2011 International Conference on Parallel Processing

Adaptive Disk I/O Scheduling for MapReduce in Virtualized Environment

Shadi Ibrahim, Hai Jin, Lu Lu, Bingsheng He*, Song Wu

Cluster and Grid Computing Lab
Services Computing Technology and System Lab
School of Computer Science and Technology
Huazhong University of Science and Technology
Wuhan, 430074, China
{shadi, hjin}@hust.edu.cn

**School of Computer Engineering*
Nanyang Technological University
Singapore, 639798
bshe@ntu.edu.sg

***Abstract*—Virtual machine (VM) interference has long been a challenging problem for performance predictability and system throughput for large-scale virtualized environments in the cloud. Such interferences are contributed by intertwined factors including the application's type, the number of concurrent VMs, and the VM scheduling algorithms used within the host. Since MapReduce has become an important data processing platform in the cloud, we investigate the impact of disk schedulers in Hadoop. Interestingly, our experimental results report a noticeable variation of the Hadoop performance between different applications when applying different disk pairs' schedulers in both the hypervisor and the virtual machines. Furthermore, a typical Hadoop application consists of different interleaving stages, each requiring different I/O workloads and patterns. As a result, the disk pairs' schedulers are not only sub-optimal for different MapReduce applications, but also sub-optimal for different sub-phases of the whole job. Accordingly, this paper presents a novel approach for adaptively tuning the disk pairs' schedulers in both the hypervisor and the virtual machines during the execution of a single MapReduce job. Our results show that MapReduce performance can be significantly improved; specifically, adaptive tuning of disk pairs' schedulers achieves a 25% performance improvement on a *sort* benchmark with Hadoop.**

***Keywords*-Virtual Machine; MapReduce; Hadoop; Disk I/O Scheduler; Meta-Scheduler;**

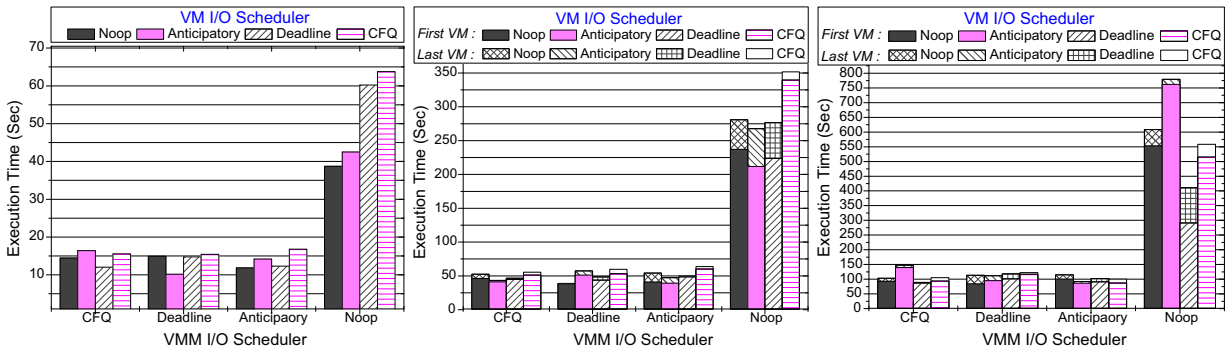
I. Introduction

Virtualization technology has become increasingly important for supporting efficient and flexible resource provisioning. By means of this technique, cloud computing provides users with the ability to perform elastic computation using large pools of VMs, without facing the burden of owning or maintaining physical infrastructure. Virtualization provides many benefits, including high resource utilization, performance isolation, ease of management, and flexibility of user-tailored environment deployment. Currently there are several commercial virtualization software (such as VMware [1], VirtualBox [2], and Windows Hyper-V [3]), while Xen [4] is the most widely used open source virtual machine monitor (VMM), for example, Amazon uses Xen as its virtualization-enabling technology for its infrastructure as a service, particularly in the Amazon Elastic Compute Cloud (EC2) [5].

Meanwhile, the MapReduce programming model [6] has become an essential component for data-intensive applications in the cloud. Due to its remarkable features including simplicity, fault tolerance, and scalability, MapReduce is by far the most powerful realization of data intensive cloud computing. It is often advocated as an easier-to-use, efficient and reliable replacement for the traditional programming model of moving the data to the computation. MapReduce has been applied widely in various fields including data- and compute-intensive applications [7][8][9]. The popular open source implementation of MapReduce, Hadoop [10], was developed primarily by Yahoo!, where it processes hundreds of terabytes of data on at least 10,000 cores, and is now used by other companies, including Facebook, Amazon, Last.fm, and the New York Times [11].

Although, MapReduce has been widely studied and considerable experience has been gained in clusters or data centers, there are few studies on the performance of MapReduce in VM-based system such as virtual clusters and clouds. Previous studies with Hadoop demonstrate a noticeable degradation in the MapReduce performance in virtual clusters and clouds [12][13][14]. Further studies reveal that the cause for such performance degradation is the VM interference, in particular the network I/O interference [15], that is, all the VMs within the same physical environment are sharing the network I/O resources. Lately, many techniques have been proposed to improve or analyze Hadoop's performance in VM-based clusters. Their main focus is to reduce the network overhead [16][17] and minimize the number of the speculative tasks caused by the network heterogeneity when varying the number of VMs which are deployed on each cluster's machines [15]. Despite the fact that the main focus of the aforementioned work is network I/O interference impacts on MapReduce performance in Xen-based cloud, all these methods require MapReduce to be aware of the runtime dynamics of virtualized environments. This global knowledge is hard to get. Even worse, that means MapReduce needs to be aware of virtualization, which hurts the simplicity and ease-to-use features.

0190-3918/11 $26.00 © 2011 IEEE
DOI 10.1109/ICPP.2011.86



(a) **One VM** (b) **Two VMs** (c) **Three VMs**

Figure 1. The variation of performance scores for different disk pairs' schedulers in Xen with different VM consolidation, *using* ***Sysbench*** *to create in parallel one process per each VM to sequentially write 1GB to 16 files*: (a) one VM is uniquely deployed within the physical machine, (b) two VMs are deployed within the physical machine and, (c) three VMs are sharing the physical machine.

