

Aggressive Server Consolidation through Pageable Virtual Machines

Anton Burtsev Mike Hibler Jay Lepreau

School of Computing, University of Utah

Abstract

We extend the Xen virtual machine monitor with the ability to host a hundred of virtual machines on a single physical node. Similarly to a demand paging of virtual memory, we page out idle virtual machines making them available on demand. Paging is transparent. An idle virtual machine remains fully operational. It is able to respond to external events with a delay comparable to a delay under a medium load.

To achieve desired degree of consolidation, we identify and leave in memory only a minimal working set of pages required to maintain the illusion of running VM and respond to external events. To keep the number of active pages small without harming performance dramatically, we build a correspondence between every event and its working set. Reducing a working set further, we implement a copy-on-write page sharing across virtual machines running on the same host. To decrease resources occupied by a virtual machine's file system, we implement a copy-on-write storage and golden imaging.

1 Introduction

Historically virtual machine monitors (VMMs) choose isolation as a primary goal. This design choice prohibits almost any form of resource sharing. The only shared resource is a physical CPU. Memory and disk space are statically preallocated upon creation of a virtual machine (VM). As a result 5-10 virtual machines easily exhaust memory resources of a physical host.

At the same time, there are multiple examples of virtual environments, where it is desirable to keep hundreds of virtual machines around. In many cases VMs are used occasionally, but have to be available on demand. Examples include long-running experiments in network testbeds like Emulab, months-long debugging and development sessions, experiments with unlimited number of nodes, when virtual machines are allocated on demand, honeypot installations, public machines in data centers and organizations, where users are allowed to keep a running copy of a VM, which is used occasionally.

Existing solutions optimize utilization of hardware resources by relying on combination of load balancing and migration. Application of these techniques is limited as they assume full load as a default case. Several systems increase server consolidation by using memory sharing across virtual machines. Potemkin [3] implements a

virtual honeypot capable of hosting hundreds of virtual machines on a single physical node. However, it limits all virtual machines to be started from the same memory image. VMWare virtual machine monitor implements a copy-on-write memory sharing [4] across virtual machines running on the same node. However, it's unable to achieve desired degree of consolidation as it doesn't attempt to page out memory of an idle virtual machine.

2 Overview

In this work we aim to run hundreds of idle or lightly-loaded virtual machines on a single node. We choose sharing and server consolidation over isolation as the primary goal.

To achieve this goal, we aggressively reclaim resources occupied by an idle VM. We detect when virtual machine becomes idle. After that we infer a minimal working set needed for a continuous operation and swap out the rest of the VM's memory. If we predict a long period of inactivity, we release the physical node entirely by migrating the virtual machine along with its local file system to a special node hosting idle VMs. Here we detect all pages, which are identical across VMs and share them in a copy-on-write manner.

Being swapped, a virtual machine remains fully operational. It processes timer events, maintains network connections, responds to requests from remote machines and local devices.

If we detect an event, which is predicted to lead to a period of active execution, we proactively start process of swapping the VM in.

3 Implementation challenges

Idleness detection: Swap out is a trade-off of whether we can benefit from the effort involved in swapping a guest operating system out and then back in, or lose on doing this expensive operation for a short period of idleness.

We have to decide when to consider a virtual machine to be idle. A good indication that VM becomes idle is the fact that it returns to the hypervisor before exhausting its scheduling quantum. Moreover, we can stress the VM further by depriving it of CPU time. For that we gradually reduce length and frequency of the VM's CPU quantum. We combine these techniques with existing approaches for detecting and predicting periods of idleness [2].

Timely response from a swapped VM: The core idea of our approach is to provide a timely response from a swapped virtual machine. Keeping the small working set

Students: Burtsev. Corresponding author: aburtsev@cs.utah.edu.
Poster web page: <http://www.cs.utah.edu/~aburtsev/rd2008.pdf>

	Working Set Size	
	4KB Pages	MBs
Idle VM	445	1.8
Idle Bash shell	725	2.9
Ping response	776	3.1
Idle ssh	805	3.2
SCP out 20MB 348KB/s	856	3.5
SCP in 20MB 348KB/s	7723	31.6
Find / -name *	6098	24.9
One minute Vim editing session	1432	5.8

Table 1: Working set size of an idle Linux VM

in memory, we create an illusion of running VM. To optimise event processing in the swapped state, we try to predict the next event and swap in the corresponding working set proactively.

