

Application Specific Linux (ASL) *

Lamia Youseff Rich Wolski Chandra Krintz

Department of Computer Science
University of California, Santa Barbara
{lyouseff, rich, ckrintz}@cs.ucsb.edu

1. SUMMARY

Linux has emerged as the system-of-choice in academic and production scientific computing settings. A key limitation to the use of Linux for high-end cluster computing however is its potential performance impact on application execution, since Linux dictate general policies that do not promote the performance of high-end scientific applications.

In this abstract, we are presenting our Application Specific Linux (ASL), a customized Linux image that enhances the performance of the scientific applications. Our Research end-goal is a software system that automatically enables high performance scientific computing on commodity systems through application-specific customization and dynamic adaptation of the Linux OS. Our research is novel in that it combines the research done in OS specialization, extensibility and minimization as well as virtual machine monitors (VMMs) into a system that automatically customizes the kernel image for a single application run and is specifically focused on the application domain of scientific computing using high-performance clusters.

2. EXTENDED ABSTRACT

Recent advances in high-performance processor and network technologies are making clusters of workstation class computers cost-effective platforms that can support the next generation of scientific applications. Low per-unit cost, advances in computing and communication power, and the availability of Linux as a free, easy-to-use, and nearly standard operating system, make high-end computing with these systems accessible both to a very large developer base and to a wide range of users. As part of this evolution, Linux has emerged as a nearly ubiquitous, open-source operating system with a wide-range of readily available programming support tools and specialized libraries. It is currently the system-of-choice in academic and production scientific computing settings and as a result, many -if not the majority of- scientific programmers, being trained today are familiar with Linux as a development platform. Moreover, its ability to be used as a locally controlled, high responsive development environment has greatly increased the programmer productivity, hence the advancement of science.

A key limitation to the use of Linux for high-end cluster computing however is its potential performance impact on application execution [3]. Linux, like other general-purpose operating systems (GPOS) with commercial application, continues to evolve to support an enormous range of user requirements, application domains,

and devices (everything from supercomputers to hand-helds). In contrast, scientific applications executing in clustered settings are frequently large, resource intensive, long-running, and use space-sharing to gain exclusive access to the machines they use through a batch system. They do not compete dynamically for processor and I/O resources like many other application domains. Therefore, the Linux OS includes many features and built-in policies that do not promote the performance of high-end scientific applications, which can retard the performance of scientific programs in high-end computing settings. In particular, scientific applications typically do not require the extensive support for fair resource sharing (since they execute in production space-shared, and not time-shared, environments) or quick response time (since they may not be interactive). None the less, the portability that Linux affords combined with the familiarity that its wide-spread popularity has bred make it a de facto standard operating system for clustered architectures.

In our Research, we are designing Application Specific Linux (ASL) to enhance the performance of Linux for scientific applications[4]. The goal of our research is to investigate techniques that maintain the ease-of-use and cost benefits of Linux while enhancing the performance achievable by high-end scientific applications executing in large-scale cluster computing settings. To enable this, we are studying ways to automatically customize the Linux instance an application uses when it is running in a “production” (i.e., non-development or debugging) setting based on the specific needs of the application itself. We are also exploiting the exclusive processor access that batch-scheduling implements to relax or eliminate unneeded mechanisms that are designed to facilitate effective time-sharing, but which introduce unnecessary overhead in a space-sharing context.

We are using both runtime and compile-time approaches to customize the Linux instance used by each application. Each of which will allow the scientific programmer to use unmodified Linux as a local development and debugging environment and then to apply our techniques as an additional compilation step before initiating high-end cluster execution.

In order to enable Linux customization, our system consists of four logical customization phases. Static and dynamic application analysis is our system’s first phase. The application as well as its potential coupling effect on the kernel is studied using Phases profiling techniques as well as kernel performance-annotated call graphs. The second phase is the static customization enhancements for the kernel image based on the outcome from the first phase, which is produced at the development site. For example, we studied in [7] the I/O pattern for MIT General Circulation Model

*This work is sponsored in part by grant from the National Science Foundation (ST-HEC-0444412).

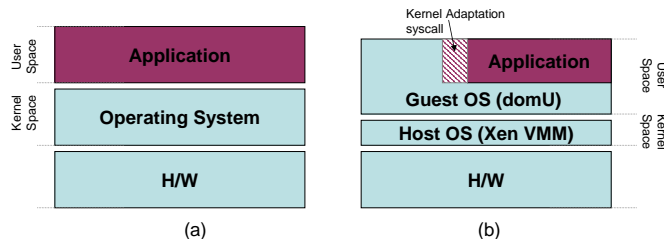


Figure 1: Deployment model for current scientific applications in (a) versus our deployment model using Xen VMM in (b).

(GCM) [5], which is a popular numerical simulation used by scientists to study oceanographic and climatologic phenomena. For this specific MIT GCM code I/O pattern, the `sys.write()` syscall was modified to enhance the application performance. The static customization phases furthermore include inlining the application code inside the kernel image to reduce the user to kernel space crossing overhead, as well as inlining some kernel modules and creating execution shortcuts in the kernel for enhanced performance of the common execution scenarios for the application, as shown in figure 1(b). The customized Linux image (including the application) is then shipped to the production environment.

