the entire running process of applications into different phases. We hope this temporal dividing could isolate the execution of I/O and computing jobs and then reduce the performance degradation caused by scheduling competition.

C. Scheduling optimization for cooperative VMs

Besides the dynamic VM migration, we also try to optimize the scheduling strategy based on the correlation between different virtual machines. As illustrated in section III, there are strong similarity between the I/O behaviors of cooperative VMs and this correlation is mainly caused by inter-VM communications. In section IV, we have used this characteristic to promote the prediction precision of I/O load state. In this section, we will discuss how to use this correlation to optimize the scheduling strategy for cooperative virtual machines.

Grouping migration: Cooperative virtual machines usually undertake different threads of parallel program or commercial server. They always perform intensive I/O operations to communicate with each other when synchronizing computing result or acquiring data. However, serious CPU competition and disordered VM scheduling could prevent inter-VM communication finishing in time and then cause serious synchronization latency or response timeout. Currently, some scholars have proved that the co-scheduling strategy is an efficient way to address the synchronization problem under virtual machine system [29]. Therefore, in our study, we also apply similar mechanism to optimize the VM migration. As illustrated in Figure 6, we divide several VMs into the same group if they have cooperative relationship (E.g. four VMs running LU program will be put into `GroupN` while four other VMs running CG program will be put into `Group(N+1)`). VMs belong to the same group will always be migrated cross different cpupools concurrently. This mechanism guarantees cooperative VMs could always benefit from I/O scheduling simultaneously when they are both in I/O-intensive load state and thus maximizes the efficiency of inter-VM communication.

However, this grouping migration strategy could also increase the difficulty of balancing the resource allocation between I/O and computing cpupools since the amount of VMs in both cpupools will vary rapidly. In this case, the resource capacity of I/O and computing cpupools needs to be adjusted more promptly to achieve an optimal configuration. In this paper, we just assigned fixed-quantity of CPU resource to I/O and cpupools to explain our conception. In future works, we will keep working on this issue and try to find a way to balance the performance between I/O and computing jobs efficiently.

Cpupool allocation optimization: Besides VM migration strategy, cross-VM correlation also influences the topology of cpupool allocation. Currently, virtual network drivers prefer to apply memory sharing mechanism(such as `grant table` in Xen) to speed up the network communication between multiple virtual machines when they are deployed in the same physical platform. In this case, the efficiency of data transmission could get notable promotion if the sharing L2 or L3 cache could be well utilized. Therefore, in our design, we always try to deploy I/O and computing cpupools in individual processor sockets to make sure cooperative VMs could take fully advantage of cache sharing when they are performing inter-VM communication. Although this isolation between I/O and computing cpupools could increase the overhead of VM migration because of cache flushing, this overhead can be efficiently controlled if we limit the frequency of VM migration into a proper range. We tested the performance of EP program when 4 virtual computing nodes periodically migrate cross two individual cpupools which both employ Credit scheduler but deployed on two different sockets. Compared to static scheduling, the performance degradation is less than 5% if the migration period is longer than 300ms. Since the VM migration frequency is strictly controlled based on the I/O intensity threshold and migration condition, we think deploying I/O and computing cpupools in separate sockets should be reasonable in our study.

It needs to point out that we only group VMs statically in this study. This method needs user to provide grouping information explicitly before virtual machines are deployed into system. However, sometimes sever provider could not get enough information from customer or user could change their application deployment during system running. In this case, some automatic recognizing mechanism would be necessary to dynamically detect the cooperation relationship between different virtual machines. Although we haven't investigated this issue in this paper, we believe some simple pattern identification methods based on I/O behavior characterizing, such as periodicity comparison or correlation analysis would be sufficient to address this problem.

