Developing Embedded Networked Products using the Simics Full-System Simulator

Jakob Engblom, David Kågedal, Andreas Moestedt, and Johan Runeson

This paper is an industrial case and tool study that describes the Simics full-system simulator and its network simulation facilities. It also gives a number of examples showing how full-system simulation of heterogeneous networked embedded systems is used in the development, debugging, and testing of networked embedded systems. Simics is a commercial product with many active customers in industry, and our experience from servicing these customers has had a great effect on the design of the tool.

***Index Terms*—Simulation software, Networks, Network Testing, Software Development**

I. Introduction

THIS paper presents the network simulation features and techniques used in the commercial full-system simulator Virtutech Simics. This simulator is in use currently at all Virtutech customers, and it has been used to simulate a number of complex commercial embedded networked systems.

The paper starts with an overview of the simulation system, continues with some industrial case studies of Simics applications, and finishes up with comparison to related work and a short discussion.

Full-System Network Simulation

A. Full-System Simulation

A *full-system simulator* is a computer program that simulates computer systems at such a level of detail that complete software stacks from real systems can run on the simulator without any modification. Basically, you get *virtual hardware*. The full-system model includes processor cores, peripheral devices, memories, and network connections [1][2][3]. Thus, full-system simulation makes it possible to simulate network nodes with all their software, from network device drivers to operating systems, network stacks, middleware, servers, and application programs.

Virtutech Simics [1] is a commercial full-system simulator that has been used in many commercial and academic projects to simulate heterogeneous networked and distributed systems (as well as single computers). At heart, Simics is an event-driven simulator where processors are treated specially for performance reasons. All I/O and other peripheral devices in a system (and thus all interconnects between systems) are simulated in a *transaction-based* style. This means that entire transactions are performed as one action; for example, sending a network packet is simulated by transmitting the complete packet at once, rather than sending the individual bytes of the packets individually.

Simics is designed to be a fast simulator, and can currently achieve speeds of up to 1900 MIPS when running single-processor workloads on top of full simulated systems with a real operating system[1]. This speed allows for real workloads to be run in simulation, including operating systems, network stacks, and complex applications. Simics runs the same binaries as the real target machine, which allows for very realistic network experiments where the setup can be identical to the real world. The software cannot tell the difference to a real hardware environment.

The basic Simics simulation of a network node is *deterministic* – from the same initial state, the simulation will simulate the exact same execution path every time the simulation is run.

Simics can execute any number of simulated machines within a single Simics process, with no limitations on the types of nodes. When simulating multiple nodes within a single Simics process, in order to gain good locality and thus good execution speed, we use *round-robin time slicing*. Each node gets to execute a certain (simulated) time before switching to the next node. In loosely coupled network scenarios, the time slice is often set to about 100k instructions (assuming homogeneous nodes).

Simics uses a completely virtual time base, which means that the relative execution speed of network nodes is respected in simulation. For example, in a scenario mixing 10 MHz sensor nodes with 500 MHz gateways, the gateway nodes will execute code 50 times faster than the sensor nodes (assuming that both have the same average clocks-per-instruction ratio). This means that the time slices for each node are adjusted to account for node speed.

When simulating large networks, we have noticed that in many cases, most nodes are idle or very lightly loaded. Simics takes advantage of this by *skipping idle time*; when a processor is waiting for the next interrupt to happen (for example, executing a HLT instruction on an x86 processor or putting a PowerPC processor into doze mode), Simics will skip ahead until the next interesting event happens. This can speed up simulation tremendously, as we avoid stepping through idle time cycle by cycle.

B. Network Link Simulation

Networks are simulated on the *packet level* in Simics.

All authors are with Virtutech AB, Stockholm, Sweden (http://www.virtutech.com). They can be reached at email addresses {jakob, david, am, runeson}@virtutech.com.

[1] Running an encryption benchmark on a simulated PowerPC 750 target machine with a Linux OS. Host was a 2.4 GHz AMD Athlon64.

