VM-based Slack Emulation of Large-scale Systems

Patrick G. Bridges and Dorian Arnold*
Department of Computer Science
University of New Mexico
Albuquerque, NM 87131
{bridges,darnold}@cs.unm.edu

Kevin Pedretti †
Scalable System Software Department
Sandia National Laboratories
Albuquerque, NM 87123
ktpedre@sandia.gov

ABSTRACT

This paper describes the design of a system to enable large-scale testing of new software stacks and prospective high-end computing architectures. The proposed architecture combines system virtualization, time dilation, architectural simulation, and slack simulation to provide scalable emulation of hypothetical systems. We also describe virtualization-based full-system measurement and monitoring tools to aid in using the proposed system for co-design of high-performance computing system software and architectural features for future systems. Finally, we provide a description of the implementation strategy and status of the proposed system.

1. INTRODUCTION

Developing hardware, system software and applications for next-generation supercomputing systems requires fast large-scale testbeds. Such testbeds allow developers to study the impact of both architectural and software changes on overall application performance. Given recent emphasis on hardware/software co-design methodologies, which rely on continuous evaluation of the impact of hardware, system software and application changes, these testbeds have become particularly important.

Our position is that virtualization-based system emulation is an effective approach for accelerating the deployment and performance of testbeds for simulating novel, highly-concurrent architectures. Furthermore, we propose leveraging recent work on slack simulation [3] in distributed virtual machine emulation; the allows distributed virtual machine monitor emulations to be only loosely synchronized, further accelerating testbed performance.

In this paper, we describe the design of an emulation-based framework that we are researching to provide a testbed for HPC system and application architects. We focus on time-to-result for large scale simulation runs of real applications instead of complete accuracy. This approach is in contrast to past work on cycle accurate node simulations [1] or high-fidelity cluster-level simulations that rely on skeleton applications or mini-apps to complete simulation runs in reasonable amounts of time [9]. We do assume that such simulations are also used as part of the development process.

Our approach focuses on the use of a virtual machine monitor (VMM) to emulate individual nodes of the target system, with the software stack under evaluation running as a *guest software stack* in a virtual machine (VM). This allows much of the guest stack to run natively, with the VMM intercepting hardware calls that require additional handling for architectural simulation and controlling the passage of time in the guest stack. We use standard time dilation techniques [6, 5] to allow a single node to emulate additional processors and nodes. Unlike past work in this area, we use slack simulation to enable the virtual machine to emulate the behavior of low-latency I/O devices such as network interface controllers (NICs). In this approach, distributed virtual machine emulations are coarsely synchronized to a global time source instead of finely synchronized on an operation-by-operation basis. The level of synchronization will be used to control emulation accuracy when evaluating network-intensive HPC workloads.

VM-based slack emulation potentially sacrifices flexibility in what systems it can evaluate. In particular, emulation works best when the system being emulated is relatively similar to the system on which the emulation is running. For example, larger system scales, systems with additional or faster processors, and new I/O devices can all potentially be emulated efficiently. Dramatically different processor or memory architectures, however, will not be feasible to emulate because they would require continual intervention by the VMM, substantially increasing time to solution.

In the remainder of this paper, we first describe several motivating examples that are driving our work in this direction. We then describe the overall architecture of the system we are building, along with selected architectural details from specific portions of the system. Finally, we describe our implementation strategy and status, discuss related work on virtualization-based system emulation, and conclude.

*This work was supported in part by the DOE Office of Science, Advanced Scientific Computing research, under award number DE-SC0005050, program manager Sonia Sachs, and by a faculty sabbatical appointment from Sandia National Labs.

†Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy under contract DE-AC04-94AL85000.

ROSS '11, May 31, 2011, Tucson, Arizona, USA

2. EXAMPLES USES

2.1 Performance-Heterogeneous Processors

Many-core systems that include processors with heterogeneous performance characteristics, particularly different clock speeds on different processors, comprise the first kind of system we seek to emulate. Such systems present interesting challenges to both HPC application and system software design, particularly for examining issues related to application load balancing, node-level resource allocation, and inter-processor communication performance. Enabling development and evaluation of new application, runtime, and system software techniques for these systems is a key motivating factor in the work described in this paper.