D. summary

Although we haven't finished our design, preliminary test results demonstrate that this asymmetric scheduling framework could efficiently reduce the performance degradation caused by interference between I/O and computing jobs. Some communication-intensive HPC programs could get significant performance improvement when running in scheduling competition environment(e.g. the Mops of CG program improved from 164.48 to 264.84 with 4 virtual computing nodes running on 2 processor cores). However, compared to NPB programs, TPC-W benchmark gets much less performance promotion. The average latency of user request reduced less than 4% when 100 parallel sessions accessed the sever simultaneously. This is because the CPU competition for TPC-W benchmark is much slighter than NPB programs during out test. The average CPU usages of front-tier and back-tier VMs of TPC-W sever are only 60% and 20% respectively while NPB programs always try to exhaust their CPU time. Therefore, the I/O performance degradation of TPC-W server caused by CPU competition actually not as serious as the one of NPB benchmark. Furthermore, the I/O behavior of TPC-W sever has more randomness and variation than NPB programs and thus decrease the accuracy of I/O state prediction which then causes much more unnecessary VM migrations. This result demonstrates that our approach could be more suitable for the cases where the performance of computing and I/O jobs need to be well balanced and the application I/O behavior has better regularity.

We are now still working on the scheduler design, especially how to balance the resource capacities between I/O and computing cpupools to achieve an optimal configuration. Other issues such as VCPU migration optimization and possible VM scheduling coordination cross multiple physical nodes are also in study now. We will describe more details about these issues after we get complete analysis results.

VI. Related Works

Currently, the behavior analysis of virtualization applications at fine granularity has not been thoroughly carried out. Most of current research work about virtual machine system concentrated on the performance evaluation and scheduler optimization. E.g., Ongaro [30] *et al.* explored the impact of a VM scheduler for various combinations of scheduling features over multiple DomUs running different types of applications. Cherkasova *et al.* studied the impact that three different schedulers, Borrowed-Virtual-Time(BVT), Credit and SEDF schedulers, have on the throughput of different I/O-intensive benchmarks [31]. Govindan *et al.* [32] proposed a communication-aware VM scheduling mechanism to improve the performance of high throughput network intensive workloads. Hwanju Kim [33] *et al.* analyzed the boost/tickle mechanism of Xen credit scheduler, and proposed a task-aware virtual machine scheduling mechanism based on inference techniques using gray-box knowledge to improve the I/O performance of mixed workloads. All of these works demonstrated a fact that the performance interference between different applications and the mismatch between resource allocation and application behavior always caused serious performance degradation and significantly reduced the system efficiency. However, current optimization for virtual machines system largely focused on the scheduling algorithm optimizing or I/O architecture adjustment based on application characteristics [5], [6], [7], [8]. They could not overcome the resource requirement contradiction between I/O and computing workloads which is the inherent defect of traditional symmetric scheduler. Since more and more complex sever applications are consolidated into virtulization platform, resource allocation control at finer granularity and more flexible adaptive scheduler framework become more necessary for future virtual machine system design.

Application behavior analysis and prediction has been a common and efficient way to improve the efficiency of hardware system or optimize the performance of traditional programs. E.g., D. Joseph*et al.* implemented Markov model in hardware to predict the prefetch addresses [9] and Chen *et al.* applied the same way to predict branches [10]. Timothy Sherwood *et al.* presented a unified profiling architecture to capture and classify phase-based program behavior on the largest of time scales [11]. Duesterwald *et al.* studied the time-varying behavior of programs using metrics derived from hardware counters on IBM Power architectures and explore the potential of incorporating prediction into adaptive systems [12]. Coskun *et al.* investigated how to use predictors for forecasting future temperature and workload dynamics, and propose proactive thermal management techniques for multiprocessor system-on-chips(MPSoCs) [13]. There are still many other related works applying application behavior analysis and predicting to promote the system performance and efficiency. However, the fine-grain characterizing and prediction of workloads under virtualized environment has not been widely studied. Therefore, in this paper, we tried to propose a runtime behavior characterizing and predicting methodology for virtualization applications. We hope these analysis could provide some inspirations and help for future intelligent and adaptive virtual machine scheduler design.

VII. Conclusion and Future Work

In order to exploit the methodology of dynamic behavior analysis for virtualization workloads, this paper proposes a runtime characterizing and predicting approach for virtual machine environment. We characterize the time-varying I/O behavior of several typical virtualization workloads and explore their predictability using several online predictors. Test results demonstrate that the I/O state variation of applications can be efficiently captured and predicted if using proper predicting model and configuration. Based on this result, we further investigate the possibility of virtual machine scheduler optimization based on I/O behavior characterizing. We discuss several important issues including I/O computing jobs isolation through asymmetric scheduling, VM dynamic migration based on execution phase tracking and scheduling optimization for cooperative virtual machines. Preliminary test results demonstrate that this method could efficiently decrease the performance degradation caused by I/O and computing competition.