The third phase takes place at the production environment, where ASL is deployed on a minimal virtual machine monitor (VMM), as shown in figure 1(b). This approach allows us also to introduce unsafe kernel customization while protecting volatile hardware resources (e.g. BIOS code). In order to ensure the feasibility of our approach, we have evaluated the performance implication of running HPC performance-oriented codes on virtual machines in [6]. We opt to deploy paravirtualized VM, in which the guest OS is aware of being virtualized. Paravirtualization is characterized by its minimal performance degradation on application execution, relative to full virtualization. Figure 2 shows the performance comparison between four kernels for several NAS [1] Parallel Benchmark (NPB) codes, on a 16 processor cluster. CHAOS [2] is an HPC performance oriented kernel developed at Lawrence Livermore National Lab (LLNL). Xen kernel is a paravirtualized 2.6.12 kernel. The two RHEL kernels are off-the-shelf Red Hat Enterprise Linux version 2.6.9 and 2.9.12 respectively. In this figure, the y -axis shows the performance of different codes on the x -axis, relative to Livermore’s CHAOS kernel. Using the averages illustrated here and several statistical techniques, we were able to empirically prove minimal performance degradation for paravirtualized systems. A comprehensive performance evaluation of paravirtualization in HPC should be found in [6].

During the execution of the application, the ASL is self evolving and dynamically adopting system which keeps modifying itself to meet the application’s requirements at runtime and enhance its performance. Meanwhile, ASL provides a kernel adaptation system call that the user can interact with to suggest dynamic adaptation of the system during runtime. This dynamic customization constitute our final customization phase.

Much prior work in the area of application-specific operating systems (OSs) has thoroughly studied extensibility, specialization, and minimization of the OS in general. However, our research is novel in that it combines and extends these efforts into a system that automatically customizes the Linux operating system for a single appli-

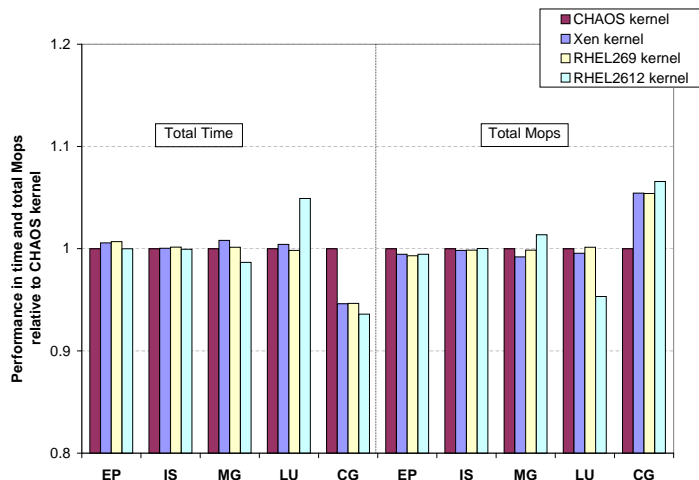


Figure 2: NAS Parallel Benchmark performance relative to CHAOS. The left half (first benchmark set) is for total time (lower is better); the right half is for Mops (higher is better).

cation run and is specifically focused on the application domain of scientific computing using high-performance clusters. At the same time, our use of Linux ensures that the environment for scientific applications developers remains familiar and unified across the development and production platforms they use; thereby promoting ease-of-use, programmer productivity, and efficient program management. Our Research’s end-goal is a software system that automatically enables high performance scientific computing on commodity systems through application-specific customization and dynamic adaptation of a low-cost, popular, and familiar Linux operating system.

9

3. REFERENCES

- [1] D. Bailey, T. Harris, W. Saphir, R. van der Wijngaart, A. Woo, and M. Yarrow. The NAS Parallel Benchmarks 2.0. *The International Journal of Supercomputer Applications*, 1995.
- [2] CHAOS Project. <http://www.cs.umd.edu/projects/hpsl/ResearchAreas/PerfPred.htm>.
- [3] R. Gioiosa, F. Petrini, K. Davis, and F. Lebaillif-Delamare. Analysis of System Overhead on Parallel Computers. In *The 4th IEEE International Symposium on Signal Processing and Information Technology (ISSPIT 2004)*, Roma, Italy, December 2004. Available from <http://www.c3.lanl.gov/~fabrizio/papers/isspit04.pdf>.
- [4] C. Krintz and R. Wolski. Using Phase Behavior in Scientific Application to Guide Linux Operating System Customization. In *Workshop on Next Generation Software at IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, April 2005.
- [5] J. Marotzke and R. G. et al. Construction of the adjoint MIT ocean general circulation model and application to Atlantic heat transport sensitivity. *Journal of Geophysical Research*, 104(C12), 1999.
- [6] L. Youseff, R. Wolski, B. Gorda, and C. Krintz. Paravirtualization for HPC Systems. In *Under submission to SuperComputing (SC 06)*, 2006.
- [7] L. Youseff, R. Wolski, and C. Krintz. Linux Kernel Specialization for Scientific Application Performance. Technical Report UCSB Technical Report 2005-29, Univ. of California, Santa Barbara, Nov 2005.