Towards Accurate Simulation of Large-Scale Systems via Time Dilation

James Edmondson and Doug Schmidt
Vanderbilt University, Nashville, TN

1 Introduction

Distributed systems, particularly heterogeneous systems, have been historically hard to validate [4]. Even at small scales—and despite significant efforts at planning, modeling, and integrating new and existing systems into a functional system-of-systems—end users often experience unforeseen (and often undesirable) emergent behavior on the target infrastructure. Some types of unexpected emergent behaviors include unwanted synchronization of distributed processes, deadlock and starvation, and race conditions in large-scale integrations or deployments [25].

Deadlock and starvation are not just limited to large-scale systems and can occur when connecting just a few computers or computer systems together. Phenomenon such as the Ethernet Capture Effect [25][27] (which is a type of race condition involving a shared bus and accumulating back off timers on resending data), once occurred on networks as small as two computers, despite decades of previous protocol use and extensive modeling. If problems like this can occur during small integrations or technology upgrades, the challenges of integrating large-scale systems containing thousands of computers, processing elements, software services, and users is even more daunting. Ideally, all technological upgrades and new protocols could be tested on the actual target infrastructure at full scale and speeds, but developers and system integrators are often limited to testing on smaller-scale testbeds and hoping that the behavior observed in the testbeds translates accurately to the target system.

Consequently, what we need are technologies and methodologies that support representative "at-scale" experiments on target infrastructure or a faithful simulation, including processor time and disk simulation, as well as network simulation. These technologies and methodologies should allow application developers to incorporate their application or infrastructure software into the simulator unmodified, so they run precisely as expected on the target system. There are many network simulators available for use—some of which we discuss in this chapter—but we also explore a new simulation technology called *time dilation* [18] , which is the process of altering time sources and disk and network transactions to allow accurate simulation of multiple virtual machines on a single host. Simulations based on time dilation allow system integrators and planners to run unmodified executables, services, and processing elements to accurately emulate CPU, network, disk, and other resources for large-scale systems in much smaller testbeds. A prototype of this time dilation technology called DieCast [19] has been implemented by researchers at the University of California at San Diego. This chapter explores the benefits of this technology to date, summarizes what testers must consider when using DieCast, and describes future work needed to mature time dilation techniques and tools for large-scale distributed systems.

2 Background

In general, large-scale distributed systems are hard to validate. Though some simulators, such as USSF described in Section 2.2, can emulate networks consisting of millions of nodes, simulation technologies that scale to this level often deviate from the performance and characteristics of target large-scale systems. This section describes related work on validating distributed systems, summarizing the pros and cons of current validation techniques with respect to their ability to address the role of time dilation in large-scale system validation. We divide related background material into two main areas: formal composition and simulation techniques.

2.1 Formal Composition Techniques

Formal composition is typically associated with modeling a target system in a computer-aided manner that ensures the distributed system is validated and will work on the target system [9]. When constructing mission and safety-critical distributed real time and embedded systems, such as flight avionics computers and nuclear power plant control systems, software developers often use formal composition techniques and tools, such as Step-Wise Refinement [6], Causal Semantics [7], Behavioral Modeling [14], and Object Modeling Technique [9], to validate their software before it goes into production.

Formal composition techniques are often time consuming to validate, however, and can be tightly coupled to a particular development context and domain. Moreover, many formal composition methods require developers to model *everything* (e.g., from processors to operating systems to the application logic) to validate the target system. When composing systems of systems with formal composition techniques in this manner, it is hard to ensure a meaningful composition of heterogeneous components that interoperate correctly [20]. Progress is being made in this area of expertise – including a recent Turing Award awarded to Edmund Clarke, Allen Emerson, and Joseph Sifakis in 2007 for their work in the field [10], but formal composition of heterogeneous hardware, software, and platforms generally remains an open challenges, especially for large-scale distributed systems.

The main thrust of current development in this area of expertise is the domain-specific modeling language (DSML) [32], which require developers to tailor a visual modeling language to a specific knowledge domain, allowing business logic programmers to create an application, device driver, etc. for a specific application need (e.g. a device driver for a particular type of hardware). DSMLs can shield developers from many tedious and error-prone hardware and operating system concerns, allowing them to focus on modeling the business application logic and allow further validation by tools developed by researchers and engineers experienced in the target domain.

Though DSMLs can simplify validation of certain software/hardware artifacts (such as device drivers on an airplane) for application developers, it is much harder to make a DSML that encompasses all hardware, device drivers, operating systems, etc. for an Internet-connected application or even a small network of application processes. One issue that hinders modeling languages from being able to completely validate arbitrary application and infrastructure software is the sheer variety of orthogonal personal computer architectures and configurations that must be expressed to ensure proper validation.

Validating an application can be simplified somewhat for homogenous configurations (e.g., all personal computers are Dell T310 with a certain number and type of processors, all running a specific version and configuration of Windows, etc.), but complexities remain due to randomized algorithms used in many page replacement, queuing, etc. components of operating systems, . Threading presents additional challenges, even in homogonous hardware configurations, since composing multiple threads has no formally recognized semantic definition that fits all possible compositions [20].

When heterogeneous hardware, operating systems, and software must be supported, validating software with formal composition techniques becomes even harder. Formally composing legacy systems with closed-source implementations into a large-scale system may be helped by solutions that allow descriptions of behaviors by the legacy system, but it is still hard to ensure that an entire large-scale system is formally composed together in a semantically meaningful way. Due to these issues, we do not discuss formal composition techniques in this chapter, but instead focus on a separate vector of validation: simulation.

2.2 Simulation Techniques

Simulation is the process of reproducing the conditions of a target platform [26], and due to its flexibility, simulation has been the de facto method for validating many networked and distributed applications. A popular simulation model is discrete event simulation, where business application logic, operating system logic, etc. are treated as distinct, discrete events that are processed by a simulator engine. Though simulation has evolved quite a bit in recent decades, simulators often must make approximations that might bring testing closer to a target system but does not precisely match what is being simulated, especially when the network connectivity, processing elements, or activities performed experience failures, intermittent behavior, or scarce resources.

Simulation technologies are particularly problematic in highly connected distributed or networked applications where internet connections with high failure rates or resends are frequent. Although simulating the Internet is generally considered infeasible [28][30], many network emulators exist that attempt to emulate Internet access times, intermittency, network congestion, etc., as well as local area network testing. Examples of these simulators include Emulab and its derivatives Netlab [35] and ISIS Lab at Vanderbilt University [21], which can simulate dozens to hundreds of processing elements and their interconnections.