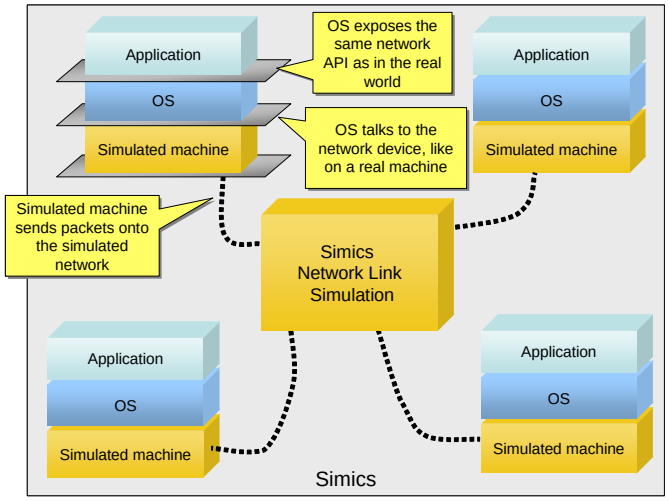

Fig. 1. Simics network simulation, showing the levels of simulation and the exposed APIs.

There are simulated network cards in all simulated nodes that send and receive packets, and network link objects representing the networks transporting the packets.

The network cards are models of real network cards and use the same device drivers as the real cards would. For example, on the PC platform we model network cards like the DEC 21140A, and for an AMCC PowerPC440GP SoC we model the on-chip Ethernet controller. This makes it possible to run unmodified software systems on top of Simics, as illustrated in Fig 1.

The network devices are connected to *network link objects* that represent the actual networks. Each link object corresponds to a point-to-point connection or a shared medium or switched networked segment. Packet transport in a network is done on a link level. All packets visible on a real Ethernet link are visible in Simics, including malformed packets, broadcast packets, etc.

Determinism is maintained at the network level, across the complete network. All network packets will be sent and received at the exact same points in time from the perspective of all nodes in the network. This is a straight-forward consequence of every node being simulated in a deterministic manner with a virtual time base.

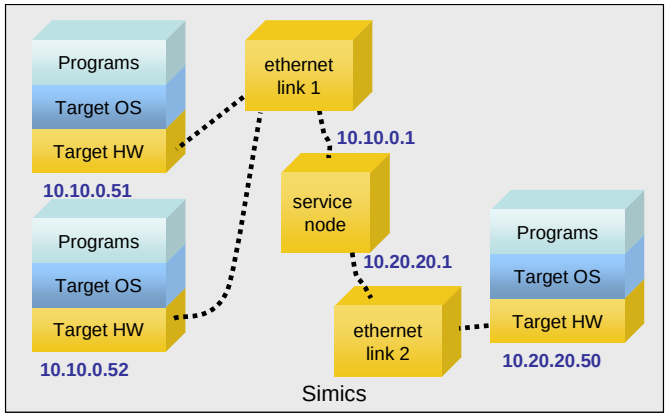

Fig. 2. Service nodes are abstract network nodes that do not run a real software stack. They are connected to the simulated network as distinct machines, and have their own MAC-level and IP-level address (shown here for an IPv4 example). Here, the service node is also used as a router between the 10.20.20.x and 10.10.0.x networks. All fully simulated machines use the service node as their default gateway for IP traffic.

Packets are delivered with a configurable *latency* (which is set per link object). To maintain determinism, the latency cannot be lower than the length of a time slice. Thus, a shorter execution time slice in a multi-machine setup gives lower network latency but also slower simulation speed. The right trade-off depends on the particular characteristics of the target systems and network protocol. In some cases, very short time slices might be more appropriate to capture all details of a network interaction, but in most cases quite long slices work very well in our experience. Packet sends are rare events compared to instruction execution, and many network protocols are built to be latency-tolerant.

Finally, machines and networks simulated in Simics support *checkpointing*: the state of all machines and the networks connecting them can be written to disk at any point in time and brought back up later to the exact same state. This is a great time saver and work method enabler.

C. *Abstract Nodes*

The nodes on the simulated networks need not be constrained to fully simulated machines only. Basic network services can be provided by *service nodes* that directly handle network traffic without simulating a computer and running a real workload on them. Service nodes are basically Simics modules that run natively on the host machine and talk to the simulated network.

The service nodes shipping with Simics provides convenience services for IPv4 networks like DHCP, BOOTP, DNS, TFTP, and ping. They can also be used as virtual *routers* between different simulated Ethernet networks, using static routing tables. Fig 2 shows an example of a setup involving a service node used as a router and two simulated networks.

