NORTHWESTERN
UNIVERSITY

Computer Science Department

Technical Report
NWU-CS-05-06
April 14, 2005

VSched: Mixing Batch And Interactive Virtual Machines Using Periodic Real-time Scheduling

Bin Lin, Peter A. Dinda

Abstract

We are developing Virtuoso, a system for distributed computing using virtual machines (VMs). Virtuoso must be able to mix batch and interactive VMs on the same physical hardware, while satisfying constraints on responsiveness and compute rates for each workload. VSched is the component of Virtuoso that provides this capability. VSched is an entirely user-level tool that interacts with the stock Linux kernel running below any type-II virtual machine monitor to schedule all VMs (indeed, any process) using a periodic real-time scheduling model. This abstraction allows compute rate and responsiveness constraints to be straightforwardly described using a period and a slice within the period, and it allows for fast and simple admission control. This paper makes the case for periodic real-time scheduling for VM-based computing environments, and then describes and evaluates VSched. It also applies VSched to scheduling parallel workloads, showing that it can help a BSP application maintain a fixed stable performance despite externally caused load imbalance.

Effort sponsored by the National Science Foundation under Grants ANI-0093221, ACI-0112891, ANI-0301108, EIA-0130869, and EIA-0224449, and aided by support from VMware and Dell. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author and do not necessarily reflect the views of the National Science Foundation (NSF), VMware, or Dell.

Keywords: real-time scheduling, interactive workloads, batch workloads, parallel workloads, grid computing, virtual machines, resource management

VSched: Mixing Batch And Interactive Virtual Machines Using Periodic Real-time Scheduling

Bin Lin Peter A. Dinda
{binlin, pdinda}@cs.northwestern.edu
Department of Computer Science, Northwestern University

Abstract

We are developing Virtuoso, a system for distributed computing using virtual machines (VMs). Virtuoso must be able to mix batch and interactive VMs on the same physical hardware, while satisfying constraints on responsiveness and compute rates for each workload. VSched is the component of Virtuoso that provides this capability. VSched is an entirely user-level tool that interacts with the stock Linux kernel running below any type-II virtual machine monitor to schedule all VMs (indeed, any process) using a periodic real-time scheduling model. This abstraction allows compute rate and responsiveness constraints to be straightforwardly described using a period and a slice within the period, and it allows for fast and simple admission control. This paper makes the case for periodic real-time scheduling for VM-based computing environments, and then describes and evaluates VSched. It also applies VSched to scheduling parallel workloads, showing that it can help a BSP application maintain a fixed stable performance despite externally caused load imbalance.

1 Introduction

We are developing Virtuoso, a middleware for virtual machine distributed computing that very closely emulates the process of buying, configuring, and using an Intel-based computer or collection of computers from a web site, a process with which many users and certainly all system administrators are familiar. Instead of a physical computer, the user receives a reference to the virtual machine which he can then use to start, stop, reset, and clone the machine. The system presents the illusion that the virtual machine is right next to the user in terms of console display, devices, and the network. Virtuoso currently uses VMWare GSX Server, a type-II virtual machine [16], as its virtual machine monitor (VMM), though other VMMs can in principle be substituted. Details about the Virtuoso implementation [36], its virtual networking system [38], its application topology inference system [17] and its dynamic adaptation system [39] can be found in the references, as can a detailed case for grid computing on virtual machines [12].

Virtuoso is designed to support a wide range of workloads that its simple user-level abstraction makes possible. Three workload types that drove our design process are:

- Interactive workloads which occur when using a remote VM to substitute for a desktop computer. These workloads include desktop applications, web applications and games.
- Batch workloads, such as scientific simulations or analysis codes. These workloads are commonplace in grid computing [13].
- Batch parallel workloads, such as scientific simulations or analysis codes that can be scaled by adding more VMs. These are also commonplace in grid computing. Typically, it is desirable for such workloads to be gang scheduled [25].

Today, both sequential and parallel batch jobs are often scheduled using advance reservations [24, 37] such that they will finish by some deadline. Resource providers in Virtuoso price VM execution according to interactivity and compute rate constraints; thus, its scheduling model must be able to validate and enforce these constraints.

An important challenge in Virtuoso is how to schedule a workload-diverse set of VMs on a single physical machine so that interactivity does not suffer and batch machines meet both their advance reservation deadlines and gang scheduling constraints. It is that challenge that VSched addresses. VSched provides a unified periodic real-time scheduling model that can address the various constraints of different kinds of VMs. VSched is an entirely user-level Linux tool that is remotely controlled by Virtuoso. Its main requirements are a 2.4 or 2.6 Linux kernel and root privileges, in addition it can make use of

the KURT high resolution timer [20] to permit very fine grain schedules. It can work with any type-II VMM that runs the VM as a Linux process.

VSched is publicly released and can be downloaded from http://virtuoso.cs.northwestern.edu.

2 VSched

VSched schedules a collection of VMs on a host according to the model of independent periodic real-time tasks. Tasks can be introduced or removed from control at any point in time through a client/server interface. Virtuoso uses this interface to enforce compute rate and interactivity commitments a provider has made to a VM.

2.1 Abstraction

The periodic real-time model is a unifying abstraction that can provide for the needs of the various classes of applications described above. In the periodic real-time model, a task is run for $slice$ seconds every $period$ seconds. Typically, the periods start at time zero. Using earliest deadline first (EDF) schedulability analysis [29], the scheduler can determine whether some set of $(period, slice)$ constraints can be met. The scheduler then simply uses dynamic priority preemptive scheduling with the deadlines of the admitted tasks as priorities.

VSched offers soft real-time guarantees. Because the Linux kernel does not have priority inheritance mechanisms, nor known bounded interrupt service times, it is impossible for a tool like VSched to provide hard real-time guarantees to ordinary processes. Nonetheless, as we show in our evaluation, for a wide range of periods and slices, and under even fairly high utilization, VSched almost always meets the deadlines of its tasks.