Emulab and its derivatives allow swapping in operating system images, applications, and test script setup to allow automatable testing. They also provide robust network emulation including bandwidth restriction, packet loss, etc. Accuracy of the simulation is left as an exercise to the developer or user and how they configure operating system images, scripts, etc. Moreover, Emulab does not explicitly support multiple virtual machines (VMs) per host, though if a user needs to scale a small testbed to a larger target system, operating system specific virtual machine managers, such as Xen [3], may be used.

Other simulators like Modelnet [34] and USSF [28] enable explicit virtualization of hosts and also include robust network emulators. Modelnet separates a testbed infrastructure into edge nodes and nodes that act as switches between the edge nodes. Vahdat et. al. [34] show that a host emulating Gigabit Ethernet in Modelnet can result in pretty accurate networked application performance simulation. The throughput difference between the emulation shown in their results, however, can differ by as much as 20%. Closer performance is possible if more than just networking is emulated, as shown in Section 4.

USSF is a simulation technology based on Time Warp and built on top of the WARPED [24] parallel discrete event simulator that claims to simulate large-scale target systems of over 100,000 nodes [29]. USSF is complicated to use and develop for, requiring the creation of topology models and then generation from application model to application code. Developers then tailor their application to the simulator, which may be unacceptable for existing code, particularly complex code that interfaces with undocumented legacy systems.

Working with USSF requires parsing a USSF model into a Topology Specification Language and then code generation via static analysis to reduce memory consumption – a major issue in WARPED libraries. The resulting code then links to WARPED libraries and the USSF kernel which interfaces to the user application. Although USSF does allow simulation of potentially millions of nodes, there is no guarantee (or even an established working estimate) that the simulation will match a real world target system because USSF development has been prioritized to operate on reduced memory per virtual machine, high cache hit ratios via file based caching with a least-recently-used policy for cache replacement, etc. and not accuracy of simulation.

3 Motivating Scenarios

To provide a solid motivation for time dilation and simulation of large-scale systems in general, this section presents several testing scenarios that require scalable and sound validation techniques. These examples focus on system environments where the computer infrastructure is mission-critical. Before we deploy large-scale mission-critical computer systems it is essential to accurately simulate such systems in a smaller, less critical infrastructure where typical functional and performance problems can be observed, analyzed, and ultimately resolved before production deployment occurs.

We first discuss a smaller situation (connecting two computers together) and specifically look into a phenomenon known as the Ethernet Capture Effect[25][27] that can manifest itself when connecting just two computers together, at least one of which is constantly sending information over an Ethernet connection. To identify the Ethernet Capture Effect during simulation requires the simulator adhering closely to actual performance on a target architecture. We next review Application Services, which are collections of interdependent systems, such as database servers, front end servers, etc. We then expand the second scenario to include collections of such services, which is our targeted large-scale scenario. We refer to these three motivating scenarios when discussing the capacities and capabilities of modeling or simulation techniques.

Ethernet Capture Effect. The Ethernet Capture Effect [25][27] is a form of emergent misbehavior that plagued system integrators during the 1990s and caused unfairness in a networking system once Ethernet hardware became fast enough to support resend frequencies that approached optimal speeds indicated in the Ethernet protocol standard. Though specific to Ethernet, similar problems during integration and scaling can occur with any other system with a shared bus. To replicate this behavior requires just two hosts, both of which send information frequently.

During the 1990s, when a collision occurred between Ethernet connected hosts, all hosts involved in the collision would randomly generate a back off time (e.g., 1-10 ms), to allow the selection of a winner to send information across the Ethernet. The winner of this contest for the shared resource would have its back off timer reset, while the loser or losers of this contest would essentially accumulate back off timers until they successfully sent information.

The problem of the Ethernet Capture Effect manifested itself when the winner of these contests had a lot of information to send and the Ethernet hardware was fast enough to allow for the winner to send its next information immediately (see Figure 1 for losing hosts increasing their backoff timers). Since its timer had been reset, this host or service would be allowed to win each contest, indefinitely, while the losing hosts would be essentially starved until the winning process or service finished. If the winning process never died, the starvation would be indefinite or last until a developer or technician reset the nodes involved.

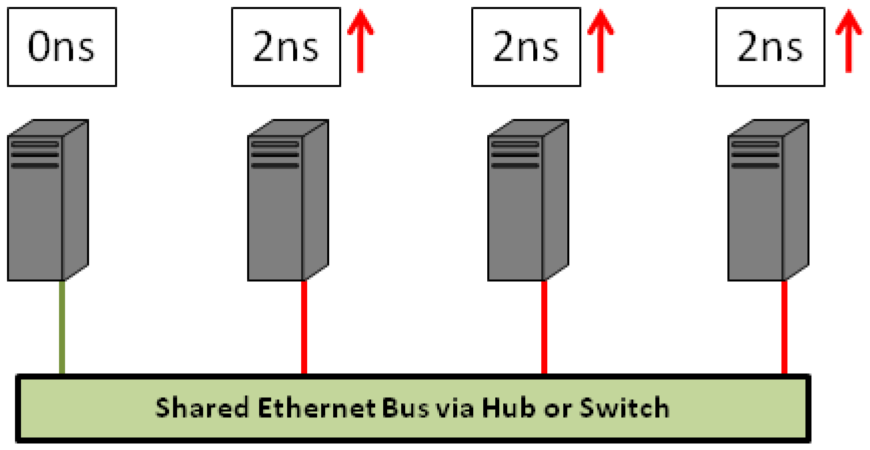

Figure 1. Example Ethernet Capture Effect in a four host system where one host is publishing constantly, and three hosts are being forced to increase their backoffs indefinitely to avoid collisions. This race condition occurred despite extensive modeling of the Ethernet protocol and decades of real world use.

Ethernet had been modeled extensively before the problem became apparent. Moreover, Ethernet devices had been in service for decades before Ethernet Capture Effect manifested itself in any type of scale, the protocol had been standardized and validated by thousands of users across the world, but small increases in hardware capabilities caused this emergent misbehavior to cripple previously functioning networks or integrations between seemingly compatible Ethernet networks of services, processes, and hosts.

The Ethernet Capture Effect represents emergent behaviors that should be caught during simulation on a testbed before deployment. If a simulator cannot catch such behaviors, major problems could manifest themselves in the production deployments.