2.2 Global Non-coherent Addressing

Globally addressable memory in distributed-memory systems is often proposed for deployment in future HPC systems, particularly latency-oriented systems designed to handle large, irregular data sets. Integrating low-latency remote DMA network devices directly with the hardware memory addressing system could dramatically simplify the system programming model by providing low-latency access to remote memory. It would also avoid the the performance penalties of previous software-based distributed shared memory systems. Because virtualization software can easily intercept virtual and physical memory accesses, global non-coherent addressing is an ideal use-case for virtualization-based emulation of future large-scale HPC systems.

2.3 Active Messaging Network Interfaces

Finally, active messages are an increasingly important mechanism for low-latency communication in future systems, with recent research demonstrating their usefulness in implementing high-performance distributed graph algorithms [13]. New network interface cards are being designed to handle active messages, for example with new control structures or the ability to offload message handlers to the NIC. Evaluation of the performance of these systems on meaningful algorithms and data sets at scale is imperative to understand the potential benefits and challenges they present. Because no real-world implementations of such cards exist, however, they are another hardware enhancement with impact across the breadth of the software stack that motivates the research described in this paper.

3. ARCHITECTURE

3.1 Overview

Figure 1 shows the general architecture of the system. This system is based on a virtual machine monitor (VMM) that intercepts relevant hardware calls from the application and system software being evaluated. The VMM handles these calls to emulate the hardware on which this application/system software is being evaluated, and also provides monitoring and control functionality.

The virtual machine monitor is the central element in this system. Its primary responsibility is to interact with the guest software stack to provide the illusion that the guest is running on the hardware being emulated. To do this, it intercepts guest software stack hardware accesses when necessary through standard virtualization techniques and performs the following tasks:

- Emulate specified processor/system performance by controlling the real and apparent flow of time in the virtual machine;
- Invoke external architectural simulation tools to perform detailed simulation of external devices;
- Synchronize the local and remote processor and node clocks to adjust for varying times due to architectural simulation costs;
- Provide performance information about the emulated machine to external monitoring tools

We describe each of these tasks in detail in the remainder of this section.

3.2 VMM-based Emulation

In addition to intercepting guest software stack calls and coordinating activity between various system components, the VMM is responsible for general processor emulation functionality. In particular, we will use VMM-based emulation to achieve the following:

1. Multiple nodes using a single node
2. Increased numbers of processors on a node
3. Increased and decreased processor speeds on a node

In each of these cases, well-known past work on virtual machine time dilation techniques [6, 5] will form the initial basis for our work. Time dilation runs each guest node in a virtual machine that runs at a *fixed* fraction of real time by scheduling the virtual machine less frequently and delivering timer interrupts more or less frequently that real time.

Time dilation is important in our approach because it allows a core or node to emulate more than one node, and provides a virtual global clock source that synchronizes the activities of all nodes in the emulated system. For example, time dilation by a factor of four allows a single node to emulate four hardware nodes. It also guarantees that time is elapsing at the same rate on all nodes in the system so that causality is preserved in communications between nodes.

3.3 External Architectural Simulation

An external architectural simulator will be used to simulate key I/O devices such as the active messaging NICs described in Section 2.3. We focus on full device simulation of such interfaces as opposed to simpler performance emulation because of the device's potential impact on system software performance and the application programming model. In addition, the use of an existing architectural simulator allows existing device models to be leveraged, and provides a straightforward path for implementing new models.

Multiple simulated architectural devices will be tied together into a distributed network simulation using an approach similar to that we used in our previous work [9]. In particular, we will use Lamport clock-style message timestamps to propagate message transmission and reception times. The global synchronization provided by a time dilation approach, subject to the complications described in the following subsection, will substitute for the periodic global barrier synchronization used in that approach.

