VSim: Simulating Multi-Server Setups at Near Native Hardware Speed

FREDERICK RYCKBOSCH, STIJN POLFLIET and LIEVEN EECKHOUT
Ghent University, Belgium

Simulating contemporary computer systems is a challenging endeavor, especially when it comes to simulating high-end setups involving multiple servers. The simulation environment needs to run complete software stacks, including operating systems, middleware, and application software, and it needs to simulate network and disk activity next to CPU performance. In addition, it needs the ability to scale out to a large number of server nodes while attaining good accuracy and reasonable simulation speeds.

This paper presents VSim, a novel simulation methodology for multi-server systems. VSim leverages virtualization technology for simulating a target system on a host system. VSim controls CPU, network and disk performance on the host, and it gives the illusion to the software stack to run on a target system through time dilation. VSim can simulate multiple targets per host, and it employs a distributed simulation scheme across multiple hosts for simulations at scale. Our experimental results demonstrate VSim's accuracy: typical errors are below 6% for CPU, disk, and network performance. Real-life workloads involving the Lucene search engine and the Olio Web 2.0 benchmark illustrate VSim's utility and accuracy (average error of 3.2%). Our current setup can simulate up to five target servers per host, and we provide a Hadoop workload case study in which we simulate 25 servers. These simulation results are obtained at a simulation slowdown of one order of magnitude compared to native hardware execution.

Categories and Subject Descriptors: C.0 [**Computer Systems Organization**]: General—*Modeling of computer architecture*; C.4 [**Computer Systems Organization**]: Performance of Systems—*Modeling techniques*

General Terms: Design, Experimentation, Measurement, Performance

Additional Key Words and Phrases: Architectural simulation, virtualization, time dilation

ACM Reference Format:
Ryckbosch, F., Polfliet, S., and Eeckhout, L. 2012. VSim: Simulating multi-server setups at near native hardware speed. ACM Trans. Architec. Code Optim. 8, 4, Article 52 (January 2012), 20 pages.
DOI = 10.1145/2086696.2086731 http://doi.acm.org/10.1145/2086696.2086731

1. INTRODUCTION

Simulation is a vital tool in contemporary experimental research and development, both in hardware and software design. Because simulation is so widely used for different purposes, there exists a whole range of simulation techniques. Emulation or functional simulation is at one end of the spectrum and models the functional aspects of a computer system only. The simplest form of emulation is interpretation in which the simulator interprets one instruction at a time [Austin et al. 2002]. Dynamic

F. Ryckbosch is supported through a doctoral fellowship by the Research Foundation-Flanders (FWO). S. Polfliet is supported through a doctoral fellowship by the Agency for Innovation by Science and Technology (IWT). Additional support is provided by the FWO projects G.0232.06, G.0255.08, and G.0179.10, the UGent-BOF projects 01J14407 and 01Z04109, and the European Research Council under the European Community's Seventh Framework Programme (FP7/2007-2013)/ERC Grant No. 259295.
Authors' address: F. Ryckbosch, S. Polfliet, and L. Eeckhout, ELIS Department, Ghent University, Sint-Pietersnieuwstraat 41, B-9000 Gent, Belgium; email: leeckhou@elis.ugent.be.

DOI 10.1145/2086696.2086731 http://doi.acm.org/10.1145/2086696.2086731

translation can significantly speed up simulation by caching previously translated instructions [Cmelik and Keppel 1994; Witchell and Rosenblum 1996]. Emulation does not produce timing information and is most useful to determine whether a design is functionally correct. At the other end of the spectrum, cycle-accurate simulation models both the timing and the functional aspects of a computer system at a very high level of detail [Binkert et al. 2006; Emer et al. 2002; Martin et al. 2005; Wenisch et al. 2006]. Hence, accuracy is excellent but simulation speed is problematic; Industry-grade simulators typically incur a slowdown of at least 5 or 6 orders of magnitude compared to native hardware execution.

Simulating servers, multi-server setups and datacenters is particularly challenging for a number of reasons. For one, simulating user-level benchmarks, such as SPEC CPU, is unlikely to be representative for server workloads. Instead, one needs the ability to simulate a complete software stack including the operating system, (process) virtual machines, middleware, application software, etc. In addition and related to this, simulating a multi-server setup not only involves modeling CPU performance but also network and disk activity. Further, the simulation technique needs to be scalable: It requires the ability to scale out and simulate a large number of nodes, and, if the goal is to use simulation to make design decisions, it is crucial that one can simulate a future target system on a contemporary host. Moreover, a developer may not have (frequent) access to a large number of host servers to run simulations. This implies the ability to simulate large target systems on smaller host systems. Finally, simulation speed is obviously a key property. Cycle-accurate simulation is clearly not the appropriate approach for simulating multi-server systems. Instead, we need fast simulation techniques that produce meaningful performance data in a reasonable amount of time. Although we want high simulation speed, we do not want to compromise on accuracy too much. The purpose of simulation is to steer decision making, hence the simulation data does not necessarily need to be cycle-accurate, yet it should be accurate enough to make good design decisions.

This paper proposes VSim, a novel full-system simulation methodology that leverages virtualization technology to simulate multi-server setups. VSim consists of a system virtual machine that runs on a host server and controls CPU, network and disk performance as perceived by the software, i.e., the software is given the illusion to run on a target system with performance properties that differ (significantly) from the host. Virtualization also enables simulating multiple target servers per host by running target servers as guest virtual machines. Distributed simulation across multiple hosts enables simulation at scale.

The key contribution made in VSim is to enable full-system simulation (including CPU, disk and network) using time dilation: VSim filters timer interrupts delivered to the guests, making the time perceived by the guest (simulated time) a dividend of the physical time on the host (simulation time or wall clock time). At the same time, VSim schedules the guests at native hardware speed on the host. This not only enables simulating multiple target servers per host but it also enables controlling the performance perceived by the guests. For example, by scheduling a guest for a longer period of time on the host, one can model a higher performance target server, i.e., the simulated target server gets more work done per unit of (simulated) time. Next to controlling CPU performance, VSim also controls network and disk bandwidth and latency: All network and disk requests pass through VSim which controls their bandwidth and latency in simulated time.

Our prototype implementation in Oracle's VirtualBox demonstrates the potential of the technique. We provide results showing that VSim can accurately model CPU, network and disk performance. In particular, we model both high-end (AMD Opteron) and low-end (Intel Atom) target CPUs on a mid-scale host system within 2.0%; we

model 1Gbit and 100Mbit target networks within 4.9%; and we model SSD (Solid State Drive) and HDD (Hard Disk Drive) disks within 3.3% and 5.5%, respectively. When put together, VSim can simulate complete computer systems running complete software stacks with good accuracy. We demonstrate this through a client-server setup running the Lucene search engine which involves simulating a client and a server running an operating system, Java virtual machine, network protocol stack and disk I/O (average error of 3.2%). We also consider case studies involving more complex setups, including the Olio Web 2.0 benchmark running a web server, file server and database server, as well as a 25-server Hadoop workload setup. VSim achieves good accuracy while incurring a simulation slowdown of one order of magnitude only compared to native hardware execution. Moreover, VSim can simulate multiple target servers per host server (up to five target servers per host server in our current setup with an average error below 5%).

Given that VSim can simulate large systems on smaller systems with good accuracy at good speed while running complete software stacks, we believe that VSim is a promising approach towards simulating systems at scale. Moreover, VSim is complementary to other simulation paradigms, i.e., whereas cycle-accurate simulation remains an important tool in the computer designer's toolbox for making detailed microarchitecture design decisions, VSim is a more appropriate tool for making system-level design decisions in multi-server setups. Finally, both software developers and system architects can benefit from VSim. Software developers can use VSim to evaluate software optimizations and tune their software before deployment on a target system. System architects can use VSim to evaluate server design trade-offs and make system-level design decisions. Service providers can potentially use VSim to find performance bottlenecks in existing systems: they can deploy an existing software stack on a VSim simulation model and then vary both hardware and software parameters to identify performance issues.