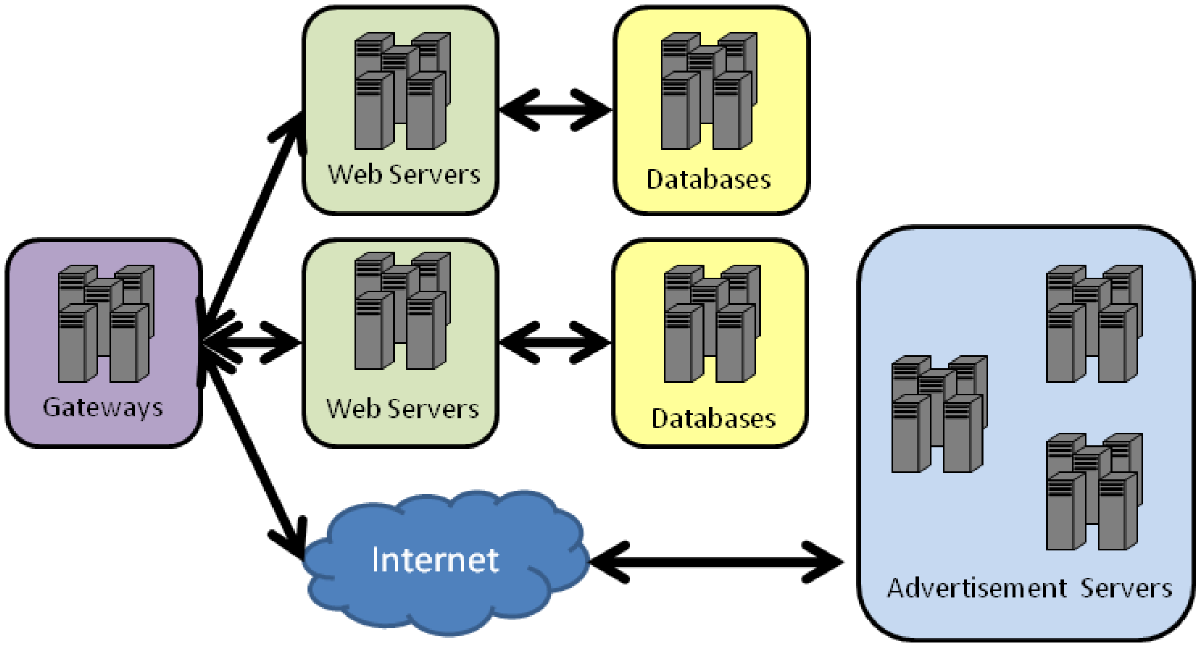

Figure 2. Example of a medium scale Application Service with application gateways communicating over local area connections to web servers and databases. Internet connections can be a large part of Application Services, as depicted by the advertisement servers interactions.

Application Services. Amazon, Google, and various other companies maintain thousands of servers providing software services for millions of end users every year. Application Services can range from a dozen to thousands of hosts (see Figure 2 for an example deployment). Though most of these service providers have proprietary networks and systems, there are other open-source auction, e-commerce, and specialty sites for general testing. Many of these proprietary and legacy systems are closed-source, meaning that formal composition methods may not be feasible for most of the utilized systems (e.g. other than a brief description of what the legacy software does, a developer may not be able to completely model much less formally compose the system without the original developer providing the completed formal model).

Consequently, Modelnet, Emulab, or similar simulators can be used to gauge and validate a target Application Service, though simulation technologies like USSF (See Section 2) may require too much modification for the simulation to be accurate enough for validation before deployment.

Large-scale systems. Large-scale systems are complex systems of systems that generally evolve over time. For the purposes of this chapter, the motivating scenario here is an integration of a dozen or more Application Services into one large-scale system that has 2,000 nodes and is comprised of heterogeneous hardware and services. Moreover, we envision a large-scale system to have subsystems linked together via a combination of Internet connections and local area networks.

Consequently, the motivating scenarios presented in this chapter cover simulation needs from a very small network of two to three nodes (e.g., Ethernet Capture Effect) to a medium sized network of a few dozen nodes (e.g., Application Services) to a large network of thousands of nodes (e.g., large-scale systems). As described below, time dilation can be used to accurately simulate these scenarios by intrinsically maintaining both accuracy and scale.

4 Applying Time Dilation with DieCast

Previous sections have examined formal composition and simulation technologies that are being used to approximate and validate large-scale networks. This section expands on these technologies to include descriptions and results of the DieCast simulator system, which is based on the time dilation principle.

4.1 Overview

Time dilation factor and scale factor. Time dilation [18][19] is the process of dividing up real time by the scale of the target system that will be emulated on a particular host. The reasoning for this partitioning is simple. Each host machine will potentially emulate numerous other hosts via virtual machines in an environment called Xen [3], preferably all requiring the same hardware, resources, and timing mechanisms found on the host machine.

A first instinct for emulating 10 hosts might be to create 10 virtual machines of the same operating system image and run them with regular system time mechanisms on the testbed hosts. An equivalent to this situation is shown in Figure 3. This approach, however, does not actually emulate the timing of the real target system because the testing system will be shared each real time second between the emulated hosts or services. This sharing can cause problems in emulation, e.g., it can affect throughput to timer firings, sleep statements, etc. Time dilation allows the system developer to adjust the passage of time to something more indicative to the actual used time allotments for each emulated host. Figure 4 below shows how time dilation affects the simulation of 9 VMs on a single host.