The ability to insert abstract nodes in a network means that the level of simulation of network nodes can be varied across the network. This has been used to implement load generators connected to a simulated network. Abstract nodes can also be used to implement light-weight nodes in the simulated network. For example, IP phone terminals in an IP telephony scenario where the phones place calls regularly and reply to signaling traffic, but there is no real software running.

D. *Distributed Simulation*

Simulation can be distributed across multiple host machines in order to support larger simulations. The basis for this is the synchronization mechanism of *Simics Central* [1]. With central, simulated network nodes can be distributed over two or more host machines (or processors of a multiprocessor host). A network interconnection forms a natural point of division in a simulated system, as it is a packet-level interface with long latencies and relatively low bandwidth (compared to the tight coupling of processors within a shared memory machine).

Simics uses a single Simics Central *server* that can be placed in any Simics process taking part in the simulation, and Simics Central *clients* that are present in all Simics processes. The clients communicate with the server to make sure that all simulations are proceeding in lock-step. Fig. 3 shows an example of a distributed simulation setup.

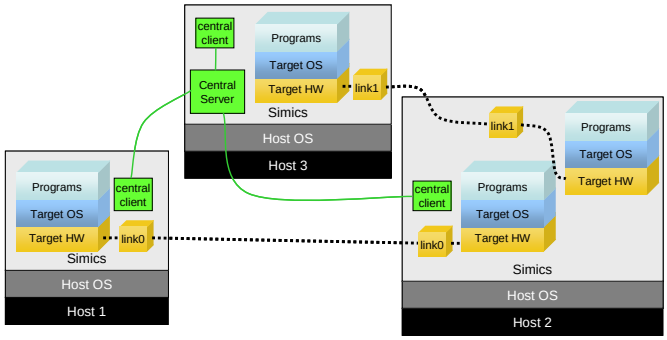

Fig. 3. Distributed simulation example showing four simulated machines spread across three host machines. The central server and its connections to the clients is shown, as well as two separate simulated networks. The networks are not connected to Central in any way, and synchronization can be used independently of networking.

The basic premise in distributed simulation is to keep all simulation processes simulating the same point in virtual time. However, achieving perfect synchronization at every point in time is too expensive in terms of overhead and having simulation processes stand idle. The Simics Central solution allows the Simics processes some slack: the distributed simulation is not completely synchronized, but there is an upper bound on how much time might differ between two different processes. The difference in time between two processes *n* and *m*, |T*n* - T*m*|, is always less than or equal to the maximum slack *D*.

Note that in Simics versions prior to 2.2, Central was integrated with each network link simulation. The separation of network links and distributed synchronization is an obvious improvement as more types of network links are added to Simics, simplifying implementation and improving scalability.

E. *Connection to Real Network*

Simics supports the interconnection of a simulated network with a real network. This *real-network connection* has turned out to be quite a useful feature. It has been used to provide stimuli to the simulated network by having external clients connect to simulated servers or vice versa. It also offers a convenient way to connect existing debug tools to the simulated machines. For example, the WindRiver Tornado tools work by having a target server executing on the target machine that talks to the PC-hosted development tools over Ethernet. With a real-network connection, these tools work with a Simics-simulated target machine just like with a real machine. Customers have also connected Simics simulated machines to real-world test tools that use Ethernet to connect to a target machine to be tested. For the testing tools, the simulated machines are just machines with an IP address like any other, there is no difference compared to talking to a real machine. This way, Simics can be used in place of a real machine without changing the existing tools setup.

There are several implementation variants for real-network connections. It is possible to use a *virtual router* to route between the simulated and a real network, or *bridge* between them at a lower level. An easy implementation is to use *port forwarding* and *Network Address Translation* (NAT) to provide real-network connections without administrative privileges on the host.

An interesting effect when interconnecting simulated and real network nodes is that the time runs at different speeds in simulation and in the real world. The simulated nodes run at an uneven speed, which might cause synchronization and time-out issues. The preferred solution to this problem is to have the real world system listen to a heartbeat from the simulated machines. Or simply make the simulated machines very fast and throttle their execution to not run faster than the real-world time.

Obviously, involving real nodes in the network simulation will break determinism and make checkpointing less powerful. Note that Simics makes it possible to record a real-network session in order to replay it with full determinism.