This paper is organized as follows. Section 2 presents and describes the VSim simulation paradigm in detail. Section 3 then elaborates on the experimental setup which we use to evaluate VSim in Section 4. Finally, we describe related work (Section 5), and we conclude (Section 6).

2. VSIM

VSim models a target computer system by running a modified virtualization layer on top of a host computer system. The virtualization layer gives the illusion to the software stack to run on the target computer system. The target computer system could be an existing or a future computer system that runs a complete software stack. The simulation is done through a virtualization layer, VSim, which is run on top of a host, and the target software stack is run on top of the virtualization layer. The host system is likely to be different from the target system, e.g., it is potentially smaller in size (e.g., has fewer servers, fewer disks, etc.). VSim is a modified system virtual machine that controls the performance observed by the software stack, i.e., the software stack is given the illusion to run on the target system.

2.1. General Concept

VSim is built on the notion of the timer interrupt. In order to understand how this is done, we first briefly revisit the purpose of the timer interrupt in a computer system. The hardware periodically generates timer interrupts and the timer interrupt handler keeps track of time by counting the number of timer interrupts. System software (operating system, hypervisor, virtual machine monitor) uses the timer interrupts to drive the scheduling of processes on the hardware. In other words, the software stack

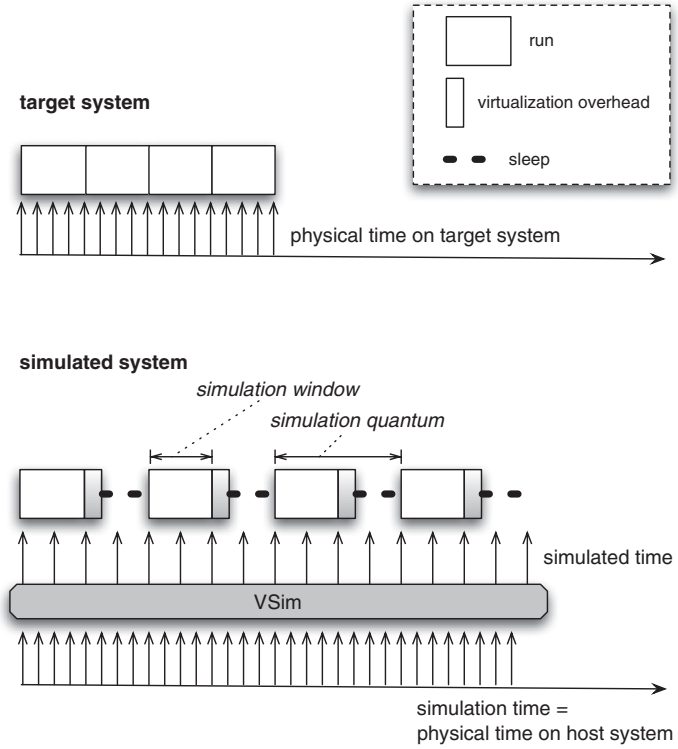

Fig. 1. Simulating one server in VSim.

observes physical time, i.e., the software stack receives and handles all timer interrupts delivered by the hardware.

The key idea of time dilation is to let VSim manipulate the timer interrupt mechanism on the host system such that the software stack is given the illusion to run on a different system, namely, the target system. VSim receives the timer interrupts and sees simulation time (i.e., physical time on the host) and does the following three actions: (i) It limits the amount of simulation time given to the software stack (i.e., it schedules the software stack for only a fraction of the simulation time); (ii) It measures the overhead due to virtualization; and (iii) It filters the timer interrupts given to the software stack. Other timing sources for the operating system in the host system, such as HPET and RDTSC, are dilated in the same way as the timer interrupts. The combination of these actions makes the software stack believe that it runs as fast as on the target system because it performs the same amount of work over the same simulated time period.

Figure 1 illustrates this for simulating a single server. While on the target system the software stack receives all the physical timer interrupts, on the simulated system, the software stack receives only a fraction of the timer interrupts. In this example, VSim filters one out of two timer interrupts and it schedules the target for a *simulation window* per *simulation quantum*. In other words, VSim schedules the target on the host during the simulation window and then puts the target to sleep until the next simulation quantum. VSim schedules the software stack on the host system such that the simulated system gets as much work done on average per unit of time as the target system: For example, the physical target system gets a unit of work done in 4 (physical)

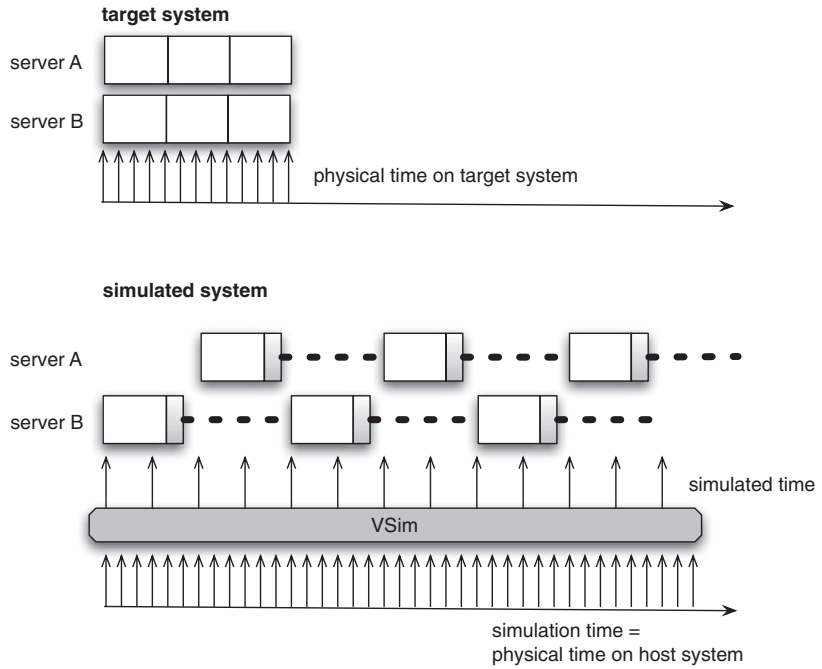

Fig. 2. Simulating multiple servers on a single host server in VSim.

time units (see Figure 1 top) and the simulated target system also gets a unit of work done in 4 (simulated) time units (see Figure 1 bottom). When the software stack is scheduled to run on the host machine it runs at the host's native speed, but the time progress given to the software stack is the simulated time, not the physical time on the host machine. The virtualization layer also measures its own overhead which it does not include when computing simulated time. The slowdown during simulation is the ratio of the simulated time divided by the simulation time, or $2\times$ in this example.

We can employ this approach for simulating multiple target servers on a single host server; Figure 2 illustrates this for two servers. Referring back to Figure 1, because of the virtualization layer overhead, we cannot run two target servers on a single host at a simulation slowdown of $2\times$ compared to native target execution, i.e., we cannot schedule units of work for both target systems within a simulation quantum. We therefore target a simulation slowdown of $3\times$ here in this example and filter one timer interrupt every three physical timer interrupts, see Figure 2, i.e., we prolong the simulation quantum.

These general concepts immediately illustrate the power of VSim. It enables simulating multiple servers running complete software stacks at high speed on a single server. By manipulating the simulated time, one can model different server configurations and study its impact on overall system performance. For example, one can model a faster CPU by filtering out more timer interrupts (i.e., the software stack is given the illusion that it gets more work done per unit of time); or one can model different network latency and bandwidth properties by controlling the latency and bandwidth of the packets as they traverse the VSim virtualization layer; or one can model different disk configurations by controlling the latency and bandwidth of disk accesses by the VSim virtualization layer. We now describe in more detail how we model CPU, network and disk performance in VSim.