Instead of changing the MapReduce code base, we investigate whether we can develop virtualization techniques to improve the performance of MapReduce in specific cases, and other data-intensive applications in general. VM interference depends on various factors. First, the type of application; second, the number of VMs; third, the VM scheduling algorithms in the hypervisor [18][19][20][21][22]. In addition, the impacts of the VM's interference caused by disk contention is as important, if not more important in some applications, as the ones caused by network contention. For example, as shown in Fig. 1, where we observe the elapsed time of the *Sysbench* benchmark [23], when sequentially writing 1GB to 16 files, varies significantly according to the number of VMs sharing the same physical machine. The elapsed time increases by a factor of 3.5X and 8.5X on average when the number of VMs deployed within a physical machine is 2VMs and 3VMs respectively. More importantly, our studies with our local cluster reveal that the performance score of I/O application varies along with the associated disk pair schedulers within both the Dom0 and DomUs, where the elapsed time varies by 16% on average when selecting different disk pairs' schedulers, regardless of the number of VMs deployed per physical node. Furthermore, we can easily see that the default pair schedulers[1](CFQ, CFQ) is not the best choice. In addition, as in the new computation to data paradigm, the data that needs to be processed by the disk is somewhat larger than the one need to be transferred by the network, in this paper, we investigate the impacts of disk schedulers in Hadoop. As far as we know, no previous studies have illustrated the impacts of VMs interference brought by disk I/O contention along with the appropriate selection of disk pair schedulers within the VMM and VMs on data intensive applications in a virtualized environment (MapReduce-based applications).

[1]In this paper, by pair schedulers, we mean a pair of disk schedulers, one within the VMM and the second within the VMs, referred to as (disk scheduler in VMM-level, disk scheduler in VMs-level).

Accordingly, a series of experiments are conducted to measure the performance of Hadoop on a Xen-based cluster with different pairs' schedulers. Interestingly, our experimental results report not only a noticeable variation of the Hadoop performance with different applications and with different pairs schedulers, but also demonstrate the opportunity to achieve better performance, for example, when we are using the pair (Anticipatory, Deadline), we achieve a 9% performance improvement for the *sort* benchmark, while using the pair (Anticipatory, CFQ), we can achieve the best performance score for the *wordcount* benchmark. Hence, different pair schedulers are only sub-optimal for different MapReduce applications. Moreover, MapReduce applications consist of different phases, in which the application performs different I/O workloads with different patterns. Therefore, taking advantage of the feature of dynamically tuning the disk schedulers in both VMM and VMs, we investigate the performance improvement that can be achieved by changing the disk pair schedulers at different points during the Hadoop execution. We implement our scheme by first manually dividing the Hadoop program into phases based on the characteristics of Hadoop applications. If there are phases in an application and a possible pair scheduler, there are unique solutions, where a solution is an assignment of disk pair schedulers to each phase.

However, dynamic disk scheduler tuning might not be able to outperform the best case when a single scheduler is used for all phases due to the penalty of the switching overhead, that is, the time cost of switching among different disk schedulers on VMMs and VMs. Even worse, the cost of switching the disk pairs' schedulers varies according to the first state and the second state. Thus we must offset the variation in the penalties of the switch cost. We present a novel heuristic that searches the space of solutions. It executes a solution and evaluates the performance score including the switch cost. Based on the evaluation, it selects the next solution to evaluate.

Performance results on Hadoop show that our heuristic finds an effective solution. Specifically we find several solutions that use different pairs' schedulers per phases that are superior to any solution that uses a single pair schedulers for all phases. For example, the *sort* benchmark using multiple pairs' schedulers outperforms the default pair scheduler (CFQ, CFQ) and the best single scheduler solution by 25% and 10% respectively. Moreover, the performance improvement is proportional to the data size, VM consolidation degree and system scale.

The primary contributions of this paper are as follows:

1. It demonstrates that significant potential exists for performance improvement in applications, particularly MapReduce applications, when choosing the appropriate disk pair schedulers within both VMMs and VMs.
2. It proposes a new methodology to adaptively tune the disk pairs' schedulers to improve the overall performance of the applications.

The rest of this paper is organized as follows: Section 2 briefly presents MapReduce and Xen I/O schedulers. Section 3 reports on our empirical study on the impacts of the different disk pairs' schedulers on Hadoop. The design of our adaptive meta-scheduler method is discussed in section 4. In sections 5, we detail the performance evaluation of our methodology and discuss our results. Section 6 discusses the related work. Finally, we conclude the paper and propose our future work in section 7.

II. Background

In this section, we briefly introduce MapReduce and Xen I/O schedulers.

A. MapReduce Programming Model

The MapReduce [6] abstraction is inspired by the Map and Reduce functions, which are commonly used in functional languages such as Lisp. Users express the computation using two functions, map and reduce, which can be carried out on subsets of the data in a highly parallel manner. The runtime system is responsible for parallelizing and fault handling.

B. Xen I/O Schedulers

The Xen hypervisor is a para-virtualizing virtual machine monitor [4][24] in which the machine architecture presented to an operating system is not identical to the underlying hardware. The Xen hypervisor is responsible for resource (CPU, memory and I/O device, etc.) allocation for the various virtual machines running on the same hardware device.

Xen is unique among VMM software because it allows users to choose among different CPU schedulers and disk I/O schedulers. The disk I/O scheduler performs two basic operations: merging and sorting. While the merging operation reduces the number transactions between the guest-OS and the VMM by merging adjacent I/O requests, the sorting operation arranges pending I/O requests in block order to minimize the seek time. There are currently four available disk I/O schedulers in the 2.6 Linux kernels: *Noop*, *Anticipatory*, *Deadline*, and *Complete Fair Queuing Scheduler (CFQ)*, (more details can be found in [18][20][25]).

