A Virtual Machine Monitor for Utilizing Non-dedicated Clusters

Kenji Kaneda
University of Tokyo
7-3-1 Hongo Bunkyo-ku
Tokyo, Japan
kaneda@yl.is.s.u-tokyo.ac.jp

Yoshihiro Oyama
University of Tokyo
7-3-1 Hongo Bunkyo-ku
Tokyo, Japan
oyama@yl.is.u-tokyo.ac.jp

Akinori Yonezawa
University of Tokyo
7-3-1 Hongo Bunkyo-ku
Tokyo, Japan
yonezawa@yl.is.s.u-tokyo.ac.jp

ABSTRACT

We have designed and implemented a virtual machine monitor (VMM) for utilizing non-dedicated clusters. The VMM virtualizes a shared-memory multi-processor machine on a commodity cluster. In addition, it hides dynamic changes of physical hardware configurations. The experimental result demonstrates the feasibility of our approach.

Categories and Subject Descriptors

D.4.1 [**Operating Systems**]: Process Management; D.4.7 [**Operating Systems**]: Organization and Design; D.4.8 [**Operating Systems**]: Performance

General Terms

Design, Measurement, Performance

Keywords

Virtual machine monitors, single system image, distributed systems

1. INTRODUCTION

It is difficult to efficiently utilize non-dedicated commodity clusters. Since resources of non-dedicated clusters are shared by multiple users, the quality as well as quantity of the resources available to an application changes constantly. Although it becomes important for parallel programs to adapt to such dynamic changes, writing adaptive parallel programs still requires large efforts. As a result, dynamic change of resource availability is one of great burdens of the wide deployment of clusters.

To address this problem, we have been developing a virtual machine monitor (VMM) for utilizing non-dedicated commodity clusters. Like existing VMMs, our VMM takes complete control of the machine hardware and creates virtual machines, each of which behaves like a complete physical machine that can run its own operating system. In contrast to the existing VMMs, our VMM has two distinguishing features. First, the VMM virtualizes a shared-memory multi-processor machine on a commodity cluster. For example, it gives users the illusion of an N-way multi-processor machine on top of N single-processor machines. Second, the VMM hides dynamic changes of physical hardware configurations. It allows a virtual machine to provide a fixed number of processors even if available physical machines are added and/or removed dynamically (See Figure 1). For example, an application running inside a virtual machine sees N processors all the time even if the number of available physical machines decreases and becomes less than N.

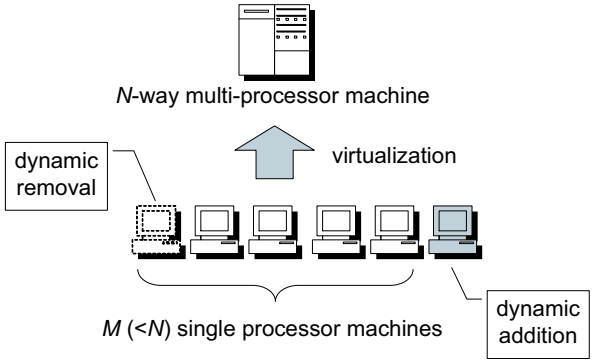

Figure 1: Creation of a virtual multi-processor machine: the number of processors of a virtual machine is constant regardless of dynamic addition and/or removal of physical machines.

The above functionality of our system greatly simplifies utilization of non-dedicated clusters. It enables parallel applications to achieve high performance with no modification of their source code. For example, our system enables parallel applications (e.g., parallel `make`) normally installed on multi-processor machines to run on clusters and adapt to dynamic changes in their execution environments.

2. IMPLEMENTATION OF THE VMM

Our VMM is designed for the x86 architecture. It virtualizes processors, shared memory, and I/O devices as follows. To virtualize processors, the VMM supports paravirtualization of the instruction set architecture [1, 4]. A guest operating system is statically modified to run opti-

SOSP'05, October 23–26, 2005, Brighton, United Kingdom.

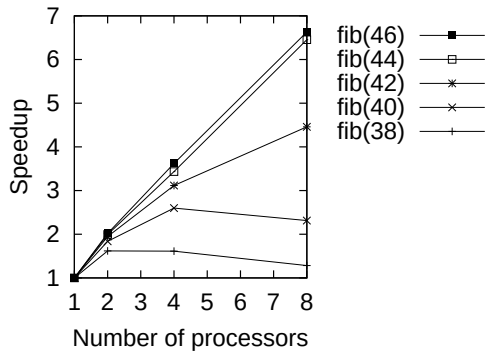

Figure 2: Speedup of parallel fibonacci

mally in a virtual machine. To provide a fixed number of virtual processors, the VMM maps one or more virtual processors to a physical processor and changes the mapping dynamically. Since the dynamic mapping may cause asymmetric speeds of virtual processors, the time ballooning technique [8] is used for load balancing.