F. *Imperfect and Limited Network Simulation*

The network simulation described so far implements a perfect network with potentially infinite bandwidth. This is sufficient to test many applications, as the main issue is to get the functionality right, and the network timing has no effect on application correctness.

The easiest limitation to add to a network is *bandwidth limitation*. The basic implementation that we use for this is to limit the stream of packets at the source, i.e. in a network device. This solution easily maintains determinism and requires no global synchronization which minimizes the performance impact.

There are also cases where *imperfect* networks are needed to test an application. For example, safety-critical applications should handle network packets being garbled or lost, and testing this in simulation is much easier than in the real world. To support such *fault injection* in the network, we provide an interface for modifying or deleting packets in the networks, at the point in time where they have left the source node.

We are also planning on extending the network instrumentation to include the ability to set the precise delivery time of individual packets, as well as the capability to repeat and inject new packets into the network. This work is motivated by our involvement in the EU RUNES project, as discussed below in Section II.D.

G. *Performance*

The network simulation per se incurs a very small overhead in the overall Simics simulation, as network packet transmissions are comparatively rare events compared to instruction execution or memory accesses (which dominate the execution time of the simulator). Thus, the performance of Simics with networks simulated is very close to the simulation of the same machines but without a network added. The use cases below will provide some anecdotal performance numbers for the whole systems.

Using distributed simulation with Simics Central does incur a performance penalty when Simics host processes are waiting for each other. Thus, distribution should only be used when absolutely necessary.

II. Use Cases

This section presents some example cases of how Simics has been used to build simulators for networked embedded

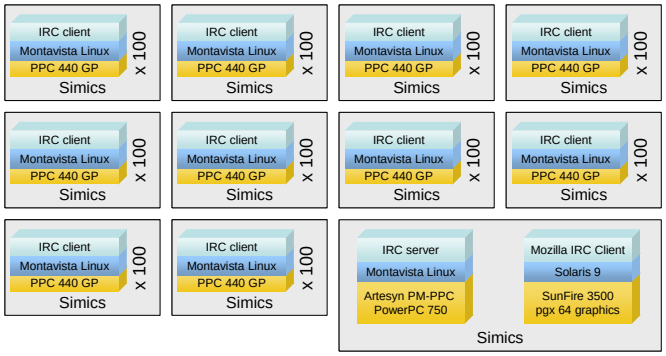

Fig. 4. Large simulated network example – 11 hosts are used to simulate 1002 machines.

systems.

A. *Very Large Network Simulation*

To demonstrate Simics network scalability for the Embedded Systems Conference 2004 [4], we created a simulated network containing 1002 machines. The network contained an IRC server running on a simulated PowerPC 750 machine, as well as an IRC graphical front-end running on a simulated UltraSPARC machine, and 1000 IRC clients running on simulated PowerPC 440GP machines. The whole network was simulated using 11 processors, as shown in Fig. 4. Since the IRC clients were mostly idle, the overall speed of the system was equal to a real-world speed of about 20 MHz for each PPC client (run on 11 1.8 GHz Opteron processors).

B. *Multiple Networks in a Large Telecom Switch*

Telecom switches are typically large embedded systems, consisting of many tens of processing and input/output cards mounted in custom or standard racks (for example, AdvancedTCA [5]). The backplane connecting the cards is a network based on standard networks such as Ethernet or ATM, and communications between cards is based on network traffic and not shared memory. In addition to the backplane, there are also multiple network connections entering the switch from the front, carrying inbound and outbound traffic.

A concrete example of such a switch has been simulated in Simics. A large configuration contains 22 cards with a mix of different types of PowerPC processors. The backplane is based on ATM, and traffic enters the switch over Ethernet links from the front. Several simulated network links are used in the simulation this system. The ATM backplane is a different network from the front-side Ethernet. Real-network connections are used to run test cases against the simulated switch using the same TTCN scripts and tools as would be used with a real switch.

Needless to say, the simulated switch passed the tests and is accepted as a fully functional simulator that can be used instead of a real switch for software testing and debugging. Using a simulator in this manner is much cheaper than buying hardware, in addition to the visibility and instrumentation features that simulation brings. The speed of the simulation is acceptable for daily use, especially with smaller configurations containing only a few cards.

