



國立臺灣大學電機資訊學院資訊工程學系

碩士論文

Department of Computer Science and Information Engineering

College of Electrical Engineering and Computer Science

National Taiwan University

Master Thesis

虛擬化多核心之省電投射技術
Energy-Efficient Mapping Technique
for Virtual Processors.



Yu-Chia Lin

指導教授：郭大維 博士

Advisor : Tei-Wei Kuo, Ph.D.

中華民國 九十八 年 六 月

June, 2009



Energy-Efficient Mapping Technique for Virtual Processors

by

YU-CHIA LIN

Supervised by

DR. TEI-WEI KUO



Submitted to

GRADUATE INSTITUTE OF
COMPUTER SCIENCE AND INFORMATION ENGINEERING

In Partial Fulfillment of the Requirements
For the Degree of
MASTER OF SCIENCE

At the
NATIONAL TAIWAN UNIVERSITY

June 2009



This dissertation is dedicated to my parents,





國立臺灣大學碩士學位論文
口試委員會審定書

虛擬化多核心之省電投射技術

Energy-Efficient Mapping Technique for Virtual
Processors

本論文係林育佳君（學號 R96922062）在國立臺灣大學資訊工程學系完成之碩士學位論文，於民國 98 年 6 月 24 日承下列考試委員審查通過及口試及格，特此證明

口試委員：

郭大維

張吉昇

(指導教授)

李政崑

石維寬

洪士濂

呂育道

系主任



中文摘要

在現今的系統設計上，虛擬化技術為解決可攜性、可維持性、發展性以及系統使用率的問題上提供了一個極佳的解決方案。本論文中，我們將專注於開發平台虛擬化技術上的省電設計。我們探討了虛擬化核心與真實核心上計算資源的對應技術，以及在考慮系統上執行工作的時間限制下，虛擬化核心與真實核心上能源消耗的關係。透過模組化每個虛擬化核心上所需的執行資源，本論文同時考慮了即時性工作以及非即時性工作上的工作量。以此設計為基礎，本論文提出了一個以微核心為基礎，支援動態電壓調整的原型架構。並評估了此架構的功能以及造成的負擔，實驗結果展示出本設計在省電支援以及系統負擔上，有著良好的結果。

關鍵字：虛擬化核心模組，開放式環境，省電設計，投射技術，虛擬化環境，微核心

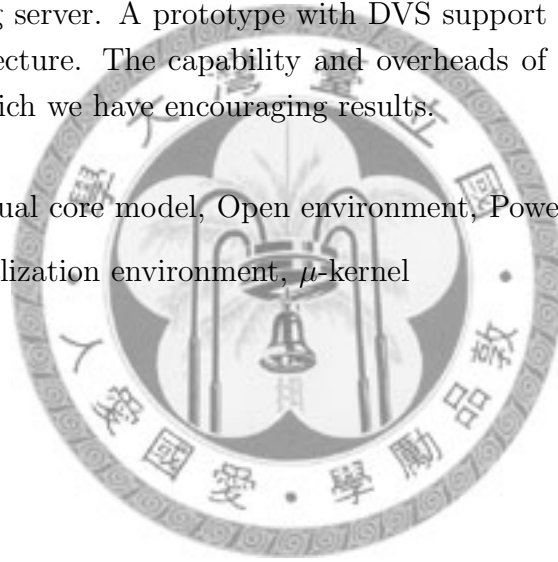




Abstract

Virtualization provides an excellent solution to resolve the portability, maintainability, development, and utilization problems in many system designs. In this paper, we are interested in energy-efficient designs for platform virtualization. In particular, we explore the computing resource mapping between virtual cores and physical cores and their energy consumption relationship when timing constraints in task executions are considered. Real-time and non-real-time task workloads are both considered in the study, where the computing needs of each virtual core is modeled with a computing server. A prototype with DVS support is implemented based on a μ -kernel architecture. The capability and overheads of the proposed design was evaluated, for which we have encouraging results.