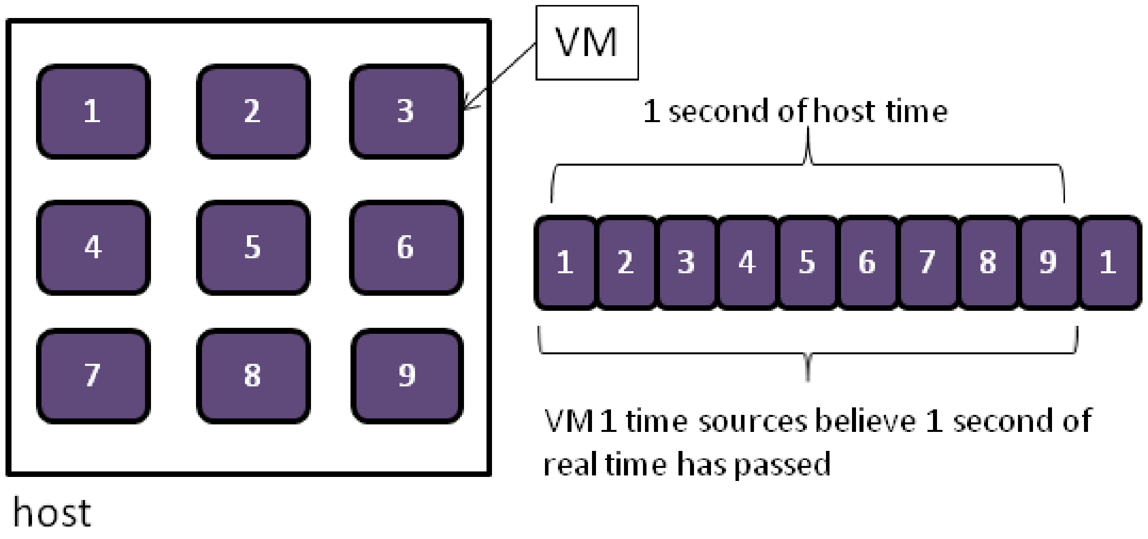

Figure 3. Most simulators do not modify time sources for virtual machines according to processor time actually used by a virtual machine. This can cause a simulation to drift from actual operation on target hardware due to queuing of timer related events.

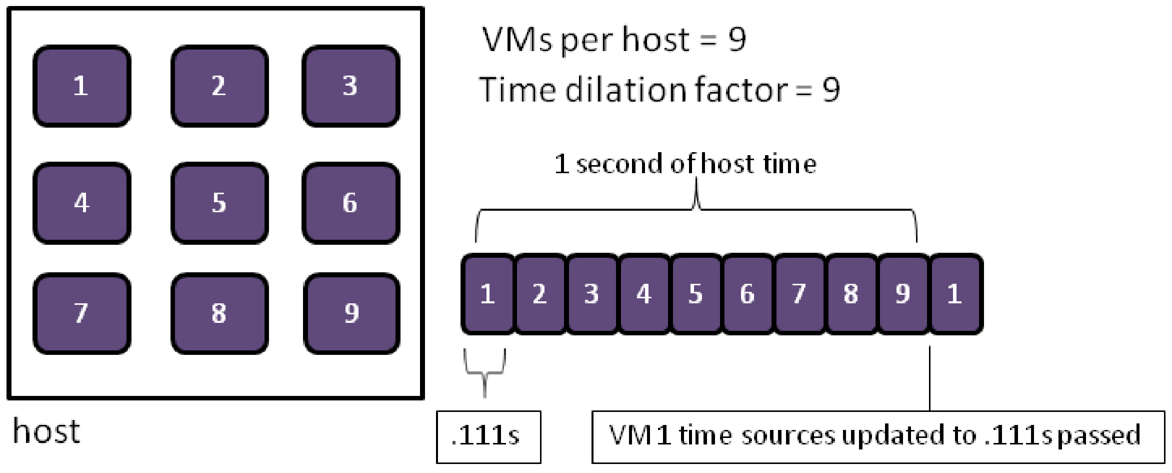

Figure 4. Basics of time dilation. When running 9 virtual machines on a single host, programmable interrupt timers, time sources, and other timer related events for each individual VM should only increment by the amount of processor time used by the VM.

Scale factor specifically refers to the number of hosts being emulated, while the time dilation factor refers to the scaling of time. Gupta et. al. [19] mention that these factors may be set to different values, but for the purpose of experimentation in the paper, time dilation factor (TDF) and scale factor (SF) are both set to the same values, thereby avoiding potential confusion.

Paravirtualized vs Fully Virtualized Virtual Machines. To accomplish Disk input/output emulation, Gupta et. al. had to deal with the intricacies of two different types of virtual machines: paravirtualized and fully virtualized. A paravirtualized virtual machine is a virtualized OS image that was limited to certain flavors of Linux and was soft emulated on the host, whereas fully virtualized requires hardware support via Intel Virtualization Technology or AMD Secure Virtual Machine, but does allow for any operating system image to be emulated directly on the hardware – rather than just a certain type of supported OS.

In their previous work on paravirtualized images [18], the DieCast authors had to create mechanisms that sat between the disk device driver and the OS, so that emulation of disk latencies, write times, etc. could be done. For the fully virtualized model of emulation, Gupta et. al. used a disk simulator called DiskSim to emulate a disk drive in memory, which gave them more control over buffering of write/read tasks and was more conducive to time dilation. DiskSim gave the DieCast developer the ability to take the number of VMs into account and slice up the read/write queuing accordingly.

The fully virtualized virtual machines implementations give DieCast users the amazing ability to plug in any operating system into the underlying Xen hypervisor and emulate a functional host according to the time slices allocated to each virtual machine via time dilation. Together with changes to the time sources, CPU scheduling, and network emulation, fully virtualized VMs give developers a lot more options and control over what they are going to be testing and amazing scalability (concerning VMs accurately simulated per host). The overestimation factor evident in any simulation without taking time dilation into account is shown in Figure 5, which is based on Figure 2c from Gupta [19]. Note that this error is a multiple of the correct target system (i.e. simulating 10 VMs per host results in inaccurate simulation by a factor of 9 on disk throughput). We also experienced this same phenomenon when simulating multiple subscribers and publishers per host in our own testing of the Quality-of-Service (QoS)-Enabled Dissemination (QED) middleware [22].

