On Application Responsiveness and Storage Latency in Virtualized Environments

Alexander Spyridakis, Daniel Raho
Virtual Open Systems
Grenoble - France
Email: {a.spyridakis, s.raho}@virtualopensystems.com

***Abstract*—Preserving responsiveness is an enabling condition for running interactive applications effectively in virtual machines. For this condition to be met, low latency usually needs to be guaranteed to storage-Input/Output operations. In contrast, in this paper we show that in virtualized environments, there is a missing link exactly in the chain of actions performed to guarantee low storage-I/O latency. After describing this problem theoretically, we show its practical consequences through a large set of experiments with real world-applications. For the experiments, we used two Linux production-quality schedulers, both designed to guarantee a low latency, and a publicly available I/O benchmark suite, after extending it to correctly measure throughput and application responsiveness also in a virtualized environment. Finally, as for the experimental testbed, we ran our experiments on the following three devices connected to an ARM embedded system: an ultra-portable rotational disk, a microSDHC (Secure Digital High Capacity) Card and an eMMC (embedded Multimedia card) device. This is an ideal testbed for highlighting latency issues, as it can execute applications with about the same I/O demand as a general-purpose system, but for power-consumption and mobility issues, the storage devices of choice for such a system are the aforementioned ones. Additionally, the lower the speed of a storage device is, the consequences of I/O-latency are more evident.**

Keywords: KVM/ARM, virtualization, responsiveness and soft-real time guarantees, scheduling, embedded systems

I. INTRODUCTION

Virtualization is an increasingly successful solution to achieve both flexibility and efficiency in general-purpose and embedded systems. However, for virtualization to be effective also with interactive applications, the latter must be guaranteed a high, or at least acceptable responsiveness. In other words, it is necessary to guarantee that these applications take a reasonably short time to start, and that the tasks requested by these applications, such as, e.g., opening a file, are completed in a reasonable time.

To guarantee responsiveness to an application, it is necessary to guarantee that both the code of the application and the I/O requests issued by the applications get executed with a low latency. Expectedly, there is interest and active research in preserving a low latency in virtualized environments [1][2][3][4][5], especially in soft and hard real-time contexts. In particular, some virtualization solutions provide more or less sophisticated Quality of Service mechanisms also for storage I/O [4][5]. However, even just a thorough investigation on application responsiveness, as related to storage-I/O latency, seems to be missing. In this paper, we address this issue by providing the following contributions.

A. Contributions of this paper

First, we show, through a concrete example, that in a virtualized environment there is apparently a missing link in the chain of actions performed to guarantee a sufficiently low I/O latency when an application is to be loaded, or, in general, when any interactive task is to be performed. To this purpose, we use as a reference two effective schedulers in guaranteeing a high responsiveness: Budget Fair Queuing [7] and Completely Fair Queuing [9]. They are two production-quality storage-I/O schedulers for Linux.

Then, we report experimental results with real-world applications. These results confirm that, if some applications are competing for the storage device in a host, then the applications running in a virtual machine executed in the same host may become from not much responsive to completely unresponsive. To carry out these experiments, we extended a publicly available I/O benchmark suite for Linux [8], to let it comply also with a virtualized environment.

As an experimental testbed, we opted for an ARM embedded system, based on the following considerations. On one hand, modern embedded systems and consumer-electronics devices can execute applications with about the same I/O demand as general-purpose systems. On the other hand, for mobility and energy-consumption issues, the preferred storage devices in the former systems are (ultra) portable and low-power ones. These devices are necessarily slower than their typical counterparts for general-purpose systems. Being the amount of I/O the same, the lower the speed of a storage device is, the more I/O-latency issues are amplified. Finally, as a virtualization solution we used the pair QEMU/KVM, one of the most popular and efficient solutions in ARM embedded systems.

B. Organization of this paper

In Section II, we describe the schedulers that we use as a reference in this paper. Then, in Section III we show the important I/O-latency problem on which this paper is focused. After that, in Section IV, we describe how we modified the benchmark suite to execute our experiments. Finally, we report our experimental results in Section V.