C. *Scalability of Self Configuring Systems*

In another case, Simics was used to test the scalability of self-configuring storage fabrics. The network consisted of many fiber-channel loops connecting PowerPC-based custom boxes together. The system is designed to automatically configure itself when it boots up, but testing this for large configurations is essentially impossible using hardware as the test engineers cannot get their hands on the 200+ machines needed to build a good test network (not to mention the hardware cost and configuration time required).

Instead, Simics was used to simulate a fabric containing more than 200 machines, with all links in place (using scripting to automatically create the network). This test was successful in demonstrating that the self-configuration did behave as expected. This example shows the value of simulation in being able to trade execution time for system size: you can do very large configurations if you are willing to sacrifice some speed.

D. *Sensor Networks*

Within the RUNES EU research project [6], Virtutech is working on building a simulation environment for development and testing of software for sensor networks. The simulation will target a system consisting of a mix of small sensor nodes based on 16-bit MSP430 processors (Telos Mote-IV) and intermediate MIPS32-based gateways, as well as x86-based PCs or other interesting standard machines interfacing to such a network. The radio networks connecting the nodes are Zigbee and WLAN, and we will include the capability to simulate imperfect networks with limited transmission ranges, packet loss, and garbled information. The geographical location of the nodes will be included in the simulation in order to correctly account for radio range and overhearing effects.

The simulation will be used to test both the basic functionality and the scalability of the middleware for sensor networks that is developed within the RUNES project. As the sensor nodes are mostly idle (duty cycle is about 1-2% to conserve battery power) and use little memory, we estimate that simulating thousands of nodes on a single PC with a reasonable slowdown is perfectly feasible.

Initially, the networks will be simulated using simple perfect transmissions between appropriate sets of nodes, and the addition of imperfections in the radio layer will be optional. This allows users to debug basic functionality separately from the behavior in the presence of noise and other imperfections in the real environment.

III. Related work

There is a range of abstract network simulators available that focus on the network itself. The best known example is probably *NS-2* [7]. In this type of simulator, the nodes are abstracted and do not run real codes or operating systems, but rather simple behavioral models or statistical traffic generators. The advantage of the abstraction is that scalability is excellent, and large networks containing thousands of nodes can be routinely simulated. The simulator itself has to contain an implementation of the network protocols used in the simulated network, which means that introducing new protocols requires significant implementation work. In contrast, a simulation on the physical level like Simics is oblivious to the protocols used.

In general, Simics also runs the actual code in each node, even if abstract nodes can also be used.

TOSSIM [8] is a simulator for sensor networks based on the TinyOS operating system. It compiles code written for the TinyOS API to run on a Windows or Linux PC, by emulating the function of the operating system and hardware below the (fairly low-level) TinyOS API. In this way, it is possible to simulate networks consisting of a thousand sensor nodes, but you do not run the real target code. Working at the same API-level abstractions, *CCSimTech* [9] simulates CAN-based embedded networks. It has been extended with technology to synchronize the speed of individual network nodes to get closer to the real system behavior [10]. The *OSE SoftKernel* [11] is an API-level simulator for the Enea OSE real-time operating system that also offers the simulation of networks of systems. In contrast, Simics runs the actual code and complete software stack of the target, with no special compilation required for simulation.

Virtual Mote Network (VMNet) [12] is a full-system full-network simulator for sensor networks. It combines the *ATEMU* [13] full-system simulator for Atmel AVR-based *Mica2 Motes* with a detailed simulated network layer. ATEMU itself is a complete virtual sensor network simulation solution, and VMNet adds more detailed radio system simulation on top of ATEMU. In contrast to the transaction-oriented Simics network simulation, these systems simulate radio networks based on sending individual bits, and nodes are interleaved per clock cycle which increases the simulation accuracy at the cost of simulation speed. They also presume homogeneous networks that all consist of Motes.

The *Virtual Network User-Mode Linux* (VNUML) [14] is a system built on top of User-Mode Linux (UML) to allow the creation of large simulated networks consisting of x86-based Linux nodes. The network simulation is based on tap devices and bridging, the same methods as used for Simics real-world networking. Simics, in contrast to VNUML, allows for network nodes to be based on arbitrary processor architectures, and to run arbitrary operating systems. VNUML also does not allow the speed of simulated nodes to be controlled to mimic heterogeneous networks.

Similarly, *virtualization systems* like VmWare [15] or *paravirtualization systems* like Xen [16] can be used to create virtual networks consisting of copies of the host machine, with potentially different operating systems. However, they cannot simulate the precise hardware of a particular target system, only generic hosts.

