On Disk I/O Scheduling in Virtual Machines

Mukil Kesavan, Ada Gavrilovska, Karsten Schwan
Center for Experimental Research in Computer Systems (CERCS)
Georgia Institute of Technology
Atlanta, Georgia 30332, USA
{mukil, ada, schwan}@cc.gatech.edu

ABSTRACT
Disk I/O schedulers are an essential part of most modern operating systems, with objectives such as improving disk utilization, and achieving better application performance and performance isolation. Current scheduler designs for OSs are based heavily on assumptions made about the latency characteristics of the underlying disk technology like electromechanical disks, flash storage, etc. In virtualized environments though, with the virtual machine monitor sharing the underlying storage between multiple competing virtual machines, the disk service latency characteristics observed in the VMs turn out to be quite different from the traditionally assumed characteristics. This calls for a re-examination of the design of disk I/O schedulers for virtual machines. Recent work on disk I/O scheduling for virtualized environments has focused on inter-VM fairness and the improvement of overall disk throughput in the system. In this paper, we take a closer look at the impact of virtualization and shared disk usage in virtualized environments on the guest VM-level I/O scheduler, and its ability to continue to enforce isolation and fair utilization of the VM’s share of I/O resources among applications and application components deployed within the VM.

1. INTRODUCTION
The evolution of disk I/O schedulers has been heavily influenced by the latency characteristics of the underlying disk technology and the characteristics of typical workloads and their I/O patterns. I/O schedulers for electromechanical disks are designed to optimize expensive seek operations [16], whereas schedulers for flash disks are generally designed to save on expensive random write operations [13, 1]. However, when such schedulers are used in OSs inside virtual machines, where the underlying disks are shared among multiple virtual machines, the disk characteristics visible to the guest OSs may differ significantly from the expected ones. For instance, the guest perceived latency of the “virtual disk” exposed to a VM, depends not only on the characteristics of the underlying disk technology, but also on the additional queuing and processing that happens in the virtualization layer. With bursty I/O and work conserving I/O scheduling between VMs in the virtualization layer, the virtual disks of guest OSs now have random latency characteristics for which none of the existing scheduling methods are designed for.

This discussion raises the question, “Should we be doing disk I/O scheduling in the VM at all in the first place?” Disk I/O scheduling, in addition to optimally using the underlying storage, also serves to provide performance isolation across applications. The most appropriate choice of an I/O scheduler is dependent on the characteristics of the workload [16, 19]. This is true in a virtualized environment too; guest OSs need to provide isolation across applications or application components running within a VM, and different VM-level disk schedulers are suitable for different workloads. However, at the virtualization layer, there is generally limited availability of information regarding characteristics of the workloads running inside VMs. Furthermore, specializing the virtualization layer for the applications that run inside virtual machines calls into question its very nature. A recent study by Boucher et al. [6] corroborates this analysis and the authors demonstrate the need for disk scheduling at the level closest to the applications, i.e., the VM guest OS, even in the presence of different storage technologies such as electromechanical disks, flash and SAN storage.

Therefore, we argue that to address the issues related to disk I/O scheduling in virtualized environments, appropriate solutions should be applied to both a) the VM-level disk scheduling entity designed to make best use of application level information and b) the VMM layer scheduling entity designed to optimally use the underlying storage technology and enforce appropriate sharing policies among VMs sharing the platform. An example of the former is that VM-level disk schedulers should not be built with hard assumptions regarding disk behavior and access latencies. Regarding the latter, VMM-level disk schedulers should be designed with capabilities for explicit management of VM service latency. Random “virtual disk” latency characteristics in the guest VMs make the design of the VM level disk scheduling solution hard, if not impossible. Similar observations may be true regarding other types of shared resources and their VM- vs. VMM-level management (e.g., network devices, TCP congestion avoidance mechanisms in guest OSs, and the scheduling of actual packet transmissions by the virtualization-layer).