II. REFERENCE SCHEDULERS

To show the application-responsiveness problem that is the focus of this paper, we use the following two storage-I/O schedulers as a reference: BFQ [6] and CFQ [9]. We opted for these two schedulers because, they, both guarantee a high

 ISBN: 978-1-61208-338-4

throughput and low latency. In particular, BFQ achieves even up to 30% higher throughput than CFQ on hard disks with parallel workloads. Strictly speaking, only the second feature is related to the focus of this paper, but the first feature is however important, because a scheduler achieving only a small fraction of the maximum possible throughput may be, in general, of little practical interest, even if it guarantees a high responsiveness. The second reason why we opted for these schedulers is that up-to-date and production-quality Linux implementations are available for both. In particular, CFQ is the default Linux I/O scheduler, whereas BFQ is being maintained separately [8]. In addition to the extended tests for BFQ and CFQ, we also identified similar behaviour with the Noop and Deadline schedulers. In the next two sections, we briefly describe the main differences between the two schedulers, focusing especially on I/O latency and responsiveness. For brevity, when not otherwise specified, in the rest of this paper we use the generic term *disk* to refer to both a hard disk and a solid-state disk.

A. BFQ

BFQ achieves a high responsiveness basically by providing a high fraction of the disk throughput to an application that is being loaded, or whose tasks must be executed quickly. In this respect, BFQ benefits from the strong fairness guarantees it provides: BFQ distributes the disk throughput (and not just the disk time) as desired to disk-bound applications, with any workload, *independently of* the disk parameters and even if the disk throughput fluctuates. Thanks to this strong fairness property, BFQ does succeed in providing an application requiring a high responsiveness with the needed fraction of the disk throughput in any condition. The ultimate consequence of this fact is that, regardless of the disk background workload, BFQ guarantees to applications about the same responsiveness as if the disk was idle [6].

B. CFQ

CFQ grants disk access to each application for a fixed *time slice*, and schedules slices in a round-robin fashion. Unfortunately, as shown by Valente and Andreolini [6], this service scheme may suffer from both unfairness in throughput distribution and high worst-case delay in request completion time with respect to an ideal, perfectly fair system. In particular, because of these issues and of how the low-latency heuristics work in CFQ, the latter happens to guarantee a worse responsiveness than BFQ [6]. This fact is highlighted also by the results reported in this paper.

III. Missing link for preserving responsiveness

We highlight the problem through a simple example. Consider a system running a guest operating system, say guest G, in a virtual machine, and suppose that either BFQ or CFQ is the default I/O scheduler both in the host and in guest G. Suppose now that a new application, say application A, is being started (loaded) in guest G while other applications are already performing I/O without interruption in the same guest. In these conditions, the cumulative I/O request pattern of guest G, as seen from the host side, may exhibit no special property that allows the BFQ or CFQ scheduler in the host to realize that an application is being loaded in the guest.

Hence, the scheduler in the host may have no reason for privileging the I/O requests coming from guest G. In the end, if also other guests or applications of any other kind are performing I/O in the host—and for the same storage device as guest G—then guest G may receive *no help* to get a high-enough fraction of the disk throughput to start application A quickly. As a conclusion, the start-up time of the application may be high. This is exactly the scenario that we investigate in our experiments. Finally, it is also important to note that our focus has been in local disk/storage, as scheduling of network-based storage systems is not always under the direct control of the Linux scheduling policies.

IV. Extension of the benchmark suite

To implement our experiments we used a publicly available benchmark suite [8] for the Linux operating system. This suite is designed to measure the performance of a disk scheduler with real-world applications. Among the figures of merit measured by the suite, the following two performance indexes are of interest for our experiments:

Aggregate disk throughput. To be of practical interest, a scheduler must guarantee, whenever possible, a high (aggregate) disk throughput. The suite contains a benchmark that allows the disk throughput to be measured while executing workloads made of the reading and/or the writing of multiple files at the same time.

Responsiveness. Another benchmark of the suite measures the *start-up* time of an application—i.e., how long it takes from when an application is launched to when the application is ready for input—with cold caches and in presence of additional heavy workloads. This time is, in general, a measure of the responsiveness that can be guaranteed to applications in the worst conditions.

Being this benchmark suite designed only for non-virtualized environments, we enabled the above two benchmarks to work correctly also inside a virtual machine, by providing them with the following extensions:

Choice of the disk scheduler in the host. Not only the active disk scheduler in a guest operating system, hereafter abbreviated as just guest OS, is relevant for the I/O performance in the guest itself, but, of course, also the active disk scheduler in the host OS. We extended the benchmarks so as to choose also the latter scheduler.