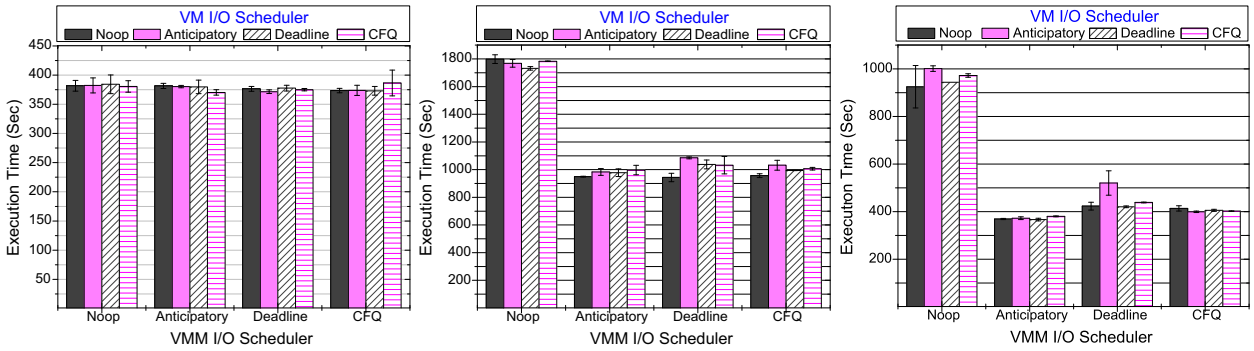
III. Empirical Study of Hadoop

A. Experimental Environment

We have conducted the experiments on a local cluster of four nodes, with 32 CPU cores in total. Each node is equipped with two quad-core 2.33GHz Xeon processors, 8GB of memory and one dedicated SATA disk of 1TB, running RHEL5 with kernel 2.6.22, and is connected with 1Gb/s Ethernet. This capable server allows us to deploy multiple virtual machines on each host. In VM-based environments, we use Xen 3.4.2 [4]. The VMs are running with RHEL5 with kernel 2.6.22. Four VMs are deployed within each host, and each VM is configured with 1GB memory and 1 VCPU pinned to its own core. Thus, the total number of VMs is 16. All results described in this paper are obtained using Hadoop version 0.19.0. Each data node processes 512MB on average and the data is stored with 2 replicas per chunk in Hadoop Distributed File System (HDFS). In all our experiments, we use the Xen credit scheduler as the default CPU scheduler [26]. All results are reported based on the average of three consecutive runs of the benchmarks.

1) Benchmarks: The performance of the disk schedulers vary with workloads. In the MapReduce application, different tasks are performed simultaneously, such as read the input of a new map task while writing the output of previous map to the local disk. Thus, selecting the appropriate disk schedulers is dominated by the I/O patterns along with the number of requests issued by the different workloads and data size. However, most of the MapReduce applications' workloads adopt a similar I/O pattern, leaving the decision of selecting the appropriate disk scheduler to the data size of each workload. Hence, MapReduce applications exhibit different behavior according to the data size of the map output and reduce input. To this end, we classify the MapReduce applications in term of disk operations into: heavy, moderate and light. The applications are termed as heavy disk operations when map output *and* the reduce output are relatively big and the applications are classified as moderate disk operations when only the map output is big, otherwise the applications are light disk operations. Accordingly, we use the three simple and widely used benchmarks to mimic the three aforementioned applications type: default wordcount, wordcount without combiner, and stream sort.

- *WordCount- With combiner*. In map stage, the map function maps data chunk (text file) into sets of key/value pairs, where the key is the read word from



(a) **Wordcount benchmark** (b) **Wordcount w/o combiner** (c) **Sort benchmark**

Figure 2. Impacts of the different I/O schedulers pairs on MapReduce performance in virtualized cluster: (a) Wordcount benchmark, (b) Wordcount without combiner benchmark, and (c) Sort benchmark.

the file and the value is a count of one (word, 1), and buffers these pairs in the buffer memory. If the buffered data reaches the buffer threshold, a combine function (reduce-type function) will be performed on the in-memory map output and outputs (word, count-in-this-part-of-the-input) pairs to disk [27]. As a result, fewer pairs will be written to disk and then read from disk in reduce phase.

- *WordCount w/o Combiner*. Similar to the wordcount benchmark but no combiner function will be applied on the in-memory map output. Thus there is a huge amount of data to be written to the disk, as the map output is nearly 1.7 times the size of the data input.
- *Stream Sort*. Each mapper sorts the data locally, and the reducer merges the result from all the mappers. The map input and the reduce output have the same data size as the input data.

2) Methodology: In our experiments we manually select the combinations of different disk I/O schedulers within the VMM and the VMs, giving the possibility of 16 disk pairs' schedulers.

B. Macroscopic Analysis

In this section, we seek to obtain a big-picture understanding the impacts of the disk I/O schedulers on Hadoop applications performances. In particular, we intend to answer the following questions:

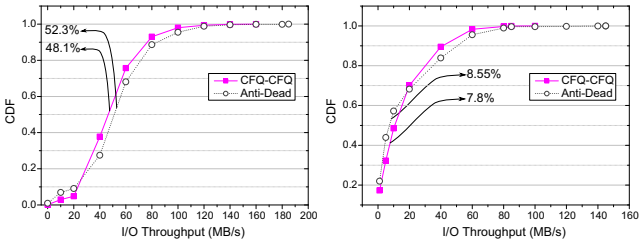
Q1. Is the default disk pair schedulers, namely (CFQ, CFQ) the optimal solution for Hadoop applications?

Q2. How significant is the impact of selecting the appropriate pair schedulers on applications performance?

Fig. 2 shows the execution time of wordcount (with and without combiner) and sort benchmarks with different disk pair schedulers in Xen. Our first observation is that the default pair scheduler (CFQ, CFQ) is not the optimal solution regardless the tested benchmark. In addition, as expected, the execution time varies according to the selected disk I/O pair schedulers as well as the running application. For instance, the execution time varies 1.5%, 29% (4.5% excluding Noop[2] in VMM), and 45% (10% excluding Noop in VMM) for the wordcount, wordcount w/o combiner, and sort benchmark, respectively. This can be explained by the data size along with I/O workload and pattern which will be operated on the data input and output during the map and reduce phases. For instance, the performance variation of the wordcount application is very small as shown in Fig. 2-a, which can be explained due to the small fraction of the data which needs to be written to the local disk, in instance the map output and the reduce input. However, the main disk operations are sequential reads when reading the data from the HDFS disk, 512 MB per VM. Thus, Anticipatory is the best disk scheduler candidate within the VMM, due to it's "seeking-conserving" behavior, which is based on the observation that a process will perform multiple I/O operations within short time duration. Moreover, as each VM performs two concurrent maps, fairness when reading the map input for each process is important to achieve better overall performance, therefore the CFQ scheduler is good disk scheduler candidate within the VMs. As a result, the disk pair scheduler (Anticipatory, CFQ) performs the best with an improvement score of 4.5% compared to the default scheduler.

Surprisingly, in the case of the wordcount without combiner, as shown in Fig. 2-b, the best solution (Anticipatory, Noop) side by side with (Deadline, Noop) outbalance the default solution by only 6%. We have expected the improvement ratio to be higher as the amount of data needs to be written to the disk is rather large compared to the default wordcount benchmark. However, this results can be explained due to the different I/O workloads that the application exhibits while running, (i.e. the main disk operations are a mix of processes doing synchronous and

[2]We additionally show the results excluding the ones when Noop is selected as disk scheduler in the VMM for both Wordcount w/o combiner and sort benchmark because the applications' performances are particularly bad.



