Journal of Systems Architecture 59 (2013) 1405–1413

Contents lists available at ScienceDirect

Journal of Systems Architecture

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

Cache isolation for virtualization of mixed general-purpose and real-time systems

Ruhui Ma [a,c], Wei Ye [a,b], Alei Liang [a,b], Haibing Guan [a,c], Jian Li [a,b,*]

[a] Shanghai Key Laboratory of Scalable Computing and Systems, Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai, China
[b] School of Software, Shanghai Jiao Tong University, Dongchuan Road 800, Shanghai 200240, China
[c] Department of Computer Science & Engineering, Shanghai Jiao Tong University, Dongchuan Road 800, Shanghai 200240, China

ARTICLE INFO

Article history:
Available online 18 July 2013

Keywords:
Cache
Soft real-time
Hypervisor
Multi-core

ABSTRACT

Shared cache has been widely used to lessen the communication overhead between multi-core processors and off-chip main memories. In a virtualization execution environment, real-time tasks supported by real-time operating systems (RTOS) need to share the last level cache with other general purpose tasks (supported by general purpose operating systems (GPOS) or execute on bare metal). Since current hypervisors are not aware of the attributes of tasks in the virtual machines, real-time tasks cannot be adequately supported by the hypervisor. Therefore, real-time task will suffer from many cache misses, and the response of a virtual machine may be poor due to cache pollution, especially if real-time tasks compete for cache resources with heavy general-purpose tasks. To address this issue, we propose two cache optimization management policies. One is cache partitioning (CAP) – which isolates the cache statically for real-time tasks and general-purpose tasks using page coloring. Hence the predictability of cache usage is greatly increased. CAP reduces both cache pollution, and the amount of cache space that real-time tasks use, even if other cache is not used by general purpose tasks. We also study allocating different proportions of the cache for real-time tasks and find that, a proper cache ratio is optimal to guarantee response time. Due to this, another dynamic cache optimization policy is proposed, called page table prefetching (PTP). PTP employs a daemon-like thread to maintain the cache, so as to make the contents used by real-time tasks be in cache. PTP is implemented by utilizing a dedicated core assigned to general-purpose tasks to do prefetching of a page table into shared physical cache for a soft real-time task. As the page table is referenced frequently by tasks, PTP gives a considerable advantage because fewer TLB misses occur. The results from the Cyclitest and CPUspec2006 benchmarks show that both CAP and PTP can provide better response for real-time tasks. CAP has benefits due to its simplicity, but PTP has advantages in flexibility and high cache utilization due to its dynamic cached maintenance. In addition, the experimental results show that PTP improves real-time performance more but sacrifices some GP performance.

1. Introduction

Recent years have witnessed the quick development of virtualization technology, which has been widely employed in various domains, such as server consolidation, datacenters, IP telephone severs, and embedded systems [1,2]. Its key functionality is providing complete isolation of hardware resources, making it possible to integrate applications with different runtime environments together on the same physical platform with a tiny cost. For example, on a smartphone, health care applications can co-exist with many general-purpose applications with the help of virtualization technology. This avoids the requirement of recompilation to fit a specific operating system.

At the same time, multi-core processors have also been widely employed in all kinds of computing platforms, and virtualization technology allows diverse applications to share the finite underlying physical hardware resources among different VMs (virtual machines), each running its own operating system (e.g. Window 7, Android, Vxworks, etc). Reasonable and efficient utilization of multi-core resources in the virtual execution environment is critical, so the VMM (virtual machine monitor) software layer not only promises performance isolation between different VMs, but also manages thread scheduling to guarantee performance, fairness or QoS (Quality of Service) [3–5].

Most multi-core processors contain shared LLC (last level cache), which has important advantages, such as better cache utilization, fast inter-core communication, and decreased aggregate

* Corresponding author. Tel.: +86 21 34207874.
E-mail addresses: ruhuima@sjtu.edu.cn (R. Ma), weiye@sjtu.edu.cn (W. Ye), liangalei@sjtu.edu.cn (A. Liang), hbguan@sjtu.edu.cn (H. Guan), li-jian@sjtu.edu.cn (J. Li).

1383-7621/$ - see front matter © 2013 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.sysarc.2013.07.004

cache footprint. On the other hand, shared LLC also results in the challenge of a competition among VMs although virtual machines can easily share underlying physical resources for multiple applications. On a multi-core processor with shared LLC, real-time tasks have to compete for cache lines with general-purpose tasks. Real-time performance will be significantly affected by LLC competition, so it becomes difficult to ensure meeting real-time constraints [6].

Although multiple applications can easily share underlying physical resources with the help of virtualization technology, managing the sharing may be difficult. Various VMs – including RT (real time) VMs running soft real-time applications and GP (general purpose) VMs running general-purpose applications – compete for shared cache, which becomes a performance bottleneck.

Resource competition for the LLC (last level cache) may significantly increase the overhead for context switches between VMs. For example, when a GPOS exhausts the cache capacity, an RTOS has to access off-chip main memory and suffer a heavy context switch overhead caused by the penalty of TLB (Translation Lookaside Buffer) misses [7]. This cache pollution can degrade the real-time performance for mission-critical applications. Therefore, it is necessary to deal with LLC pollution by advanced cache isolation methods.