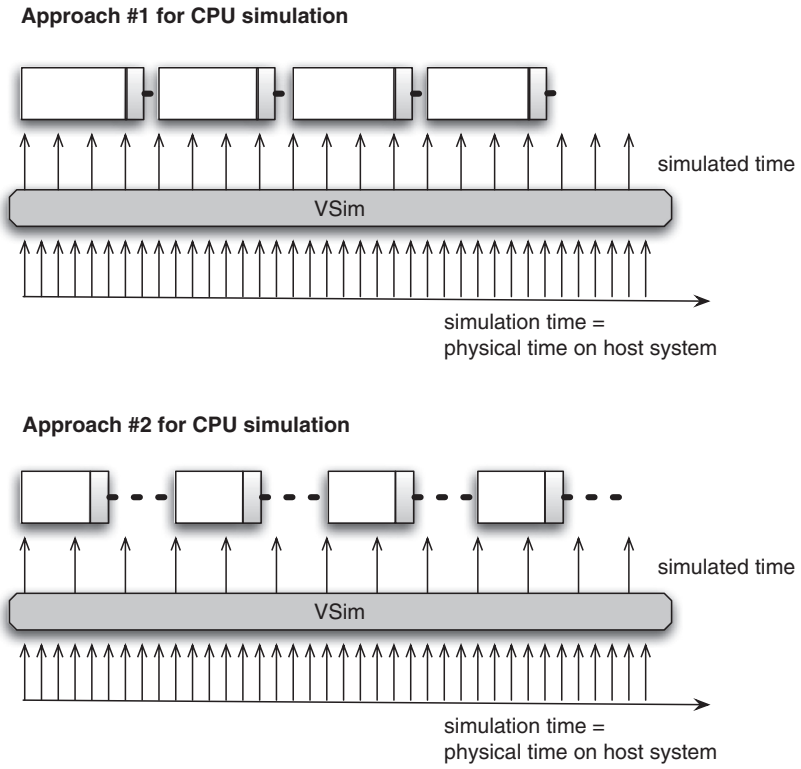

Fig. 3. Two approaches for CPU simulation.

2.2. CPU Simulation

Modeling CPU performance can be done following two approaches, see also Figure 3. The first approach keeps the timer interrupt filtering constant (e.g., VSim filters one out of two timer interrupts) and adjusts the amount of work done per simulation quantum. The second approach, which is dual to the first one, is to keep the amount of work done per simulation quantum constant and to adjust the timer interrupt filtering by VSim. In Figure 3, both approaches model a target system with a target performance of 1.5 relative to the baseline system shown in Figure 1. Approach #1 performs 6 units of work per 4 simulated time units, and approach #2 performs 4.5 units of work per 3 simulated time units.

Both approaches are in essence the same, however, we opt for approach #1 in VSim because if one is to simulate a heterogeneous system consisting of multiple servers with different relative speeds, approach #1 naturally keeps simulated time synchronized among the simulated servers. In approach #1, heterogeneity is modeled by executing different units of work per simulation quantum, i.e., VSim would execute more units of work per simulation quantum for a high-performance server as compared to a less powerful server. In approach #2 on the other hand, VSim would keep the amount of work done constant per server per simulation quantum and would filter out more timer interrupts for the high-end server than for the low-end server. As a result, time would progress faster for the high-end server than for the low-end server, and thus, time would diverge on the simulated servers, and a mechanism would need to be put in place to synchronize time among the simulated servers. Approach #1 does not need such a synchronization mechanism.

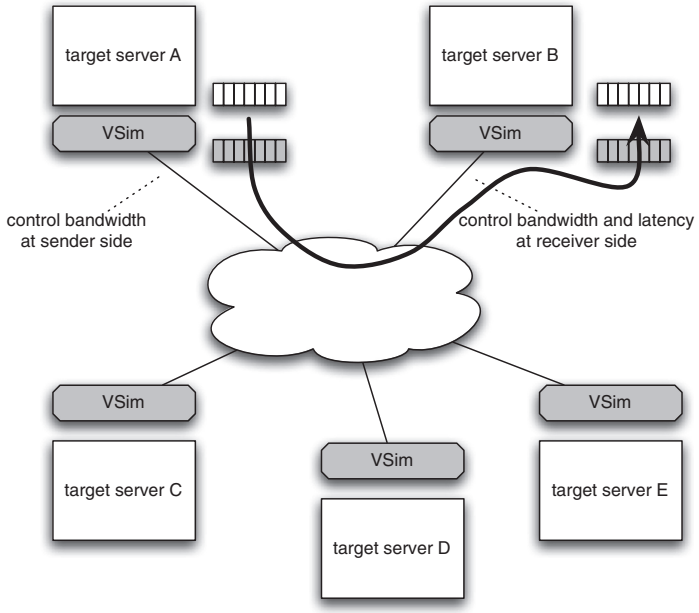

Fig. 4. Network simulation in VSim.

Our current implementation of VSim requires its users to specify the speed of the target system relative to the host system. For example, if one wants to simulate a particular workload on a target server that achieves twice the performance of the host server, VSim will execute two units of work per simulation quantum, following approach #1. The relative speed between the target system and host system is workload-dependent. For example, the relative performance difference between a high-end processor and a low-end processor is likely to be different (i.e., smaller) for a memory-intensive workload compared to a compute-intensive workload.

As part of our future work, we plan to leverage performance counters on the host and build performance models that estimate target performance based on host performance counter values. This would eliminate the need for offline profiling to determine relative host/guest performance, and it would enable accounting for an application's time-varying execution behavior.

2.3. Network Simulation

In order to be able to simulate a target network configuration on a host network, one needs the ability to control network bandwidth and latency. VSim does this by intercepting all network I/O and by manipulating bandwidth and latency. Figure 4 illustrates this. At the sender side, VSim controls the outgoing network bandwidth. This is done by keeping a buffer in VSim that holds all the outgoing network packets and sends out the packets at the bandwidth of the target network link. At the receiver side, VSim accepts all the incoming packets and holds them temporarily in a buffer. VSim delivers the packets to the target software stack at the rate determined by the target NIC bandwidth. When to deliver the packets to the target software stack is determined by the network latency. This is done through a table lookup based on the packet's sender IP address. Adjusting the sender-to-receiver network latency enables modeling different network topologies, e.g., different latencies need to be set for mesh, ring and tree networks. Other network topologies and policies can be modeled as well,

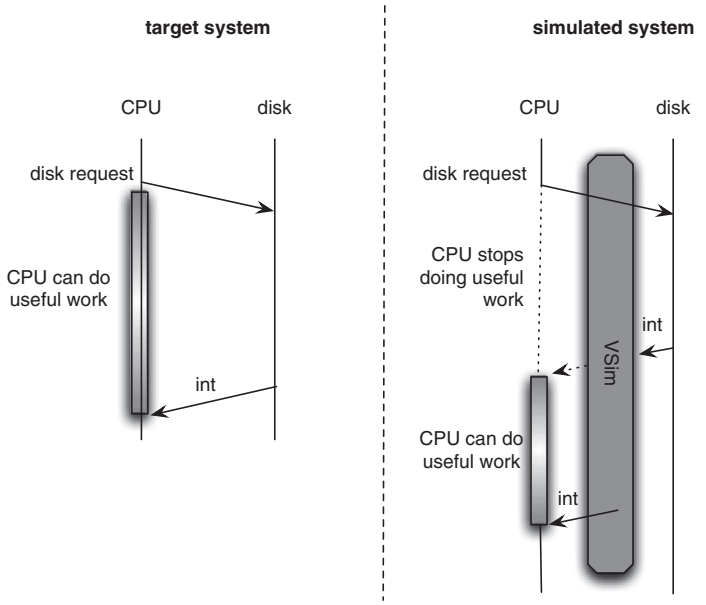

Fig. 5. Disk simulation in VSim.

including QoS-aware policies with specific bandwidth and latency properties between specific nodes in the network, and/or bandwidth and latency properties that depend on packet priority.