In typical soft and hard embedded real-time systems, the $(period, slice)$ constraint of a task is usually measured in the microseconds to low milliseconds. VSched is unusual in that it supports periods and slices ranging into days. While fine, millisecond and sub-millisecond ranges are needed for highly interactive VMs, much coarser resolutions are appropriate for batch VMs.

It is important to realize that the ratio $slice/period$ defines the *compute rate* of the task.

Batch VMs Executing a VM under the constraint $(period, slice)$ for T seconds gives us at least $slice \times \lfloor T/period \rfloor$ seconds of CPU time within T seconds. In this way, the periodic real-time model can be used to express a deadline for the entire execution of the batch VM.

Batch parallel VMs A parallel application may be run in a collection of VMs, each of which is scheduled with the same $(period, slice)$ constraint. If each VM is given the same schedule and starting point, then they can run in lock step, avoiding the synchronization costs of typical gang scheduling. If the constraint accurrately reflects the application's compute/communicate balance, then there should be minimal undesired performance impact as we control the execution rate. Because the schedule is a reservation, the application should be impervious to external load.

Interactive VMs Based on an in-depth study of users operating interactive applications such as word processors, presentation graphics, web browsers, and first-person shooter games, we have reached a number of conclusions about how to keep users of such applications happy [18]. The points salient to this paper are that the CPU rates and jitter needed to keep the user happy is highly dependent on the application and on the user. We believe we need to incorporate direct user feedback in scheduling interactive applications running in VMs.

In earlier work [27], we explored using a single "irritation button" feedback mechanism to control VM priority. This approach proved to be too course-grained. The two-dimensional control possible with the $(period, slice)$ mechanism is much finer-grained. An important design criterium for VSched is that a VM's constraints can be changed very quickly (in milliseconds) so that an interactive user can improve his VM's performance immediately or have the system migrate it to another physical machine if his desired $(period, slice)$ is impossible on the original machine.

2.2 Related work

Existing approaches to scheduling VMs running under a type-II VMM on Linux (and other Unixes) are insufficient to meet the needs of the workloads listed above. By default, these VMs are scheduled as ordinary dynamic-priority processes with no timing or compute rate constraints at all. VMWare ESX server [40] and virtual server systems such as Ensim [11] improve this situation by providing compute rate constraints using weighted fair queuing [4] and lottery scheduling [41]. However, these are insufficient for our purposes because they either provide no timing constraints or do not allow for the timing constraints to be smoothly varied. Fundamentally, they are rate-based. For example, an interactive VM in which a word processing application is being used may only need 5% of the CPU, but it will need to be run at least every 50 ms or so. Similarly, a VM that is running a parallel application may need 50% of the CPU, and be scheduled together with its companion VMs. The closest VM-specific scheduling approach to

ours is the VServer [28] slice scheduling in the PlanetLab testbed [34]. However, these slices are created a priori and fixed. VSched provides dynamic scheduling.

Periodic real-time scheduling systems for general-purpose operating systems have been developed before. Most relevant to our work is Polze's scheduler [33], which created soft periodic schedules for multimedia applications by manipulating priorities under Windows NT. DSRT [6], SMART [32], and Rialto [26] had similar objectives. In contrast, VSched is a Linux tool, provides remote control for systems like Virtuoso, and focuses on scheduling VMs. Linux SRT, defunct since the 2.2 kernel, was a set of kernel extensions to support soft real-time scheduling for multimedia applications under Linux [23]. The RBED system [35] also provides real-time scheduling for general Linux processes through kernel modifications. The Xen [9] virtual machine monitor uses BVT [10] scheduling with a non-trivial modification of Linux kernel. In contrast to these systems, VSched can operate entirely at user-level.

There have been several hard real-time extensions to Linux. The best known of these are Real-time Linux [42], RTAI [8], and KURT [20]. We examined these tools (and Linux SRT as well) before deciding to develop VSched. For our purposes the hard real-time extensions are inappropriate because real-time tasks must be written specifically for them. In the case of Real-time Linux, the tasks are even required to be kernel modules. We can optionally use KURT's UTIME high resolution timers to achieve very fine grain scheduling of VMs in VSched.

3 System design

VSched uses the schedulability test of the earliest-deadline-first (EDF) algorithm [29, 30] to do admission control and EDF scheduling to meet deadlines. It is a user-level program that uses fixed priorities within Linux's SCHED_FIFO scheduling class and SIGSTOP/SIGCONT to control other processes, leaving aside some percentage of CPU time for processes that it does not control. The resolution at which it can schedule depends on timer resolution in the system, and thus its resolution depends on the Linux kernel version and the existence of add-on high-resolution timers. VSched consists of a parent and a child process that communicate via a shared memory segment and a pipe. The following describes the design of VSched in detail.

3.1 Algorithms

A well-known dynamic-priority algorithm is EDF (Earliest Deadline First). It is a preemptive policy in which tasks are prioritized in reverse order of the impending deadlines. We assume that the deadlines of our tasks occur at the ends of their periods, although this is not required by EDF.

Given a system of n independent periodic tasks, there is a fast algorithm to determine if the tasks, if scheduled using EDF, will all meet their deadlines:

$$U(n) = \sum_{k=1}^{n} \frac{slice_k}{period_k} \leq 1 \qquad (1)$$

Here, $U(n)$ is the total utilization of the task set being tested. Equation 1 is both a necessary and sufficient condition for any system of n independent, preemptable tasks that have relative deadlines equal to their respective periods to be schedulable by EDF [30].

3.2 Mechanisms

SCHED_FIFO Three scheduling policies are supported in the current Linux kernel: SCHED_FIFO, SCHED_RR and SCHED_OTHER. SCHED_OTHER is the default universal time-sharing scheduler policy used by most processes. It is a preemptive, dynamic-priority scheduler. SCHED_FIFO and SCHED_RR are intended for special time-critical applications that need more precise control over the way in which runnable processes are selected for execution. Within each policy, different priorities can be assigned, with SCHED_FIFO priorities being strictly higher than SCHED_RR priorities which are in turn strictly higher than SCHED_OTHER priorities. SCHED_FIFO priority 99 is the highest priority in the system and it is the priority at which the scheduling core of VSched runs. The server front-end of VSched runs at priority 98. No other processes at these priority levels are allowed.