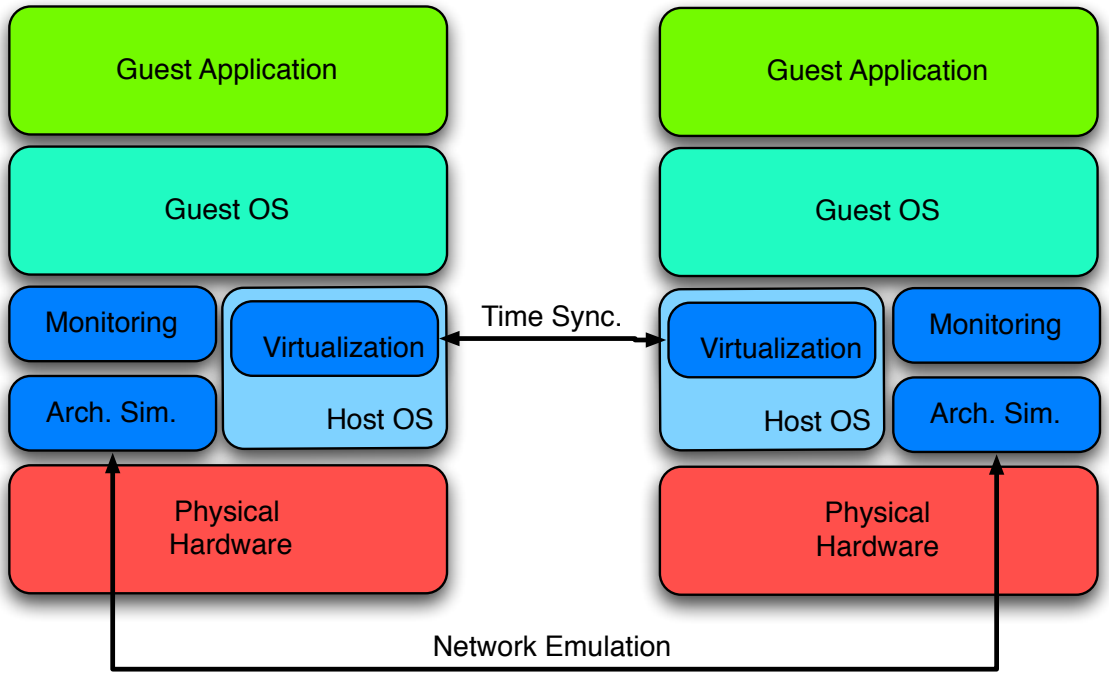

Figure 1: High-level Architecture of Large-scale VM-based Testbed

3.4 Distributed Slack Simulation

The simulation capabilities described above necessitate occasionally pausing simulation of a guest core or node for a relatively substantial length of time. As a result, time in different cores and nodes may occasionally advance at different rates, unlike in traditional time dilation systems. If these differences are not accounted for, simulation accuracy can suffer. For example, if time on node A is progressing at a significantly slower rate than on node B and node A sends a message to node B, the time at which node B receives the message from node A may be incorrect by a large amount. Traditional approaches to addressing this problem, for example optimistic parallel discrete event simulation [4], have runtime costs that are potentially expensive and are complex to implement in a virtual machine monitor setting.

We plan to use dilated time simply as a *target* rate at which guest time should advance instead of a fixed rate at which time must advance to address this issue. Doing so will keep independent cores approximately synchronized while still allowing guests to deviate from dilated time when necessary for simulation purposes. When such deviation happens, the virtual machine will need to gradually control guest time back towards the target time. This is approach trades away some accuracy for improved simulation time, and is a form of slack simulation [3], a recent approach to speeding up parallel simulation systems.

To control guest time that has deviated from the target time back towards the target this, the virtual machine will have to slowly advance time more quickly than usual in the guest, for example by injecting timer interrupts slightly more quickly. The VM must, however, bound how much it adjusts time forward; when the guest is only occasionally monitoring low accuracy timers (e.g. the PIT timer), the VM will be able to advance time more quickly in the guest towards target time. When the guest is quickly polling high-resolution timers like the timestamp counter, however, the VM cannot radically adjust guest time without potentially breaking guest software stack behavior.

3.5 Dynamic Time Dilation

We also plan to explore dynamically adjusting the time dilation factor across nodes, because correctly setting time dilation factor is vital for trading off emulation accuracy and time-to-result. For example, if guest simulated time deviates by large amounts or diverges from dilated time, emulation accuracy can suffer, and dilating time further (trading off time-to-result) can be used to improve emulation accuracy. Similarly, if the emulation spends large amounts of time with no VMs to dilate time appropriately, reducing time dilation can improve simulation speed without sacrificing accuracy.

To deal with global effects of changing the time dilation factor, we are exploring gossip-based approaches that include periodic global agreement, similar to our past work on load balancing [14]. By including time information in transmitted messages, something that is already necessary for accurate network emulation (see Section 3.3), individual nodes will be able slowly change their time dilation factor and stay approximately in sync with the remainder of the simulation. Larger changes in time dilation factor that are more likely to lead to de-synchronization of nodes, will still require some form of global agreement.