2.4. Disk Simulation

Figure 5 illustrates how we simulate disk I/O in VSim. On a target system, the CPU sends the disk request. A DMA transfer is then initiated to copy the data between disk and main memory, and the CPU resumes doing useful work. In the simulated system, VSim intercepts the disk request, the DMA transfer is initiated and the CPU is stopped (i.e., VSim no longer schedules units of work and simulated time is stopped for the target CPU). When the DMA transfer is completed, the disk sends a interrupt to the CPU, which is intercepted by VSim. The interrupt is held back by VSim. VSim reschedules the target CPU which can then perform useful work and advance simulated time. The amount of work done by the target CPU (also the time during which the interrupt is held back) is determined by the latency of the (simulated) disk operation. The disk operation latency is determined by whether the disk operation is a read or write, the size of the disk request (how many blocks need to be written or read), the offset on the disk, and the disk status. VSim signals to the target CPU when the disk operation has been completed, and delivers the interrupt.

The reason for stopping the CPU while sending the disk request to the disk on the host, is to be able to simulate an SSD target disk on an HDD host disk. An SSD disk features shorter latencies, hence, stopping the CPU while the host disk fetches the data and rescheduling the CPU for a period of time equal to the access latency of the target disk is the only way one can simulate a fast disk on a slow one.

Stopping the CPU upon a disk I/O request has its implications for keeping the various simulated servers synchronized, i.e., simulated time is stopped for CPUs that perform disk I/O while simulated time advances for the other CPUs. Without appropriate counteractions, this would cause simulated time to diverge across the simulated servers. VSim accounts for this by keeping track of how long a CPU is stopped while performing

the physical disk access on the host server. At the end of the simulation quantum, the CPUs that did not perform disk I/O are stopped to allow for the CPUs that did perform disk I/O to catch up.

2.5. Synchronizing Across Host Servers

So far, we were concerned with simulating multiple target servers on a single host server, and we already dealt with the issue of synchronizing simulated time across the simulated target servers in this setting. However, eventually, the goal is to simulate a multitude of target servers on a number of host servers. This brings in the difficulty of synchronizing simulated time across the various host servers. This is the classical simulated time synchronization problem in parallel discrete event simulation (PDES) [Fujimoto 1990].

There exist a number of approaches for how to deal with this synchronization problem in distributed simulation. One approach is to synchronize at each simulation quantum [Reinhardt et al. 1993], i.e., all the simulated servers (on the various hosts) need to reach the end of the simulation quantum before simulation is started for the next quantum. While this would yield high accuracy, it is likely to have an impact on simulation speed which is bounded by the slowest simulated server. At the opposite side of the spectrum, lax synchronization does not synchronize between the simulated servers and let all simulated servers run freely. Between these two opposite ends of the spectrum one can imagine different policies [Chen et al. 2009; Miller et al. 2010; Falcón et al. 2008], e.g., synchronize every n simulation quanta, or track the delta in simulated time between the various simulated servers and hold simulated servers for which this delta exceeds a specified threshold. VSim currently implements lax synchronization, but we plan to study and implement other synchronization mechanisms that balance simulation accuracy and speed as part of our future work.

2.6. Strengths

VSim has a number of strengths.

—VSim can simulate different multi-server architectures by varying CPU, network and disk performance. The architecture parameters that can be varied are the number of CPUs, CPU performance, network bandwidth and latency, and disk latency. VSim can also be used to model heterogeneous system architectures with different types of CPUs, disks, and network topologies and configurations.
—VSim can simulate entire software stacks. VSim is implemented in a system virtual machine (either a hosted virtual machine or a hypervisor) hence, it runs an unmodified operating system and a complete application stack on top of it, including virtual machines, middleware, application servers, etc.
—VSim can simulate a large target system on a small host system, i.e., VSim simulates multiple target servers on a single host server through multiplexing.
—VSim exploits multi-threaded parallelism in the host server: Simulating multiple target servers is parallelized on the host server by running each core of the target servers on the different cores and/or SMT hardware threads on the host.
—VSim is a scalable simulation approach and allows for simulating multi-server setups on a smaller number of host servers. In other words, VSim is a distributed simulation approach and time is synchronized across the various simulated servers.
—VSim runs at near native hardware speed. Simulation speed is limited by the number of target servers one wants to simulate per host server and the virtualization overhead. In our setup, the simulation slowdown of VSim is limited to one order of magnitude ($10\times$) relative to native hardware execution while being able to simulate up to 5 target machines per host machine.

Table I. CPUs Considered in the Evaluation

	core type	# cores	frequency	L1 cache	L2 cache	L3 cache
Intel Atom	in-order	2	1.6GHz	56KB private	512KB private	—
AMD Opteron 2212 HE	out-of-order	2	2GHz	128KB private	1MB private	—
AMD Opteron 2350	out-of-order	4	2GHz	128KB private	512KB private	2MB shared

2.7. Limitations

In spite of the important strengths mentioned, VSim also has some limitations.

—VSim's scalability is limited by the amount of physical memory in the simulation host. VSim needs to keep track of the memory state for all the target servers that it simulates per host server. This limitation can be overcome by adding more main memory to the host. Or, in addition, memory page sharing among the guests (the simulated servers) can lower memory pressure on the host server [Gupta et al. 2008].
—For a given workload, the performance of the guest server is modeled as a fixed factor of the host platform's performance. Determining relative guest/host performance is done offline. However, as mentioned before, one could leverage performance counters on the host to predict target performance in an online fashion, which would also enable modeling time-varying workload behavior; this is an interesting avenue for future work.
—Processor and memory subsystem performance is modeled as a single performance factor. By using hardware performance counters and an online performance model, as mentioned above, one could more accurately determine the impact of CPU and memory performance. Again, we leave this as part of future work.
—VSim does not provide enough detail for performing detailed microarchitecture simulations. Instead, VSim's key feature is to simulate and provide insight in the scaling behavior of multi-server workloads running on (potentially heterogeneous) multi-server setups.

3. EXPERIMENTAL SETUP

3.1. VSim Implementation and Configuration

We implemented a VSim prototype in VirtualBox v3.1.2. VirtualBox is a hosted system virtual machine—a hosted VM runs on top of an operating system in contrast to a hypervisor or virtual machine monitor which runs on bare metal. We set the simulation window to 10ms and the simulation quantum to 100ms in all of our experiments. We experimentally evaluated different values for the simulation window and quantum, and we found the above values to be effective. This setting yields a 10× simulation slowdown compared to native hardware execution in all of our experiments.

3.2. Hardware Platforms

We consider three CPUs: Intel Atom 330, AMD Opteron 2212, and AMD Opteron 2350. These machines have vastly different properties as shown in Table I. The Intel Atom processor is a dual-core multi-threaded in-order processor [Halfhill 2008], while the AMD Opteron processors are multicore out-of-order processors. The AMD Opteron machines both implement the K10 microarchitecture [Keltcher et al. 2007] but their multicore architecture differs: The 2212 features two cores and private 1MB L2 caches, and the 2350 features 4 cores with private 512KB L2 caches and a shared 2MB L3 cache.

We consider two disk types in our evaluation, a hard disk drive (HDD) and a solid state disk (SSD), see Table II. The HDD and SSD differ substantially in terms of their properties. We quantify the various disk parameters, such as sequential read/write

Table II. The SSD and HDD Disks Considered in the Evaluation Along with Their Properties According to IOzone

HDD	Western Digital Caviar Blue 500GB, 16MB cache, SATA 3Gb/s, 7200 rpm IOzone: seq read 29.6MB/s, random read 0.6MB/s, seq write 27.6MB/s, random write 0.3MB/s
SSD	Intel X25-M 80GB, SATA 3Gb/s IOzone: seq read 36.6MB/s, random read 16.7MB/s, seq write 30MB/s, random write 28.5MB/s

Table III. CPU Benchmarks Used in This Study

Benchmark	Suite	Description
blackscholes	PARSEC	Option pricing with Black-Scholes PDE
blastp	BioPerf	Identification of similar protein sequences in a database
bodytrack	PARSEC	Body tracking of a person
ce	BioPerf	Finds structural similarities between pairs of proteins
ferret	PARSEC	Content similarity search server
freqmine	PARSEC	Frequent itemset mining
h264dec	MediaBench II	H.264 video decoding
h264enc	MediaBench II	H.264 video encoding
raytrace	PARSEC	Real-time raytracing
specjbb2005	SPECjbb	Evaluates Java server performance.
streamcluster	PARSEC	Solves online clustering problem
swaptions	PARSEC	Pricing of a portfolio of swaptions

bandwidth and random read/write bandwidth, using IOZone which is a filesystem I/O benchmark [Norcott].