Toward this end, we first provide experimental evidence of varying disk service latencies in a VM in a Xen [5] environment and how this breaks the inter-process performance isolation inside the VM. Next, we propose and implement simple extensions to current Linux schedulers (i.e., the VM-level part of our solution), including the anticipatory [9] and
the CFQ [4] schedulers, and study the modified schedulers’ ability to deal with varying disk latency characteristics in virtualized environments.

Preliminary results indicate that the simple extensions at VM-level may improve the performance isolation between applications inside the VM in the case of the anticipatory disk scheduler. Finally, we motivate the need for suitable extensions to VMM-level I/O schedulers, necessary to derive additional improvements for different performance objectives in the VM.

2. RELATED WORK

Most prior studies on disk I/O schedulers have concentrated on workloads running on native operating systems [16, 19]. These studies primarily shed light on the appropriate choice of an I/O scheduler based on workload characteristics, file system and hardware setup of the target environment. Recently, there has been some interest in the virtualization community to understand the implications of using I/O schedulers developed for native operating systems in a virtualized environment. Boutcher et al. [6] investigate the right combination of schedulers at the VM and VMM level to maximize throughput and fairness between VMs in general. They run representative benchmarks for different combinations of VM and host I/O schedulers selected from the common Linux I/O schedulers, such as noop, deadline, anticipatory and CFQ. Our work is different from theirs in the sense that we study the ability of a given VM’s I/O scheduler to enforce isolation and fairness between applications running inside that VM. In fact, one of their key conclusions points out that there is no benefit (in terms of throughput) to performing additional I/O scheduling in the VMM layer. However, we show later on in this paper 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.

The Virtual I/O Scheduler (VIOS) [20] provides fairness between competing applications or OS instances in the presence of varying request sizes, disk seek characteristics and device queuing. This scheduler is a work conserving scheduler which in the presence of multiple VMs with bursty I/O characteristics, would still result in random request latency characteristics inside a guest VM with steady I/O. This would result in insufficient performance isolation between the different applications that run inside a VM much in the same way as the other schedulers analyzed in this paper.

Gulati et al. [7] devise a system for proportional allocation of a distributed storage resource for virtualized environments (PARDA) using network flow control methods. They employ a global scheduler that enforces proportionality across hosts in the cluster and a local scheduler that does the same for VMs running on a single host. This scheduling architecture is similar in principle to the one proposed by the Cello disk scheduling framework [21] for non-virtualized environments. However, in the PARDA framework there are potentially three levels of scheduling: a disk scheduler inside the VM and the two others mentioned before. The interactions between these multiple levels of scheduling and the need for coordination between these levels in order to maintain performance guarantees and isolation at the application level (as opposed to just at the VM granularity) has not been studied in their work or, to the best of our knowledge, in any previous system. The results we present in our paper provide experimental evidence of the issues with such lack of coordination and motivate the need for future system designs to take coordination into account explicitly in order to achieve desired performance properties at the end application level.

3. TESTBED

Our work is conducted on a testbed consisting of a 32 bit, 8 core Intel Xeon 2.8 GHz machine with 2GB of main memory, virtualized with Xen 3.4.2, and para-virtualized Linux 2.6.18 guest VMs. All VMs are configured with 256MB of RAM and 1 VCPU pinned to its own core to avoid any undue effects of the Xen CPU scheduler. The virtual disks of the VMs are file-backed and placed on a 10,000 RPM SCSI disk separate from the one used by Xen and Domain-0. We use the following benchmarks to evaluate the system:

- PostMark [11] (PM) – which provides workload that generates random I/O operations on multiple small files, typical of internet servers; and
- “Deceptive Idleness” (DI) – a streaming write and synchronous read benchmark from [14, 19], reproduced in Table 1 for convenience.

All measurements are reported based on averages of three consecutive runs of the benchmarks. The page cache in both Domain-0 and the VMs are flushed between consecutive runs to avoid caching effects.