The goal of this distributed management of time dilation between nodes is to allow nodes making heavy use of simulation features to be more heavily dilated than those that do

not. In addition, allowing the time dilation factor to vary over the course of the run can potentially reduce the time required to complete a full emulation run by allowing emulation to run with less time dilation when less simulation is required.

3.6 VM-based Monitoring and Analysis

Performance analysis for this emulation framework is necessary for two primary reasons: 1) to help users understand and evaluate the performance of their applications at a micro-level and 2) to help us understand the behavior of the framework itself. To do this, the virtual machine monitor will provide abstractions and mechanisms for *cross-stack performance monitoring and analysis* to external monitoring tools.

The VMM provides a useful vantage point that makes it possible for profilers to span at least four layers: the application, the OS, the "hardware" interface exported by the VMM and the actual hardware. This is possible because, many instruction-visible events are naturally intercepted by the VMM (*e.g.*, interrupts) or can be funneled through the VMM via mechanisms such as virtual address translation (*for example*, access to a specific memory region). Second, microarchitectural events can be monitored by virtualizing the performance counters. The key challenges are leveraging familiar abstractions for accessing performance data and and providing this data to higher-level guest OSes or applications and the enablement of non-intrusive analyses.

4. IMPLEMENTATION PLAN AND STATUS

4.1 Virtual Machine Monitor

We are basing our implementation of the proposed architecture around the Palacios virtual machine monitor that we have previously developed to support lightweight virtualization in HPC environments [8]. Palacios is an HPC-oriented virtual machine monitor designed to be embedded into a range of different host operating systems, including the lightweight kernels [11], Linux variants potentially including the Cray Linux Environment [12], the MINIX microkernel, and others. Recent work has shown that Palacios can virtualize thousands of nodes of a Cray XT class supercomputer with less that 5% overhead [7]. Palacios's combination of low overhead on HPC systems and embeddability into traditional HPC operating systems, both lightweight and commodity-based, makes it an ideal platform for our research.

Time dilation support.

To support time dilation and in Palacios, we are augmenting Palacios time management with the necessary scheduling and timer interrupt control features. In particular, to slow down guest time, Palacios uses two quantities:

Target cycle rate, the number of cycles that the guest should see execute per *emulated* second of guest time.

Target time rate, the time dilation factor for this guest which determines at what rate timer interrupts are delivered to the guest compared to real time.

At emulation boot time, Palacios uses the sum of the target cycle rates of all of the virtual processors on each host specified to determine the minimum required target time rate for the virtual machines it will emulate. For example, if virtual machine monitor must emulate 4 3Ghz processors using 1 2Ghz core, it sets the minimam required target time rate to 6 ($(4*3Ghz)/2Ghz$). Note that the target time rate can be adjusted upward from this point, but cannot go any lower than this minimum.

Given a taret cycle rate and target time rate, Palacios then schedules guest cores so that each receives the appropriate number of cycles in each one emulated second. In the example above, for example, Palacios needs to use a 2Ghz processor to give each of 12 virtual cores three billion cycles in 6 seconds of real time. In this simple example, that is done simply by giving each core 1/12th of the host processor, but in more complicated cases with higher specified time dilation, Palacios may idle the core periodically so that the correct number of guest cycles elapse for each second of emulated guest time.

One important impact of this is that the guest timestamp counter (TSC) does not have to be completely virtualized; TSC offsetting supported by both Intel VT virtualization and AMD SVM virtualization is sufficient. This is important because virtualizing the TSC is potentially very expensive—full TSC virtualization turns an instruction that takes at worst tens of cycles into one that takes tens of thousands of cycles.

Architectural and Slack Simulation Support.

In addition to the time dilation support mentioned above, we have also added the ability to pause, unpause, and sychronize guest time to a provided reference time source to Palacios. In particular, Palacios now keeps track of how often timer events are read or injected into the guest, and uses this information to bound how quickly it offsets guest time towards the desired target time rate. This limits guest-visible timer inaccuracy while still allowing Palacios to control time passage in the guest for slack simulation purposes.

VM-VM Communication.

For VM-to-VM communication, we are relying on RDMA communication facilities provided by the host operating system in which Palacios is embedded, for example Infiniband device support. The low latencies provided by such devices are essential for fast simulation of low-latency network devices, and support for accessing such devices is already being added to Palacios as part of another project.