3.3. CPU Benchmarks

Table III lists the CPU benchmarks used in this study. They are taken from a variety of sources such as PARSEC [Bienia et al. 2008], BioPerf [Bader et al. 2005], MediaBench II [Lee et al. 1997], and SPECjbb2005; the PARSEC benchmarks are multi-threaded and model Recognition, Mining and Synthesis (RMS) workloads.

The target servers (as well as the simulated servers) run the Ubuntu 9.04 server operating system. The Java virtual machine involved in some of our workloads is the Sun JRE6.

4. EVALUATION

We now validate and evaluate VSim. This is done in a number of steps. We first validate VSim against real hardware, and we consider the various subcomponents in isolation: CPU, network, and disk. Subsequently, we put it together and validate VSim for simulating a client-server setup including CPU, network, and disk. Finally, we demonstrate the utility of VSim for exploring the system architecture of a multi-tier Web 2.0 server application along with a 25-server Hadoop workload setup. For all of our experiments we do 20 runs (both on hardware and within the VSim simulator), and we report 99% confidence intervals.

4.1. CPU Validation

We first focus on validating the CPU model in VSim. We consider two target servers, the Intel Atom machine and the AMD Opteron 2350 server. The host server in all of our experiments is the AMD Opteron 2212 with HDD.

4.1.1. Validation. In these experiments we first run the benchmarks on the target processors and the host processor, and we determine the performance of the target processor relative to the host processor. All benchmarks run a single thread for now; we consider multi-threaded runs later. We clustered the benchmarks into three groups according to their relative performance. There are three groups for the Intel Atom target: 60% performance relative to the AMD Opteron 2212 host (or 40% worse performance on

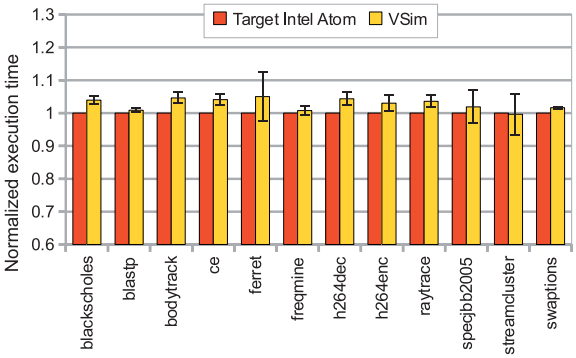

Fig. 6. Validation of the CPU model of the Intel Atom target on the AMD Opteron 2212 host.

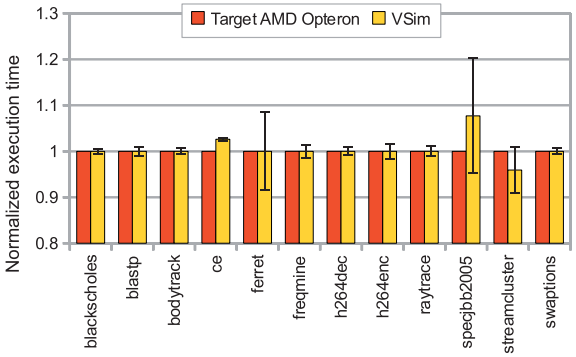

Fig. 7. Validation of the CPU model of AMD Opteron 2350 target on the AMD Opteron 2212 host.

the Intel Atom compared to the AMD Opteron 2212; this cluster includes ferret), 30% relative performance (includes blastp and swaptions), and 40% relative performance (includes all the other benchmarks). Similarly, there are three groups for the AMD Opteron 2350 target: 170% relative performance (70% better performance; includes ferret and streamcluster), 130% relative performance (includes ce and SPECjbb), and 100% (same performance; includes all the other benchmarks).[1] We refer to these clusters as the "target Intel Atom" and the "target AMD Opteron 2350," respectively, as this is the target performance VSim should model. We then run each of the benchmarks in VSim on the host server, and we set the performance target according to the above performance targets, e.g., the ferret benchmark is run at a 60% and 170% performance target when simulating the Intel Atom and the AMD Opteron 2350, respectively. We then report simulated time and compare against the target performance numbers.

The results of this validation experiment are shown in Figures 6 and 7 for the Intel Atom and AMD Opteron 2350, respectively. These graphs report normalized execution time for the two target platforms, and the simulated execution time by VSim. The ideal target execution time falls within the confidence intervals of the simulated execution times for all benchmarks, with an average error of 2.85% and 1.20% for the Intel Atom and AMD Opteron, respectively. These results demonstrate VSim's ability to accurately model CPU performance.

[1]Classifying benchmarks into three clusters introduces an average error of 7.6% and a maximum error of 26.8% for blastp on the Intel Atom, and an average error of 5.3% and a maximum error of 11.7% on the AMD Opteron 2350. These errors are a result of the clustering and not VSim.

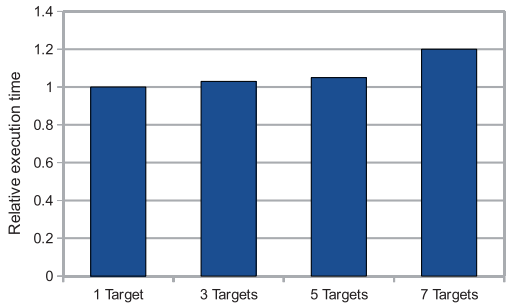

Fig. 8. Evaluating VSim scalability in terms of simulating multiple targets per host.

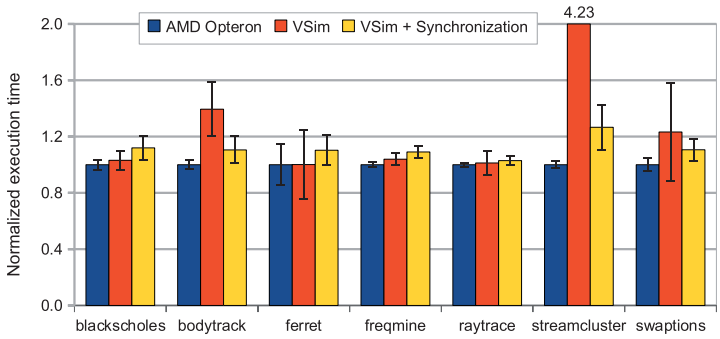

Fig. 9. CPU validation for multi-threaded workloads.

4.1.2. Scalability. The above experiments simulated a single target on a single host. We now evaluate VSim's scalability in terms of how many targets VSim can simulate on a single host with good accuracy. Figure 8 shows the average execution time across all benchmarks when simulating multiple targets normalized to simulating a single target, i.e., we run multiple copies of the same target on a single host and compute the average execution time reported by VSim across the targets; we then normalize against the execution time for a single target, and we then report the average across all of the benchmarks. The error remains small (less than 5%) for an increasing number of targets, up to 5 targets. The error increases to almost 20% for 7 targets. The reason for this increase is that VirtualBox (in which VSim is currently implemented) is a hosted virtual machine. This implies that VSim is not in control when it comes to scheduling the target server simulations; instead, the underlying operating system is. As a result, the operating system can dynamically change the order of the target simulations, and target simulations may need to wait before being rescheduled. The waiting time, however, is viewed by VSim as sleep time. Hence, the actual simulated execution time may diverge from what is perceived by VSim, and hence simulation errors are introduced. If VSim were in control of scheduling its target simulations, e.g., if VSim were implemented in a hypervisor, it is likely to expect that VSim will scale further beyond 5 targets per host. The maximum number of simulated targets is 10 given the 10ms simulation window versus 100ms simulation quantum settings and can only be achieved if VSim's virtualization overhead is very small.