Host-cache flushing. As a further subtlety, even if the disk cache of the guest OS is empty, the throughput may be however extremely high, and latencies may be extremely low, in the guest OS, if the zone of the guest virtual disk interested by the I/O corresponds to a zone of the host disk already cached in the host OS. To address this issue, and avoid deceptive measurements, we extended both benchmarks to flush caches at the beginning of their execution and, for the responsiveness benchmark, also (just before) each time the application at hand is started. In fact the application is started for a configurable number of times, see Section V.

Workload start and stop in the host. Of course, responsiveness results now depend also on the workload in execution in the host. Actually, the scenario where the responsiveness in a Virtual Machine (VM) is to be

TABLE I. Storage devices used in the experiments

Type	Name	Size	Read peak rate
1.8-inch Hard Disk	Toshiba MK6006GAH	60 GB	10.0 MB/s
microSDHC Card	Transcend SDHC Class 6	8 GB	16 MB/s
eMMC	SanDisk SEM16G	16 GB	70 MB/s

carefully evaluated, is exactly the one where the host disk is serving not only the I/O requests arriving from the VM, but also other requests (in fact this is the case that differs most from executing an OS in a non-virtualized environment). We extended the benchmarks to start the desired number of file reads and/or writes also in the host OS. Of course, the benchmarks also automatically shut down the host workload when they finish.

The resulting extended version of the benchmark suite is available here [10]. This new version of the suite also contains the general scripts that we used for executing the experiments reported in this paper (all these experiments can then be repeated easily).

V. EXPERIMENTAL RESULTS

We executed our experiments on a Samsung Chromebook, equipped with an ARMv7-A Cortex-A15 (dual-core, 1.7 GHz), 2 GB of RAM and the devices reported in Table I. There was only one VM in execution, hereafter denoted as just *the* VM, emulated using QEMU/KVM. Both the host and the guest OSes were Linux 3.12.

A. Scenarios and measured quantities

We measured, first, the aggregate throughput in the VM while one of the following combinations of workloads was being served.

In the guest. One of the following six workloads, where the tag **type** can be either **seq** or **rand**, with **seq**/**rand** meaning that files are read or written sequentially/at random positions:

1r-type one reader (i.e., one file being read);
5r-type five parallel readers;
2r2w-type two parallel readers, plus two parallel writers.

In the host. One of the following three workloads (in addition to that generated, in the host, by the VM):

no-host_workload no additional workload in the host;
1r-on_host one sequential file reader in the host;
5r-on_host five sequential parallel readers in the host.

We considered only sequential readers as additional workload in the host, because it was enough to cause the important responsiveness problems shown in our results. In addition, for each workload combination, we repeated the experiments with each of the four possible combinations of active schedulers, choosing between BFQ and CFQ, in the host and in the guest.

The main purpose of the throughput experiments was to verify that in a virtualized environment both schedulers achieved a high-enough throughput to be of practical interest. Both schedulers did achieve, in the guest, about the same (good) performance as in the host. For space limitations, we do not report these results, and focus instead on the main quantity of interest for this paper. In this regard, we measured the start-up time of three popular interactive applications of different sizes, inside the VM and while one of the above combinations of workloads was being served.

The applications were, in increasing-size order: *bash*, the Bourne Again shell, *xterm*, the standard terminal emulator for the X Window System, and *konsole*, the terminal emulator for the K Desktop Environment. As shown by Valente and Andreolini [6], these applications allow their start-up time to be easily computed. In particular, to get worst-case start-up times, we dropped caches both in the guest and in the host before each invocation (Section IV). Finally, just before each invocation a timer was started: if more than 60 seconds elapsed before the application start-up was completed, then the experiment was aborted (as 60 seconds is evidently an unbearable waiting time for an interactive application).

We found that the problem that we want to show, i.e., that responsiveness guarantees are violated in a VM, occurs regardless of which scheduler is used in the host. Besides, in presence of file writers, results are dominated by fluctuations and anomalies caused by the Linux write-back mechanism. These anomalies are almost completely out of the control of the disk schedulers, and not related with the problem that we want to highlight. In the end, we report our detailed results **only with file readers**, **only with BFQ** as the active disk scheduler **in the host**, and **for *xterm***.

B. Statistics details

For each workload combination, we started the application at hand five times, and computed the following statistics over the measured start-up times: minimum, maximum, average, standard deviation and 95% confidence interval (actually we measured also several other interesting quantities, but in this paper we focus only on application responsiveness). We denote as a *single run* any of these sequences of five invocations. We repeated each single run ten times, and computed the same five statistics as above also across the average start-up times computed for each repetition. We did not find any relevant outlier, hence, for brevity and ease of presentation, in the next plots we show only averages across runs (i.e., averages of the averages computed in each run).