4.2 Simulator Integration

To support novel HPC hardware devices, we are working on integrating the Structural Simulation Toolkit (SST) architectural simulator with Palacios. SST is a parallel discrete event simulator that provides a modular framework for constructing hardware device models at various levels of fidelity. SST can be used to simulate large-scale systems, and is itself an MPI program. The downside to SST's software-based approach is performance. Our goal in integrating Palacios with SST is to achieve higher levels of simulation performance by executing most code at near native speed in the hardware-accelerated virtual machine environment, and only passing control to SST when necessary.

The general structure of our proposed architecture is shown in Figure 2. Palacios already provides mechanisms for hooking specific instructions and regions of guest memory such that they always cause a VM exit, passing control from the

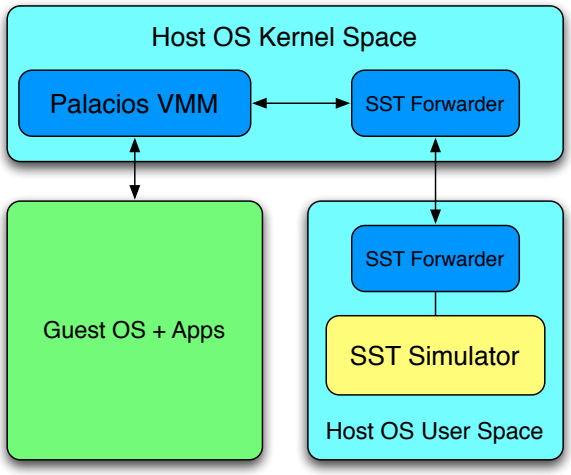

Figure 2: High-level Architecture of Palacios VMM and SST Architectural Simulator Integration

guest back to Palacios for handling. At this point, Palacios forwards information about the event to the SST forwarding agent in host OS kernel-space, which then forwards it the SST instance running in user-space. This is analogous to how KVM interacts with Qemu on Linux. An alternative and potentially higher performance approach would be to integrate SST directly with Palacios and have the combination execute fully in host OS kernel, but we expect that it would be difficult to adapt SST for the limitations of running in kernel-space. In our proposed scheme, the forwarding agent is responsible for translating between Palacios events and the SST events needed to interacting with the SST instance. The timing information in SST events will be used by Palacios to control the progression of time in the guest environment, therefore providing a realistic passage of time.

The primary limitation of this approach is that it limits the extent of what can be simulated. Obviously, if every guest instruction causes a VM exit, performance will be worse than when simulating with SST alone due to the increased overhead. The best situation will be when the vast majority of instructions are executed natively, and only a small percentage are forwarded to SST for handling. We expect that simulating relatively self-contained hardware features such as network interfaces and global address space schemes will demonstrate this behavior and perform well with our approach.

4.3 Monitoring and Analysis

To provide monitoring and analysis support, our baseline approach is to extend the Performance API (PAPI) [2] to support VMM performance counters. Our initial focus will be on the high-level PAPI interface that supports simple (start, stop and read) event measurements. Later, we will include support for PAPI's low-level, programmable interface that supports grouping related events to provide higher-level information. Using this approach means that the myriad of existing PAPI-based tracing, profiling and analysis tools would become usable with our VMM framework.

Additionally, we will explore mechanisms that perform rudimentary performance analyses. Our approach is to build a framework we call VMM Tuning and Analysis (VTAU) using the TAU [10] framework. The VTAU framework will be used to wrap performance critical components (for example, functions, code regions and loops) with tracing or profiling code as appropriate. This will allow us to control the collection of timing and event information mapped to VMM functionality. Our VTAU framework will be able to leverage the feature-rich visualizations of the TAU framework.

5. RELATED WORK

A number of systems have used techniques similar to the ones we suggest for simulating or emulating large-scale systems. As mentioned above, DieCast's time dilation approach [6, 5] is closely related to our work, and forms a partial basis for the system we propose. Unlike the system we propose, however, DieCast makes only limited use of architectural simulation, in particular only for high-latency devices such as disk systems. This avoids the time synchronization issues inherent in simulating low-latency devices, but limits DieCast's usefulness in studying the impact of novel low-latency I/O devices in large-scale systems.

Also closely related to the system we propose is past work on cluster-based simulation of cluster systems [9]. This system uses network simulation such as we propose in combination with a fine-grained processor simulator, and allows for detailed simulation and analysis of processor and memory system changes not possible in the system we propose. Because of its reliance on cycle-accurate simulation, however, its runtime is bounded by the the runtime of individual node simulations, which can result in slowdowns of several orders of magnitude. As a result, this and similar systems are most appropriate for studying the performance of benchmarks and simplified mini-applications, not full applications such as we seek to study.

