Netbus: A Transparent Mechanism for Remote Device Access in Virtualized Systems

Sanjay Kumar, Sandip Agarwala, Karsten Schwan
College of Computing, Georgia Institute of Technology
{*ksanjay, sandip, schwan*}*@cc.gatech.edu*

1 Introduction

Virtual Machine Monitors (VMMs) (e.g., Xen and VMWare) support the creation and execution of multiple virtual machines (VMs) on the same platform, and they enforce the isolation properties necessary to make the underlying shared platform resources appear exclusive to each VM. Toward these ends, VMMs export virtual instances of physical resources to VMs and they offer secure methods for sharing them. For I/O devices, such methods include time-sharing, space-sharing, and exclusive use. This paper argues the importance of VMM-level support for transparent access to remote devices. By decoupling device locations from the VMs that access them, we enable device consolidation, flexibility in VM configurations and seamless VM migration. By enabling virtual device migration, such support will make it easier to develop the load balancing methods envisioned for next generation datacenters. VMM-level support for access to remote devices provides complete *transparency* in accessing remote vs. local devices, at a level of abstraction not visible to guest operating systems. Specifically, since the hypervisors or VMMs that control the hardware platform already virtualize the platform's physical resources, it becomes possible to extend their per-platform methods for device virtualization into *remoting* methods that make the physical locations of devices entirely transparent to guest operating systems. Toward this end, this paper briefly describes a new abstraction termed *Netbus*, which provides VMs with transparent access to remote devices. For full description, please refer to [1]. To support seamless VM migration, it describes *virtual device migration* mechanism built using Netbus, which not only enables a VM to continuously access its IO devices, including remotely after migration, but also, to seamlessly hot-swap devices, to replace remote with local devices whenever indicated or necessary, without any noticiable downtime.

Netbus currently targets LAN-centric environments with single administrative domain that offer strong connectivity, high levels of cross-machine network bandwidth and low network latency (e.g., datacenters, home and office LANs etc.). The Netbus architecture utilizes the underlying device virtualization mechanism of the VMM to extend the per platform mechanism over the network.

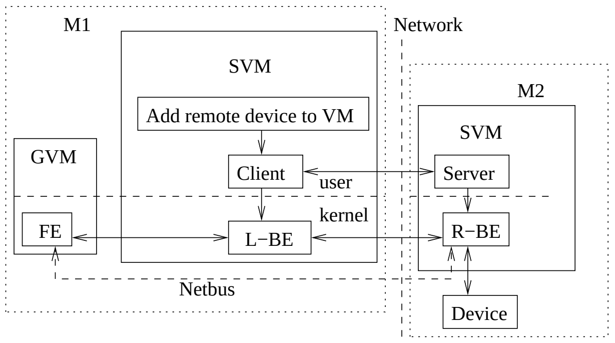

Figure 1: Netbus Software Architecture

2 Netbus Software Architecture

The following exposition of Netbus assumes a VMM using the split device driver stacks (Frontend(FE)-Backend(BE) communication mechanism), as implemented in paravirtualized Xen VMs. The software architecture of Netbus for FE/BE-based VMM implementations is depicted in Figure 1. To initialize a device, the Netbus server running in the SVM on host M2 having the device, exports it (i.e., publishes it) across the network to a designated host or a set of hosts if the device is to be shared. When a remote device is being added to guest VM G1 running on host M1, the Netbus client in SVM establishes a connection with the Netbus server and executes required authentication and authorization