Keywords: Virtual core model, Open environment, Power-aware design, Mapping technique, Virtualization environment, μ -kernel





Contents

Abstract	ix
Contents	xi
List of Figures	xiii
List of Tables	xv
1 Introduction	1
2 System Architecture and Research Motivation	5
3 Power-Aware Virtualization System	9
3.1 A Virtual-Core Server Model	10
3.2 Hypervisor Functionalities	14
3.3 Implementation Remarks	17
4 Performance Evaluation	19
4.1 System Implementation	19
4.2 Experimental Setup	21
4.2.1 Experimental Results	22
5 Conclusions	27



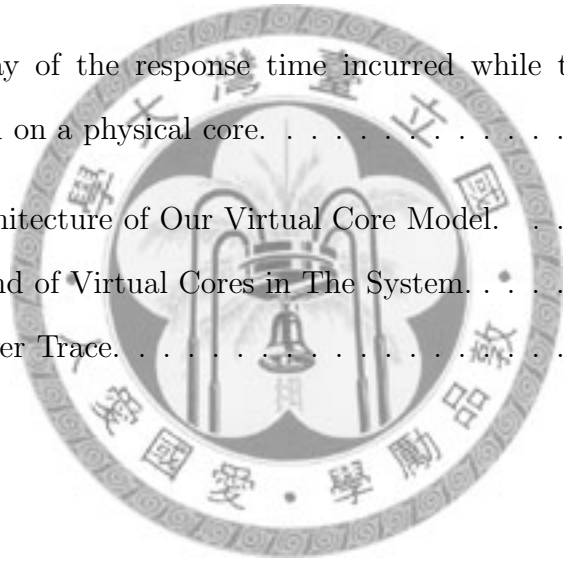
Bibliography

29



List of Figures

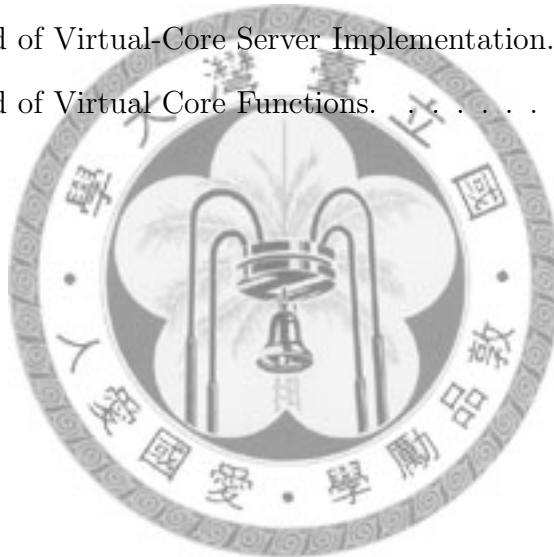
2.1	A Virtualization Example.	5
2.2	Two virtual cores over one physical core.	6
3.1	The delay of the response time incurred while the virtual core is emulated on a physical core.	12
4.1	The Architecture of Our Virtual Core Model.	20
4.2	The Trend of Virtual Cores in The System.	23
4.3	The Power Trace.	24





List of Tables

4.1	DVS Level Support in Davinci on ARM side.	22
4.2	The Behavior of Virtual Cores.	22
4.3	Overhead of Virtual-Core Server Implementation.	24
4.4	Overhead of Virtual Core Functions.	24





Chapter 1

Introduction

The growing variety and rapid evolving of hardware architectures and platforms has resulted in tremendous challenges in the development of system and application software. Virtualization, that provides an abstraction layer between hardware platforms and their executing software, provides an excellent alternative to simplify the development and deployment of applications and to improve the system resource utilization. Such observations motivate this work in the exploring of energy-efficient virtualization designs.