The *M5* simulation framework [17] is similar to Simics in that it uses full-system simulation to construct networks. However, the focus of M5 is more on the computer architecture side of networked systems than on the network as such, and the networks targeted are smaller than those we simulate with Simics. Simics executes code much faster than M5, and M5 is currently limited to a single Alpha target system and the PISA research instruction set.

IV. Discussion

Simics is a full-system simulator with network simulation abilities that has been used to simulate a large number of different embedded (and non-embedded) networked systems. Simics has been designed with to achieve simulation scalability and speed while maintaining an interesting level of detail (compared to a real target). The current compromise has been proven to be very useful in practice, allowing simulators to be used instead of real hardware in many real development projects. The simulation runs the same software as the real-world system, and as more and more functions migrate to software from custom hardware, analyzing and testing the real software in a realistic environment is crucial. The level of abstraction used in Simics network simulation has been proven appropriate for this purpose.

Acknowledgment

Thanks to the whole team at Virtutech who have worked very hard for many years to make Simics the wonderful tool that it is today.

References

[1] Peter S. Magnusson, Magnus Christensson, Jesper Eskilson, Daniel Forsgren, Gustav Hållberg, Johan Högberg, Fredrik Larsson, Andreas Moestedt, Bengt Werner. “Simics: A Full System Simulation Platform”, *IEEE Computer*, Feb 2002.

[2] Alaa R. Alameldeen, Milo M.K. Martin, Carl J. Mauer, Kevin E. Moore, Min Xu, Daniel J. Sorin, Mark D. Hill and David A. Wood. “Simulating a $2M Commercial Server on a $2K PC”, *IEEE Computer*, Feb 2003.

[3] Jakob Engblom. Full-System Simulation, *Proceedings of the European Summer School on Embedded Systems (ESSES)*, Sep 2003.

[4] Embedded Systems Conference: http://www.esconline.com

[5] AdvancedTCA at the PCIMG: http://www.picmg.org/newinitiative.stm

[6] RUNES EU Project, http://www.ist-runes.org.

[7] NS-2: http://www.isi.edu/nsnam/ns/.

[8] P. Levis, N. Lee, M. Welsh, and D. Culler. Tossim: Accurate and Scalable Simulation of Entire TinyOS Applications, *Proceedings of the First ACM Conference on Embedded Networked Sensor Systems (SenSys 2003)*, Nov 2003.

[9] Anders Möller, “A Simulation Technology for CAN-based Systems”, *CAN Newsletter*, Dec 2004.

[10] Jakob Engblom and Magnus Nilsson. Time Accurate Simulation: Making a PC Behave Like a 8-bit Embedded CPU, Technical Report 2002-024, Dept. of Information Technology, Uppsala University, 2002.

[11] *OSE SoftKernel Environment Datasheet*, ENEA Embedded Technology, 2004.

[12] Hejun Wu, Qiong Luo, Pei Zheng, Bingsheng He, and Lionel M. Ni. Accurate Emulation of Wireless Sensor Networks, *Proceedings of the Workshop on Building Intelligent Networks (BISON)*, Oct 2004.

[13] Jonathan Polley, Dionysys Blazakis, Jonathan McGee, Dan Rusk, John S. Baras. ATEMU: A Fine-grained Sensor Network Simulator, *Proceedings of the First IEEE International Conference on Sensor and Ad Hoc Communication Networks (SECON'04)*, Oct 2004.

[14] Fermín Galán, David Fernández, and Tomás de Miguel. “Study and Emulation of IPv6 Internet-Exchange-Based Addressing Models”, *IEEE Communications Magazine*, Jan 2004.

[15] VmWare: www.vmware.com

[16] P Barham, B Dragovic, K Fraser, S Hand, T Harris, A Ho, R Neugebauer, I Pratt, and A Warfield. Xen and the Art of Virtualization, Proceedings of 19th ACM Symposium on Operating Systems Principles (SOSP2003), October 2003.

[17] Nathan L. Binkert, Erik G. Hallnor, and Steven K. Reinhardt. Network-Oriented Full-System Simulation using M5, *Proceedings of the Sixth Workshop on Computer Architecture Evaluation using Commercial Workloads (CAECW)*, Feb 2003.