4.1.3. Multi-threaded workloads. Figure 9 shows results for the multi-threaded benchmarks in our benchmark suite, with each benchmark running four threads. We model the AMD Opteron 2350 as the target on the AMD Opteron 2212 host. We consider two

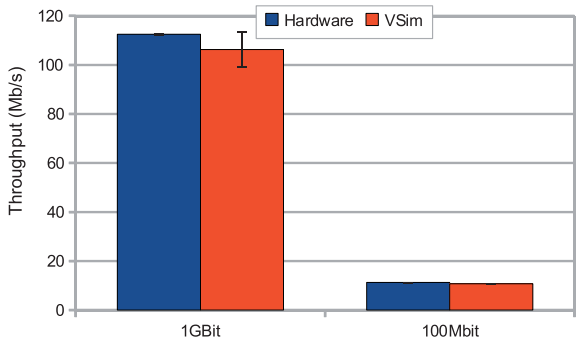

Fig. 10. Network validation.

versions of VSim here. The first version, called 'VSim' in Figure 9, does not synchronize the scheduling of the threads, i.e., this implies that threads may be scheduled in subsequent simulation quanta. The second version on the other hand, called "VSim + synchronization" in Figure 9, synchronizes thread scheduling: It co-schedules the threads on separate cores in the same simulation quantum. We observe good accuracy for all benchmarks when co-scheduling the threads. When not synchronizing thread scheduling, we observe large errors for one benchmark, streamcluster, because of its lock-intensiveness: Timing-sensitive behavior at a granularity smaller than the simulation quantum is unlikely to be accurately modeled because a lock held by a thread upon the end of a simulation window will be held until the next simulation quantum, which prevents other contending threads from making progress. This problem is overcome by co-scheduling lock-intensive threads, see the "VSim + synchronization" bars in Figure 9.

4.2. Network Validation

We now validate the network simulation approach in VSim. We consider a 1Gbit host network and model 1Gbit and 100Mbit target networks at a simulation slowdown of 10×. The workload considered here is ftp which transfers an 800MB file between two servers. We report throughput in megabytes per second and validate the simulation results against real hardware, see Figure 10. The throughput numbers reported by VSim very closely match the throughput numbers obtained on real hardware for both the 100Mbit and 1Gbit target networks: The real numbers fall within the confidence interval obtained through simulation. We obtain an average error of 4.4% and a maximum error of 15% for the 1Gbit network, and an average error of 3.6% and a maximum error of 6% for the 100Mbit network.

4.3. Disk Validation

Figures 11 and 12 validate the disk models in VSim for the HDD and SSD, respectively, using the IOzone filesystem benchmark[2]; the HDD is the host. We make a distinction between reading and writing the disk, and we consider different disk access patterns: sequential, strided and random. Sequential access means accessing subsequent 4KB blocks; strided access of 2 means accessing every other 4KB block on disk; etc. VSim's accuracy is good: the real hardware measurements always fall within the confidence intervals of the simulation. VSim is able to accurately model the disk latency difference between HDD and SSD. We report an average error of 3% and maximum error of 6% for SSD and an average error of 5% and maximum error of 10.7% for HDD. SSD

[2]http://www.iozone.org/.

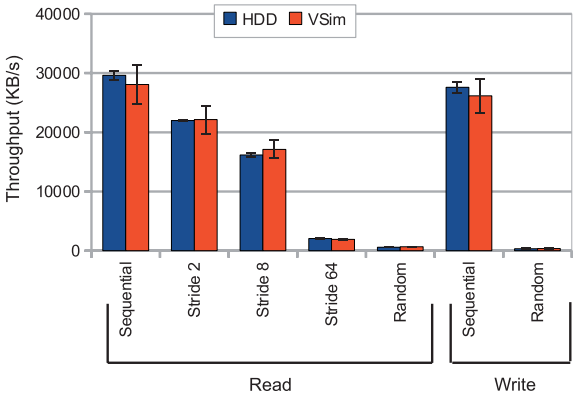

Fig. 11. Disk HDD validation.

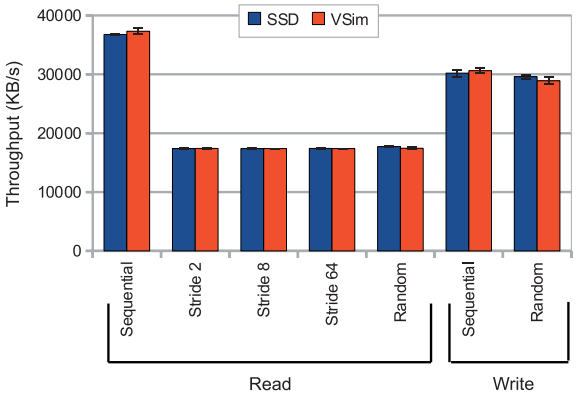

Fig. 12. Disk SSD validation.

achieves higher disk bandwidth for sequential reads, big strided reads (64 blocks), as well as random reads and writes; HDD achieves a higher throughput for 2-strided reads compared to SSD. VSim tracks these differences in disk throughput accurately compared to real hardware.

4.4. Lucene Indexing Benchmark

Now that we have validated all of the VSim subcomponents, we are ready to put everything together and evaluate VSim's accuracy while considering CPU, network, and disk. We consider a text search engine written in Java, called Lucene, and we consider two experiments. In the first experiment, Lucene builds up an index for 10K Wikipedia documents and we compare three scenarios: the Wikipedia documents are held in (i) main memory, (ii) SSD, and (iii) HDD. Figure 13 compares the simulated execution time using VSim run on the AMD Opteron 2212 (the host, with HDD)—note the entire system, including CPU, network, and disk, is simulated—against the execution time on the AMD Opteron 2350 (the target). We report an average error of 2.2% and a maximum error of 4.2%. The confidence intervals for the simulation and real hardware execution times overlap.

In the second experiment we consider a client-server setup. The client simulates a number of concurrent users sending queries to the Lucene index stored on the server. We use siege as a stress test on the client side; the clients send requests to the server

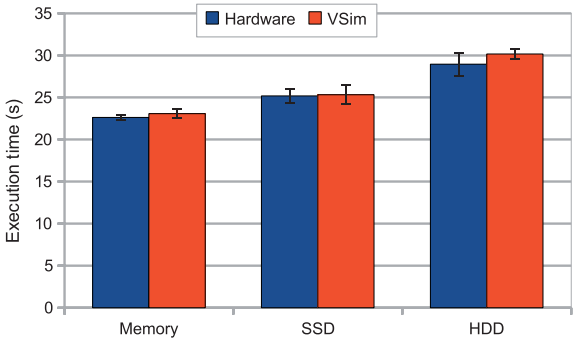

Fig. 13. Lucene index building on 10K Wikipedia documents held in memory, SSD, and HDD.

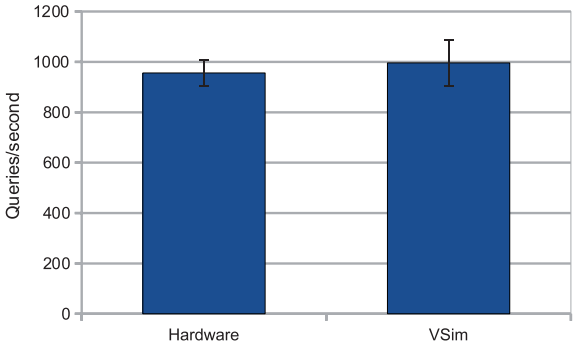

Fig. 14. A client-server setup with a client, modeling multiple concurrent users querying the Lucene index stored on the server.

with zero think time. The client is run on the AMD Opteron 2212 and the server is run on the AMD Opteron 2350. Further, the Lucene index is stored on HDD, and the client and server are connected through a 1Gbit network. VSim simulates the client and server machines on a single host machine (the AMD Opteron 2212). Figure 14 compares VSim against the hardware experiment. Again, VSim is accurate compared to real hardware as the confidence intervals overlap with an average error of 4.2% (maximum error of 14%).