SCHED_FIFO is a simple preemptive scheduling policy without time slicing. For each priority level in SCHED_FIFO, the kernel maintains a FIFO queue of processes. The first runnable process in the highest priority queue with any runnable processes runs until it blocks, at which point it is placed at the back of its queue. When VSched schedules a VM to run, it sets it to SCHED_FIFO and assigns it a priority of 97, just below that the VSched server front-end. No other processes at this priority level are allowed.

The following rules are applied by the kernel: A SCHED_FIFO process that has been preempted by another process of higher priority will stay at the head of the list for its priority and will resume execution as soon as all processes of higher priority are blocked again. When a SCHED_FIFO process becomes runnable, it will be inserted at the end of the list for its priority. A system call to `sched_setscheduler` or

`sched_setparam` will put the SCHED_FIFO process at the end of the list if it is runnable. No other events will move a process scheduled under the SCHED_FIFO policy in the queue of runnable processes with equal static priority. A SCHED_FIFO process runs until either it is blocked by an I/O request, it is preempted by a higher priority process, or it calls `sched_yield`. The upshot is that the process that VSched has selected to run is the one with the earliest deadline. It will run whenever it is ready until VSched becomes runnable.

Timers After configuring a process to run at SCHED_FIFO priority 97, the VSched core waits (blocked) for one of two events using the `select` system call. It continues when it is time to change the currently running process (or to run no process) or when the set of tasks has been changed via the server front-end.

The resolution that VSched can achieve is critically dependent on the available timer. Under the standard 2.4.x Linux kernel, the timer offers 10 ms resolution. For many applications this is sufficient. However, especially interactive applications, such as games or low-latency audio playback require finer resolution. When running on a 2.6.x Linux kernel, VSched achieves 1 ms resolution because the timer interrupt rate has been raised to 1000 Hz. The UTIME component of KURT-Linux [20] uses the motherboard timers to deliver asynchronous timer interrupts with resolution in the tens of μs. In VSched, we call `select` with a non-null timeout as a portable way to sleep with whatever precision is offered in the underlying kernel. Since UTIME extends `select`'s precision when it's installed, VSched can offer sub-millisecond resolution in these environments. Note, however, that the overhead of VSched is considerably higher than UTIME, so the resolution is in the 100s of μs.

SIGSTOP/SIGCONT By using EDF scheduling to determine which process to raise to highest priority, we can assure that all admitted processes meet their deadlines. However, it is possible for a process to consume more than its slice of CPU time. VSched can optionally limit a VM to exactly the slice that it requested by using the SIGSTOP and SIGCONT signals to suspend and resume the VM. Although this adds overhead, we envision this as critical in a commercial environment.

3.3 Structure

VSched consists of a server and a client, as shown in Figure 1. The VSched server is a daemon running on Linux that spawns the scheduling core, which executes the scheduling scheme described above. The VSched

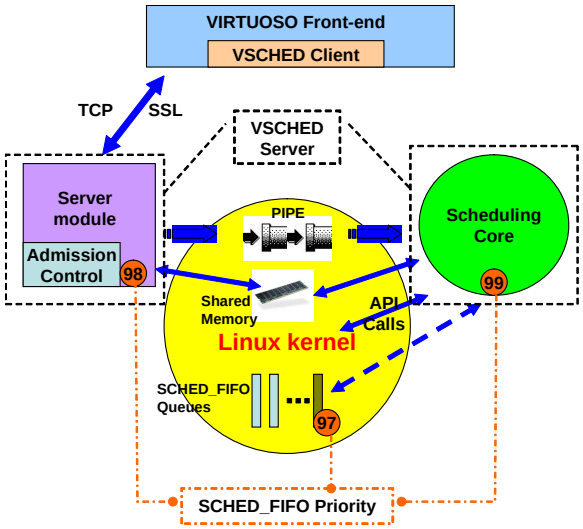


Figure 1. Structure of VSched.

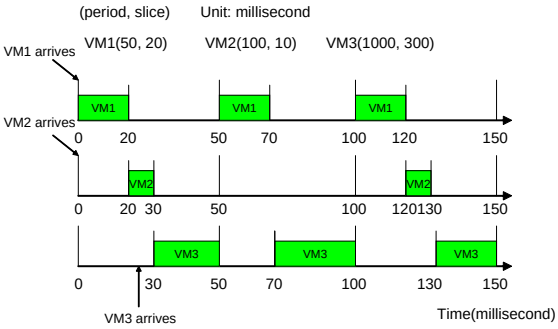


Figure 2. A detailed VSched schedule for three VMs.

client communicates with the server over a TCP connection that is encrypted using SSL. Authentication is accomplished by a password exchange. The server communicates with the scheduling core through two mechanisms. First, they share a memory segment which contains an array that describes the current tasks to be scheduled as well as their constraints. Access to the array is guarded via a semaphore. The second mechanism is a pipe from server to core. The server writes on the pipe to notify the core that the schedule has been changed.

Client interface Using the VSched client, a user can connect to VSched server and request that any process be executed according to a period and slice. Virtuoso keeps track of the *pids* used by its VMs. For example, the specification (3333, 1000 ms, 200 ms) would mean that process 3333 should be run for 200 ms every 3000 ms. In response to such a request, the VSched server

Machine 1: Pentium 4, 2.00GHz, 512MB Mem, Linux version 2.4.20-31.9 (Red Hat Linux 9.0)
Machine 2: Dual CPUs (Pentium III Coppermine, 1.0 GHZ), 1G Mem, non-SMP Linux kernel 2.4.18 patched with KURT 2.4.18-2
Machine 3: Pentium 4, 2.00GHz, 512MB Mem, Linux version 2.6.8.1 (Red Hat Linux 9.0)

Figure 3. Testbed Machines

Kernel version	Machine (from Figure 3)	Utilization Range	Period Range	Slice Range	Deadlines per combination
Linux kernel 2.4.20-31.9	1	10% - 99% (increasing by 10%)	1016 ms - 16 ms (decreasing by 40)	105.8 ms - 1.6 ms	1000
KURT 2.4.18-2	2	1% - 99% (increasing by 1%)	10.1 ms - 1.1 ms (decreasing by 1)	9.999 ms - 0.011 ms	2000
Linux kernel 2.6.8.1	3	1% - 99% (increasing by 1%)	101 ms - 1 ms (decreasing by 10)	99.99 ms - 0.01 ms	2000

Figure 4. Evaluation scenarios.

determines whether the request is feasible. If it is, it will add the process to the array and inform the scheduling core. In either case, it replies to the client.