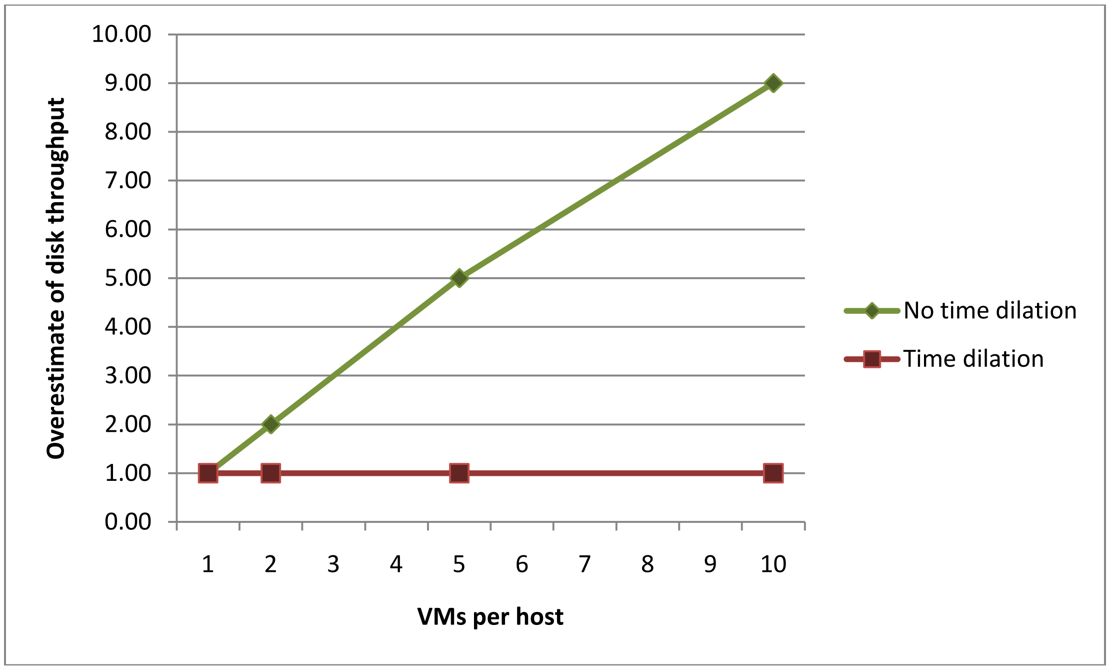

Figure 5. Analysis of the overestimation of disk throughput without time dilation versus using CPU and disk time dilation scaling.

CPU Scheduling. To implement properly scaled CPU scheduling, Gupta et. al. had to intercept and scale a number of time sources appropriately. Timer interrupts (e.g. the Programmable Interrupt Timer), specialized counters (e.g. TSC on Intel platforms), and external time sources (e.g. network time protocol) were all intercepted and scaled before being handed to VMs and their applications.

Timing is more intricate, however, than just allotting 1/(time dilation factor) time to each VM. For IO bound VMs, they may not use their full CPU allocation and this could skew the non-IO bound VM CPU usage upwards and affect all timing dependent aspects of the emulation. Consequently, the Gupta et. al. devised a credit-based CPU scheduler in Xen to support a debit/credit scheme, where IO bound VMs could relinquish CPU, but no VM used more than its exact allocated share of a real time unit. If a VM did not use its share before blocking, it received a debit to put towards its next time slice, and the CPU scheduler took into account total usage over a second to make sure that non-IO bound VMs were not monopolizing the CPU and skewing results.

Network Emulation. Network emulation in the DieCast implementation is accomplished via capturing all network traffic and routing it through a network emulator. Though Gupta et. al. mention that DieCast has been tested with Dummynet, Modelnet, and Netem, all experiments presented in their paper used Modelnet as the network emulator. Since time dilation scales time down, emulating network throughput and latency turns out to be a relatively straight forward task (compared to CPU scheduling). To emulate a 1 Gbps network to a scaled host with time dilation factor of 10, the emulator simply ships 100 Mpbs (1/10 that total) to the host. Latencies are easy to mimic also, since each VM is slowed down to 1/10 speed, and consequently, a system requiring 100 microsecond latency on the target system could be emulated with data arriving every 1 ms.

Time dilation is a powerful tool for network emulation. Gupta et. al.'s first paper on time dilation [18] shows how time dilation can be used to emulate network throughput and speeds that are larger than the network capacity available over a link. As a result, not only can a tester using DieCast simulate ultra high network capacities internal to nodes (e.g. when all VMs are on the same host and the network link is completely emulated), but testers may also scale the number of VMs per host while simultaneously scaling the network capacity between hosts, if required. The key to this ability, once again, is effectively slowing down each VM according to a time dilation factor. If all VMs are operating at that time scale, then the network can be emulated at a factor equal to the time dilation factor.

4.2 Application to Motivating Scenarios

Prior work has focused on time dilation in two scenarios: scaling network characteristics [18] and validating large-scale applications [19]. In the first paper covering time dilation in networked applications, Gupta et. al. [18] scale network bandwidth and response time when needed via time dilation in both CPU and network emulation. This work showed many results on how time dilation brought TCP throughput and response times between a simulated and baseline system into harmony in tests between two machines. These results show that time dilation can accurately simulate traffic characteristics (network capacity, response time, and traffic flows down to the TCP layer). As a first step in reflecting our motivating scenarios, therefore, time dilation can accurately simulate scaling the number of virtual machines (VMs) and reflecting a target system when the target system is small – down to 1 to 2 machines if needed.

On the surface, this would appear to signal that the DieCast implementation of time dilation would be able to mimic the Ethernet Capture Effect, presented in Section 3, between two or more Ethernet capable hosts. Ethernet is at a lower OSI layer than TCP, however, and it is unlikely that a race condition like the Ethernet Capture Effect, which is essentially a hardware-related phenomenon that requires ultra fast publication to manifest, will occur when a simulator is slowing down publication rates by splitting processor time between multiple VMs. Consequently, the Ethernet Capture Effect and phenomenon like it demonstrate a small kink in the armor of time dilation based systems and motivates the need to still do real world testing whenever possible to further validate distributed systems.

In another paper on time dilation Gupta et. al. [19] performed testing in two different types of scenarios: (1) a baseline that is run without time dilation on the actual hardware – 40 machines on their testbed and (2) a scaled version running on much less hardware – usually 4 machines scaled to 40 machines via a time dilations factor of 10. Several types of distributed systems were tested in the papers on time dilation, but we will be focusing on specific tests that reflect our motivating scenarios: RUBiS (an academic tool that resembles an auction site, along with automatable item auctioning, comments, etc.), Isaac (a home grown internet service architecture that features load balancers, front end servers, database servers, clients and application servers) and PanFS (a commercial file system used in clustering environments).

Gupta et. al. experimented with the DieCast system with time dilation factors of 10 (meaning that 10 virtual machines were running per host) and showed the effects of turning off

CPU scheduling, Disk IO, etc. to see how the system diverged from the actual baseline when time dilation was not applied in each of the time dilation mechanisms. When CPU and Disk IO time dilation were turned off, the graphs diverge drastically for all experiments – often deviating by factors of two or more in perceived network throughput, disk IO, etc. These particular results show us the problems with only using network emulation to validate target systems – namely that these testbed systems may misrepresent the target systems by not just small error ratios but factors of error. This result is unfortunately the case when using Modelnet, Emulab, or other network emulation frameworks while simulating multiple virtual machines per node with ns, VMWare, Xen, or other systems without time dilation. This differential in network throughput, disk throughput, or latency could mean developers missing emergent behavior on the testbed that will occur on the actual target hardware.