Virtualization techniques can be classified into resource virtualization, application virtualization and platform virtualization techniques according to the perspective of their abstractions: *Resource virtualization* provides a layer of abstraction between (physical) resources, such as disk space, CPU cycles, and RAM, and the semantic activities in the consuming of the resources [8, 38]. *Application virtualization* (also known as application service virtualization) improves the portability, manageability and compatibility of applications by encapsulating them from the underlying operating system on which they are executed [4, 15]. *Platform virtualization*, that hides the physical characteristics of hardware platforms from users

by emulating multiple virtual machines (VMs), have many implementations in the market already. Well-known products are such as Xen [2], VMWare [40], Virtual PC [21] and VirtualBox [23] for personal computers and/or enterprise systems. Another implementation is the hypervisor of OpenSPARC, that provides a high-performance firmware-level abstraction, referred to as the logical domain, to enhance the system portability [36]. Moreover, VirtualLogix [24] and μ -kernel such as L4:Fiasco [17] and OKL4 [22] are designed for embedded systems with the considerations of security, robustness, isolation and real-time performance guarantee [10–12, 34]. This work would focus itself on platform virtualization because of its wide applicability in various domains.

In the past decades, a number of excellent research results on virtualization were also proposed in the academics. For example, research on the performance enhancement of the system by means of the executions of multiple operating systems was presented by Murata et al. [32]. When system security is considered, Criswell et al. [5] designed a virtual instruction set to provide safe execution environments. Seshadri et al. [37] guaranteed the code integrity with a hypervisor-based solution. Moreover, with secure operating systems [13, 20] upon a minimized software level based on the L4 μ -kernel architecture, the system security can be guaranteed at the operating system and application levels. Härtig [9] presented a high-performance design of μ -kernel for embedded systems. The dependability can also be improved by using unmodified device drivers in virtual machines [27].

In this paper, we are interested in energy-efficient designs for virtualization because of its importance and the lack of sufficient study in that direction. The closest related work is the power control architecture with on-line adjustments by Nathuji et al. [33]. A similar framework was proposed by Stoess et al. [39] to

account and allocate the energy consumption of virtual machines. An architecture was also proposed by Wang et al. [42] to minimize the energy consumption under response time constraints. However, many past results do not consider real-time tasks or are only dedicated to enterprise systems. In this paper, we should consider both real-time and non-real-time tasks and *dynamic voltage scaling* (DVS) in energy-efficient designs. In particular, we explore the computing resource mapping between virtual cores and physical cores and their energy consumption relationship when timing constraints in task executions are considered. We propose to model the computing needs of a virtual core with a server model and propose a DVS policy to minimize the energy consumption without any potential violation of timing constraints. The proposed methodology is realized based on a μ -kernel architecture. A series of experiments was conducted. It was shown that the energy consumption of a multimedia application was reduced by $XX\%$ with less than 5% overhead in terms of the system performance.

The rest of the paper is organized as follows. Section 2 provides the system architecture and the research motivation. The design and analysis of the proposed virtualization methodology is presented in Section 3. Section 4 addresses implementation issues and provides the evaluations of the proposed virtualization methodology. Section 5 concludes this paper.



Chapter 2

System Architecture and Research

Motivation

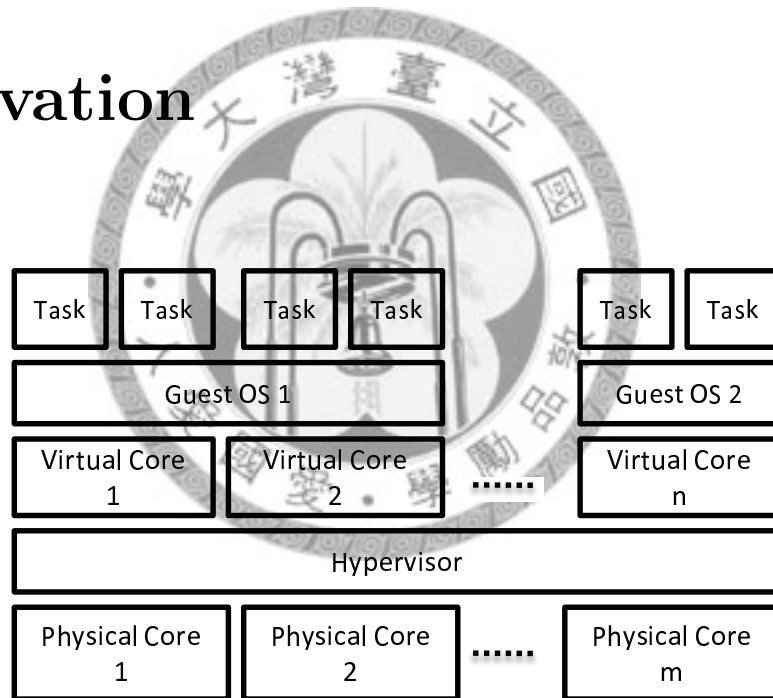


Figure 2.1: A Virtualization Example.

