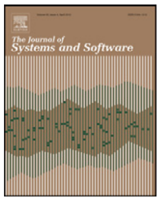
The Journal of Systems and Software 86 (2013) 367–376

Contents lists available at SciVerse ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss



A high performance inter-domain communication approach for virtual machines

Yuebin Bai[a,c,*], Yao Ma[a], Cheng Luo[b], Duo Lv[a], Yuanfeng Peng[a]

[a] State Key Laboratory of Software Development Environment, Beihang University, Beijing 100191, China
[b] Department of Computer Science, The University of Tokyo, Tokyo, Japan
[c] Science and Technology on Information Transmission and Dissemination in Communication Networks Laboratory, Shijiazhuang 050081, China

ARTICLE INFO

Article history:
Received 28 September 2011
Received in revised form 15 August 2012
Accepted 23 August 2012
Available online 1 September 2012

Keywords:
Virtualization
Inter-VM communication
Overhead
Communication path

ABSTRACT

In virtualization technology field, researches mainly focus on strengthening the isolation barrier between virtual machines (VMs) that are co-resident within a single physical machine. At the same time, there are many kinds of distributed communication-intensive applications such as web services, transaction processing, graphics rendering and high performance grid applications, which need to communicate with other virtual machines at the same platform. Unfortunately, current inter-VM communication method cannot adequately satisfy the requirement of such applications. In this paper, we present the design and implementation of a high performance inter-VM communication method called IVCOM based on Xen virtual machine environment. In para-virtualization, IVCOM achieves high performance by bypassing some protocol stacks and privileged domain, shunning page flipping and providing a direct and high-performance communication path between VMs residing in the same physical machine. But in full-virtualization, IVCOM applies a direct communication channel between domain 0 and Hardware Virtualization based VM (HV^2M) and can greatly reduce the VM entry/exit operations, which has improved the HV^2M performance. In the evaluation of para-virtualization consisting of a few of benchmarks, we observe that IVCOM can reduce the inter-VM round trip latency by 70% and increase throughput by up to 3 times, which prove the efficiency of IVCOM in para-virtualized environment. In the full-virtualized one, IVCOM can reduce 90% VMX transition operations in the communication between domain 0 and HV^2M.

1. Introduction

Virtual machines (VMs) technology was introduced in the 1960s, and reached prominence in the early 1970s. It can create virtual machines which can provide function and performance isolation across applications and services that share a common hardware platform. At the same time, VMs can improve the system-wide utilization efficiency and provide lower overall operation cost of the system. With the advent of low-cost minicomputers and personal computers, the need for virtualization declined (Figueiredo et al., 2005). As growing interest in improving the utilization of computing resources through server consolidation, VM technology is regaining the spotlight again and is widely used in many fields. Now, there are many virtual machine monitors (VMMs) such as VMware (Sugerman et al., 2001), Virtual PC, UML, and Xen (Barham et al., 2003). Among them, Xen develops a technique known as para-virtualization (Whitaker et al., 2002), which offers virtualization with low-overhead and has attracted much attention from both the academic VM and the enterprise market. However, para-virtualized approach has its intrinsic shortcoming that it has to modify the OS kernel on it to recuperate the processor's virtualization holes. To implement full-virtualization (Adams and Agesen, 2006) on x86 platform, some processor manufacturers propose their own hardware assisted technologies to support full virtualization such as Intel's VT technology, and AMD's Pacifica technology.

In spite of the recent advance in the VM technology, virtual network performance remains a major challenge (Menon et al., 2006). Some researches done by Menon et al. (2005) show that Linux guest domain has far lower network performance than native Linux in the scenarios of inter-VM communication. The communication performance between two processes in their own VMs on the same physical machine is even worse than expected, which is mainly due to the virtualization technology's central characteristic of isolation. For example, a distributed HPC application may have two processes running in different VMs that need to communicate using messages over MPI libraries. Another example is network transaction. In order to satisfy a client transaction request, a web service running in one VM may need to communicate with a database server which is running in another VM. Even routine inter-VM communication, such as file transfers, may need to cross the isolation barrier frequently. In the examples above, we would like

* Corresponding author. Tel.: +86 10 8233 9020.
E-mail addresses: byb@buaa.edu.cn (Y. Bai), mayao@cse.buaa.edu.cn (Y. Ma).

0164-1212/$ – see front matter © 2012 Elsevier Inc. All rights reserved.
http://dx.doi.org/10.1016/j.jss.2012.08.054

to use a direct and high performance communication mechanism, which can minimize the communication latency and maximize the throughput.

In this paper, basing on the research about virtualization technology and multi-core technology, we combine the two technologies and propose a high performance inter-VM communication method. The rest of this paper is organized as follows: Motivation of this work is described in Section 2. Section 3 gives a brief view of Xen network background. Section 4 presents the related work. Section 5 presents the design and implementation of inter-VM communication method (IVCOM) as well as the extension to hardware-based virtual machine. Section 6 discusses the overhead and the detailed performance evaluation of IVCOM in para-virtualization. Section 7 shows the experiments in full-virtualization. Section 8 draws the conclusion.