This paper focuses on a case study at system level to integrate a real-time operating system (RTOS) with a general-purpose operating system (GPOS). We use a static cache isolation method to prevent cache pollution in order to achieve both high system performance and better real-time response. A dynamic cache maintenance method is proposed to keep a cache partition for an RTOS from being polluted by applications in a GPOS, and it is compared with a static cache partition policy based on page coloring. In order to retain the real-time performance of mission-critical applications in the RTOS, an isolated cache space is preserved for it.

Cache partitioning has already been employed to address cache pollution [3,5,8,9–15], and a performance improvement is obtained through the elimination of undesired replication of cache lines. This can to some extent resolve the problem of cache pollution and efficiently utilize cache space for each thread. But most existing studies [3,5,12,13,15] were executed on a simulator, rather than on a real physical machine. Although simulation is flexible, it possesses several limitations in evaluating these cache partitioning methods. Simulation platforms have such limited capacity that most current large, dynamic real programs cannot be executed on them, and the most serious problems in the real world are omitted. On a typical simulator, only a few billion instructions for a program can be simulated, which is equivalent to about one second of execution on a real physical machine. Also, the characteristics of an OS are not adequately exhibited in simulation, since the overall impact of the OS cannot be observed in a period of simulation time. Some research results [8,9–11,14] on cache partitioning are based on real physical machines, but they focus on global system performance improvement, without considering real-time response.

In this paper, the VMM KVM (Kernel virtual machine) [17] is selected to support our VMM framework, which is a new virtualization solution for Linux. The methods examined in this paper employ only a slight modification on the operating system's memory allocation mechanism, therefore our work would be easy to port to other VMMs like Xen [18].

To fulfill the requirements of soft real time applications, we present a new static cache management scheme, CAP (cache partitioning), to address real-time issues with a KVM-based framework. This work, based on a page coloring technique [11,15], is directly applied in the physical machine rather than in a simulator. CAP was built in order to balance real-time capability and the overall performance. It requires only a little modification of the Linux kernel to fully support the CAP policy. By assigning cache space adaptively, the system can achieve better performance and promise executing real-time applications successfully. Finally, we have tested CAP with extensive experiments as to its effectiveness in improving the performance of GP applications and promising better RT response.

To provide a stronger safety guarantee for RT VMs when competing with GP VMs requires a more dynamic restriction on GP VMs in cache management. Therefore, we propose another cache management module, PTP (page table prefetch), in our KVM-based framework. It is assigned to a dedicated core running general-purpose applications. This mechanism sacrifices some performance of general-purpose applications to support soft real-time tasks by reserving some cache for them. Obviously, this cache management mechanism is not ideal for managing cache resources on popular platforms, since it only emphasizes soft real-time applications. The experiments show that PTP is better than CAP at improving the responsiveness.

The rest of the paper is organized as follows: Section 2 introduces our KVM-based method to combine RTOS and GPOS as well as pointing out the cache pollution problem. In Section 3, we describe the CAP and PTP schemes. Then Section 4 gives evaluation results for them. Section 5 discusses related work and Section 6 offers conclusions and discusses our plans for future work.

2. System architecture and problem statement

The combination of a RTOS and off-the-shelf time-sharing GPOS by using virtualization technology is a common approach to get both a large application base and timely deterministic response. A famous use case is that in industrial control or healthcare, a RTOS is used to take over the time-critical tasks and a GPOS is used beside it to run some monitor applications. We propose a framework to run an RTOS and a GPOS together.

2.1. System architecture combining RTOS and GPOS

A mobile device, like a smartphone, may run multiple operating systems under the overall control of a hypervisor, to gain better utilization of its resources just like we introduced in the previous section. In this situation, a variety of OSs will run concurrently, some of them as normal applications, and others demanding soft real time features. So we use GPOSes to contain normal applications and RTOSes to contain the ones which need real-time response.

Without loss of generality, we build one RTOS and one GPOS on a KVM hypervisor and we use Linux with PREEMPT-RT [19,20] as the RTOS. KVM is a highly efficient and straightforward virtualization solution for Linux, which uses the hardware virtualization extension and is built as a Linux kernel module. It is designed as a full system virtualization solution, which allows users to run unmodified guest operating systems. KVM presents an interface to the guest system completely similar to the underlying hardware, for Linux on X86 with hardware virtualization extensions. We use KVM in our framework to run two guests, one running a real-time OS and one a general-purpose OS.

To ensure the RTOS runs on physical CPUs as soon as possible when the RT task is issued, we pin the RTOS to a dedicated core, and other processes including the GPOS to another core, as shown in Fig. 1. Generally, a modern processor has many cores. Those cores always have several levels of caches where the lower-level caches are isolated but the last level cache (LLC) is shared between the cores.

The system architecture combining RTOS and GPOS (referred as "SCRG" below) handles a common use case. Applications range from a pure data service like consolidation of datacenter servers to more complicated SIP (session initiation protocol) and