4.5. Case Study #1: Olio Web 2.0

As a case study to illustrate VSim's utility for driving multi-server setup design decisions and optimizations, we consider the Olio benchmark, which is also used in Cloudstone [Avetisyan et al. 2010], a benchmark developed for evaluating Web 2.0 and cloud computing performance. Olio is a realistic Web 2.0 social-events application: a client (rain [Beitch et al. 2010], a Markov-chain based workload generator) that sends requests to the Web server which then interfaces with a file server and a database server. We assume that each of the servers (Web server, file server, and database server) runs on a separate physical server. We explore different trade-offs in performance versus cost while changing on which CPU node each server is run. The total cost of ownership (TCO), or cost for short, includes hardware purchasing cost and energy cost (both empowering and cooling the servers) assuming a 3-year depreciation. We consider three configurations, from left-to-right in Figure 15: (i) All servers run on Intel Atom 330 nodes; (ii) The Web server is run on AMD Opteron 2350, and the file and database

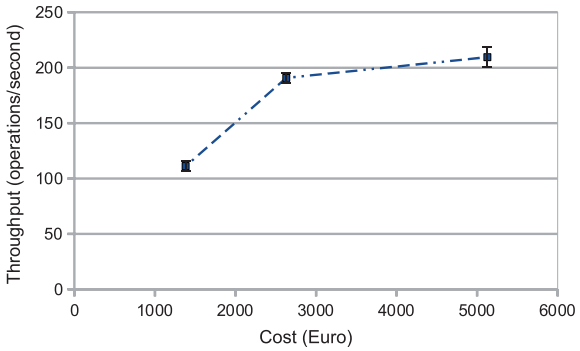

Fig. 15. Comparing three datacenter configurations for the Olio Web 2.0 benchmark in terms of performance (vertical axis) and cost (horizontal axis). Leftmost point: All servers run on Intel Atom 330 nodes; middle point: The Web server is run on AMD Opteron 2350, and the file and database servers run on Intel Atom 330; rightmost point: All servers run on AMD Opteron 2350 nodes.

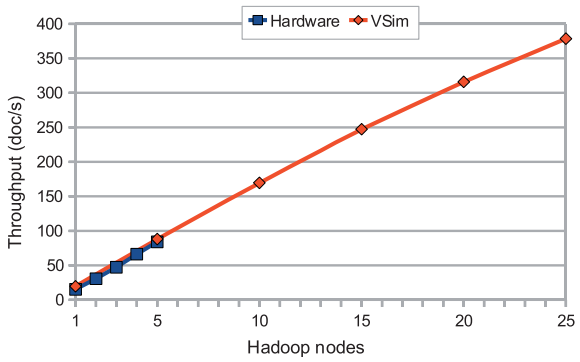

Fig. 16. Simulating up to 25 target servers running the Hadoop workload on 5 host servers.

servers are run on Intel Atom nodes; and (iii) All servers run on AMD Opteron 2350 nodes. We run the various target servers on a single host (the AMD Opteron 2212) at a 10× simulation slowdown. The interesting observation is that the heterogeneous configuration (middle configuration) (Web server run on AMD Opteron 2350, and the file and database servers run on Intel Atom 330) yields the best performance-cost trade-off for this particular workload. Its performance is nearly as good as running all three servers on AMD Opteron 2350, yet its cost is significantly lower and its performance is substantially better than running all servers on Intel Atom. The reason is that the Web server caches file and database requests using a memcached service, i.e., a memcached service runs on the Web server and handles recently accessed files and database records, and thereby reduces the load on the file and database servers. As a result, less powerful server nodes for the file and database servers achieve comparable overall system performance, yet their cost (and energy consumption) is substantially lower.

4.6. Case Study #2: 25-Server Hadoop Workload

Our second case study involves a Hadoop workload with a distributed (MapReduce-style) version of word count on 100K Wikipedia documents. We use Apache Hadoop 0.20, the OpenJDK 6 JVM, and the Ubuntu 10.04 server on the software side. Our target platform consists of 25 AMD Opteron 2212 servers with HDD interconnected

through 1Gb ethernet. We simulate all target servers on five host servers at a slowdown of one order of magnitude. Figure 16 shows the throughput as a function of the number of target servers. The graph shows two curves: the simulated curve with up to 25 target servers along with the real hardware curve up to 5 servers. VSim achieves the same throughput as the real hardware; we cannot validate beyond 5 target servers because we do not have access to more than 5 target servers. We observe almost linear scaling, although we observe a slightly sublinear scaling beyond 20 target servers. The key message is that VSim can be used for studying scale-out behavior.

5. RELATED WORK

The simulation approach most closely related to VSim probably is COTSon. COTSon [Argollo et al. 2009] is an open-source simulation framework developed by HP Labs. COTSon targets cluster-level systems consisting of multiple multicore processor nodes connected through a network, i.e., it targets both scale-up (i.e., multicore and manycore processor simulation) as well as scale-out (i.e., simulation of a multinode cluster). COTSon uses the AMD SimNow full-system simulator to functionally simulate each node in the cluster. Each COTSon node further consists of timing models for the disks, network card interface and the CPU (i.e., processor and memory). The various COTSon nodes are interconnected through a network mediator. There is at least one key difference between VSim and COTSon. VSim models target performance by moderating the amount of work done per simulation quantum for each of the simulated targets. COTSon, on the other hand, uses a performance model that is fed by a dynamic trace of instructions which is analyzed and based on which performance is estimated. These differences lead to different trade-offs in the simulation space. VSim is faster (10× slowdown versus 50× for COTSon) and consumes much less memory (20MB memory overhead for VSim versus 2GB for COTSon), hence, VSim can simulate both more targets per host and larger systems at higher speed compared to COTSon—VSim is more scalable. A limitation of VSim is that it is tied to the host server's instruction-set architecture (ISA); this is not necessarily the case for the COTSon approach which leverages functional simulation rather than virtualization.

SimOS [Rosenblum et al. 1997] was the first simulator achieving the high execution speeds required to simulate a complete system including the operating system. Binary translation was employed to achieve this high execution speed. SimOS includes CPU, disk an network simulation making it suitable for simulating networked systems. SimOS formed the foundation for the VMWare hypervisor; hardware vendors included special instructions to speed up system virtualization. VSim uses hardware assistance in combination with time dilation to achieve high simulation speeds, and it enables simulating multiple guests on a single host.

Open Cirrus [Avetisyan et al. 2010] is an open cloud-computing research testbed that was initiated by a collaborative group of researchers in both industry and academia. Open Cirrus' primary goal is to provide a distributed set of federated datacenters as a testbed for system-level cloud computing research: It provides open-source software stacks and APIs, it enables systems-level research; it provides experimental data sets and it allows for studying application development for cloud computing.

Gupta et al. [2006] propose time dilation for network simulation. Time dilation provides the illusion to an operating system and its applications that time is passing at a rate different than physical time. They implement time dilation in the Xen virtual machine by filtering delivered timer interrupts. Gupta et al. use time dilation for simulating network devices only. VSim in contrast models complete computer systems, including CPU, network and disk activity through time dilation. Moreover, VSim enables running multiple target servers per host server (this was not explored in the Gupta et al. paper) which is required for simulating scale-out scenarios.