2. Motivation

While enforcing isolation is an important requirement from the viewpoint of security of individual software components, it also can result in significant communication overhead because different software components also need to communicate with each other across the isolation barrier to achieve application objectives in specific scenarios. We take Xen as an example, and analyze it to find out the reason for communication performance loss.

Xen is an open source hypervisor running between hardware and operating systems (OS). It virtualizes all resources over the hardware and provides the virtualized resources to OS running on Xen. Each OS is called a guest domain or domain U, and the only one privileged OS for hosting the application level management software is called domain 0. Now, Xen supports two virtualization ways: full-virtualization and para-virtualization. It can provide virtual machine performance close to a native one by para-virtualization.

In para-virtualization, Xen exports virtualized network devices to each domain U replacing the actual network drivers that can interact with the real network card within domain 0. Domain 0 communicates with Domain U by means of a split network driver architecture shown in Fig. 1.

The domain 0 hosts the backend of the split network driver called netback, and the domain U hosts the frontend called netfront. They interact by using high-level network device abstraction rather than low-level network hardware specific mechanisms. The split drivers communicate with each other through two producer–consumer ring buffers. The ring buffers are a standard lockless shared memory data structure built on grant table and event channels which are two primitives in Xen architecture.

The grant table can be used to share pages between domain U and domain 0. The frontend of the split driver in domain U can notify Xen hypervisor that a memory page can be shared with domain 0. Domain U then passes a grant table reference through the event channel to domain 0 that copies data to or from the memory page of domain U. Once completing the page access, domain U removes the grant reference. Page sharing is useful for synchronous I/O operations such as sending packets through a network device. Meanwhile, domain 0 may not know the destination domain for an incoming packet until the entire packet has been received. In this situation, domains 0 will first DMAs the packet into its own memory page. Then, domain 0 can choose to copy the entire packet to the domain U′ memory. If the packet is large, domain 0 will notify Xen hypervisor that the page can be transferred to the target domain U. The domain U then initiates a transfer of the received page and returns a free page back to hypervisor for the next one.

In full-virtualization, Intel VT technology defines two modes: root mode and non-root mode for virtual machines. In root mode, virtual machine has the whole privilege and full control of the processor(s) and other platform hardware. In non-root mode, virtual machine can only operate with limited privilege. Corresponding to two modes, VT provides a new form of processor operation called virtual machine extension (VMX) operation. There are two kinds of VMX operation: VMX root operation and VMX non-root operation. In general, a VMM will run in VMX root operation and guest software will run in VMX non-root operation. The transition between VMX root operation and VMX non-root operation is called VMX transition. There are two kinds of VMX transitions. Transition into VMX non-root operation is called VM entry. Transition from VMX non-root operation to VMX root operation is called VM exit.

In Xen, domain 0 runs in root mode and Hardware Virtualization based VM (HV^2M) runs in non-root mode. When applications in HV^2M need to access hardware resources such as IO devices, VMX transition will occur. First of all, the current running scene will be saved into a virtual machine control structure (VMCS), and the root mode scene will be load from it. By this way, the HV^2M domain is scheduled out and domain 0 is scheduled in. Then it is time for handling the real IO request by device module. After that, domain 0 is scheduled out and the domain switches back to HV^2M. After analysing the process of VMX transition, we can find the actual IO handle cost only take little part in the total overhead in one switch. As all privileged access such as IO access or interrupt will be handled in this way, the performance of HV^2M degrades about 10–30%.

By detailing the network communication process between VMs in para-virtualization, we can find that there is no direct communication channel between guest VMs and all communications need to be forwarded by domain 0 which decrease both the performance of the communication between guest VMs and the performance of domain 0.

In full-virtualization, the communication between VMs under different modes is not smooth. Ideally, we would like that the communication between VMs on the same physical machine should be simple and direct, and has high performance. Therefore, we propose a new communication method that can set up direct communication channel between VMs.

3. Related work

There have been some researches to improve the inter-VM communication performance in virtualization environment. For example, XWay (Kim et al., 2008), XenSockets (Zhang et al., 2007) and IVC (Huang et al., 2007) have developed tools that are more efficient than traditional communication path that needs to via domain 0. XWay provides transparent inter-VM communication for TCP-oriented applications by intercepting TCP socket calls beneath the socket layer. It requires extensive modifications to the implementation of network protocol stack in the core OS, since Linux does not seem to provide a transparent netfilter-type hooks to intercept messages above TCP layer. XenSockets is a one-way communication pipe between two VMs, which is based on shared memory. It defines a new kind of socket, with associated connection establishment and read–write system calls that provide interface to the underlying inter-VM shared memory communication mechanism. In order to use these calls, user applications and libraries need to be modified. XenSockets is suitable for applications that are high throughput distributed stream systems, in which latency requirement is relaxed, and that can perform batching at the receiver side. IVC is a user level communication library intended for message passing HPC applications. It can provide shared memory communication across VMs that reside within the same physical machine. It also provides a socket-style user-API, through which an IVC aware application or library can be written. IVC is beneficial for HPC applications that can be modified to explicitly use the IVC API. In other