As the number of cores per system grows significantly in the next few years, how to provide an effective system design that can be adaptive to the development or the deployment of multi-core platforms has become a very critical issue. Among possible design alternatives, virtualization is a popular concept to resolve the problem, e.g., [2, 32, 41]. As shown in Figure 2.1, virtualization is usually realized by

having a hypervisor over physical cores to emulate selected virtual cores so that selected guest operating systems could run with their target multi-core configurations. With the help of virtualization, the number of virtual cores and their characteristics could be realized before the deployment of a target system. It is the obligation of the vendors to ensure that the final and underlying hardware can support the required configuration. Note that there is, in general, no constraint on the ratio between the number of virtual cores and that of physical cores.

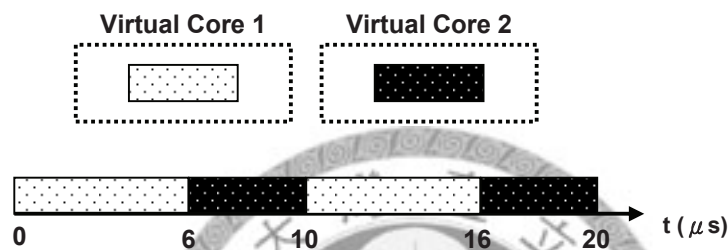
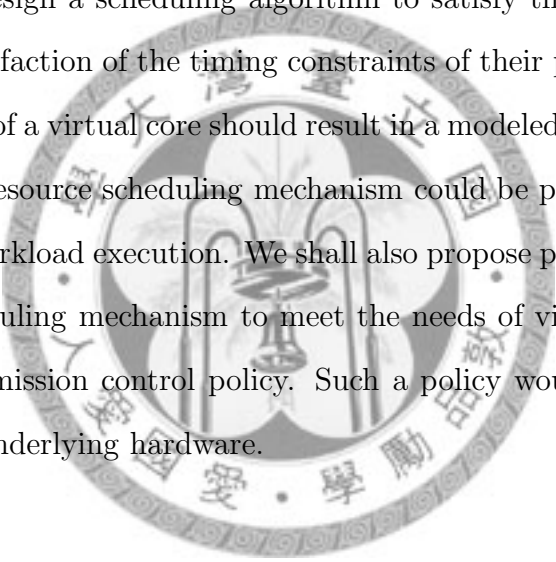


Figure 2.2: Two virtual cores over one physical core.

Different from the past work, we are interested in the dynamic-voltage-scaling (DVS) implementation issues of virtual cores. The adjustment of the operating frequency of a virtual core might result in proper dynamic voltage scaling (or even the turning-off) of some physical core and/or trigger the service adjustment mechanism of a hypervisor to emulate selected virtual cores at their proper speeds. Such an adjustment procedure might generate some potential problem in timing constraint violations. For example, consider an embedded system with 2 physical cores that both operate at an operating frequency f , where each physical core serves one virtual core in an initial configuration. Suppose that the operating frequency of the first virtual core is now set as $0.6f$, and that of the second virtual core is also adjusted as $0.4f$. After the frequency adjustment, the hypervisor might decide to turn off one physical core and to let two virtual cores share one physical core for energy saving, as shown in Figure 2.2. Suppose that two virtual cores use the first $6\mu s$ and the last $4\mu s$, respectively, for every $10\mu s$ time period over the physical core.

Such a virtual core emulation might come to a deadline violation problem when the second virtual core must run a real-time task with a period $5\mu s$ and an execution time of $2\mu s$ (at the operating frequency f). Such an observation reveals a constraint on how to emulate virtual cores over physical cores when timing constraints of processes must be satisfied.