To virtualize shared memory, the VMM uses a mechanism similar to software distributed shared memory. The VMM implements the consistency protocol of the shared memory with the virtual memory page protection mechanism of physical machines. The current consistency algorithm is based on that of Ivy [6].

To virtualize I/O devices, the VMM prepares a central server that keeps track of the states of all the devices. The VMM communicates with the server whenever a virtual processor issues an I/O operation.

3. CURRENT STATUS

We implemented a prototype of the VMM and conducted experiments. The current implementation builds a virtual 8-way multi-processor machine on top of eight physical machines and hosts Linux kernel 2.4 for SMP. We ran coarse-grain parallel tasks inside the virtual machine and measured the execution time to demonstrate the feasibility of our approach. Specifically, we ran eight processes that calculate a fibonacci number simultaneously on a 1-way, ..., 8-way multi-processor machine built on top of 1, ..., 8 physical machines respectively.

Figure 2 shows the experimental result. `fib(n)` denotes the calculation of `n`th fibonacci number. As shown in this figure, the program achieved better speedup as tasks were coarser. For example, the execution of `fib(46)` with an 8-way multi-processor machine is about 6.6 times faster than that with a 1-way processor machine.

4. RELATED WORK

vNUMA [2] virtualizes a cc-NUMA machine on top of physical machines with the Itanium architecture. The major difference between our system and vNUMA is that vNUMA does not support dynamic addition and/or removal of physical machines.

Virtual Iron [9] builds a virtual multi-processor machine on top of clusters. The virtual machine tolerates dynamic changes of physical hardware configurations. The basic mechanism of Virtual Iron is similar to that of our system. A comparison between Virtual Iron and our system has not been made yet since details of Virtual Iron are not yet public (2005/10/14).

5. FUTURE WORK

We plan a number of extensions and improvements to our system. For example, we plan to implement a fault tolerance mechanism. Since a machine crash is a frequent event in commodity clusters, in which a large number of machines involve, we require that the system can continue to run even if some machines fail. The implementation of the fault tolerance mechanism will be based on techniques such as the checkpointing/recovery [3] and replication for VMMs [7].

We also plan to improve the performance of the memory consistency algorithm. For example, according to the specification of the IA-32 memory model [5], an individual processor can delay the propagation of its writes to remote processors' memory until the processor executes a synchronous instruction or an atomic instruction.

In addition, we plan to evaluate our system using real-world applications such as SPLASH-2 and Apache with more than eight machines.

More information can be found at `http://www.yl.is.s.u-tokyo.ac.jp/~kaneda/vmp/`.

6. REFERENCES

[1] 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. In *Proc. of SOSP*, pages 164–177, 2003.

[2] M. Chapman and G. Heiser. Implementing Transparent Shared Memory on Clusters Using Virtual Machines. In *Proc. of the USENIX Annual Technical Conference*, pages 383–386, 2005.

[3] E. N. (Mootaz) Elnozahy and Lorenzo Alvisi and Yi-Min Wang and David B. Johnson. A Survey of Rollback-recovery Protocols in Message-passing Systems. *ACM Computing Surveys (CSUR)*, 34(3):375–408, 2002.

[4] H. Eiraku and Y. Shinjo. Running BSD Kernels as User Processes by Partial Emulation and Rewriting of Machine Instructions. In *Proc. of BSDCon*, pages 91–102, 2003.

[5] Intel Corporation. *IA-32 Intel Architecture Software Developer's Manual Volume 3: System Programming Guide*, 2003.

[6] Kai Li and Paul Hudak. Memory Coherence in Shared Virtual Memory Systems. *ACM Trans. on Computer Systems (TOCS)*, 7(4):321–359, 1989.

[7] Thomas C. Bressoud and Fred B. Schneider. Hypervisor-based fault tolerance. *ACM Trans. on Computer Systems (TOCS)*, 14(1):80–107, 1996.

[8] V. Uhlig, J. LeVasseur, E. Skoglund, and U. Dannowski. Towards Scalable Multiprocessor Virtual Machines. In *Proc. of VM*, pages 43–56, 2004.

[9] Virtual Iron Software. `http://www.virtualiron.com/`.