VSched allows a remote client to find processes, pause or resume them, specify or modify their real-time schedules, and return them to ordinary scheduling. Any process, not just VMs, can be controlled in this way.

Admission control VSched's admission control algorithm is based on Equation 1, the admissibility test of the EDF algorithm. As we mentioned above, it is both a necessary and sufficient condition. Instead of trying to maximize the total utilization, we allow the system administrator to reserve a certain percentage of CPU time for SCHED_OTHER processes. The percentage can be set by the system administrator when starting the VSched daemon.

Scheduling core The scheduling core is a modified EDF scheduler that dispatches processes in EDF order but interrupts them when they have exhausted their allocated CPU for the current period. If configured by the system administrator, VSched will stop the processes at this point, resuming them when their next period begins.

Since a task can miss its deadline only at a period boundary, the scheduling core makes scheduling decisions only at period boundaries, i.e., at the points when a task exhausts its slice for the current period, or when the server indicates that the task set and its constraints have changed. In this way, unlike a kernel-level scheduler [1, 2, 3, 5, 19, 31], VSched is typically invoked only at the rate of the task with the smallest period.

When the scheduling core receives scheduling requests from the server module, it will interrupt the current task and make an immediate scheduling decision based on the new task set. The scheduling request can be a request for scheduling a newly arrived task or for changing a task that has been previously admitted.

Figure 2 illustrates the scheduling of three virtual machines with different arrival times.

4 Evaluation

Our evaluation focuses on the resolution and utilization limits of VSched running on several different platforms. We attempt to answer the following questions: what combinations of period and slice lead to low deadline miss rates and what happens when the limits are exceeded?

We ran our evaluation for three different environments, as shown in Figure 3. The key differences between these environments are the processor speed (1 GHz P3 versus 2 GHz P4) and the available timers (2.4 kernel, 2.4 with KURT, and 2.6 kernel). *For space reasons, we present results for machine 1 only, a stock Red Hat installation that is the most conservative of the three. The final paper will show complete results.*

4.1 Methodology

Our primary metric is the *miss rate*, the number of times we miss the deadlines of a task divided by the total number of deadlines. For tasks that miss their deadlines, we also collect the *miss time*, the time by which the deadline was overrun. We want to understand how the miss rate varies with period and slice (or, equivalently, period and utilization), the number of VMs, and by how much we typically miss a deadline when this happens.

We evaluate first using randomly generated testcases, a testcase being a random number of VMs, each with a different $(period, slice)$ constraint. Next, we do a careful deterministic sweep over period and slice for a single

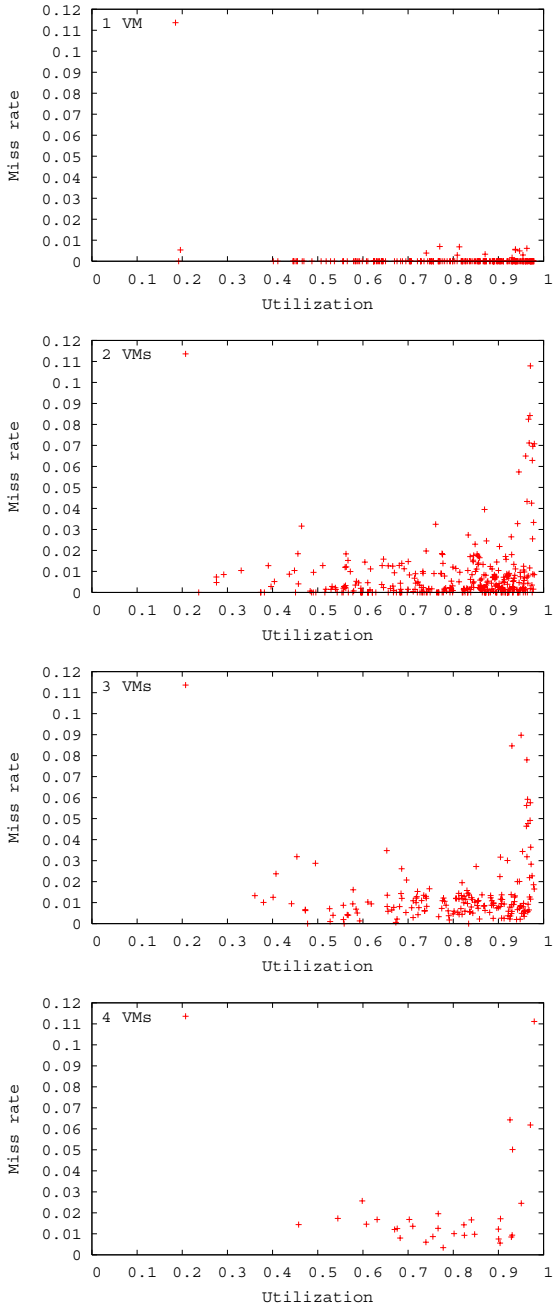


Figure 5. Miss rate as a function of utilization, Random study on Machine 1 (2 GHz P4, 2.4 kernel).