Our default heuristics is to prepare for processing of the most frequent events, which are incoming network packets and timer interrupts. Therefore, we keep in memory an active set corresponding to these events. If we predict that the next event is wake up of a guest task, we try to extract information about the task from the guest OS and swap in its working set.

To estimate a working set size for typical “idle” activities, we instrumented the Xen VMM to measure a working set size of a Linux VM (Figure 1). The initial experiments show that in a typical idle case, the VM’s working set varies from 1.8MB to 3.2MB. This shows the possibility to host several hundreds of VMs on a single host.

Memory management: There is a general observation allowing us to significantly reduce amount of memory occupied by a single virtual machine. Virtual installations run VMs from a small set of preinstalled operating system images. As a result, many identical memory pages can be found and shared across virtual machines. Shared pages can be identified either by comparing page table hashes [4], or, more efficiently, by monitoring VM’s disk I/O [1].

The Xen VMM provides two modes of managing VM’s memory. Normally Xen employs advantages of paravirtualization and speeds up memory translation by directly exposing physical memory layout to a guest operating system. This means that it is impossible to implement a copy-on-write sharing transparently to the VM.

In the second mode of operation Xen maintains a shadow page table for every VM. Normally, the shadow page table is synchronised with the one used internally by the VM. However, both page tables may not be identical. The shadow page table translation comes at a performance price. Earlier approaches used the shadow page tables to implement a copy-on-write memory sharing [3]. At the moment, it is unclear if such solution is optimal.

File system sharing and migration: Similarly to the memory, file systems of all virtual machines initialized from the same system image are almost identical, and vary only with a small set of write changes. To reflect this fact, we use a “golden” file system image, from which we start

all VMs.¹ The golden image remains unchanged and thus can be shared across any number of VMs. To provide the VM with a write access, we developed a fast versioning block-level storage solution. Thus, a file system of any VM is a pair of golden image and a relatively small set of write changes. The golden image is cached on all nodes. This allows us to migrate a file system by transferring only the small set of write changes.

If we want to run hundreds of VMs on a single node, we have to consider the fact that any file system uses some memory for caching its metadata information. In case of a versioning storage, the metadata is extended with information about write updates. We plan to explore how this scaling can be achieved efficiently.

Network stack virtualization: Migration of multiple virtual machines on a single node results in a folding of a complex network topology into a single physical node. This requires an adequate support from a VMM’s network stack. The Xen VMM uses the Linux network stack to provide network connections across VMs.

The use of a software 802.11q VLANs allows us to avoid “revisitation” problem for nontrivial topologies on a single node. However, we have to be creative about instantiating hundreds of networks on a single host, as the Linux protocol stack is not designed to provide this support efficiently. Therefore, we work on coming up with a reasonably light-weight and efficient network stack virtualization in Linux.

V2P migration: There are cases when performance or realism of execution on a real hardware becomes essential. Naturally, this contradicts to our desire to run everything inside a VM. To meet both requirements, we explore a possibility of implementing a novel feature of converting a virtual system into physical and back on the fly, without rebooting or interrupting guest OS execution. We believe that the latest hardware virtualization support makes it possible.²

References

- [1] Edouard Bugnion, Scott Devine, Kinshuk Govil, and Mendel Rosenblum. Disco: running commodity operating systems on scalable multiprocessors. *ACM Trans. Comput. Syst.*, 15(4):412–447, 1997.
- [2] Richard A. Golding, Peter Bosch II, Carl Staelin, Tim Sullivan, and John Wilkes. Idleness is not sloth. In *USENIX Winter*, pages 201–212, 1995.
- [3] Michael Vrable, Justin Ma, Jay Chen, David Moore, Erik Vandekieft, Alex C. Snoeren, Geoffrey M. Voelker, and Stefan Savage. Scalability, fidelity, and containment in the potemkin virtual honeyfarm. *SIGOPS Oper. Syst. Rev.*, 39(5):148–162, 2005.
- [4] C. Waldspurger. Memory resource management in VMware ESX server. In *OSDI*, December 2002.

¹There is a golden image for every type of guest operating system

²VMWare VMM already implements this as “hot cloning”