In this paper, we shall address three major design issues regarding the DVS support of virtual cores: (1) How to model the DVS needs of a virtual core, (2) how to map the DVS needs of virtual cores into the DVS settings of physical cores, and (3) how to design a scheduling algorithm to satisfy the needs of virtual cores, including the satisfaction of the timing constraints of their processes. The modeling of the DVS needs of a virtual core should result in a modeled workload for the virtual core such that a resource scheduling mechanism could be proposed for a hypervisor to schedule the workload execution. We shall also propose proper scheduling policies to drive the scheduling mechanism to meet the needs of virtual cores and derive a corresponding admission control policy. Such a policy would help to estimate the demands to the underlying hardware.






Chapter 3

Power-Aware Virtualization

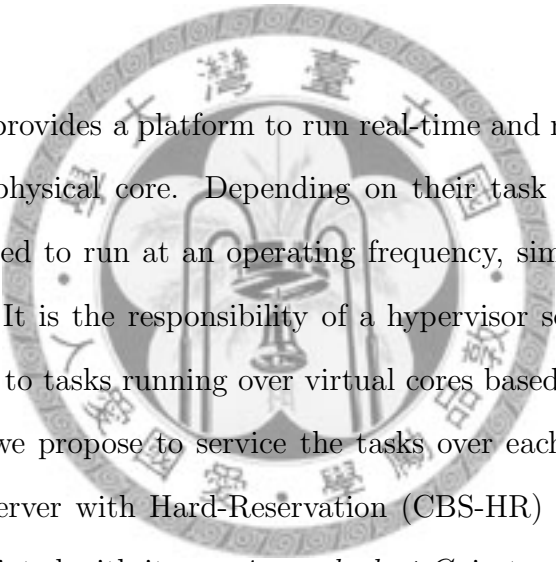
System



This section presents our approach for the DVS support of virtual cores. With the considerations of both the real-time and non-real-time task workloads over virtual cores, a resource-reservation-based model is considered for the modeling of the computing demands of each virtual core in Section 3.1. In Section 3.2, a DVS-based scheduling policy is proposed for the DVS adjustment of physical cores and the emulation of virtual cores while timing constraints of real-time tasks are guaranteed. Implementation remarks are later presented with the considerations of overheads and the support of multiple physical cores.

3.1 A Virtual-Core Server Model

The purpose of this section is to explore the modeling of the computing needs of virtual cores and their emulation over physical cores. A Constant-Bandwidth-Server-based model is presented to model the computing needs of each virtual core with resource reservation [1, 35]. In this paper, we are interested in the system design issues for the sharing of a physical core for multiple DVS virtual cores, where no global resource synchronization is considered among tasks over multiple virtual cores.



A virtual core provides a platform to run real-time and non-real-time tasks in a way similar to a physical core. Depending on their task workloads, virtual cores might be configured to run at an operating frequency, similar to their counterpart physical cores. It is the responsibility of a hypervisor software to allocate proper execution cycles to tasks running over virtual cores based on their workload characteristics. Thus, we propose to service the tasks over each virtual core by a Constant Bandwidth Server with Hard-Reservation (CBS-HR) server, where each CBS-HR server is associated with its *maximum budget* C_i in terms of the execution cycles and a *replenish period* T_i [30, 31]. Constant Bandwidth Server (CBS) [1] is a resource reservation algorithm in which servers are scheduled under the Earliest-Deadline-First (EDF) scheduling algorithm, that always schedules the task with the nearest (absolute) deadline. In order to prevent CBS from suffering the deadline aging problem [31], CBS-HR is extended from CBS such that it can guarantee a fixed execution budget C_i during every period T_i .