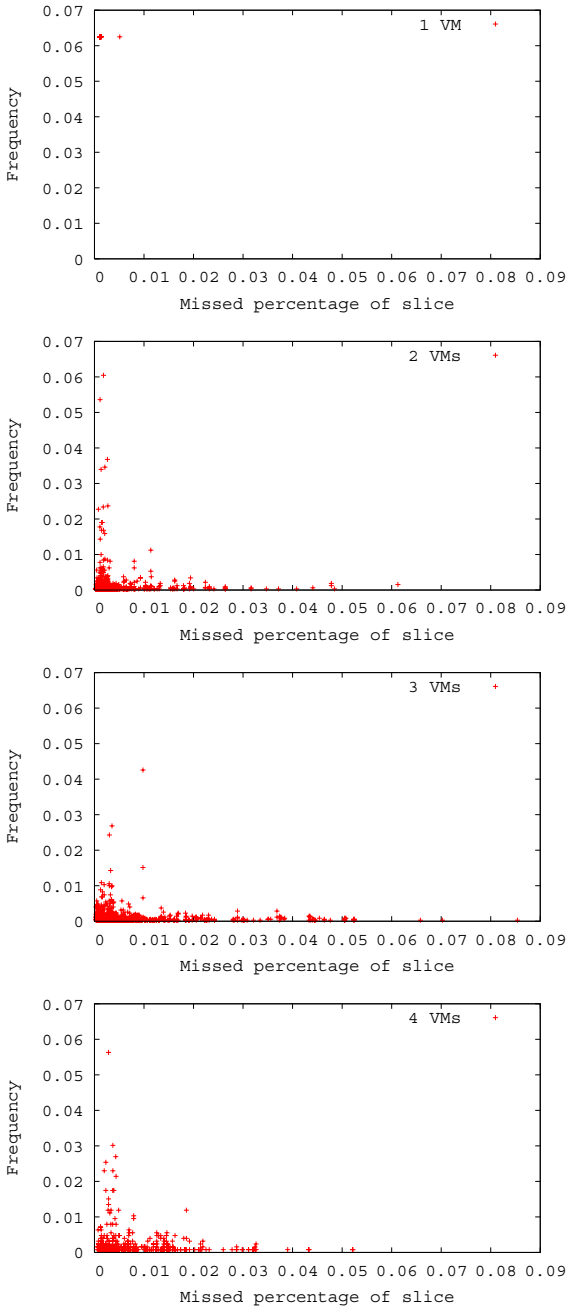
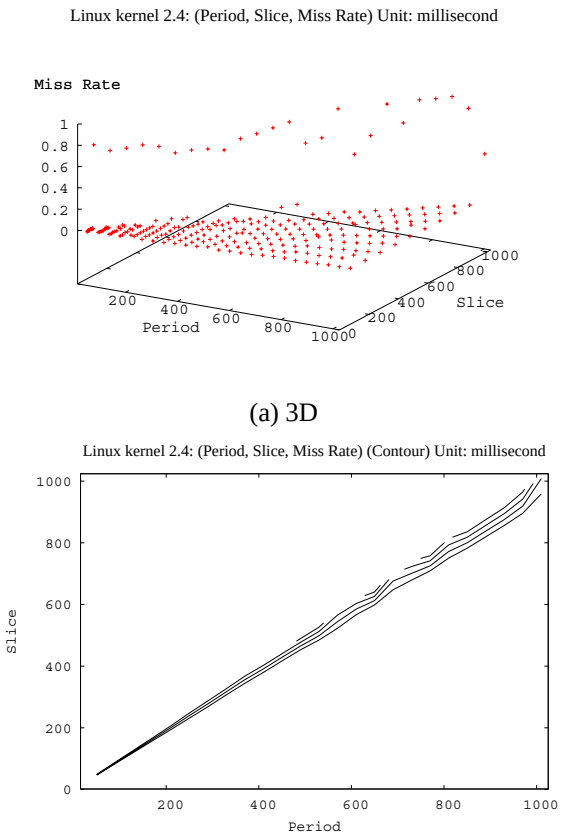


Figure 6. Distribution of missed percentage of slice, Random study on Machine 1 (2 GHz P4, 2.4 kernel).

VM. In both cases, Figure 4 shows the range of parameters.

4.2 Randomized study

Figure 5 shows the miss rates as a function of the total utilization of the VMs for one through four VMs. Each point corresponds to a single randomly generated test-



(a) 3D

(b) Contour

Figure 7. Miss rate as a function of period and slice for Machine 1 (2 GHz P4, 2.4 kernel).

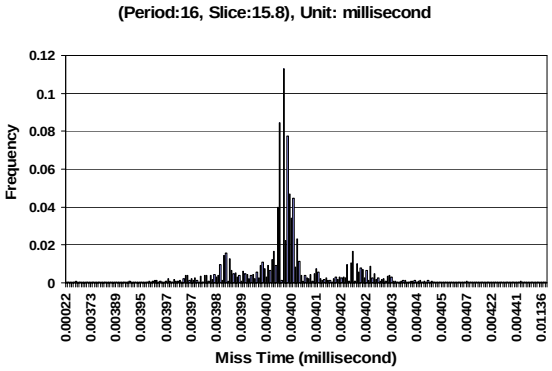


Figure 8. Distribution of miss times when utilization is exceeded for Machine 1 (2 GHz P4, 2.4 kernel).

Configuration	Maximum Utilization	Minimum Resolution
Machine 1	0.90	10 ms
Machine 2	0.75	0.2 ms
Machine 3	0.98	1 ms

Figure 9. Summary of performance limits on three platforms.

case. The miss rates are low, indendent of total utilization, and largely independent of the number of VMs after two VMs. Going from one to two VMs introduces the need for more frequent context switches.

Figure 6 shows the distribution of the ratio of miss time to slice size. All misses that do occur miss by less than 9%.

4.3 Deterministic study

In this study, we scheduled a single VM, sweeping its period and slice over the values described in Figure 4. Our goal was to determine the maximum possible utilization and resolution, and thus the safe region of operation for VSched on the different platforms.

