Satori: Enlightened page sharing

Grzegorz Miłoś, Derek G. Murray, Steven Hand
University of Cambridge Computer Laboratory
Cambridge, United Kingdom
First.Last@cl.cam.ac.uk

Michael A. Fetterman
NVIDIA Corporation
Bedford, Massachusetts, USA
mafetter@nvidia.com

Abstract

We introduce Satori, an efficient and effective system for sharing memory in virtualised systems. Satori uses enlightenments in guest operating systems to detect sharing opportunities and manage the surplus memory that results from sharing.

Our approach has three key benefits over existing systems: it is more able to detect short-lived sharing opportunities, it is efficient and incurs negligible overhead, and it maintains better performance isolation between virtual machines.

We have implemented and evaluated a prototype of Satori for the Xen virtual machine monitor. In our evaluation, we show that Satori quickly exploits up to 94% of the maximum possible sharing with insignificant performance overhead. Furthermore, we demonstrate workloads where the additional memory improves macrobenchmark performance by a factor of two.

1 Introduction

An operating system can almost always put more memory to good use. By adding more memory, an OS can accommodate the working set of more processes in physical memory, and can also cache the contents of recently-loaded files. In both cases, cutting down on physical I/O improves overall performance. We have implemented Satori, a novel system that exploits opportunities for saving memory when running on the Xen virtual machine monitor (VMM). In this presentation, we will explain the policy and architectural decisions that make Satori efficient and effective, and evaluate its performance.

Previous work has shown that it is possible to save memory in virtualised systems by sharing pages that have identical [5] and/or similar [2] contents. These systems were designed for unmodified operating systems, which impose restrictions on the sharing that can be achieved. First, they detect sharing opportunities by periodically scanning the memory of all guest VMs. The scanning rate is a trade-off: scanning at a higher rate detects more sharing opportunities, but uses more of the CPU. Secondly, since it overcommits the physical memory available to guests, the VMM must be able to page guest memory to and from disk. We observe in our evaluation that this can lead to poor performance.

We propose to present Satori in two parts: first, we will outline the major design decisions that differentiate...
Satori from other systems (Section 2), then we will describe how we implemented a prototype of Satori on the Xen VMM (Section 3). Finally, we will present the results of a thorough evaluation of Satori’s performance, including its effectiveness at finding sharing opportunities and its impact on overall performance (Section 4).

2 Design decisions

We will first present the major design decisions that differentiate Satori from existing page sharing schemes, including those found in VMWare ESX Server [5] and Difference Engine [2]. Figure 1 shows the life-cycle of a page that participates in sharing. This diagram raises three key questions, which we will address in the presentation:

How are duplicates detected? We propose sharing-aware block devices as a low-overhead mechanism for detecting duplicate pages. Since a large amount of sharing originates within the page cache [3], we exploit this fact by monitoring data as it enters the cache.

How are memory savings distributed? When \( n \) identical pages are discovered, these can be represented by a single physical page, and \( n - 1 \) pages are saved. We propose distributing these savings to guest VMs in proportion with their contribution towards sharing.

What happens when sharing is broken? Shared pages are necessarily read-only. When a guest VM attempts to write to a shared page, the hypervisor usually makes a writable private copy of the page for the guest. We propose that the guest itself provides a list of volatile pages that may be used to provide the necessary memory for private copies. In addition, we obviate the need for copying in certain cases.

3 Implementation

We implemented Satori for Xen version 3.1 and Linux version 2.6.18 in 11551 lines of code (5351 in the Xen hypervisor, 3894 in the Xen tools and 2306 in Linux). We chose Xen because it has extensive support for paravirtualised guests [1]. In this part of the presentation, we will describe how we implemented the design decisions from Section 2.

Our changes can be broken down into three main categories. We first modified the Xen hypervisor, in order to add support for sharing pages between VMs. Next, we modified the blktap driver to add support for sharing-aware block devices. Finally, we enlightened the guest operating system, so that it can take advantage of additional memory, and repay that memory when sharing is broken. The guest OS enlightenments were based on porting some parts of IBM’s Collaborative Memory Management (CMM) system to Xen [4].

4 Evaluation

To characterise Satori’s performance, we conducted an evaluation in three parts. First, we profiled the opportunities for page sharing under different workloads. In contrast with previous work, we specifically considered the duration of each sharing opportunity, as this is crucial to the utility of page sharing. We then measured the effectiveness of Satori, and found that it is capable of quickly detecting a large amount of sharing. Finally, we measured the effect that Satori has on performance, in terms of the benefit when sharing is enabled, and the overhead on I/O operations.

For reasons of brevity, we cannot present the full evaluation results here. However, we note that Satori exploits up to 94% of the total sharing opportunities for some workloads. We also note that the overhead of duplicate detection (without exploiting sharing opportunities) is statistically insignificant. When sharing is enabled, the additional page cache capacity improves performance for some I/O-bound macrobenchmarks.

5 Conclusions

In this abstract, we have introduced Satori, which employs enlightenments to improve the effectiveness and efficiency of page sharing in virtualised environments. In our presentation, we will identify several cases where the traditional page sharing approach (i.e. periodic memory scanning) does not discover or exploit opportunities for sharing; and show that, by using information from the guest VMs, and making small modifications to the operating systems, it is possible to discover a large fraction of the sharing opportunities with insignificant overhead.

Our implementation has concentrated on sharing-aware block devices. In the future we intend to add other enlightened page sharing mechanisms – such as long-lived zero-page detection, page-table sharing and kernel text sharing – which will improve Satori’s sharing discovery rate.
References