4. ISSUES WITH SHARED DISK USAGE

We next experimentally analyze the implications of shared disk usage in virtualized environments on the VMs’ performance and the ability of their guest OSs to manage their portion of the I/O resources.

4.1 Virtual Disk Latency Characteristics

First, we measure the observed disk service latencies inside a VM running the DI benchmark from Table 1, simultaneously with five other VMs running the PostMark benchmark and generating background I/O load. We use the Linux blktrace facility inside the DI VM to record I/O details on the fly. The blktrace output is sent to a different machine over the network, instead of writing to disk, to prevent any self-interference in the measurement. In this test, the Domain-0 I/O scheduler is set to the anticipatory scheduler.

<table>
<thead>
<tr>
<th>Program 1:</th>
</tr>
</thead>
<tbody>
<tr>
<td>while true</td>
</tr>
<tr>
<td>do</td>
</tr>
<tr>
<td>dd if=/dev/null of=file count=2048 bs=1M</td>
</tr>
<tr>
<td>done</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Program 2:</th>
</tr>
</thead>
<tbody>
<tr>
<td>time cat 50mb-file &gt; /dev/null</td>
</tr>
</tbody>
</table>

Table 1: Deceptive Idleness Benchmark: Asynchronous Write and Synchronous Read

Program 2:
corresponding to read and write requests, respectively). A naive disk scheduler, like the deadline disk I/O scheduler, for example, may assume prematurely that a process performing synchronous I/O has no further requests and in response, switches to servicing the process performing asynchronous requests. This problem is solved by the use of bounded anticipatory idle periods to wait for the request from a process doing synchronous I/O, thereby preventing the prematurely made inappropriate decisions. The Linux implementation of anticipatory scheduling uses a default anticipatory timeout (antic_expire) of around 6ms for synchronous processes, set most likely assuming a disk service latency of around 10ms in the worst case. This parameter can be tuned to obtain the appropriate trade-off between disk throughput and deceptive idleness mitigation.

We use the DI benchmark, which represents a workload prone to deceptive idleness, in a VM, executing alone or along with 4 other VMs running the PostMark benchmark. The schedulers in all VMs are set to anticipatory. The scheduler in the Domain-0 is varied across noop, anticipatory and CFQ. The execution time for the read portion of the DI benchmark is plotted against different configurations in Figure 2. The read times for all scheduler combinations with no consolidation (first bar in each set) is significantly lower than when the DI workload is run consolidated (second bar in each set). The reason for this is that in the presence of consolidation, the widely varying latencies of the virtual disks exposed to the VMs render the static setting of the anticipation timeout in the VM-level scheduler ineffective. In addition, the random latency characteristics also affect the process scheduling inside the VMs where a process blocked on a long-latency request is scheduled out for longer, whereas a process blocked on a small-latency I/O request is not. This also affects the per process thinktimes computed by the anticipatory scheduling framework and would eventually lead to it not anticipating at all for the synchronous process, thereby breaking its design.

4.3 CFQ Scheduler

The goal of the Completely Fair Queuing (CFQ) scheduler [4] is to fairly share the available disk bandwidth of the system between multiple competing processes. The Linux implementation of CFQ allocates equal timeslices between processes and attempts to dispatch the same number of requests per timeslice for each process. It also truncates the timeslice allocated to a process if it has been idle for a set amount of time. The idle time benefit is disabled for a process if it “thinks” for too long between requests or is very seek-intensive; process thinktime and seekiness is computed online and is maintained as a decaying frequency table. We evaluate the ability of the CFQ scheduler to provide fairness

![Figure 1: Disk latencies observed inside a VM running in a consolidated virtualized environment. Min = 38 us, Max = 1.67 s, Avg = 116 ms, std. deviation = 227 ms.](image1)

![Figure 2: Deceptive Idleness in VM Disk I/O](image2)

Table 2: CFQ Fairness between Processes inside a VM. Titles for Columns 2,3 and 4 represent the VMScheduler-Domain0Scheduler used for the experiment.