Currently, more and more traditional server applications are consolidated into virtualization environment to reduce cost and increase flexibility. In this case, assigning adequate computing resource to specific applications at proper timing has become an important prerequisite for system optimization. Because of the diversity and strong variation of applications, accurately characterizing and predicting the runtime behavior of workloads would be necessary for correct scheduling decision making. In this paper, we characterize the I/O behavior of several typical virtualization workloads and preliminarily discuss the possibility of optimizing virtual machine scheduler using asymmetric scheduling. In future works, we will further complete our design and exploit other important issues including resource allocation balance between I/O and computing jobs and possible VM scheduling coordination cross multiple physical nodes. We hope this methodology could break the restriction of traditional scheduler design and provide more space for virtual machine scheduling optimization.

References

[1] Y. Koh, R. Knauerhase, P. Brett, M. Bowman, null Zhihua Wen, and C. Pu, “An analysis of performance interference effects in virtual environments,” *IEEE International Symmposium on Performance Analysis of Systems and Software*, pp. 200–209, 2007.

[2] P. Apparao, R. Iyer, X. Zhang, D. Newell, and T. Adelmeyer, “Characterization & analysis of a server consolidation benchmark,” in *Proceedings of the fourth ACM SIGPLAN/SIGOPS international conference on Virtual execution environments(VEE ’08)*, 2008, pp. 21–30.

[3] P. Padala, X. Zhu, Z. Wang, S. Singhal, and K. Shin, “Performance evaluation of virtualization technologies for server consolidation,” *HP Laboratories Technical Report*, 2007.

[4] N. E. Jerger, D. Vantrease, and M. Lipasti, “An evaluation of server consolidation workloads for multi-core designs,” in *IISWC ’07: Proceedings of the 2007 IEEE 10th International Symposium on Workload Characterization*.

Washington, DC, USA: IEEE Computer Society, 2007, pp. 47–56.

[5] G. Liao, D. Guo, L. Bhuyan, and S. R. King, “Software Techniques to Improve Virtualized I/O Performance on Multi-core Systems,” in *Proceedings of the 4th ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS’08)*, Nov. 2008, pp. 161–170.

[6] K. K. Ram, J. R. Santos, Y. Turner, A. L. Cox, and S. Rixner, “Achieving 10 Gb/s Using Safe and Transparent Network Interface Virtualization,” in *Proceedings of the 5th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments (VEE’09)*, Mar. 2009, pp. 61–70.

[7] S. R. Seelam and P. J. Teller, “Virtual I/O Scheduler: a Scheduler of Schedulers for Performance Virtualization,” in *Proceedings of the 3th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments (VEE’07)*, June 2007, pp. 105–115.

[8] A. .Menon, A. L. Cox, and W. Zwaenepoel, “Optimizing Network Virtualization in Xen,” in *Proceedings of the USENIX 2006 Annual Technical Conference (USENIX’06)*. Berkeley, CA, USA: USENIX Association, June 2006, pp. 2–2.

[9] D. Joseph and D. Grunwald, “Prefetching Ssing Markov Predictors,” in *Proceedings of the 24th Annual International Symposium on Computer Architecture*, June 1997.

[10] I.-C. Chen, J. T. Coffey, and T. N. Mudge, “Analysis of Branch Prediction via Data Compression,” in *Proceedings of the 7th International Conference on Architectural Support for Programming Languages and Operating Systems*, Oct. 1996, pp. 128–137.

[11] S. S. T. Sherwood and B. Calder, “Phase Tracking and Prediction,” in *Proceedings of the 30th International Symposium on Computer Architecture, ISCA-30*, June 2003.

[12] E. Duesterwald, C. Cascaval, and S. Dwarkadas, “Characterizing and predicting program behavior and its variability,” in *Proceedings of the 12th International Conference on Parallel Architectures and Compilation Techniques*, ser. PACT ’03, 2003.

[13] A. Coskun, T. Rosing, and K. Gross, “Utilizing predictors for efficient thermal management in multiprocessor socs,” *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 28, no. 10, pp. 1503–1516, 2009.