(a) **VMM I/O throughput (MB/s)** (b) **VM I/O throughput (MB/s)**

Figure 3. CDFs of the I/O throughput in both VMM and VM (average of the four VMs) in the cases of the two disk pairs' schedulers (CFQ, CFQ) and (Anticipatory, Deadline).

asynchronous requests, roughly corresponding to reading the maps' inputs and writing both the maps' outputs in the mappers and the reduces' inputs in the reduces, in addition to only synchronous requests when reading the reduces' inputs). Thus, all the disk pairs' schedulers including the best solutions are going to be sub-optimal in different points of the job execution progress. This is similar to our case: VMM treats all the VMs as "process" and within each VM two maps and two reduces are running simultaneously, thus the map/reduce read or write operation for the two running instances are relatively close. In the current setting, since each VM has only one core, we provision each VM at most two Map or Reduce tasks (Hadoop default settings).

The previous discussion leads to the very important observation that, there is no single optimal disk pair schedulers for Hadoop applications. Moreover, different stages of MapReduce requires quite different I/O patterns: sequential I/Os at the beginning of Map, random writes at the end of Map, network I/O at the beginning of Reduce, sequential writes at the end of Reduce. The interleaving of such stages makes all the disk pair schedulers only sub-optimal for different stages. Thus, none of the disk pair scheduler is an optimal solution to any specific Hadoop application. The same observation applies strongly to the sort benchmark, where the best pair solution is (Anticipatory, Deadline) which outperforms the default pair scheduler by 9%, as shown in Fig. 2-c.

C. Microscopic Analysis

In this section we complement our macroscopic analysis with a detailed analysis of *sort* benchmark with two different disk pairs schedulers, the default one (CFQ, CFQ) and (Anticipatory, deadline), and then we examine the performance score of the different disk pair schedulers in different stages of the job.

Table I
PERFORMANCE SCORE OF THE DIFFERENT DISK PAIRS' SCHEDULERS WITH SORT BENCHMARK (*average out of 3 runs*).

VM	VMM			
	CFQ	Deadline	Anticipatory	Noop
CFQ	**402**	436	375	962
Deadline	405	415	**365**	927
Anticipatory	399	516	369	987
Noop	413	418	370	915

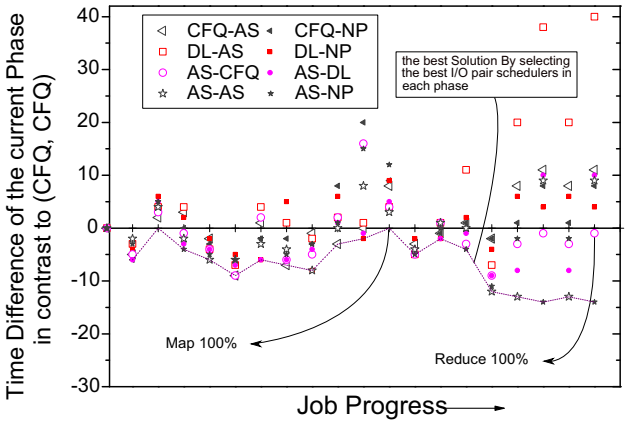


Figure 4. Performance score of the different disk I/O scheduler in different points of the sort benchmark: *Each point presents the MapReduce performance under different disk pairs' schedulers, where CFQ: CFQ, DL: Deadline, AS: Anticipatory, and NP: Noop.*

Table I presents the execution time of the sort benchmark with different disk pairs' schedulers. The pair schedulers (Anticipatory, Deadline) outperforms the default one (CFQ, CFQ) by 9%. To empirically explain these results, we have compared the I/O throughput in both VMM and VMs in one physical machine while running the sort benchmark. Fig. 3-a shows the cumulative distribution function (CDF) of the I/O throughput in the VMM. The Anticipatory scheduler in the VMM achieves better throughput, a maximum of 184 MB/s and 52.3 MB/s on average, while the CFQ achieves throughput of 159 MB/s on maximum and 47.1 MB/s on average. Fig. 3-b shows the cumulative distribution function (CDF) of the I/O throughput in the VMs (the average of the four VMs). The pair schedulers (Anticipatory, Deadline) achieves better throughput of 9.4, 9.6, 8.3, and 6.9 MB/s in each VM (8.55 MB/s on average), while (CFQ, CFQ) achieves 8.4, 7.8, 7, and 8.1 MB/s (7.8 MB/s on avarage). Hence, (Anticipatory, Deadline) achieves better overall performance while the (CFQ, CFQ) achieves better fairness amongst the different VMs.

Fig. 4 supports our earlier observation - the interleaving of different stages during MapReduce job makes all the disk pairs' schedulers sub-optimal for different stages - by showing the performance score (running time) in different points of the job with different disk scheduler pairs in contrast to the default pair (CFQ, CFQ), the base line. We can clearly see that the (Anticipatory, Deadline) pair is not the optimal pair schedulers for the sort application, although it achieves the best performance. More importantly, we can clearly see that the optimal solution, which selects the best pair schedulers for each sub-phase, could achieve a

performance improvement of 26% and 15% in contrast to the default pair (CFQ, CFQ) and (Anticipatory, Deadline), respectively. Accordingly, this paper presents a novel approach for adaptively tuning the disk I/O pairs schedulers in *Dom0* and *DomUs* during the execution of single MapReduce job.

IV. A META-SCHEDULER FOR ADAPTIVE DISK I/O SCHEDULER SELECTION

As described in section 2.2, Xen provides different CPU and I/O schedulers. While we would need to reboot the physical machine when switching between the two CPU schedulers; we can easily, on the fly, switch among the disk schedulers in Dom0 and DomUs. This enables us to develop a meta-scheduler for selecting the I/O scheduler at runtime with little overhead. Previous studies [18][20] revealed that different VMM I/O schedulers can significantly affect the application performance. More interestingly, different guest I/O scheduler can also affect the application performance considering the applications running on other VMs within the same physical machine.

Accordingly, we propose a novel methodology for adaptive disk I/O scheduler tuning, which is a framework for executing a single MapReduce job with several disk pair schedulers. The basic idea is to first divide the MapReduce job into phases and execute a series of experiments, with each phase we assign a prescribed pair of schedulers. During each experiment, we measure the performance score and then use a heuristic to choose the assignment of the disk pair schedulers for the next phase of the experiment.