Figure 7 shows the miss rate as a function of the period and slice for Machine 1. The top graph is a 3D representation of this function, while the bottom graph is a contour map of the function. Clearly, utilizations to within a few percent of 100% are possible with nearly 0% miss rate.

Deadline misses tend to occur in one of two situations:

- Utilization misses: The utilization needed is too high (but less than 1).
- Resolution misses: The period or slice is too small for the available timer and VSched overhead to support.

Figure 8 illustrates utilization misses on Machine 1. Here, we are requesting a period of 16 ms (feasible) and a slice of 15.8 ms (feasible). However, this utilization of 98.75% is too high for to be able to schedule it VSched would require slightly more than 1.25% of the CPU. The figure shows a histogram of the miss times. Notice that the vast majority of misses miss by less than 405 μs, less than 3% of the period.

Figure 9 summarizes the utilization and resolution limits of VSched running on our different configurations. Beyond these limits, miss rates are close to 100%, while within these limits, miss rates are close to 0%.

5 Mixing batch and interactive VMs

To see the effect of VSched on an interactive VM used by a real user, one of the authors ran an interactive VM with fine-grain interactive programs together with a batch VM and reported his observations. The test machine had the following configuration:

- Pentium 4, 2.20GHz, 512MB Mem, Linux version 2.6.3-7mdk (Mandrake Linux 10.0)
- VMware GSX Server 3.1
- VSched server running as a daemon
- Interactive Windows XP Professional VM
- Batch VM running Red Hat Linux 7.3. A process was started in the batch VM that consumed CPU cycles as fast as possible and periodically sent a UDP packet to an external machine to report on progress.

The author tried the following activities in the interactive VM:

- Listening to MP3 music using Microsoft Media Player
- Watching MPEG video clip using Microsoft Media Player
- Playing DOOM [21] and QUAKE II [22] (3D first person shooter games)
- Browsing the web using Internet Explorer, using multiple windows, Flash Player content, saving pages, and performing fine-grain view scrolling.

We set the batch VM to run 1 minute every 10 minutes (10% utilization). The user was given control of the period and slice of his interactive VM. For each activity, he tried different combinations of period and slice to determine qualitatively which were the minimum acceptable combinations. Figure 10 summarizes his observations.

The final paper will discuss these results in detail.

These qualitative results are very promising. They suggest that by using VSched we can run a mix of interactive and batch VMs together on the same machine without having them interfere. The results also indicate that there is considerable headroom for the interactive VMs. For example, we could multiplex nearly 9 Windows VMs with users comfortably playing QUAKE II in each of them on one low-end P4 computer. Given the fast reaction time of VSched to a schedule change (typically within a few milliseconds), we have high hopes that the end-users of interactive machines will be able to dynamically adjust their VM's constraints for changing needs.

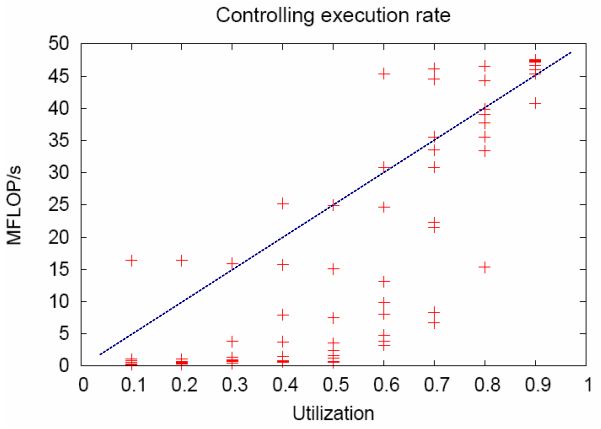


Figure 11. Compute rate as a function of utilization

The same holds true for the users of batch VMs. Indeed, the VSched abstraction provides for a continuum from fine grain interactivity to very course grain batch operation, all on the same hardware.

6 Scheduling batch parallel applications

Can we use the periodic real-time model of VSched to (a) linearly control the execution rate of a parallel application running on VMs mapped to different hosts; and (b) protect such an application from external load. Recall that parallel applications are typically run on either a space-shared machine or using gang-scheduling in order to avoid performance-destroying interactions.

To provide initial answers to these questions, we run a synthetic Bulk Synchronous Parallel (BSP [15]) benchmark, patterns, written for PVM [14]. Patterns is configured to run all-to-all communication pattern on four nodes of a cluster (2.0 GHz Xeon, 1.5 GB RAM, Gigabit Ethernet interconnect). The compute/communicate ratio is set to 0.5. We schedule the program on each of the nodes using VSched. We use the execution rate of the program in MFLOP/s as our metric.

6.1 Controlling execution rate

The goal of this experiment was to determine if, for a desired utilization, there is a $(period, slice)$ constraint that achieves the utilization while resulting in only a corresponding decrease in actual execution rate. We used periods of 20, 30, ..., 100 ms and slices of 0.1, 0.2, ..., 0.9 times the period.

Figure 11 shows the relationship between MFLOP/s and utilization ($slice/period$). As is marked on the

(period, slice)(ms)	Quake(with sound)	MP3 playback	MPEG(with sound) playback	Web Browsing
5, 1	good	good	good	good
6, 1	good	good	good	good
7, 1	good	good	ok(can't tell)	good
8, 1	ok(can't tell)	good	tiny video jitter	good
9, 1	small jitter	good	very small video jitter	good
10, 1	very small jitter	ok(can't tell)	small video jitter, sound jitter	good
15, 1	jitter	noisy	video jitter, sound jitter	good
20, 1	jitter	noisy	video jitter, sound jitter	good
30, 1	jitter	noisy	video jitter, sound jitter	jitter
20, 10	good	good	good	good
30, 10	video ok, sound jitter	noisy	video jitter, sound jitter	good
30, 15	good	noisy	video ok(can't tell), sound jitter	good
50, 10	jitter	noisy	video jitter, sound jitter	small jitter
100, 80	ok(can't tell)	good	good	good
200, 100	much jitter	very noisy	much jitter	jitter
300, 100	much jitter	very noisy	much jitter	jitter

Figure 10. Summary of qualitative observations from running various interactive applications in an Windows VM with varying period and slice. The machine is also running a batch VM simultaneously with a (10 min, 1 min) constraint.