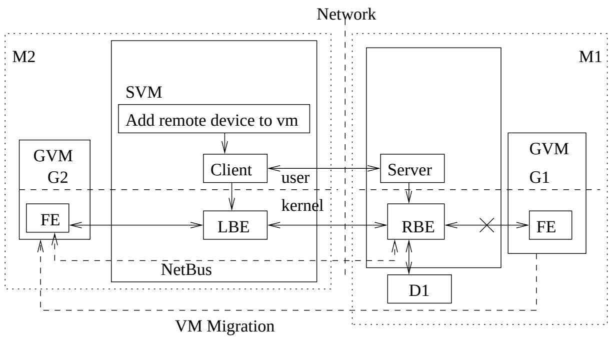

Figure 2: Virtual Device Migration

actions. When those succeed, client and server both inform their respective BEs about this connection information and henceforth, all communications between the BE drivers on both machines are carried out via this one-to-one connection. The local BE (LBE) makes the virtual device appear inside guest VM where it gets initialized by its respective driver. When the FE from the guest VM accesses the device by making requests to its corresponding LBE driver, the LBE forwards this request to the remote host's BE (RBE). The RBE then makes the actual request to the device and returns the response to the LBE. The LBE in turn returns the response to the FE. The RBE also provides asynchronous device events (e.g., interrupts) to the LBE which forwards them to the FE. This mechanism effectively provides an abstraction of a *network bus* over which the FE can interact with a Remote BE (RBE). While the above description assumes a para-virtualized FE-BE mechanism for I/O device virtualization, the Netbus architecture itself is not restricted to such an environment. It can also be applied to a fully-virtualized case, where software I/O device emulation is used to provide virtual devices to guest VMs.

3 Using Virtual Device Migration

Netbus enables live VM migration of GVMs, with continuous access to its devices. This mechanism is termed as *virtual device migration* since logically the virtual device is also migrated along with the VM. Virtual Device Migration is depicted in Figure 2. Assume that a guest VM named G1 is migrating from host machine M1 to host machine M2, while G1 is accessing a device D1 on M1. During VM migration, G1 is frozen and the FE-BE communication is suspended. During this suspension, BE (RBE) breaks its connection with the FE, but it does not break its connection with the device. On the destination host M2, new VM G2 is created, and its OS pages are filled from the suspended VM G1. G2 uses G1's configuration so that it exactly looks like G1. During G2's creation, however, its configuration is modified with respect to device D1, so that D1 appears as a remote device in G2. In this fashion, Netbus ensures the establishment of a valid communication channel between the two BEs (LBE and RBE). Next, just before the migrated VM G2 is un-paused on host M2, its FE establishes a connection with the local backend (LBE). At this point, G2 is un-paused and migration completes, and communication between FE and LBE on M2 resumes. While all subsequent accesses to device D1 use Netbus, the entire process of virtual device migration is transparent to the guest VM.

3.1 Pending IO Transactions

A potential issue for virtual device migration is that there may be pending IO transactions at the time G1 is suspended. More precisely, there may be pending IO transactions in BE submitted by G1's FE. By the time these IO transactions complete, the VM has already migrated and the BE can't return the IO results to the FE. To deal with the problem, the virtual device of the migrating VM is brought into a state where there are no pending IO operations to the backend driver, termed the *quiescent* state. To do this, during the suspend operation, the communication channel from FE to BE is suspended (but not in the other direction), so that the FE ceases to make additional requests to the BE, queuing them instead. In addition, suspend waits until all pending I/O operations are complete. At this point, the virtual device is in a quiescent state and can be migrated. During the resume operation on M2, FE first issues the queued requests to BE before resuming normal operation.

3.2 Device Hot-Swapping

Device hot-swapping permits a VM to dynamically rewire its local/remote device connections while the device is in operation. This is particularly useful when after migration, a VM wishes to switch from the remote to a local device to improve device throughput, remove network dependence or to shutdown the remote host. Transparent hot-swapping, however, requires that a 'similar' device be present locally (e.g., a disk with the same content as the original disk), implying the need to properly deal with device contents and state. The new device is brought into the same state as the original device before switching the LBE connection from the RBE to the local device, and resuming device operation. For devices like NICs, frequent hot-swapping is reasonable due to their small internal states. For disks and similarly state-rich devices, hot-swapping is likely to remain infrequent.

References

[1] KUMAR, S., AGARWALA, S., AND SCHWAN, K. Netbus: A Transparent Mechanism for Remote Device Access in Virtualized Systems. Tech. Rep. GIT-CERCS-07-08, Apr. 2007.