In this section we first describe our methodology in dividing the MapReduce job into different phases. Next, we discuss the penalties of switching among different disk pairs' schedulers, and then we discuss our method for choosing an assignment of disk pair schedulers to each phase.

A. MapReduce Phase Detection

By design, the MapReduce program is divided into two phases, map and reduce. The map phase starts by reading the data it needs to process then the application-specific map, sort and spill is performed. The reduce phase starts by receiving the data from different maps and then writing it to the local disk, so finally the application-specific reduce function is performed. However, the concurrent execution of map and reduce makes it hard to identify the different phases.

To identify the different phases in the MapReduce program we use static program analysis to identify the access pattern of Hadoop program, according to the amount of work per subtask. Thus, as the subtasks' workload is either computation or disk request or bandwidth request, we can easily reduce the space of profiled option into - resource utilization space:

- *Computation*. From start to the first map output fetching to the disk.
- *Computation, disk and network I/O-intensive*. From the first map is fetched to disk until all maps are done.
- *Disk and network I/O-intensive*. From all maps are done until the shuffle phase is done.
- *Computation and disk I/O-intensive*. From the shuffle is done till the job is completely finished.

Bearing in mind that (1) the MapReduce programming model provides load balanced execution of the maps and reduces among the nodes, that is mostly every data node (TaskTracker) executing the same number of Map and Reduce tasks, and (2) the cost of the frequent switch of schedulers in running time, we finally divide the MapReduce program into three distinct phases:

Ph1. Starts when starting the job and ends when all the maps are completely done. This means that each node is running (CPU-intensive and disk and network I/O-intensive)

Ph2. Starts when the maps are done and ends when the shuffle is done. This means that each node is running (disk and network I/O-intensive)

Ph3. Represent the sort and reduce function. This means that each node is running (CPU-intensive and disk I/O-intensive).

However, the length of the second phase is highly depending on the number of maps that are expected to be executed for each node as shown in Table II. Therefore, as in our example, we used the case when each node is performing 8 maps. The second phase will be very short and we integrate it with phase three because the performance improvement which we could achieve by switching to another disk pair scheduler is very small taking in consideration the cost of switching the disk pairs' schedulers.

In the current setting, since MapReduce assumes that different stages are synchronized in each VM, applying the meta-scheduler method is effective and close to optimal, however this assumption will not hold in the case of slow nodes or tasks or when the cluster is shared by many users, which needs a more fine-grained meta-scheduler at the individual VM level and/or in the VMM level.

B. Cost of Tuning Disk Pairs Schedulers

In order to find the time cost when switching between two solutions, we start a dd command that writes 600MB of zeroes from /dev/zero to a file in parallel on four machines within the same physical machine. We calculate the switch cost as: $Cost_{switch} = Time_{withTwoSolutions} - \frac{1}{2} \times (Time_{Solution1} + Time_{Solution2})$.

Table II
PERCENTAGE OF NON-CONCURRENT SHUFFLE PHASE IN THE SORT BENCHMARK WITH DIFFERENT WAVES NUMBER:
$$\textit{No. of waves} = \frac{\textit{No. of data blocks}}{\textit{No. of data nodes} \times \textit{No. of slots per data node}}$$

Waves No.	1	1.5	2	2.5	3	3.5	4	4.5	5
Percent (%)	29.5	17	10.9	6.4	5.3	3.4	2.1	2.3	1.4

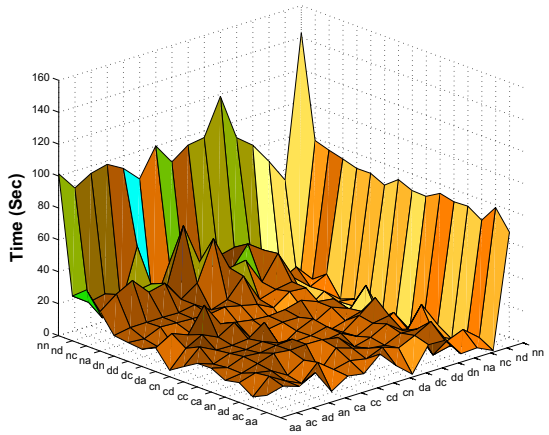


Figure 5. Cost variation when switching between different disk pairs' schedulers: *The X and Y axis represent the first state and the second one and* ***ii*** *represents the pair scheduler (VMM, VM) where i = {c: CFQ, d: Deadline, a: Anticipatory, and n: Noop}*

Fig. 5 demonstrates the time cost when switching between two different pairs' schedulers. We observe that the time cost of disk I/O switch varies according to the first state and the second state, from 4 seconds on average to 142 seconds. More interestingly, we observe that the switching cost is not commutative, and even worse, re-assigning the same disk I/O scheduler pair is costly, particularly when using the switch command and the first state and second state are the similar. However, we also demonstrate the same experiments with different VM consolidation and as expected the switching cost is proportional to the VM consolidations. We are currently working to provide deeper analysis of the disk I/O schedulers switching cost using a more accurate method and with different scenarios such as switching the disk schedulers within the VMs while fixing the disk scheduler within the VMM and vice versa.

C. Heuristically Assignment of the Disk Pair Schedulers to Phase

In order to effectively assign **S** disk pairs' scheduler to a **P** distinguished phases in a single MapReduce job, we need to find the best solution in a space of S^P of possible solutions, which can be a challenge due to the exponential number of combinations. Although the number of candidate solutions for the example above is 16^2 which is relatively small, we consider in our design the cases with larger number of phases as in fine-grained phases detection or a chain of MapReduce jobs (e.g., those specified in Pig [28]). As the number of phases increases, the number of possible solutions dramatically increases. Therefore, a brute force approach of enumerating the cost of all possible solutions is not practical, and we should consider the adaptation algorithm with little overhead. Therefore, we use a heuristic method to find the best disk pair schedulers in each phase, and then we move

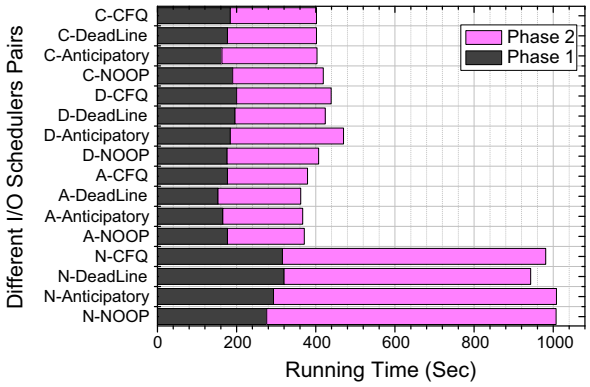


Figure 6. Performance score of the different disk pairs' schedulers in two phases for the sort benchmark.

