Virtual Machines

Lecture #10: Thursday, 1 May 2003
Lecturer: Jayanth Gummaraju, Garrett Smith
Scribe: Joel Coburn, Abhyudaya Chodisetti

1 Introduction

Convential computer systems architecture has well defined interfaces between applications, OSes, and hardware. Major design tasks are decoupled so that hardware and software designers can work independently. As hardware and software have continued to grow in complexity, there is an increasing problem of interoperability and backward compatibility. Also, optimization across major interfaces is difficult. Virtual machines provide a layer between hardware and software. This adds flexibility through cross-platform software portability and can also provide performance optimization.

The papers discussed ([1],[2]) present Disco and Co-designed virtual machines.

2 Disco: Running Commodity Operating Systems on Scalable Multiprocessors([1])

This paper talks about an efficient way of running commodity Operating Systems on multi-processors. The authors show that by inserting an additional layer of software between the hardware and operating system (HAL: Hardware Abstraction Layer), commodity operating systems can run on scalable multi-processors sharing major resources transparently. HAL acts like a Virtual Machine Monitor (VMM) in that multiple copies of operating systems run on a single scalable computer. The authors also show that performance degradation due to the extra level of indirection is minimal.

Using VMMs to scale existing operating systems for multi-processors offer several advantages. First, VMMs offer flexibility and scalability in that potentially different operating systems can be run on the same hardware in parallel. Second, minimal changes are needed to the existing OS code, leading to higher reliability and availability of the system. Third, VMMs enable fault containment: the failures in the software system can be contained within virtual machines rather than spreading to the entire machine. Fourth, VMMs offer portability - the same operating system can be run on different hardware. For example, the authors show that the same operating system can be run on a UMA (Uniform Memory Access) or NUMA (Non-uniform Memory Access) by hiding the NUMAness using techniques like page replication and page migration. Finally, VMMs

also enable sharing resources transparently at a fine granularity. In particular, the authors show how the disk and memory resources can be shared by virtual machines running in parallel.

The paper derives its strength by showing how an old concept of virtual machines can be innovatively used to run commodity operating systems on multi-processors with minimal changes. The paper also describes new techniques to hide NUMAness of multi-processors viz. page replication and page migration. The paper also gives detailed experimental results, showing the overhead of using VMM on different parts of the system, and also demonstrating performance benefits of page migration and page replication.

The paper doesn't have any significant shortcomings. However, it would be interesting to see a couple more experiments and results. First, how would running eight copies of IRIX on Flash (8 processor NUMA machine) compare to running IRIX on eight networked workstations (sharing files using NFS for example). Second, a study on how engineering, raytrace, and database applications scale using VMs, in addition to the pmake workload (scalability of which is demonstrated) would be interesting. Finally, it would be interesting to see how the concept of VMs can be extended to modern day CMPs (Chip Multi-Processors).

3 Achieving High Performance via Co-designed Virtual Machines([2])

The idea behind Co-Designed Virtual Machines is using Virtual Machines for performance improvement. Previous uses of virtual machines were portability (Java), Compatibility (FX!32), or virtualization (the original use of VMs). With these virtual machines the only thoughts of performance was in reducing the slowdown caused by the virtualization. Co-Designed VM's use the virtualization to optimize the running binary for the specific processor it is running on, thus providing a speedup and allowing the immediate use of new hardware features.

There are several terms introduced by the paper: V-ISA, the virtual ISA that is presented to the software by the VM and I-ISA, the ISA implemented by the hardware. The Virtual Machine sits between the I-ISA and the V-ISA and implements the parts of the V-ISA not provided by the hardware. With Co-Designing, the creators of the system can decide on the boundary between the VM and the I-ISA. In previous systems the I-ISA was already fixed because the ISA was designed separately from the VM. With Co-Design, special features targeted at VM's can be provided, and tradeoffs might be different if VM's are going to translate running code.

Some example optimizations suggested are: tuning the branch prediction global history buffer length; using the VM to select traces to schedule at runtime, with the hardware providing performance information about which branches are taken, and the software using this to analyze large portions of code and generate traces; using software to decide when to use data values in predicting branches; and re-organizing the memory layout of

a Java application to improve locality of popular data structures.

The key aspects of Co-Design:

The hardware provides:

1. **Performance information.** This is used by the VM to decide what portions of the code to optimize and allows the VM to tune the optimizations to the current behavior of the application.

2. **Configurability.** The hardware can provides special instructions that allow the VM to re-configure the processor so that it is better suited for the running applications. These configurations changes can be as small or as far reaching as desired.

The VM software performs:

1. **Analysis.** The software uses its ability to perform more complex analysis and inspect large regions of code at once to perform more detailed analysis than hardware can.