The RUBiS completion times for file sharing and auctions on scaled systems closely mirrored the actual baselines for time dilation, while the non-TDF enforced versions of Xen did not. The testing data for RUBiS in particular is fairly abundant with Gupta et. al. showing results for CPU usage, memory profiling, throughput, and response times closely mirroring the baseline target system. As the number of user sessions increased to 5,000 or more, the deviation by non-time dilation scaled systems in response time was 5 to 7 times more for non-time dilated than it was on the time dilated system and the target system. To make matters worse, the average requests per second were less than half of the performance on the target system when not using time dilation with just 10 virtual machines per host.

The more complicated Isaac scenario, consisting of load balancers and multiple tiered layers of hosts and services, not only appropriately mimicked request completion time, but the system also resembled the base system on time spent at each tier or stage in the transaction, and matched closely during IO bound transactions and CPU bound ones. Moreover, the DieCast developers caused failures to occur in the database servers in the Isaac scenario at specific times and compared the baseline performance to the time dilated and non-time dilated simulations. Without time dilation, the simulated experiments did not follow the baseline experiments. In addition, when requests through the Isaac system were made more CPU intensive (generating lots of SHA-1 hashes) or more database intensive (each request caused 100 times larger database access), the time dilation simulation was within 10% deviation of the baseline at all times – while non-dilated ended up requiring 3x more time than the baseline to complete the CPU stress tests.

The final tests on the commercial PanFS system showed similar aggregate file throughput to the baseline and has also allowed Panasas, the company that makes PanFS, to more accurately test their file system according to target client infrastructures before deployment, which they had not previously been able to do due to their clients having much larger clustering infrastructures than the company had available for testing. While RUBiS and Isaac represent classic application services scenarios, the PanFS results were especially interesting because PanFS is regularly used by clients with thousands of nodes (i.e. large scale systems). Gupta et. al. were able to test the system on 1,000 hosts against time dilation on just 100 hosts and closely mirrored the real world performance and validated that the PanFS servers were actually reaching their theoretical throughput bottlenecks. The PanFS results show us that time dilation may be ready to make the leap to larger testing configurations.

4.3 Future work

Though DieCast provides an accurate approximation of a large-scale distributed system, it may mask timing related race conditions (like the Ethernet Capture Effect and TCP Nagle algorithm problems noted in Mogul's paper [25]) due to DieCast slowing the entire system down and potentially missing boundary conditions that could have happened, for instance, in between real target timer events (which were instead queued up and fired in quick succession). One of our motivating scenarios (The Ethernet Capture Effect) may not have been caught by systems using time dilation because by slowing down an application's Ethernet traffic resend rate (by interrupting its resends to run other VMs), we may have allowed the backoff accumulator to reset. The lesson here is that, despite close approximation to the actual target system, DieCast and time dilation may not help catch all types of race conditions and unexpected emergent behavior.

Gupta et. al. admit that DieCast may not be able to reproduce certain race conditions or timing errors in the original services [19]. The system also has no way to scale memory usage or disk storage, and this can be a large limiting factor when a testbed host system is unable to emulate the time dilation factor of the target system (e.g. a 100 hosts on the testbed with 4GB of RAM trying to emulate 1000 hosts each requiring 1GB of dedicated RAM a piece).

Potential vectors of interest might include augmenting the DiskSim to allow virtualization of memory on disk (possibly by further scaling down of time) to allow for the increased latency of disk drives emulating RAM memory access times. DieCast may also be a good vehicle to implement emergent "signatures" detection algorithms [25] into the testing phases and development cycles of large scale system development.

5 Addressing Issues of Time Dilation in Physical Memory

Section 4 discussed how DieCast can be used to validate large-scale network topologies for production applications. DieCast has built-in support for scaling disk input/output, networking characteristics, and time sources, but it has no mechanisms for scaling memory. This section, therefore, discusses options available to developers when memory scaling is necessary, including how to determine memory requirements to support scaled experiments and custom modifications that may be made to DieCast or virtual machine managers like Xen to enable scaling memory.

5.1 Overview

One aspect of validation that the DieCast implementation of time dilation does not solve is the situation where a host cannot emulate the physical memory required by the user-specified number of VMs per host. Physical memory is a scarce resource, and if it runs out during emulation of the target infrastructure, virtual machine managers like Xen will begin to emulate physical memory using virtual memory on the host. Virtual memory is typically set aside from a hard drive, which has orders of magnitude worse fetch time than physical memory.

While emulating in virtual memory will not stop the VMs from functioning (unless even virtual memory is exhausted), it may result in major timing differences between the testbed system and the target system. If the time dilation factor were set to 10 (e.g., for 10 VMs hosted per host) and we only had enough physical memory to mimic target system performance for 5 of those VMs properly before virtual memory was used where the target system wouldn't have done so, we would likely miss race conditions, deadlock, and other types of emergent misbehavior during testing.

When using time dilation solutions, users should have options to address this issue. We evaluate potential solutions in this section.

5.2 Solutions

More memory or more hosts. Adding more physical memory may be possible, especially with increased adoption of 64 bit architectures and modern operating systems's ability to support more than 4GB of physical memory. This solution is attractive, and though processor speeds appear to have plateaued recently, availability of larger, inexpensive physical memory continues to increase. Still, users of time dilation systems or any other simulation system need to make sure that the amount of available memory exceeds the memory profiles required by all individual VMs. We discuss a reliable method for doing so here.

One option is to profile a single VM's physical memory usage using Linux's top utility, Window's Task Manager, or any other type of monitoring tool. To properly conduct such a memory profile, the VM must not only be up and running when profiling the memory, but also performing in the same type of role that it will be used in the target system (e.g. serving as a database system during memory profiling). Developers would then need to multiply this maximum physical memory used during the memory profiling session by the number of VMs that the host will be running and add an additional physical memory overhead required by the Xen VM Manager in the case of DieCast, or whatever technology is managing the VMs, and the actual host operating system.

Once these memory overheads are calculated, developers should be able to arrive at the required physical memory for host systems. If implementers or validation testers are unsure of the amount of overhead required by host operating system and the VM manager, it may be best to multiply the amount required by a certain percentage, and remember: it is much better to have more physical memory than required than not enough, when trying to get an accurate simulation of a target system with time dilation or any simulation system.