<table>
<thead>
<tr>
<th>Workload</th>
<th>clq-noop</th>
<th>clq-as</th>
<th>clq-clq</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 VM</td>
<td>1.0 ±0.0</td>
<td>1.0 ±0.0</td>
<td>1.0 ±0.0</td>
</tr>
<tr>
<td>2 VMs</td>
<td>0.82 ±0.18</td>
<td>0.82 ±0.17</td>
<td>0.99 ±0.14</td>
</tr>
<tr>
<td>Adaptive(s VMs)</td>
<td>0.84 ±0.17</td>
<td>0.59 ±0.06</td>
<td>0.96 ±0.03</td>
</tr>
</tbody>
</table>

Next, we measure the ability of the most common Linux disk I/O schedulers – the anticipatory and CFQ schedulers – to enforce their workload level performance guarantees in such an environment.

4.2 Anticipatory Scheduler

The anticipatory scheduler [9] addresses the problem of deceptive idleness in disk I/O in a system with a mix of processes doing synchronous and asynchronous requests (roughly
between processes inside a VM, when the VM is running alone vs. consolidated with other VMs.

We run two instances of the PostMark benchmark inside a VM and use the read throughput achieved for each instance to compute the Throughput Fairness Index \[10\] between the two instances. The range of the index ranges from 0 to 1 with 0 being the least fair and 1 being completely fair. The write throughput is excluded from the fairness calculation in order to avoid errors due to write buffering in Domain-0. Also, the VM running both PM instances is given two VCPUs in order to prevent the process scheduling inside the VM from skewing the results too much. The rest of the VMs in the consolidated test case are all running a single copy of the PostMark benchmark in order to generate a background load. The average fairness measures and its standard deviation across multiple test runs are shown in Table 2 for different combinations of VM and Domain-0 I/O schedulers. There are two key results that can be observed from the first two rows in the table.

First, it can be seen that, inside a VM, the average fairness index between processes decreases and the variation in the fairness index across multiple runs of the same experiment increases when it is consolidated with other VMs. The reason for this is that the random request service latency characteristics of the virtual disk, and the static setting of the tunable parameters (especially the timeslices and the idle timeout), result in an unequal number of requests being dispatched between multiple processes during each timeslice. A long-latency request causes process blocking and idle time-out expiration whereas short-latency requests do not.

Second, the choice of the Domain-0 I/O scheduler plays an important role in achieving fairness inside a VM. This can be seen by the difference in the average fairness indices and their deviation measured for different Domain-0 schedulers. Having the CFQ scheduler in both the VM and Domain-0 results in lesser fairness degradation between processes in the VM as the virtual disk service latencies would vary within a smaller range due to the inherent fairness of the algorithm, resulting from its bounded time slices of service.

5. ACHIEVING DESIRABLE DISK SHARING BEHAVIOR

The key takeaways from the previous section are that the static determination of VM I/O scheduler parameters and the random request service latencies in the virtualization layer are the primary contributors to the failure of the VM level schedulers when it comes to inter-process isolation. In this section we discuss the properties required for a complete solution to disk I/O scheduling in a virtualized environment, at different levels of the system. We realize a partial solution at the VM level and motivate the need for a holistic solution to transcend and require cooperation from all layers of the system – VM, VMM and possibly hardware, for VMM bypass devices.