6. CONCLUSIONS

In this position paper, we have described the architecture of a virtualization-based emulation system. This design is based on the novel combination of a number of existing techniques, including time dilation, slack simulation, and network simulation and emulation, with additional techniques to improve their performance. The resulting system seeks to provide fast, full-scale emulation of future large-scale architectures. Such emulations will aid the development both hardware and software for upcoming exascale class supercomputers.

7. REFERENCES

[1] Bohrer, P., Elnozahy, M., Gheith, A., Lefurgy, C., Nakra, T., Peterson, J., Rajamony, R., Rockhold, R., Shafi, H., Simpson, R., Speight, E., Sudeep, K., Hensbergen, E. V., and Zhang, L. Mambo – a full system simulator for the PowerPC architecture. *ACM SIGMETRICS Performance Evaluation Review 31*, 4 (Mar. 2004), 8–12.

[2] Browne, S., Dongarra, J., Garner, N., Ho, G., and Mucci, P. A portable programming interface for performance evaluation on modern processors. *The International Journal of High Performance Computing Applications 14*, 3 (Fall 2000), 189–204.

[3] Chen, J., Annavaram, M., and Dubois, M. Exploiting simulation slack to improve parallel simulation speed. In *Proceedings of the International Conference on Parallel Processing* (Los Alamitos, CA, USA, 2009), IEEE Computer Society, pp. 371–378.

[4] Fujimoto, R. M. Parallel discrete event simulation. *Communications of the ACM 33*, 10 (1990), 30–53.

[5] Gupta, D., Vishwanath, K. V., and Vahdat, A. Diecast: testing distributed systems with an accurate scale model. In *Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation* (Berkeley, CA, USA, 2008), NSDI'08, USENIX Association, pp. 407–422.

[6] Gupta, D., Yocum, K., McNett, M., Snoeren, A. C., Vahdat, A., and Voelker, G. M. To infinity and beyond: time-warped network emulation. In *Proceedings of the 3rd conference on Networked Systems Design & Implementation - Volume 3* (Berkeley, CA, USA, 2006), NSDI'06, USENIX Association, pp. 7–7.

[7] Lange, J., Pedretti, K., Dinda, P., Bridges, P. G., Bae, C., Soltero, P., and Merritt, A. Minimal-overhead virtualization of a large scale supercomputer. In *Proceedings of the 2011 ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments (VEE 2011)* (Newport Beach, CA, March 2011).

[8] Lange, J., Pedretti, K., Hudson, T., Dinda, P., Cui, Z., Xia, L., Bridges, P., Gocke, A., Jaconette, S., Levenhagen, M., and Brightwell, R. Palacios and Kitten: New high performance operating systems for scalable virtualized and native supercomputing. In *Proceedings of the 24th IEEE International Parallel and Distributed Processing Symposium* (April 2010).

[9] León, E. A., Riesen, R., Maccabe, A. B., and Bridges, P. G. Instruction-level simulation of a cluster at scale. In *Proceedings of the 2009 ACM/IEEE International Conference on Supercomputing (SC 2009)* (November 2009).

[10] Mohr, B., Brown, D., and Malony, A. D. TAU: A portable parallel program analysis environment for pC++. In *Conference on Algorithms and Hardware for Parallel Processing* (1994), pp. 29–40.

[11] Riesen, R., Brightwell, R., Bridges, P. G., Hudson, T., Maccabe, A. B., Widener, P. M., and Ferreira, K. B. Designing and implementing lightweight kernels for capability computing. *Concurrency and Computation: Practice and Experience 21*, 6 (April 2009), 791–817.

[12] Wallace, D. Compute Node Linux : Overview, progress to date, and roadmap. In *Proceedings of the 2007 Cray User Group Annual Technical Conference* (May 2007).

[13] Willcock, J., Hoefler, T., Edmonds, N., and Lumsdaine, A. AM++: A Generalized Active Message Framework. In *Proceedings of the Nineteenth International Conference on Parallel Architectures and Compilation Techniques (PACT'10)* (September 2010).

[14] Zhu, W., Bridges, P. G., and Maccabe, A. B. Lightweight application monitoring and tuning with embedded gossip. *IEEE Transactions of Parallel and Distributed Systems (TPDS) 20*, 7 (July 2009), 1038–1049.