C. Results

Figure 1 shows our results with the hard disk (Table I). The reference line represents the time needed to start *xterm* if the disk is idle, i.e., the minimum possible time that it takes to start *xterm* (a little less than 2 seconds). Comparing this value with the start-up time guaranteed by BFQ with no host workload, and with any of the first three workloads in the guest (first bar for any of the *1r-seq*, *5r-seq* and *1r-rand* guest workloads), we see that, with all these workloads, BFQ guarantees about the same responsiveness as if the disk was idle. The start-up time guaranteed by BFQ is slightly higher with *5r-rand*, for issues related, mainly, to the slightly coarse time granularity guaranteed to scheduled events in the kernel in an ARM embedded system, and to the fact that the reference time itself may advance haltingly in a QEMU VM.

In contrast, again with no host workload, the start-up time guaranteed by CFQ with *1r-seq* or *1r-rand* on the guest is 3

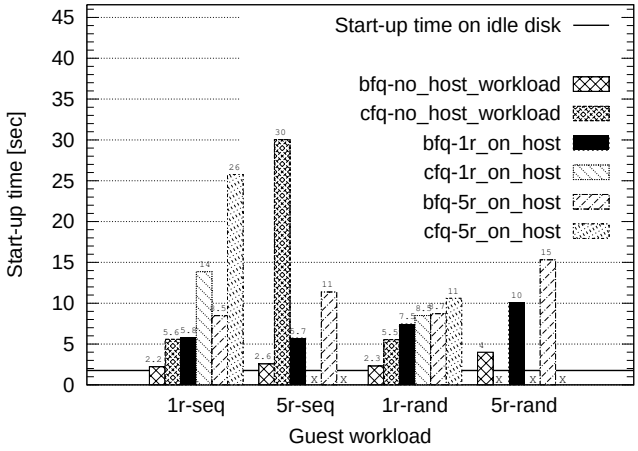

Figure 1. Results with the hard disk (lower is better).

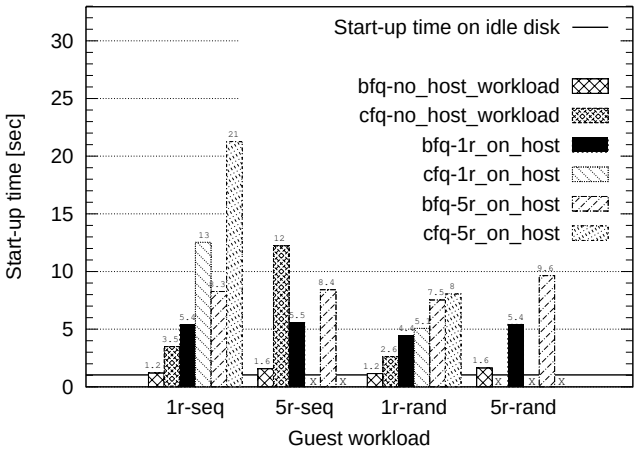

Figure 2. Results with the microSDHC CARD (lower is better).

times as high than on an idle disk, whereas with *5r-seq* the start-up time becomes about 17 times as high. With *5r-seq* the figure reports instead an **X** for the start-up time of CFQ: we use this symbolism to indicate that the experiment failed, i.e., that the application did not succeed at all in starting before the 60-second timeout.

In view of the problem highlighted in Section III, the critical scenarios are however the ones with some additional workload in the host; in particular, *1r_on_host* and *5r_on_host* in our experiments. In these scenarios, both schedulers unavoidably fail to preserve a low start-up time. Even with just *1r_on_host*, the start-up time, with BFQ, ranges from 3 to 5.5 times as high than on an idle disk. The start-up time with CFQ is much higher than with BFQ with *1r_on_host* and *1r-seq* on the guest, and, still with *1r_on_host* (and CFQ), is even higher than 60 seconds with *5r-seq* or *5r-rand* on the guest. With *5r_on_host* the start-up time is instead basically unbearable, or even higher than 60 seconds, with both schedulers. Finally, with *1r-rand* all start-up times are lower and more even than with the other guest workloads, because both schedulers do not privilege much random readers, and the background workload is generated by only one reader.