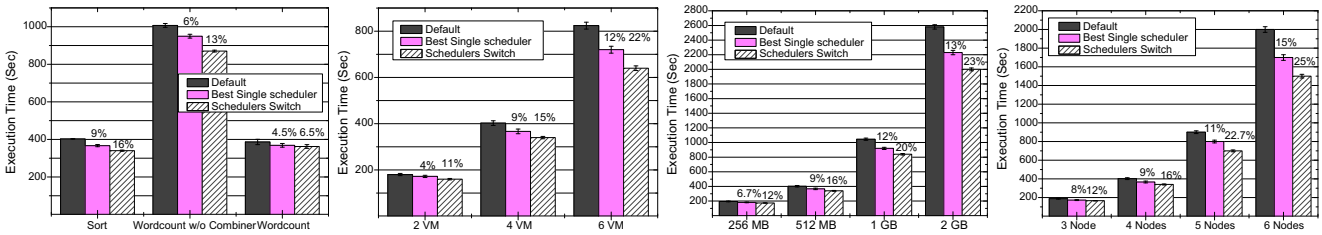
on to the next phase. Once it moves on to another phase, the scheduler for the preceding phase has been determined.

Initially, we execute the job completely using single pair schedulers, and then we find the performance score of each phase with each pair schedulers as shown in Fig. 6. Resulting with **P** sets of disk pairs schedulers per phases, refereed as **SP**. $SP = \{s_i^j\}$, where s_i^j indicates the disk pair schedulers in the phase p_i.

To achieve the best performance for a specific phase we test the application with the best pair schedulers, therefore, we sort the pairs' schedulers in each phase in descending order according to their performance score. Then we compare the current disk pair schedulers with the next pair schedulers (second best scheduler pair) with

Algorithm 1: An Heuristics Algorithm for Meta-Scheduler

Input: S: set of possible disk pairs' schedulers and P: the number of possible phases.
Description: find the best solution, assignment of disk pair schedulers to a phase. The pairs' schedulers are sorted in descending order according to their performance score in each phase for the single MapReduce job as shown in Fig. 6.
Output: $Sol = \{s_i\}$ where $1 =< i < P$
foreach $p_i \in P$ **do**
  /*process the I/O disk schedulers pair according the their performance score in each phase*/
  $j \leftarrow 0$
  while $Hadoop_{time}(Sol_{i-1}, s_i^j, S_{i+1}) > Hadoop_{time}(Sol_{i-1}, s_i^{j+1}, S_{i+1})$ **do**
    $j \leftarrow j + 1$
  end
  if *The last non-Zero value in* Sol_{i-1} *is the same as* s_i^j **then**
    $Sol_i = Sol_{i-1} \cup \{0\}$
    else
    $Sol_i = Sol_{i-1} \cup \{s_i^j\}$
  end
end



(a) **Different workloads** (b) **Different VM consolidation** (c) **Different data size** (d) **Different cluster scale**

Figure 7. Performance score of the adaptive disk I/O scheduler in different scenarios: (a) different workloads, (b) different VM consolidation, (c) different data size, and (d) different cluster scale.

respect to the elapsed time $Hadoop_{time}(Sol_{i-1}, s_i^j, S_{i+1})$. $Hadoop_{time}(Sol_{i-1}, s_i^j, S_{i+1})$ represents the elapsed time of the application using the already fixed phases Sol_{i-1}, and the current disk pair schedulers and more importantly S_{i+1} which indicates the best disk pair schedulers for all the left phases together, considering all the left phases as one integrated phase. Using S_{i+1} will guarantee a fair test for the two tested solutions, especially with the existence of the variation in the cost switch among the different solutions. If the performance of the second solution is better than the current one, it is accepted. Then it recursively tries the next lower solution. The pair schedulers is determined when the new performance score is worse than the current one. After selecting the pair schedulers, it checks the last item in the solution set, thus if the new solution is similar then we assign a value 0, which means no switch should be used and then it moves on to the next phase. While the heuristic method does not guarantee to find an optimal pairs' schedulers set, it is able to find a good one in a running time at most $P \times S$, as reflected in our experiments in the later section.

V. Performance Evaluation

We run a suite of experiments evaluating our adaptive disk pair schedulers with different MapReduce workloads, different VM consolidations, different data sizes and finally different cluster scales.

Different workloads. We have applied the same methodology in the previous study on the sort application on wordcount benchmark (with and without combiner). Fig. 7-a shows the results on four-nodes cluster when deploying four virtual machines per physical machine. We fix the data distribution per data node to 512MB. We observe that using our meta-scheduler method outperforms the default disk pair schedulers and the single best disk pairs' schedulers. In addition, the improvement score varies according to the running applications. For instance, for wordcount applications, the performance improvements are 6.5% and 2% in contrast to the default and best disk pair schedulers respectively. Surprisingly, even though the reduce output is relatively small, selecting the best pair schedulers in this phase, which is very short compared to the first phase as shown in Fig. 8, has improved the overall performance by 2%. In contrast, for the wordcount application without combiner function, the performance improvements are 13% and 7%, respectively, which can be explained because the second phase is relatively short compared with the first phase, as shown in Fig. 8. Our meta-scheduler outperforms the default pair schedulers and the best single pair schedulers by 16% and 7%. However, it is expected the performance improvement over the best single pair schedulers would be bigger, as both phases are nearly equal in time as shown in Fig. 8. This can be explained because both solutions' performances are very close in the second phase. However, we are going to extend our current study by applying more fine-grained phase detection in our methodology.

Different VM consolidations. Fig. 7-b presents the execution times of sort benchmark with different VM consolidations in three scenarios. We vary the number of VMs per physical node while we fix the same amount of data distribution per data node to 512MB. We observe that our adaptive disk I/O scheduler is superior to both the default disk pair schedulers and best single disk pair schedulers. More importantly the improvement score is proportional to the VM consolidation degree. For instance, in the case of 2VMs per node, the best single pair scheduler achieves improvement of 4% and this improvement is gradually increasing as the VM consolidation degree increases, with 9% and 12% when 4VMs and 6VMs are deployed. Our adaptive solution achieves a performance improvement of 11%, 15%, and 22%. Here the improvement score is strongly affected by the disk scheduler pairs' switch cost.