5.1 Service Latency Adaptation inside VM

Our analysis in the previous section points to the fact that the algorithmic parameters exposed as tunables in both the anticipatory scheduler and the CFQ scheduler, if set dynamically based on observed service latencies over time windows, may be able to improve the ability to enforce application isolation in the VMs. We develop a basic implementation of such a facility in the Linux I/O scheduler framework as follows. We measure the virtual disk service latencies for synchronous requests inside the VM and maintain a decaying frequency table of mean disk service latency (exponentially weighted moving average) at the generic elevator layer. The decay factor is set such that the mean latency value decays to include only 12% of its initial value over 8 samples, ensuring that our service latency estimates adapt quickly (this is also the default decay value used for the process thinktimes and seekiness estimates in Linux). Then we compute a latency scaling factor for the disk service latency with an assumption of 10ms service latency as the baseline disk latency in the non-virtualized case (it is pertinent to note that most of the default parameters in the Linux disk schedulers are also likely derived based on the very same assumption). Finally, we use this scaling factor to scale up the default algorithmic parameters for the anticipatory and CFQ schedulers, over time, to see if that results in better achievement of their algorithmic objectives.

5.1.1 Adaptive Anticipatory Scheduler

For the anticipatory scheduler, we scale up the anticipation timeout (antic_expire) using the latency scaling factor over time. When the virtual disk latencies are low a small scaling of the timeout is sufficient to prevent deceptive idleness, whereas when the latencies are high a larger scaling of the timeout value may be required to achieve the same. Note that such dynamic setting of the timeout value ensures that we attain a good trade-off between throughput (lost due to idling) and deceptive idleness mitigation. Setting a high value for the scaling factor (increasing idling time) only happens when the disk service latencies themselves are higher. This may not necessarily cause a significant loss in throughput, because submitting a request from another process instead of idling is not going to improve throughput if the virtual disk itself does not get any faster than it is at the current period. A higher anticipation timeout might also be capable of absorbing process scheduling effects inside the VM. The results for the adaptive anticipatory scheduler are shown in Figure 2. The read time with our modified implementation (third bar in the different scheduler combinations) shows that it is possible to mitigate the effects of deceptive idleness by adapting the timeout. An interesting related observation is that the level to which the improvement is possible varies for different Domain-0 schedulers: noop - 39%, anticipatory - 67% and cfq - 36%. This again points to the fact that the I/O scheduler used in Domain-0 is important for the VM’s ability in enforcing I/O scheduling guarantees. Different Domain-0 I/O schedulers likely have a different service latency footprint inside the VMs, contributing to different levels of improvement.

5.1.2 Adaptive CFQ Scheduler

We use the scaling factor described previously to scale several tunables of the CFQ scheduler. These are listed below:\(^1\):

- \texttt{cfq.slice.sync} - represents the timeslice allocated to processes doing synchronous I/O. The default value

\(^1\)All the default values mentioned are assuming a kernel clock tick rate of 100 Hz.
in Linux is 100ms.

- `cfq_slice_async` - represents the timeslice allocated to processes doing asynchronous I/O. The default value in Linux is 40ms.
- `cfq_slice_idle` - the idle timeout within a timeslice that triggers timeslice truncation. The default value in Linux is 10ms.
- `cfq_fifo_expire_sync` - the deadline for read requests. The default value in Linux is 125ms.
- `cfq_fifo_expire_async` - the deadline for write requests. The default value in Linux is 250ms.

As explained with the adaptive anticipatory scheduler, the use of large values for the timeslices does not necessarily result in reduced throughput if the virtual disk latencies themselves are high. The inter-process fairness results for the adaptive CFQ scheduler shown in the last row of Table 2. The results indicate that the adaptive setting of the CFQ parameter alone is insufficient to ensure the improved improvement in fairness across all schedulers. As explained previously, with randomly varying virtual disk latencies, the number of requests dispatched per process timeslice is bound to vary across timeslices. A long-latency request is likely to result in early expiration of the idle-timeout as it causes the issuing process to block for longer on the request. On the other hand, short-latency requests (e.g., writes getting buffered in Domain-0) result in more temporally adjacent requests from the process being serviced in the same timeslice. This non-determinism in number of requests processed per timeslice is not solved by merely scaling the timeslices and the idle timeout as long as the virtual disk latencies vary too much. In other words, fairness is a much stricter performance objective than deceptive idleness mitigation (i.e., prevent writes starving reads at the expense of writes).