2. **Optimization.** The software performs optimizations based on the analysis. It can dynamically compile sections of code and optimize them, or it can re-arrange the layout of memory (eg Disco).

3. The VM software can also reconfigure the hardware based on its analysis of the applications behavior.

The Co-Design paper proposes that Virtual Machines can be used for performance enhancement in addition to their previous uses. This suggests new areas for research and this paper gives some example optimizations, but it doesn't go into any details of how to implement any optimizations or what speedups could be expected.

4 Discussion

4.1 ISA Requirements

The first question of the discussion focused on the requirements for an instruction set to be virtualizable. An instruction set is virtualizable if it can implement all of the guest OS's actions that interact with hardware resources. The behavior of instructions should not change in user or privileged modes. Memory instructions should not depend on physical locations but rather virtual addresses. Also, instructions that change the visible state of the machine should always trap in user mode.

4.2 Disco

After establishing ISA requirements, the class discussed problems with virtualizing Disco. The virtual CPUs of Disco provide an abstraction of the MIPS R10000 processor to match the FLASH machine. This was a poor choice for Disco because the R10000 does not support the complete virtualization of the kernel virtual address space.

In the Disco paper([1]), eight copies of IRIX were run on Disco to utilize all the processing cores. In a CMP without a VMM, eight copies of IRIX could be run independently on eight processors. However, this relies on complete parallelization of the application, whereas Disco provides flexibility in resource usage. Partitioning the system into different virtual machines provides increased scalability.

Processor communication is achieved in Disco through the use of a normal network protocol. Normal network protocols are not designed for a CMP architecture where there is low-latency interconnect. So this situation requires specialized protocols which look similar to the standard protocols like TCP/IP but are optimized.

4.3 Hardware support for VMs

The next question centered on the hooks that should be provided by the hardware to run a VM. Since the overhead of a VM should be minimal, Disco runs as much as possible in user mode and only translates what is privileged. The ISA interface to the application is different from the implementation ISA, which leads to the co-design of virtual machines. This involves the use of interpreters, just-in-time compilation, and dynamic compilation. An essential part of this design is to use an OS that expects to be virtualized. Disco also uses hardware counters to keep track of cache misses, and can then move data to the desired local node.

Another point of discussion was the implementation of dynamic reconfiguration in reconfigurable architectures. In CMPs, it is possible to configure local memory (FIFOs, caches, scratchpad, etc.), the instruction set, interconnection networks, and the number of processors/memories to run programs.

4.4 Dynamic Profiling

A VM can provide dynamic allocation of resources because it can monitor both hardware and software utilization. The VM can monitor the number of threads and then assign the required number of processors. This type of a system may have the ability to perform dynamic profiling to adjust resources and utilization. This is a problem that cannot be solved statically by a compiler. Thread-level speculation is a method that is ideal for dynamic compilation because the run-time system can adjust the amount of speculation to match the needs of the executing program.

4.5 Overhead Issues

A chief concern in VMs is reducing overhead on the system. This can be done through automated mechanisms such as hardware counters. It is useful to have the hardware attempt to summarize the events of execution, as this reduces the price of profiling normally caused by the software. Also, the usefulness of a VM is questionable when warm-up time is high due to a constantly changing application. In these instances, the overhead of dynamic profiling may be too great to justify its expense. If profiling and updating are continuous, this problem may be alleviated.

4.6 Co-designed VMs

The co-design paper([2]) looked at the potential for improved performance and flexibility through co-operative design of the hardware and software for a VM. The design presented is similar to the Transmeta Crusoe, where the main goal was to maintain binary compatibility and exploit VLIW performance. Crusoe is essentially a VLIW machine with a VM to run x86 code.

To support a VM, hardware support is needed. Performance can improve through cache translations of commonly executed code. There is a significant problem with exceptions in that dynamic compilation re-orders program execution. Trace-caching is a tecnique of virtualizing in hardware. This is advantageous over software virtualization because of speed and a large instruction window when compared to a dynamic compiler.

4.7 Reliability/Fault Tolerance

Reliability and fault tolerance can be improved with VMs. Typically this can be done with the OS, but the VM is a smaller piece of code that is easier to manage. While a VM cannot help with hardware faults, it does provide a single point of failure which makes it possible to closely track faults.

References

[1] E. Bugnion, S. Devine and M. Rosenblum. Disco: Running Commodity Operating Systems on Scalable Multiprocessors. *Proceedings of 16th Symposium on Operating Systems Principles*, Saint Malo, France, October 1997.

[2] J.Smith, S. Sastry, T. Heil, T. Bezenek. Achieving High Performance via Co-designed Virtual Machines. *Proceedings of the International Workshop on Innovative Architecture*, Maui, HI, November 1999.