Adding more hosts may also be a feasible solution to this scenario if developers can afford to add more hosts to the testbed system. Gupta et. al. recommend a time dilation factor of 10 [19], and although there was not much reasoning or testing presented in the paper to support this time dilation factor, persons using DieCast may be best served by following this advice and keeping the host to VM ratio at 10 or less (i.e. 10 VMs per host at a maximum).

The authors believe that these two solutions (adding more memory or adding more hosts) are probably feasible for the vast majority of validation requirements. The next proposed solution tries to cover the other portion of testbed emulation of a target infrastructure.

Memory emulation. This solution requires the most augmentation to a time dilation system like DieCast and is the most likely to not exactly mimic the target system. This solution, however, may be the only option available when the amount of physical memory required for the testbed system to match the target infrastructure is so large that obtaining enough physical memory is infeasible.

As an example, consider a situation where a testbed system is composed of 10 hosts, and a target infrastructure has 1,000 nodes. If we were to equally distribute the 1,000 VMs required over the 10 hosts, we would require each host to emulate 100 VMs, requiring at least a time dilation factor of 100 to accurately mimic operation of the target system. Assuming that each VM requires a physical memory profile of 4 GB to accurately reflect operation of a target system a total of 400 GB of physical memory must be installed on each host, before taking into account the memory required by the host operating system and VM manager.

Assuming an overhead of 20% of the VM requirement for the latter (400 GB x .2 = 80 GB for a total of 480 GB required per host), if our hosts actually have just 4 GB of installed memory, this situation will result in a simulation that does not accurately reflect timing of target systems, due to virtual memory being much slower than the physical memory used on the target system. A potential solution to this situation is to completely emulate the instruction set for all VMs on the host and run most of the VMs on virtual memory with a time dilation factor that reflects usage of virtual memory instead of physical memory. This solution will result in a significant increase in the amount of time an experiment will be required to run, which is a real problem with most simulators and takes away much of the reason to use DieCast or a custom time dilation implementation.

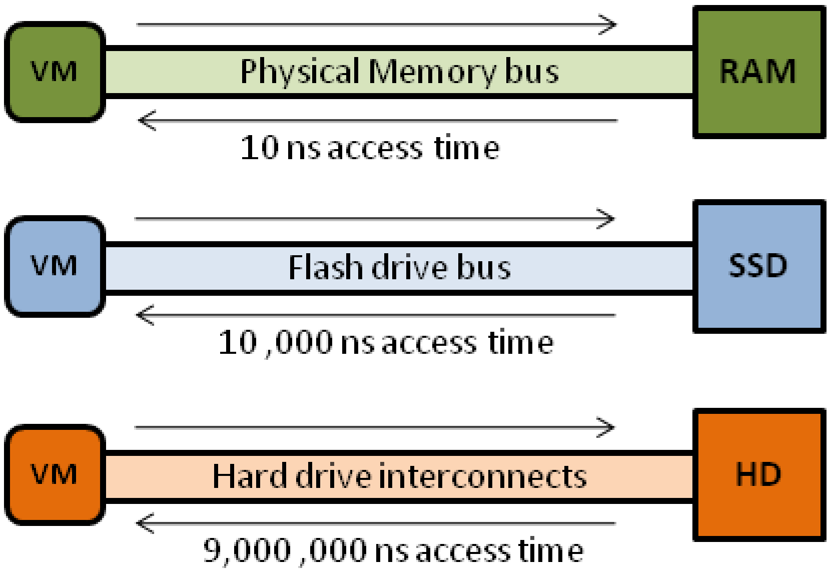

Figure 6. Memory access time comparison between physical memory (RAM), SSD flash, and traditional hard drives. All numbers are approximations to provide scale.

Figure 6 shows the difference between accessing physical memory and a hard drive for your memory needs. The difference between access time in physical memory and hard drive data is typical six orders of magnitude. Consequently, emulating all virtual machines in virtual memory and adjusting the time dilation accordingly to the access time difference could lead to a time dilation simulation taking over 1 million times longer with emulation on hard disks and over 10 thousand time longer with emulation on a flash memory type drive (shown as SSD for Solid-State Drive in Figure 6). SSD flash cards or hard drives are currently able to supplement system memory with over 64 GB of flash memory. Obtaining the 480 GB of additional memory for the ten thousand times longer run time system could potentially be possible via USB hubs or similar technologies.

6 Concluding Remarks

Time dilation is a versatile emerging technology that helps developers and testers ease the validation of medium- to large-scale systems. Prior work on DieCast has provided developers with CPU scheduling, network emulation, and disk emulation that is informed by time dilation mechanisms, resulting in extremely accurate test runs on reduced hardware. DieCast does not require developers to remodel their business application logic, piggyback unrelated libraries, generate simulation glue code, or do many things required by other simulation frameworks and technologies typically used for large to even ultra large scale target architectures. Instead, it allows for developers to compile their projects and code as they normally would do, harness the power of the Xen hypervisor – a stable, well supported virtual machine manager for Linux, and run multiple VMs per machine in a way that more accurately reflects true performance of a target system.

Though time dilation shows promise, it does not identify all types of problems that affect mission-critical distributed systems. This chapter explained how Ethernet problems, such as the Ethernet Capture Effect, might be masked by slowing down the host and not witnessing some of the race conditions that could appear in target systems due to queuing of timer firings and other related issues. Time dilation also currently has a higher memory footprint than some network simulators (if that is all that developers require) that are able to reduce memory requirements via shared data structures, operating system emulation, etc. While these network simulators might be useful to determine if the target system works if provided with certain operating systems or configurations and modeling of environments, implementations of time dilation like DieCast also allow developers to test their business logic for distributed or networked applications on real operating systems with excellent approximations of time source progression, disk queuing, and network throughput and latency.

8 References

1. Aguilera, M. K., J. C. Mogul, J. L. Wiener, P. Reynolds, and A. Muthitacharoen. "Performance Debugging for Distributed Systems of Black Boxes." Proceedings of the 19th ACM Symposium on Operating System Principles, 2003.
2. Armando, A., et al. "The AVISPA Tool for the Automated Validation of Internet Security Protocols and Applications ." Lecture Notes in Computer Science, 2005: 281-285.
3. Barham, P., et al. "Xen and the Art of Virtualization." Proceedings of the 19th ACM Symposium on Operating System Principles, 2003.
4. Basu, A., M. Bozga, and J. Sifakis. "Modeling Heterogeneous Real-time Components in BIP." Fourth IEEE International Conference on Software Engineering and Formal Methods. 2006. 3-12.
5. Batory, Don, and Bart J. Geraci. "Composition Validation and Subjectivity in GenVoca Generators." IEEE Transactions on Software Engineering. 1997.
6. Batory, Don, Jacob Neal Sarvela, and Axel Rauschmayer. "Scaling Step-Wise Refinement." IEEE Transactions on Software Engineering, 2004.
7. Bliudze, S., and J. Sifakis. "Causal semantics for the algebra of connectors." Formal Methods for Components and Objects, 2008: 179-199.