[14] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, “Xen and the Art of Virtualization,” in *Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP’03)*, Oct. 2003, pp. 164–177.

[15] A. Kivity, Y. Kamay, D. Laor, U. Lublin, and A. Liguori, “KVM: The Linux Virtual Machine Monitor,” in *Proceedings of the 2007 Ottawa Linux Symposium (OLS ’07)*, Aug. 2007.

[16] A. I., A. J.M., H. A.M., K. R., and M. V., “An Analysis of Disk Performance in VMware ESX Server Virtual Machines,” in *Proceedings of the 6th Workshop on Operating System and Architectural Support for the on Demand IT InfraStructure (WWC-6)*, Oct. 2003, pp. 65–76.

[17] Y. Hu, X. Long, J. Zhang, J. He, and L. Xia, “I/O Scheduling Model of Virtual Machine Based on Multi-core Dynamic Partitioning,” in *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, ser. HPDC ’10, 2010.

[18] “NPB,” http://www.nas.nasa.gov/Resources/Software/npb.html.

[19] “TPC-W,” http://www.tpc.org/tpcw/.

[20] “TPC-W-NYU,” http://www.cs.nyu.edu/pdsg/.

[21] “The jboss application server,” http://www.jboss.org.

[22] “Mysql,” http://www.mysql.com.

[23] “TPC-W-UVA,” http://www.cs.virginia.edu/ th8k/downloads/.

[24] Q. He, S. Zhou, B. Kobler, D. Duffy, and T. McGlynn, “Case study for running hpc applications in public clouds,” in *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, ser. HPDC ’10, 2010, pp. 395–401.

[25] J. E. Simons and J. Buell, “Virtualizing high performance computing,” *SIGOPS Oper. Syst. Rev.*, vol. 44, pp. 136–145, December 2010.

[26] M. Kesavan, A. Gavrilovska, and K. Schwan, “On disk i/o scheduling in virtual machines,” in *Proceedings of the 2nd conference on I/O virtualization*, ser. WIOV’10, 2010, pp. 6–6.

[27] A. S. Dhodapkar and J. E. Smith, “Managing Multiconfiguration Hardware via Dynamic Working Set Analysis,” in *Proceedings of the 29th International Symposium on Computer Architecture, ISCA-29*, May 2002.

[28] N. OTSU, “A thresholding selection method from gray-level histogram,” *IEEE Transactions on Systems, Man and Cybernetics*, vol. 9, no. 1, pp. 62 – 66, 1979.

[29] C. Weng, Z. Wang, M. Li, and X. Lu, “The hybrid scheduling framework for virtual machine systems,” in *Proceedings of the 2009 ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*, ser. VEE ’09, 2009.

[30] D.Ongaro, A. L. Cox, and S. Rixner, “Scheduling I/O in Virtual Machine Monitors,” in *Proceedings of the 4th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments (VEE’08:)*, Mar. 2008, pp. 1–10.

[31] L. Cherkasova, D. Gupta, and A. Vahdat, “Comparison of the Three CPU Schedulers in Xen,” *SIGMETRICS Perform. Eval. Rev.*, vol. 35, no. 2, pp. 42–51, 2007.

[32] S. Govindan, A. R. Nath, A. Das, B. Urgaonkar, and A. Sivasubramaniam, “Xen and Co.: Communication-aware CPU Scheduling for Consolidated Xen-based Hosting Platforms,” in *Proceedings of the 3th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments (VEE’07)*, June 2007, pp. 126–136.

[33] H. Kim, H. Lim, J. Jeong, H. Jo, and J. Lee, “Task-aware Virtual Machine Scheduling for I/O Performance,” in *Proceedings of the 5th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments (VEE’09)*, Mar. 2009, pp. 101–110.

Yanyan Hu born in 1984, is currently a Ph.D. candidate of computing science at Beihang University, Beijing, China. He received his BS and MS degrees from Beihang University and Beijing University of technology in 2004 and 2007, respectively. His current research interests include operating system, virtualization and cloud computing.

Xiang Long born in 1963. Ph.D. He is currently professor and Ph. D. supervisor at Beihang University. His current research interests include embedded systems design, operating system and network and information security.

Jiong Zhang born in 1976. Ph. D. He is now a lecturer of Beihang University. His current research interests include embedded systemsoperating system and virtualization.