RC23694 (W0508-091) August 16, 2005
Computer Science

IBM Research Report

P.R.O.S.E. :
Partitioned Reliable Operating System Environment

Eric Van Hensbergen
IBM Research Division
Austin Research Laboratory
11501 Burnet Road
Austin, TX 78758

Research Division
Almaden - Austin - Beijing - Haifa - India - T. J. Watson - Tokyo - Zurich

P.R.O.S.E.
Partitioned Reliable Operating System Environment

Eric Van Hensbergen
(bergevan@us.ibm.com)
IBM Research

Abstract

This document re-evaluates the software stack in the light of para-virtualization technology and hypervisor support within next generation processors and operating systems. We describe an infrastructure enabling the use of logical partitions (LPARs) for the execution of stand-alone applications along side traditional operating systems. The design goal is to provide an environment allowing normal users to execute, interact and manage these custom kernels in much the same way they would with typical applications. The development environment is a set of modular component libraries providing necessary system services, and a familiar debug environment provided by exposing partition memory and control interfaces to a "controller" partition. We describe the implementation of our prototype using the IBM research hypervisor along with the Linux kernel and explore potential applications which could benefit from this new environment.

1. Introduction

Virtualization technology has existed since the early days of computer science [Singh04] , providing mechanisms to safely partition larger mainframes into discrete virtual machines. There has been much recent interest in applying virtualization technology to provide an easy means of partitioning commodity clusters within data centers to efficiently multiplex hardware resources while providing security and quality of service guarantees to customers. This has resulted in mainstream interest in having virtualization support incorporated into microprocessor design. While architectures such as the PowerPC have always provided some level of support for virtualization, recent chips from IBM, AMD, and Intel provide unprecedented support for system partitioning [Tsao04][VTwp][Pacifica].

These new hardware virtualization features support more efficient partitioning of system resources and create an opportunity to rethink the systems software stack. *Hypervisors*, the software agents which manage partitioning of the system, subsume a substantial portion of roles typically performed by an operating system. By embracing this fact, one can engineer the hypervisor to collaborate with operating systems in order to provide a more efficient virtualized environment. This approach is often called partial virtualization, or para-virtualization [Whitaker02]. Recent research projects such as *Xen* have shown minimal performance degradation from running in such para-virtualized environments for typical workloads even without extensive hardware support [Barham03].

Virtualization techniques have been primarily limited to providing private partitions allowing multiple operating systems (or multiple instances of the same operating system) to run simultaneously. If partitions communicate at all, they do so through virtual I/O devices such as emulated Ethernet controllers. While this isolation provides an excellent mechanism for server consolidation or the ability to run multiple environments on a desktop, it doesn't leverage the full potential of the technology.

We propose extending the use of hypervisors to provide partitions for application execution. By using a library-OS model combined with a unified resource sharing mechanism, we enable practical “stand-alone” application partitions. The rest of this paper summarizes our design goals, describes our prototype implementation, explores potential applications which could benefit from such an environment, and reports a preliminary evaluation on hardware for a sparse memory update benchmark.

2. Design

The hypervisor forms the foundation of the system software stack. It provides high-level memory, page table and interrupt management as well as partition scheduling policies.

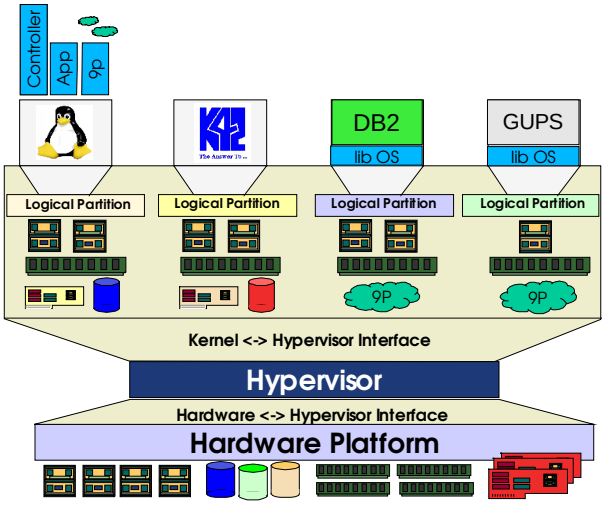

Illustration 1: Para-virtualized Software Stack

Our approach extends the idea of hosting multiple operating systems on a virtualized machine to using partitions to host applications. These applications can be services such as databases or they can be end-user applications such as the high performance computing challenge workloads [HPCC]. Since they essentially "own the virtual machine" there is an increased level of control and determinism during execution. They can also benefit from custom system components such as specialized schedulers and memory allocators. This approach has the same motivations and intent as *Exokernels* [Engler95], but leverages the more secure protection mechanisms of virtualization and enables interaction with legacy operating systems and drivers executing in other partitions.