5.2 Service Latency Smoothing in VMM

The previous subsection shows that adaptive derivation of VM disk I/O scheduling parameters alone (i.e., fixing just one layer of our system) is not sufficient to ensure the improved improvement of VM level disk I/O performance objectives. The adaptive CFQ scheduler’s inability to achieve fairness across processes in the VM is primarily due to the random virtual disk latencies determined by the I/O scheduling that is done at the VMM layer on behalf of all VMs. This points to the need to explicitly manage and shape the guest perceived disk latency. Ideally, the rate of change of virtual disk request latency should be gradual enough for the VM-level schedulers to adapt gracefully to available server levels. In addition, such shaping of the observed request latency characteristics also serves to improve the accuracy of the adaptive virtual disk latency measurement inside the VM. The implication of this improved accuracy is that the algorithmic parameters can be scaled just enough to apply desired service objectives without being overly conservative in disk idling thereby losing out on throughput. Our recent work [12] experimentally demonstrates that, for network devices, such shaping of VM perceived request latency in the VMM provides better achievement of network performance objectives in VMS with active TCP congestion avoidance mechanisms.

5.3 VMM Bypass Disk I/O

Recent trends in hardware support for virtualization have made CPU [15, 2] and memory [8, 3] be virtualized inexpensively. For I/O, device-level hardware support for virtualization has existed in more specialized I/O technologies, such as InfiniBand. Broader penetration of similar VMM-bypass solutions for virtualized I/O have only recently been gaining attention, through technologies such as SR-IOV. However, studies that quantify the benefits and overheads of these solutions for disk devices have been far and few in between. While we have not experimentally evaluated such devices, we believe that the choices of the hardware level scheduler that shares the underlying disk device between multiple bursty VMs has a similar impact on the request latency perceived inside a VM, as with the software-based I/O virtualization solutions evaluated in this paper. For example, scheduling and ordering VM requests focused solely on improving disk throughput might cause variations in the latency of a VM’s requests when run together with other VMs. In fact, our group’s work with virtualized InfiniBand devices has demonstrated the presence of such variations for network I/O [17, 18]. As we already demonstrate in the prior sections, such uncertainty in the request latencies makes it hard for the VM level schedulers to enforce application level performance objectives. Therefore, we believe that the focus of such hardware-level scheduling methods, should not just be the overall improvement of disk throughput and bandwidth fairness amongst VMs, but also the appropriate shaping of the I/O request latency of a given VM when servicing multiple bursty VMs.

6. CONCLUSIONS AND FUTURE WORK

In this paper we demonstrate that virtual disks exposed to VMs on virtualized platforms have latency characteristics quite different from physical disks, largely determined by the I/O characteristics of other VMs consolidated on the same node, and by the behavior of the disk scheduling policy in the virtualization layer. This not only affects VM performance, but limits the ability of the VM-level schedulers to enforce isolation and fair utilization of the VM’s share of I/O resources among applications or application components within the VM. In order to mitigate these issues, we argue the need for both VM-, VMM-level and possibly hardware level modifications to current disk scheduling techniques. We implement and evaluate a VM-level solution for two common Linux schedulers – anticipatory and CFQ. Our basic enhancements to the anticipatory scheduler result in improvements in application performance and the VM-level process isolation capabilities. The case of the CFQ scheduler, however, provides additional insights into the required VMM-level behavior and the enhancements of their schedulers necessary to achieve further improvements, as well as, more generally, design guidelines for next generation disk I/O schedulers for the virtualized environment.

Future work will include the realization and study of a complete solution to disk I/O scheduling with different storage technologies including SAN solutions and realistic datacenter workloads. In addition, while the improvements presented in Section 5.1.1 are significant, we recognize that our simplistic adaptation function may not have general applicability and that further investigation of the benefits, limitations and the nature of the adaptation for other workload patterns is necessary. We plan to pursue this next.
7. REFERENCES