Different data sizes. Fig. 7-c demonstrates the results of sort benchmark on four-nodes cluster when deploying four virtual machines per physical machine. We vary the data distribution on each data node to 256MB, 512MB, 1GB, and 2GB. Obviously, the performance improvements are gradually increasing as the data size is increasing because of two reasons, first the number of I/O operations bigger and second the feasibility of our phase detection, see Table II. The non-concurrent shuffle is gradually decreasing as the number of maps "data size" increase which results with more clearly and equally two phases as shown in Fig. 8.

Different cluster scales. Fig. 7-d presents the execution

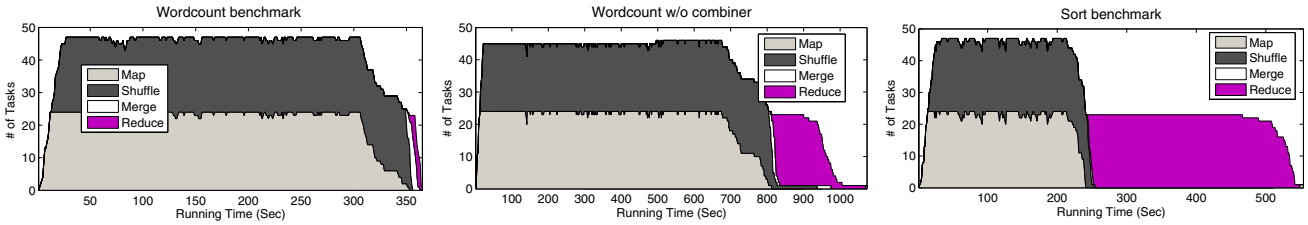


Figure 8. Different phases during MapReduce job for different benchmarks.

times of the sort benchmark with different physical cluster scales. We vary the number of used physical nodes to 3, 4, 5, and 6 nodes and kept the same number of VMs per node (4 VMs per node). The adaptive disk I/O scheduler can achieve better performance improvement as the physical cluster is scaling up. These results are expected as the improvement in each physical node is nearly the same, while here the improvement score is affected by the network transfer in all-to-all communication.

VI. Related Work

We divide previous work into two parts: the impacts of I/O schedulers on applications' performance and improving MapReduce in virtual-based environment.

Impact of I/O scheduler on application's performance. There are two representative studies [25][29] on the impact of I/O schedulers in native operating system on the application performance. Pratt and Heger [25] and Seelam et al. [29] have done a thorough evaluation of the different Linux I/O schedulers and their effects on different workload. Some studies have been dedicated to improving application's performance by using heuristic I/O schedulers [30], and using intelligence I/O schedulers [31][32].

Recent studies [18][20] have examined the impacts of the choice of disk I/O scheduler in both VMM and VMs on application performance. Kesavan et al. [20] have reported that the choice of an appropriate I/O scheduler at the VMM layer has a significant impact on the inter-application isolation and performance guarantees inside a given VM.

Our work is different from the aforementioned works in the sense that we study the impact of I/O schedulers on parallel application with multiple I/O workloads, with emphasis on MapReduce applications. We demonstrate that using multiple disk pairs' schedulers for a single job is feasible and advantageous.

Improving MapReduce performance in virtual cluster. There have been a few studies on improving MapReduce performance in Xen-based virtual clusters. Zaharia et al. [15] have proposed a new scheduling algorithm called Longest Approximate Time to End (LATE) to improve the performance of Hadoop in a heterogeneous environment, brought by the variation of VM consolidation amongst different physical machines, by preventing the incorrect execution of speculative tasks.

Ibrahim et al. [17] have proposed CLOULET a new framework for the MapReduce model by decoupling the storage unit, namely HDFS, from the computation unit, namely VMs, and keeping the data transfer physical node based, thus minimizing the impact of the I/O virtualization on the Hadoop execution.

As far as we know, we are the first to investigate the disk scheduler's impacts on MapReduce performance and propose an optimized solution without any modification to the Hadoop or the application code.

VII. Conclusion and Future Work

Virtualization has become the fundamental technique for cloud computing. As data-intensive applications become popular in the cloud, their performance on the virtualized platform calls for empirical evaluations and technical innovations. In this study, we investigate the performance of the popular data-intensive system, MapReduce, on a virtualized platform. Our detailed study reveals that different disk pair schedulers within the virtual machine manager and virtual machines cause a noticeable variation in the Hadoop performance in virtual cluster. We address this problem through developing a meta-scheduler for selecting a suitable disk pair schedulers. Given an application, a program is divided into phases; currently we use coarse-grained phase detection, using the program progress. A novel heuristic successively evaluates solutions (phase-scheduler assignments). The program is executed, and the performance score is measured. Then, the heuristic evaluates the solution based on the best performance score. Performance results are obtained on our local virtual cluster which is based on a Xen hypervisor. We find that for most of the Hadoop benchmarks, using multiple-pairs in a single application can provide a better performance score over any single-pair solution. More importantly, our solution provides up to 25% performance improvement over the default virtual Hadoop cluster configuration. Moreover, the performance improvement is proportional to the data size, VM consolidation and system scale.

In considering future work, we want to explore extending our dynamic scheduler and performance switch for a wider diversity of applications. In addition, we intend to build a fine-grained control method to dynamically switch the disk pairs' schedulers. The fine-grained control method is

using information from the VMs within the same physical node and is based on the status of the VMs' I/O (i.e. the number of request); using this we can switch to the most suitable pair schedulers. Ultimately we would want to build a general prediction model for the scheduler switch. As a more long-term agenda, we are going to investigate the cause of cost when switching amongst different disk pairs' schedulers and minimize such cost. This will improve the design and performance of our methodology.

VIII. Acknowledgments

This work is supported by the National 973 Basic Research Program of China under grant No. 2007CB310900, China Next Generation Internet Project under grant CNGI2008-109, the Important National Science & Technology Specific Projects under grant 2009ZX03004-002, the International Cooperation Program of Wuhan Technology Bureau under grant 201171034311, the National High-Tech R&D Plan about Manufacture Cloud, and the Inter-disciplinary Strategic Competitive Fund of Nanyang Technological University 2011 No.M58020029.

References

[1] Vmware Homepage, http://www.vmware.com/, 2010.

[2] Virtual Box, http://www.virtualbox.org/, 2010.

[3] Windows Hyper-V Server, www.microsoft.com/hyper-v-server/, 2011.

[4] Xensource Homepage, http://www.xensource.com/, 2011.

[5] Amazon Elastic Compute Cloud (EC2), http://aws.amazon.com/ec2/, 2011.