An alternative approach to building simulators and testbeds is to build high abstraction models. For example, Ranganathan and Leech [2007] use utilization traces from real deployments in conjunction with high-level models that correlate resource utilization to power and performance. Meisner and Wenisch [2010] model datacenter workload behavior using queuing models.

6. CONCLUSION

Simulating multi-server systems is challenging. It requires the ability of running complete software stacks, while modeling CPU, disk, and network activity. In addition, it needs good simulation speed and accuracy, while being scalable. This paper presented VSim, a novel simulation paradigm which offers a unique trade-off in speed versus accuracy versus scalability while being able to model complete systems, a promising approach for simulating systems at scale. VSim leverages virtualization technology to run multiple target servers as guests on a host and manipulate CPU, disk, and network performance as observed by the guests through time dilation such that the software stacks are given the illusion of running on the target system. The implementation of VSim in VirtualBox and the evaluation presented in this paper illustrate its accuracy: 2.0%, 4.4%, and 4.9% average error for modeling CPU, disk, and network performance, respectively; complete workloads (Lucene and Olio) involving CPU, disk, and network activity are shown to be accurately modeled in VSim (average error of 3.2%). These results are obtained at a simulation slowdown of only one order of magnitude compared to native hardware speed, and our current implementation can simulate up to five target servers per host. We reported on two case studies illustrating how VSim can be used for making design trade-offs and for exploring workload scaling behavior.

ACKNOWLEDGMENTS

We thank the anonymous reviewers for their constructive and insightful feedback.

REFERENCES

ARGOLLO, E., FALCÓN, A., FARABOSCHI, P., MONCHIERO, M., AND ORTEGA, D. 2009. COTSon: Infrastructure for full system simulation. *SIGOPS Operat. Syst. Rev. 43,* 1, 52–61.

AUSTIN, T., LARSON, E., AND ERNST, D. 2002. SimpleScalar: An infrastructure for computer system modeling. *IEEE Computer 35,* 2, 59–67.

AVETISYAN, A., CAMPBELL, R., GUPTA, I., HEATH, M., KO, S., GANGER, G., KOZUCH, M., OHALLARON, D., KUNZE, M., KWAN, T., LAI, K., LYONS, M., MILOJICIC, D., LEE, H. Y., SOH, Y. C., MING, N. K., LUKE, J. Y., AND H. NAMGOONG, H. 2010. Open cirrus: A global cloud computing testbed. *IEEE Computer 43,* 4, 42–50.

BADER, D. A., LI, Y., LI, T., AND SACHDEVA, V. 2005. BioPerf: A benchmark suite to evaluate high-performance computer architecture on bioinformatics applications. In *Proceedings of the IEEE International Symposium on Workload Characterization (IISWC)*. 163–173.

BEITCH, A., LIU, B., YUNG, T., GRIFFITH, R., FOX, A., AND PATTERSON, D. A. 2010. Rain: A workload generation toolkit for cloud computing applications. Tech. rep. UCB/EECS-2010-14, Electrical Engineering and Computer Sciences, University of California at Berkeley.

BIENIA, C., KUMAR, S., SINGH, J. P., AND LI, K. 2008. The PARSEC benchmark suite: Characterization and architectural implications. In *Proceedings of the International Conference on Parallel Architectures and Compilation Techniques (PACT)*. 72–81.

BINKERT, N. L., DRESLINSKI, R. G., HSU, L. R., LIM, K. T., SAIDI, A. G., AND REINHARDT, S. K. 2006. The M5 simulator: Modeling networked systems. *IEEE Micro 26,* 4, 52–60.

CHEN, J., ANNAVARAM, M., AND DUBOIS, M. 2009. SlackSim: A platform for parallel simulation of CMPs on CMPs. *ACM SIGARCH Comput. Architect. News 37,* 2, 20–29.

CMELIK, B. AND KEPPEL, D. 1994. SHADE: A fast instruction-set simulator for execution profiling. In *Proceedings of the ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*. 128–137.

EMER, J., AHUJA, P., BORCH, E., KLAUSER, A., LUK, C.-K., MANNE, S., MUKHERJEE, S. S., PATIL, H., WALLACE, S., BINKERT, N., ESPASA, R., AND JUAN, T. 2002. Asim: A performance model framework. *IEEE Computer 35,* 2, 68–76.

FALCÓN, A., FARABOSCHI, P., AND ORTEGA, D. 2008. An adaptive synchronization technique for parallel simulation of networked clusters. In *Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. 22–31.

FUJIMOTO, R. M. 1990. Parallel discrete event simulation. *Comm. ACM 33,* 10, 30–53.

GUPTA, D., LEE, S., VRABLE, M., SAVAGE, S., SNOEREN, A. C., VARGHESE, G., VOELKER, G. M., AND VAHDAT, A. 2008. Difference engine: Harnessing memory redundancy in virtual machines. In *Proceedings of the 8th USENIX Symposium on Operating System Design and Implementation (OSDI)*. 309–322.

GUPTA, D., YOCUM, K., MCNETT, M., SNOEREN, A. C., VAHDAT, A., AND VOELKER, G. M. 2006. To infinity and beyond: Time-warped network emulation. In *Proceedings of the International Symposium on Networked Systems Design and Implementation (NSDI)*.

HALFHILL, T. R. 2008. Intel's tiny Atom. *Microprocess. Rep. 22*, 1–13.

KELTCHER, C. N., MCGRATH, K. J., AHMED, A., AND CONWAY, P. 2007. The AMD Opteron processor for multiprocessor servers. *IEEE Micro 23,* 2, 66–76.

LEE, C., POTKONJAK, M., AND MANGIONE-SMITH, W. H. 1997. MediaBench: A tool for evaluating and synthesizing multimedia and communications systems. In *Proceedings of the 30th Annual IEEE/ACM Symposium on Microarchitecture (MICRO)*. 330–335.

MARTIN, M. K., SORIN, D. J., BECKMANN, B. M., MARTY, M. R., XU, M., ALAMELDEEN, A. R., MOORE, K. E., HILL, M. D., AND WOOD, D. A. 2005. Multifacet's general execution-driven multiprocessor simulator (GEMS) toolset. *ACM SIGARCH Comput. Architect. News 33,* 4, 92–99.

MEISNER, D. AND WENISCH, T. F. 2010. Stochastic queuing simulation for data center workloads. In *Proceedings of the Workshop on Exascale Evaluation and Research Techniques (EXERT)*.

MILLER, J. E., KASTURE, H., KURIAN, G., GRUENWALD III, C., BECKMANN, N., CELIO, C., EASTEP, J., AND AGARWAL, A. 2010. Graphite: A distributed parallel simulator for multicores. In *Proceedings of the International Symposium on High Performance Computer Architecture (HPCA)*. 295–306.

NORCOTT, W. D. IOzone filesystem benchmark. http://www.iozone.org/.

RANGANATHAN, P. AND LEECH, P. 2007. Simulating complex enterprise workloads using utilization traces. In *Proceedings of the Workshop on Computer Architecture Evaluation using Commercial Workloads (CAECW)*.

REINHARDT, S. K., HILL, M. D., LARUS, J. R., LEBECK, A. R., LEWIS, J. C., AND WOOD, D. A. 1993. The wisconsin wind tunnel: Virtual prototyping of parallel computers. In *Proceedings of the ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*. 48–60.

ROSENBLUM, M., BUGNION, E., DEVINE, S., AND HERROD, S. A. 1997. Using the SimOS machine simulator to study complex computer systems. *ACM Trans. Model. Comput. Simul. 7,* 1, 78–103.

WENISCH, T. F., WUNDERLICH, R. E., FERDMAN, M., AILAMAKI, A., FALSAFI, B., AND HOE, J. C. 2006. SimFlex: Statistical sampling of computer system simulation. *IEEE Micro 26,* 4, 18–31.

WITCHELL, E. AND ROSENBLUM, M. 1996. Embra: Fast and flexible machine simulation. In *Proceedings of the ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*. 68–79.

Received July 2011; revised October 2011; accepted November 2011