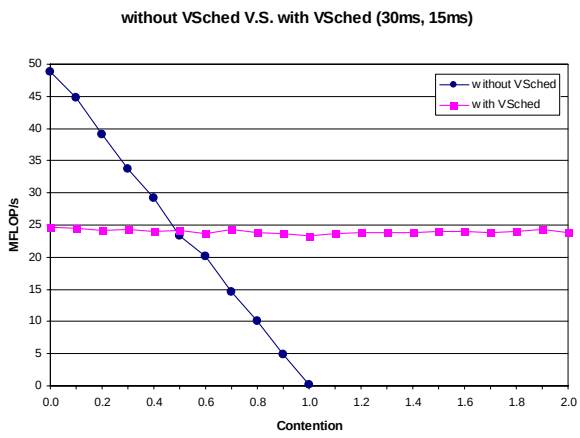


Figure 12. Compute rate as a function of contention

graph, there are choices of $(period, slice)$ that allow us to change utilization while keeping the actual program execution rate rigidly tied to it. As we decrease utilization, the duration of a compute phases increases, but the communication phase stays largely the same.

6.2 Ignoring external load

Any coupled parallel program can suffer drastically from external load on any node; the program runs at the speed of the slowest node. The periodic real-time model of VSched can shield the program from such external load, preventing the slowdown.

Here we execute patterns on four nodes with a $(period, slice)$ that results in it running at about 50% of its maximum possible rate. On one of the nodes, we apply external load, a program that contends for the CPU using load trace playback techniques [7]. Contention is defined as the average number of contention processes that are runnable. With a contention level of 1.5, if there is one other runnable process, one not scheduled with VSched, it runs at $1/(1.5+1) = 40$ % of the maximum possible rate on the system.

Figure 12 illustrates the results. With VSched, patterns executes at about 25 MFLOP/s regardless of the amount of contention introduced. On the other hand, without VSched, the node with the contending program slows as more contention is introduced, slowing down all the other nodes as well. Beyond a contention of 1.0, patterns slows to a crawl without VSched, and we do not plot those points.

We conclude that VSched can help a BSP application maintain a fixed stable performance under a specified compute rate constraint despite external load.

7 Conclusions and future work

We have motivated the use of the periodic real-time model for virtual-machine-based distributed computing; the model allows us to straightforwardly mix batch and interactive VMs and allows users to succinctly describe their performance demands. We have designed and im-

plemented a user-level scheduler for Linux that provides this model. We evaluated its performance on several different platforms and found that we can achieve very low deadline miss rates up to quite high utilizations and quite fine resolutions. Our scheduler has allowed us to mix long-running batch computations with fine grain interactive applications such as first-person-shooter games with no reduction in usability of the interactive applications. It also lets us schedule parallel applications, effectively controlling their utilization without adverse performance effects, and allowing us to shield them from external load.

We are now working on how to choose $(period, slice)$ constraints for different kinds of VMs, particularly interactive VMs in which users may have varying demands. We envision a graphical tool for such VMs that indicates to the user what his current efficiency (cycles used as opposed to cycles allocated) and cost is, and then allows him to directly manipulate period and slice. If the period and slice are not feasible on the current machine, a VM migration will be initiated.

VSched is publicly released and can be downloaded from http://virtuoso.cs.northwestern.edu.

References

[1] ANDERSON, J., AND SRINIVASAN, A. Pfair scheduling: Beyond periodic task systems. In *Proceedings of the 7th International Conference on Real-Time Computing Systems and Applications* (2000).

[2] BARUAH, S. K., COHEN, N. K., PLAXTON, C. G., AND VARVEL, D. A. Proportionate progress: a notion of fairness in resource allocation. In *Proceedings of the twenty-fifth annual ACM symposium on Theory of computing* (1993), ACM Press, pp. 345–354.

[3] BARUAH, S. K., GEHRKE, J., AND PLAXTON, C. G. Fast scheduling of periodic tasks on multiple resources. In *Proceedings of the 9th International Symposium on Parallel Processing* (1995), IEEE Computer Society, pp. 280–288.

[4] BENNETT, J., AND ZHANG, H. Worst-case fair weighted fair queueing. In *Proceedings of IEEE INFOCOM 1996* (March 1996), pp. 120–127.

[5] CHANDRA, A., ADLER, M., AND SHENOY, P. Deadline fair scheduling: Bridging the theory and practice of proportionate-fair scheduling in multiprocessor servers. In *Proceedings of the 7th IEEE Real-Time Technology and Applications Symposium (RTSA)* (June 2001).

[6] CHU, H.-H., AND NARHSTEDT, K. Cpu service classes for multimedia applications. In *Proceedings of the IEEE International Conference on Multimedia Computing and Systems* (June 1999).

[7] DINDA, P. A., AND O'HALLARON, D. R. Realistic CPU workloads through host load trace playback. In *Proc. of 5th Workshop on Languages, Compilers, and Run-time Systems for Scalable Computers (LCR2000)* (May 2000).

[8] DOZIO, L., AND MANTEGAZZA, P. Real-time distributed control systems using rtai. In *Proceedings of the Sixth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing* (2003).

[9] DRAGOVIC, B., FRASER, K., HAND, S., HARRIS, T., HO, A., PRATT, I., WARFIELD, A., BARHAM, P., AND NEUGEBAUER, R. Xen and the art of virtualization, 2003.

[10] DUDA, K. J., AND CHERITON, D. R. Borrowed-virtual-time (bvt) scheduling: supporting latency-sensitive threads in a general-purpose scheduler. In *SOSP '99: Proceedings of the seventeenth ACM symposium on Operating systems principles* (1999), ACM Press, pp. 261–276.

[11] ENSIM CORPORATION. http://www.ensim.com.