Figures 2 and 3 show our results with the two flash-based devices. At different scales, the patterns are still about the same as with the hard disk. The most notable differences are related to CFQ: on one side, with no additional host workload, CFQ achieves a slightly better performance than on the hard disk, whereas, on the opposite side, CFQ suffers from a much higher degradation of the performance, again with respect to the hard-disk case, in presence of additional host workloads.

To sum up, our results confirm that, with any of the devices considered, responsiveness guarantees are lost when there is some additional I/O workload in the host.

VI. Conclusion and Future Work

In this paper, we showed both theoretically and experimentally that responsiveness guarantees, as related to storage I/O, may be violated in virtualized environments. Even with schedulers, which target to achieve low latency through heuristics, the problem of low responsiveness still persists

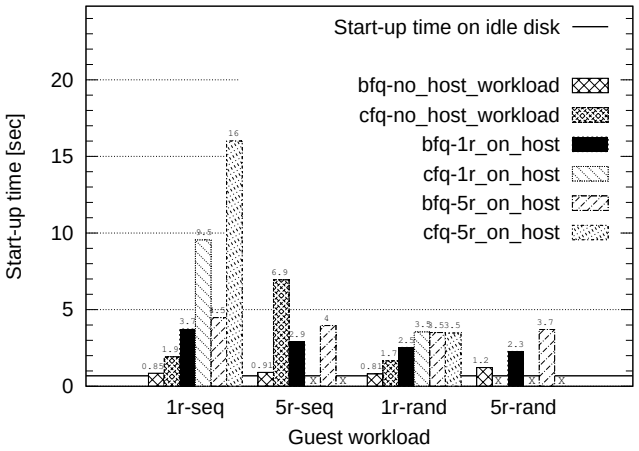

Figure 3. Results with the eMMC (lower is better).

in virtual machines. The host receives a mix of interactive and background workloads from the guest, which can completely contradict per process heuristics by schedulers such as BFQ. We are currently devising a solution for preserving responsiveness also in virtualized environments. The target of this approach is specifically for embedded systems and the KVM on ARM hypervisor, which introduces the concept of coordinated scheduling between the host/guest scheduler and KVM itself. Besides, we also plan to extend our investigation to latency guarantees for soft real-time applications (such as audio and video players), and to consider more complex scenarios, such as more than one VM competing for the storage device.

Acknowledgments

The authors would like to thank Paolo Valente for providing details about BFQ and his support for the benchmark suite. We also thank the anonymous reviewers for their precious feedback and comments on the manuscript.

REFERENCES

[1] Jaewoo Lee, et. al., “Realizing Compositional Scheduling through Virtualization”, IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS’12), April 2012, pp. 237-246.

[2] K. Sandstrom, A. Vulgarakis , M. Lindgren, and T. Nolte, “Virtualization technologies in embedded real-time systems”, Emerging Technologies & Factory Automation (ETFA), 2013 IEEE 18th Conference on, Sept. 2013, pp. 1-8.

[3] Z. Gu and Q. Zhao, “A State-of-the-Art Survey on Real-Time Issues in Embedded Systems Virtualization”, Journal of Software Engineering and Applications, Vol. 5 No. 4, 2012, pp. 277-290.

[4] Storage I/O Control Technical Overview [retrieved: April, 2014]. http://www.vmware.com/files/pdf/techpaper/VMW-vSphere41-SIOC.pdf

[5] Virtual disk QoS settings in XenEnterprise [retrieved: April, 2014]. http://docs.vmd.citrix.com/XenServer/4.0.1/reference/ch04s02.html

[6] P. Valente and M. Andreolini, “Improving Application Responsiveness with the BFQ Disk I/O Scheduler”, Proceedings of the 5th Annual International Systems and Storage Conference (SYSTOR ’12), June 2012, p. 6

[7] F. Checconi and P. Valente, “High Throughput Disk Scheduling with Deterministic Guarantees on Bandwidth Distribution”, IEEE Transactions on Computers, vol. 59, no. 9, May 2010.

[8] BFQ homepage [retrieved: April, 2014]. http://algogroup.unimore.it/people/paolo/disk_sched

[9] CFQ I/O Scheduler [retrieved: April, 2014]. http://lca2007.linux.org.au/talk/123.html

[10] Extended version of the benchmark suite for virtualized environments [retrieved: April, 2014]. http://www.virtualopensystems.com/media/bfq/benchmark-suite-vm-ext.tgz

 ISBN: 978-1-61208-338-4