[6] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," *Proc. of the 6th Usenix Symposium on Opearting Systems Design and Implementation (OSDI 2004)*, San Francisco, California, USA, Dec. 6-8, 2004, pp. 137-150.

[7] R. E. Bryant, "Data-intensive supercomputing: The case for disc," *Technical Report*, CMU-CS-07-128, Department of Computer Science, Carnegie Mellon University, 2007.

[8] B. S. He, M. Yang, Z. Y. Guo, R. S. Chen, W. Lin, B. Su, and L. D, "Comet: Batched Stream Processing for Data Intensive Distributed Computing," *Proc. of the ACM Symposium on Cloud Computing (SOCC 2010)*, Indiana, USA, June. 10-11, 2010, pp. 63-74.

[9] A. W. McNabb, C. K. Monson, and K. D. Seppi, "Parallel pso using mapreduce," *Proc. of the IEEE Congress on Evolutionary Computation 2007*, Singapore, Sep. 25-28, 2007, pp. 7-14.

[10] Hadoop Homepage, http://lucene.apache.org/hadoop, 2010.

[11] Hadoop: Applications Powered by Hadoop, http://wiki.apache.org/hadoop/Powered, accessed September 2010.

[12] S. Ibrahim, H. Jin, L. Lu, L. Qi, S. Wu, and X. Shi, "Evaluating mapreduce on virtual machines: The hadoop case," *Proc. of the First International Conference on Cloud Computing (CloudCom 2009)*, Springer LNCS, Dec. 1-4, 2009, pp. 519-528.

[13] B. S. He, M. Yang, Z. Y. Guo, R. S. Chen, W. Lin, B. Su, H. Y. Wang, and L. D. Zhou, "Wave Computing in the Cloud," *Proc. of the Usenix Workshop on Hot Topics in Operating Systems (HotOS 2009)*, Monte Verita, Switzerland, May. 18-20, 2009, pp. 5-5.

[14] S. Ibrahim, H. Jin, L. Lu, B. S. He, L. Qi, and S. Wu, "LEEN: Locality/Fairness- aware Key Partitioning for MapReduce in the Cloud," *Proc. of the IEEE 2nd International Conference on Cloud Computing Technology and Science (CloudCom 2010)*, Indiana, USA, Nov. 30-Dec. 03, 2010, pp. 17-24.

[15] M. Zaharia, A. Konwinski, A. D. Joseph, R. H. Katz, and I. Stoica, "Improving mapreduce performance in heterogeneous environments," *Proc. of the 8th Usenix Symposium on Opearting Systems Design and Implementation (OSDI 2008)*, San Diego, CA, USA, Dec. 8-10, 2008, pp. 29-42.

[16] Tashi Homepage, http://www.pittsburgh.intel-research.net/projects/tashi/, accessed September 2009.

[17] S. Ibrahim, H. Jin, B. Cheng, H. Cao, S. Wu, and L. Qi, "Cloudlet: towards mapreduce implementation on virtual machines," *Proc. of the ACM Symposium on High Performance Distributed Computing (HPDC 2009)*, ACM Press, June. 11-13, 2009, pp. 65-66.

[18] D. Boutcher and A. Chandra, "Does virtualization make disk scheduling pass?," *ACM SIGOPS Operating Systems Review*, Vol. 44, Jan. 2010, pp. 20-24.

[19] L. Cherkasova, D. Gupta, and A. Vahdat, "Comparison of the three cpu schedulers in xen," *SIGMETRICS Performance Evaluation Review*, Vol. 35, Sep. 2007, pp. 42-51.

[20] M. Kesavan, A. Gavrilovska, and K. Schwan, "On disk i/o scheduling in virtual machines," *Proc. of the 2nd workshop on I/O Virtualization (WIOV 2010)*, Pennsylvania, USA, Mar. 13, 2010, pp. 6-6.

[21] D. Ongaro, A. L. Cox, and S. Rixner, "Scheduling i/o in virtual machine monitors," *Proc of the fourth ACM SIGPLAN/SIGOPS international conference on Virtual Execution Environments (VEE 2008)*, Seattle, WA, USA, Mar. 5-7, 2008, pp. 1-10.

[22] S. Ibrahim, B. S. He, and H. Jin, "Towards Pay-As-You-Consume Cloud Computing," *Proc. of the IEEE 8th International Conference on Services Computing (SCC 2011)*, Washington, USA, July. 4-9, 2011.

[23] System performance benchmark, http://sysbench.sourceforge.net/, 2011.

[24] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. L. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the art of virtualization," *Proc. of the nineteenth ACM Symposium on Operating Systems Principles (SOSP 2003)*, New York, USA, Oct. 19-22, 2003, pp. 164-177.

[25] S. L. Pratt and D. A. Heger, "Workload dependent performance evaluation of the linux2.6 i/o schedulers," *Proc. of the Linux Symposium 2004*, Ottawa, Canada, July. 21-24, 2004, pp. 425-448.

[26] Xen Wiki: Credit Scheduler, http://wiki.xensource.com/xenwiki/CreditScheduler, 2010.

[27] Hadoop Wiki: How Map and Reduce operations are actually carried out, http://wiki.apache.org/hadoop/HadoopMapReduce, 2011.

[28] C. Olston, B. Reed, U. Srivastava, R. Kumar and A. Tomkins, "Pig Latin: A Not-So-Foreign Language for Data Processing," *Proc. of the 2008 ACM SIGMOD international conference on Management of Data (SIGMOD 2008)*, Vancouver, Canada, June. 9-12, 2008, pp. 1099-1110.

[29] S. R. Seelam, R. Romero, P. J. Teller, and W. Buros, "Enhancements to linux i/o scheduling," *Proc. of the Linux Symposium 2005*, Ottawa, Canada, July. 20-23, 2005, pp. 175-192.

[30] S. R. Seelam, J. S. Babu, and P. Teller, "Automatic i/o scheduler selection for latency and bandwidt optimization," *Proc. of the Workshop on Operating System Interference in High Performance Applications (OSIHPA 2005)*, Saint Louis, USA, Sep. 17, 2005, pp. 1-5.

[31] Y. Zhang and B. K. Bhargava, "Self-learning disk scheduling," *IEEE Transaction on Knowledge and Data Enginnering*, Vol. 21, Jan. 2009, pp. 50-65.

[32] K. Lund and V. Goebel, "Adaptive disk scheduling in a multimedia dbms," *Proc. of the eleventh ACM international conference on Multimedia (MM 2003)*, Berkeley, CA, USA, Nov. 2-8, 2003, pp. 65-74.