8. Buyya, R., and M. Murshed. "GridSim: a toolkit for the modeling and simulation of distributed resource management and scheduling for Grid computing." Concurrency and Computation: Practice and Experience, 2003: 1175-1220.
9. Cheng, B.H.C., L.A. Campbell, and E.Y. Wang. "Enabling Automated Analysis through the Formalization of Object-Oriented Modeling Diagrams." International Conference on Dependable Systems and Networks. 2000. 305-314.
10. Clarke, E., A. Emerson, and J. Sifakis. "Model Checking: Algorithmic Verification and Debugging." Communications of the ACM, 2009: 74-84.
11. Coppit, D., and K.J. Sullivan. "Formal Specification in Collaborative Design of Critical Software Tools." Third IEEE International High-Assurance Systems Engineering Symposium. 1998. 13-21.
12. Coppit, D., and K.J. Sullivan. "Sound methods and effective tools for engineering modeling and analysis." Conference on Software Engineering. 2003.
13. Edmondson, J., and B. Parker. "Application of Simulation in Computer Architecture." International Conference on Education and Information Systems, Technologies and Applications. Orlando, FL, 2006.
14. Engels, G., J.M. Küster, R. Heckel, and L. Groenewegen. "A methodology for specifying and analyzing consistency of object-oriented behavioral models." ACM SIGSOFT Software Engineering Notes 26, no. 5 (2001): 186-195.
15. Feiler, P., et al. Ultra-Large-Scale Systems Study Report. Pittsburgh: Carnegie Mellon University, 2006.
16. Fujimoto, R.M., K. Perumalla, A. Park, H. Wu, M.H. Ammar, and G.F. Riley. "Large-Scale Network Simulation: How Big? How Fast?" 11th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems. Orlando, Fl., 2003. 116-123.
17. Gössler, G., S. Graf, M. Majster-Cederbaum, M. Martens, and J. Sifakis. "An Approach to Modelling and Verification of Component Based Systems ." SOFSEM 2007: Theory and Practice of Computer Science, 2007: 295-308.
18. Gupta, D., K. Yokum, M. McNett, A. C. Snoeren, G. M. Voelker, and A. Vahdat. "To Infinity and Beyond: Time-Warped Network Emulation." Proceedings of the 3rd USENIX Symposium on Networked Systems Design and Implementation. 2006.
19. Gupta, Diwaker, Kashi V Vishwanath, and Amin Vahdat. "DieCast: Testing Distributed Systems with an Accurate Scale Model." 5th USENIX Symposium on Networked System Design and Implementation. San Francisco, CA, 2008. 407-422.
20. Henzinger, T. and Sifakis, J. "The Embedded Systems Design Challenge." Lecture Notes in Computer Science. 1-15. 2006.
21. Hill, J., J. Edmondson, A. Gokhale, and D.C. Schmidt. "Tools for Continuously Evaluating Distributed System Qualities." IEEE Software, 2010: 65-71.
22. Loyall, J., et al. "QoS Enabled Dissemination of Managed Information Objects in a Publish-Subscribe-Query Information Broker." SPIE Defense Transformation and Net-Centric Systems conference. Orlando, 2009.
23. Loyall, J.P., et al. "Dynamic Policy-Driven Quality of Service in Service-Oriented Systems." 13th International Symposium on Object/Component/Service-oriented Real-time Distributed Computing. Carmona, Spain, 2010.

24. Martin, D.E., McBrayer, T.J., and Wilsey, P.A. "WARPED: A Time Warp simulation kernel for analysis and application development." 29th Hawaii Internetional Conference on System Sciences. 1996.
25. Mogul, J.C. "Emergent (Mis)behavior vs. Complex Software Systems." 1st ACM SIGOPS/EuroSys European Conference on Computer Systems. Leuven, Belgium, 2006. 293 - 304.
26. Oxford English Dictionary. Oxford University Press, 2010.
27. Ramakrishnan, K. R., and H Yang. "The Ethernet Capture." Proc. IEEE 19th Local Computer Networks Conference. Minneapolis, MN, October 1994.
28. Rao, D.M., and P.A. Wilsey. "An Ultra-Large-Scale Simulation Framework." Journal of Parallel and Distributed Computing 62, no. 11 (2002).
29. Rao, D.M., and P.A. Wilsey. "Simulation of Ultra-large Communication Networks." Seventh IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems. 1999. 112-119.
30. Riley, G.F., and Ammar, M.H. "Simulating Large Networks – How Big is Big Enough?" Grand Challenges in Modeling and Simulation. 2002.
31. RUBiS. http://rubis.objectweb.org.
32. Schmidt, D. "Model-Driven Engineering." IEEE Computer Society, 2006: 25-32.
33. Srivastava, A., K. Hacker, K. Lewis, and T.W. Simpson. "A method for using legacy data for metamodel-based design of large-scale systems." Structural and Multidisciplinary Optimization 28, no. 2-3 (2004): 146-155.
34. Vahdat, A., et al. "Scalability and Accuracy in a Large-Scale Network Emulator." Proceedings of the 5th Symposium on Operating Systems Design and Implementation. 2002. 271-284.
35. White, B., Lepreau, J., Stoller, L., Ricci, R., Guruprasad, S., Newbold, M., Hibler, M., Barb, C., and Joglekar, A. "An Integrated Experimental Environment for Distributed Systems and Networks." 5th Symposium on Operating Systems Design and Implementation (OSDI). 2002.
36. Xiong, M., J. Parsons, J. Edmondson, H. Nguyen, and D.C. Schmidt. "Evaluating Technologies for Tactical Information Management in Net-Centric Systems." efense Transformation and Net-Centric Systems conference. Orlando, FL, 2007.
37. Yaun, G.R., D. Bauer, H. L. Bhutada, C. D. Carothers, M. Yuksel, and S. Kalyanaraman. "Large-scale network simulation techniques: examples of TCP and OSPF models." ACM SIGCOMM Computer Communication Review 33, no. 3 (2003): 27 - 41.