High Performance Cloud Computing is Nix

Francisco J. Ballesteros
Noah Evans
Charles Forsyth
Gorka Guardiola
Jim McKie
Ron Minnich
Enrique Soriano

Bell Laboratories
2018 Antwerp, Belgium

Abstract

Cloud computing uses virtualisation on shared hardware to provide each user with the appearance of having a private system of their choosing running on dedicated hardware. Virtualisation enables the dynamic provisioning of resources with demand, and the possibilities of entire system backup, migration, and persistence; for many tasks, the flexibility of this approach, coupled with savings in cost, management, and energy, are compelling.

However, certain types of tasks — such as compute-intensive parallel computations — are not trivial to implement as cloud applications. For instance HPC tasks consist of highly interrelated subproblems where synchronization and work allocation happen at fixed intervals. Interrupts from the hypervisor and the operating system add "noise" and, thereby, random latency to tasks, slowing down all the other tasks by making them wait for such slowed-down tasks to finish. The effect may cascade and seriously disrupt the regular flow of a computation.

To avoid latency issues, HPC tasks are performed on heavily customized hardware that provides bounded latency. Unlike cloud systems, these systems are typically not time-shared; instead of the illusion of exclusivity, individual users are given fully private allocations for their task running as a single-user system. However programming for single user systems is difficult, users prefer programming environments that mimic the time-sharing environment of their desktop. This desire leads to conflicting constraints between the desire for a convenient and productive programming environment and the goal of maximum performance. This conflict has lead to an evolution of HPC operating systems towards providing the full capabilities of a commodity time-sharing operating system, the IBM Compute Node Kernel, a non-Linux HPC kernel, has changed in the last ten years to become more and more like Linux. On the other hand, Linux systems for HPC are increasingly pared down to the minimal subset of capabilities in order to avoid timesharing degradation at the cost of compatiblity with the current Linux source tree. This convergent evolution has lead to an environment where HPC kernels sacrifice performance for compatiblity with commodity systems while Commodity systems sacrifice compatibility for performance, leaving both issues fundamentally unresolved.

We offer a solution that bridges this gap between performance and expressivity, providing bounded latency and maximum computing power on one hand and the rich programming environment of a commodity OS on the other. Based on the reality of coming many-core processors, we provide an environment in which users can be given dedicated, non-preemptable cores on which to run; and in which, at the same time, all the services of a full-fledged operating system are available, this allows us to take the lessons of HPC computing, bounded latency and exclusive use, and apply them to cloud computing. As an example of this approach we present NIX, a prototype operating system for future manycore CPUs. Influenced by

our work in High Performance computing, both on Blue Gene and more traditional clusters, NIX features a heterogeneous CPU model and a change from the traditional Unix memory model of separate virtual address spaces. NIX partitions cores by function: Timesharing Cores, or TCs; Application Cores, or ACs; and Kernel Cores, or KCs. There is always at least one TC, and it runs applications in the traditional model. KCs are cores created to run kernel functions on demand. ACs are entirely turned over to running an application, with no interrupts; not even clock interrupts. Unlike traditional HPC Light Weight Kernels, the number of TCs, KCs, and ACs can change as needs change. Unlike traditional operating systems, applications can access services by sending a message to the TC kernel, rather than by a system call trap. Control of ACs is managed by means of intercore-calls. NIX takes advantage of the shared-memory nature of manycore CPUs, and passes pointers to both data and code to coordinate among cores.

The full paper presents a description and benchmark evaluation of NIX under standard HPC workloads.