Application partitions are executable from the command line of a traditional "controller" operating system and users interact with the application via familiar mechanisms such as standard I/O. Likewise, the application partition has full access to the controller's resources such as the file system via familiar library interfaces. To facilitate development and debug, partition memory and run-time control interfaces are accessible to the end-user on the “controller”. A library of system service building blocks such as thread support, memory allocators, page table management, and other components is provided to allow developers to build more complicated custom application kernels.

3. Implementation

We have based our prototype on the architecture employed by the IBM research hypervisor, *rHype*. rHype is a small (~10k lines of code), low-latency, modular, multi-platform (supports x86 and PowerPC) para-virtualization engine. Logical partitions (LPARs) access rHype services through a "system call" like mechanism known as a *hcall*.

A set of applications and scripts are used to create, manage, and destroy partitions. A special device can be memory mapped to provide a view into individual partition memory regions. Shared memory channels are established through static buffers compiled into the application kernel executables.

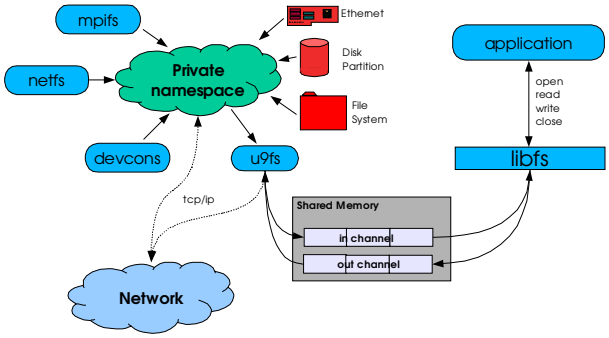

Illustration 2: Partition Resource Sharing

I/O to and from these application partitions is resolved by communication through these shared memory channels to the controlling partition. This same mechanism can be used to directly share devices such as disks or share system services such as networking stacks. In order to minimize complexity and code size, we chose Plan 9's 9P [9man] as a single unified resource sharing protocol.

During partition initialization, helper applications are executed to gateway system resources such as the console and IP stack via the 9P protocol. These resources are then mounted using the Linux 9P client [v9fs] to a private file name space. Another helper application, u9fs, is used to export the private name space through the shared memory channels to the application partition. Effort is underway to consolidate the gateways, v9fs, and export functionality into a single kernel module.

In order to create a more traditional user environment for these applications, a Linux binfmt module is used to automatically manage launching of the partition configuration utilities and helper applications. The binfmt modules are the portion of the Linux kernel which manage execution of different executable formats. We simply create a new executable format for application partitions and this mechanism together with the resource sharing infrastructure create a user environment that appears identical to normal applications – including standard I/O.

4. Application

We envision several potential applications for such an infrastructure. We are most interested in exploring the potential improvements by applying the hybrid model to databases, high performance computing, real-time and multimedia applications.

Applications with immense memory footprints, such as databases or Java virtual machines, typically attempt to manage their own memory and paging behavior. In many cases, operating system facilities and policies interfere with their ability to efficiently manage these resources. During certain operations, such as sparse access and update of data, page table miss penalties can dominate overall performance. Large page support provides some relief, but is not readily available in stock Linux environments and does not scale to the memory hierarchies of the near future. Virtualization allows these applications more explicit control of their memory hierarchy. Custom application partitions can even operate with virtual memory disabled, removing the overhead of page table operations and miss exceptions.

High performance computing (HPC) applications could also benefit from more explicit control over hardware resources. Like databases, they are characterized as having large data sets with sparse access patterns. Additionally, HPC workloads often employ custom programming and threading models. Such customizations benefit from being able to "push the operating system out of the way" and interact directly with the hardware.

When HPC applications run on a cluster of machines, tight timing constraints and short communication latencies are vital to overall system productivity and performance. Hypervisors can provide a very thin, low-latency control layer which can provide more deterministic scheduling policies.

The top-level hypervisor scheduler can be configured in such a way to isolate one partition from non-deterministic behavior of other partitions. The policy can be setup to allow a single partition to run to completion, favoring its execution over all others. These same policies can be used to guarantee frame rates in a game or meet latency constraints for an industrial control application.

5. Evaluation

We have performed a preliminary evaluation using identically configured IBM BladeCenter JS20 server blades. One of the blades runs the benchmark on top of a standard Linux kernel without a hypervisor. The second blade uses rHype in order to run Linux as a controller partition while running the application in its own partition.

The benchmark we are using is the DARPA HPC Discrete Math Benchmark, which does sparse access and update of a large table. We evaluated the application with a range of table sizes and recorded the amount of time it took to perform the updates.