[12] FIGUEIREDO, R., DINDA, P. A., AND FORTES, J. A case for grid computing on virtual machines. In *Proceedings of the 23rd International Conference on Distributed Computing Systems (ICDCS 2003)* (May 2003).

[13] FOSTER, I., AND KESSELMAN, C., Eds. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, 1999.

[14] GEIST, A., BEGUELIN, A., DONGARRA, J., JIANG, W., MANCHECK, R., AND SUNDERAM, V. *PVM: Parallel Virtual Machine*. MIT Press, Cambridge, Massachusetts, 1994.

[15] GERBESSIOTIS, A. V., AND VALIANT, L. G. Direct bulk-synchronous parallel algorithms. *Journal of Parallel and Distributed Computing 22*, 2 (1994), 251–267.

[16] GOLDBERG, R. Survey of virtual machine research. *IEEE Computer* (June 1974), 34–45.

[17] GUPTA, A., AND DINDA, P. A. Inferring the topology and traffic load of parallel programs running in a virtual machine environment. In *Proceedings of the 10th Workshop on Job Scheduling Strategies for Parallel Processing (JSPPS 2004* (June 2004).

[18] GUPTA, A., LIN, B., AND DINDA, P. A. Measuring and understanding user comfort with resource borrowing. In *Proceedings of the 13th IEEE International Symposium on High Performance Distributed Computing (HPDC 2004)* (June 2004).

[19] HOLMAN, P., AND ANDERSON, J. Guaranteeing pfair supertasks by reweighting. In *Proceedings of the 22nd IEEE Real-time Systems Symposium* (December 2001), pp. 203–212.

[20] HOUSE, S., AND NIEHAUS, D. Kurt-linux support for synchronous fine-grain distributed computations. In *Proceedings of the Sixth IEEE Real Time Technology and Applications Symposium (RTAS 2000)* (2000).

[21] ID SOFTWARE. Doom95. http://www.pcgameworld.com/details.php/get/5001.

[22] ID SOFTWARE. Quakeii. http://www.idsoftware.com/games/quake/quake2/.

[23] INGRAM, D., AND CHILDS, S. The linux-srt integrated multimedia operating system: bringing qos to the desktop. In *Proceedings of the IEEE Real-time Technologies and Applications Symposium (RTAS)* (2001).

[24] JACKSON, D., SNELL, Q., AND CLEMENT, M. Core algorithms of the maui scheduler. In *Proceedings of the 7th Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP 2001)* (2001), pp. 87–102.

[25] JETTE, M. Performance characteristics of gang scheduling in multiprogrammed environments. In *Proceedings of the 1997 ACM/IEEE conference on Supercomputing* (1997), pp. 1–12.

[26] JONES, M., MCCULLEY, D., FORIN, A., LEACH, P., ROSU, D., AND ROBERTS, D. An overview of the rialto real-time architecture. In *Proceedings of the 7th ACM SIGOPS European Workshop* (1996).

[27] LIN, B., AND DINDA, P. User-driven scheduling of interactive virtual machines. In *Proceedings of the Fifth International Workshop on Grid Computing (Grid 2004)* (November 2004).

[28] LINUX VSERVER PROJECT. http://www.linux-vserver.org.

[29] LIU, C. L., AND LAYLAND, J. W. Scheduling algorithms for multiprogramming in a hard real-time environment. *Journal of the ACM 20*, 1 (January 1973), 46–61.

[30] LIU, J. *Real-time Systems*. Prentice Hall, 2000.

[31] MOIR, M., AND RAMAMURTHY, S. Pfair scheduling of fixed and migrating periodic tasks on multiple resources. In *IEEE Real-Time Systems Symposium* (1999), pp. 294–303.

[32] NIEH, J., AND LAM, M. The design, implementation, and evaluation of SMART: A scheduler for multimedia applications. In *Proceedings of the 16th ACM Symposium on Operating Systems Principles* (October 1997).

[33] POLZE, A., FOHLER, G., AND WERNER, M. Predictable network computing. In *Proceedings of the 17th International Conference on Distributed Computing Systems (ICDCS '97)* (May 1997), pp. 423–431.

[34] RIPEANU, M., BOWMAN, M., CHASE, J., FOSTER, I., AND MILENKOVIC, M. Globus and planetlab resource management solutions compared. In *Proceedings of the 13th IEEE Symposium on High Performance Distributed Computing (HPDC 2004)* (June 2004).

[35] SCOTT A. BRANDT, SCOTT BANACHOWSKI, C. L., AND BISSON, T. Dynamic integrated scheduling of hard real-time, soft real-time and non-real-time processes. In *Proceedings of the 24th IEEE Real-Time Systems Symposium* (December 2003).

[36] SHOYKHET, A., LANGE, J., AND DINDA, P. Virtuoso: A system for virtual machine marketplaces. Tech. Rep. NWU-CS-04-39, Department of Computer Science, Northwestern University, July 2004.

[37] SNELL, Q., CLEMENT, M., JACKSON, D., AND GREGORY, C. The performance impact of advance reservation meta-scheduling. In *Proceedings of the 6th Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP 2000)* (2000), pp. 137–153.

[38] SUNDARARAJ, A., AND DINDA, P. Towards virtual networks for virtual machine grid computing. In *Proceedings of the 3rd USENIX Virtual Machine Research And Technology Symposium (VM 2004)* (May 2004).

[39] SUNDARARAJ, A., GUPTA, A., AND DINDA, P. Dynamic topology adaptation of virtual networks of virtual machines. In *Proceedings of the Seventh Workshop on Langauges, Compilers and Run-time Support for Scalable Systems (LCR 2004)* (October 2004).

[40] VMWARE. Vmware esx server-cpu resource management. http://www.vmware.com/support/esx/doc/res_cpu_esx.html.

[41] WALDSPURGER, C. A., AND WEIHL, W. E. Lottery scheduling: Flexible proportional-share resource management. In *Proceedings of the First Symposium on Operating Systems Design and Implementation* (1994), Usenix.

[42] YODAIKEN, V., AND BARABANOV, M. A real-time linux, 1997.