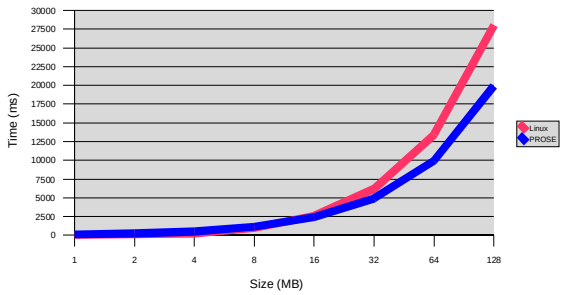

Illustration 3: Sparse Memory Update Performance

Our infrastructure starts out slightly slower than standard Linux, but then begins to outperform the standard implementation for larger table sizes. Poor performance at smaller table sizes is due to the overhead of running virtualized infrastructure – the system must context switch between the controller partition and the application. These partition context switches carry a higher overhead than the transition between user-space and kernel-space. This penalty is worst-case in our test configuration due to both partitions sharing a single CPU. We expect better results when controller Linux is modified to properly yield while idle and in multiprocessor configurations where I/O and application partitions can each be given their own CPU.

We take advantage of the fact that PowerPC virtualization technology allows us to run our application partition without virtual memory translation. By running in in *Real Mode* we avoid suffering any penalty for TLB misses resulting in superior performance as table sizes increase.

6. Future Work

We have only begun to evaluate the PROSE infrastructure with simple applications. Library support and evaluation must be expanded to more complicated multi-threaded and cluster workloads in order to evaluate the value of specialized OS components and I/O modules.

Our existing shared-memory I/O is somewhat primitive and can be vastly improved by leveraging higher performance communication mechanisms provided by the hypervisors. Another major design goal which remains to be implemented and evaluated is failure detection, recovery, and fail-over of the I/O channels which provide resources and services to the application partitions.

7. Conclusions

Commodity support for virtualization technology represents a paradigm shift in the computer industry. It can support more effective utility models of computing. It is prudent we explore how to restructure our system and application software stacks to take advantage of new virtualization technology being introduced into commodity hardware. It is our belief that cooperative virtual partitions will lead to more stable, secure, and productive computing environments.

8. Acknowledgment

This work would not be possible without the contributions of Jimi Xenidis, Michal Osterowski, Orran Krieger, and the rest of the rHype team. This work was supported in part by the Defense Advanced Research Projects Agency under contract no. NBCH30390004.

9. References

[Barham03] Xen 2002, Paul R. Barham, Boris Dragovic, Keir A. Fraser, and et al. , ucam-cl-tr-553, January 2003, University of Cambridge, Computer Laboratory.

[CellularDisco] Cellular disco: resource management using virtual clusters on shared-memory multiprocessors, Kingshuk Govil, Dan Teodosiu, Huang Yongqiang, and Mendel Rosenblum, 2000, ACM Transactions on Computer Systems, vol 18:3 , 229-262.

[devfs] Linux Devfs (Device File System FAQ), http://www.atnf.csiro.au/people/rgooch/linux/docs/devfs.html.

[Disco] Disco: Running Commodity Operating Systems on Scalable Multiprocessors, Edouard Bugnion, Scott Devine, Kinshuk Govil, and Mendel Rosenblum, 1997, ACM Transactions on Computer Systems, vol 15:4 , 412-447.

[Engler95] Exokernel: An operating system architecture for application-level resource management, Dawson R. Engler, M. Frans Kaashoek, and James O Toole, Jr., 1995, Proceedings 15th Symposium on Operating Systems Principles, 251-267.

[HPCC] High Performance Computing Challenge, http://icl.cs.utk.edu/hpcc/.

[Love03] Linux Kernel Development, Robert Love, 2003.

[9man] Plan 9 Programmer s Manual, Volume 1, AT & T Bell Laboratories, Murray Hill, NJ, 1995.

[Pacifica] AMD Virtualization Codenamed “Pacifica” Technology, Secure Virtual Machine Architecture Reference Manual, AMD, May 2005

[procfs] Linux Kernel Procfs Guide, http://www.kernelnewbies.org/documents/kdoc/procfs-guide/lkprocfs-guide.html.

[Singh04] An Introduction To Virtualization, Amit Singh, http://www.kernelthread.com/publications/virtualization, 2004.

[Tsao04] Server Consolidation Using POWER5 Virtualization White Paper, H. Tsao and B. Olszewski, http://www- 1.ibm.com/servers/eserver/pseries/hardware/whitepapers/570_serverconsol.html , 2004.

[VTwp] Enhanced Virtualization on Intel Architecture-based Server, Intel Solutions White Paper, March 2005.

[Whitaker02] Denali: Lightweight virtual machines for distributed and networked application, A. Whitaker, M. Shaw, and S. Gribble, 2002, Proceedings of the USENIX Annual Technical Conference.