We propose to model the workload characteristics over a virtual core α_i by three parameters (C_i, T_i, F_i) : C_i , T_i , and F_i denote the number of the total execution

cycles of its tasks, the replenish period of its corresponding CBS-HR server, and its user-set operating frequency, respectively. A legal setting of the maximum budget (also denoted as C_i) and the replenish period (also denoted as T_i) of the corresponding CBS-HR server of a virtual core should satisfy the inequality $\frac{C_i}{T_i} \geq F_i^1$. The replenish period T_i of a virtual core (or its CBS-HR server) should also comply with the timing constraints of its real-time tasks. For example, consider the execution of a real-time task τ_i with c_i execution cycles and a relative deadline p_i (as shown in the example of Section 2). p_i should be able to be divided by the replenish period T_i of a virtual core. Consider a set of real-time independent periodic tasks $\{\tau_1, \tau_2, \dots, \tau_n\}$ to execute over a virtual core, where independent tasks could preempt the executions of each another at any time. Let c_i and p_i denote the maximum number of execution cycles and the period of a task τ_i of the set, respectively. Suppose that the EDF scheduling algorithm is adopted to schedule the task set. The following lemma could be proved for the setting of the budget and replenish period of a CBS-HR server:

Lemma 3.1 *Given a set of real-time independent periodic tasks $\{\tau_1, \tau_2, \dots, \tau_n\}$, a CBS-HR server with the replenish period $T = \text{gcd}(p_1, p_2, \dots, p_n)$ and the budget $C \geq \sum_{i=1}^n \frac{c_i}{p_i} \cdot T$ could guarantee the schedulability of the task set.*

Proof. In order to meet all timing constraints of this task set, the operating frequency of the virtual core F_v can not be less than $\sum_{i=1}^n \frac{c_i}{p_i}$, because a set of real-time tasks can be schedulable on a processor under the EDF scheduling algorithm if and only if its total utilization $\sum_{i=1}^n \frac{c_i/F_v}{p_i}$ is no more than 1 [29]. Therefore, while these

¹The budget and the replenish period of a CBS-HR server could be theoretically different from the number of the total execution cycles of the tasks and the replenish period of its corresponding virtual core, respectively. In this paper, we would let them be the same to simplify the discussion and implementation.

n real-time tasks have the highest priority or there is no other task on the virtual core, they will be schedulable if F_v is $\sum_{i=1}^n \frac{c_i}{p_i}$.

Let C_k be the execution cycles that a virtual core can guarantee within any p_k , while we set the replenish period and maximum budget of the virtual core according to our configuration. Then, we have

$$C_k = \lfloor \frac{p_k}{T} \rfloor C = \frac{p_k}{T} C \geq p_k \sum_{i=1}^n \frac{c_i}{p_i} \text{ for } k = 1, 2, \dots, n.$$

That is, from the perspective of any task τ_k , the virtual core can provide sufficient execution cycles within its relative deadline p_k such that the virtual core will behave like a processor whose operating frequency is $\frac{C_k}{p_k}$ which is no less than F_v . Since it is similar to that these tasks are executed over a processor whose operating frequency is no less than F_v , the total utilization will be no more than 1. Therefore, the timing constraints of these tasks can be satisfied.

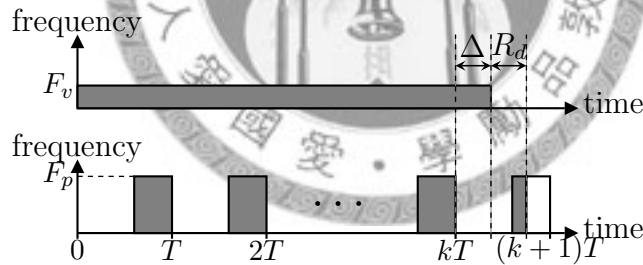


Figure 3.1: The delay of the response time incurred while the virtual core is emulated on a physical core.

Unlike real-time tasks where the timing constraints are crucial, the concern of non-real-time tasks is different. Such as multimedia applications and batch applications, users might be only interested in the throughput provided by the virtual core which can be directly reflected by the operating frequency specified by users. When these tasks are executed over a virtual core, users might also need to take care of the response time which is affected by the replenish period. Thus, they need

to determine the replenish period [3] such that the reasonable delay of the response time is achievable. However, as the granularity on the executions of non-real-time tasks is not as crucial as that of real-time tasks, users do not have to treat non-real-time tasks as one kind of real-time tasks by setting the maximum budget and the replenish period according to Lemma 3.1.

As shown in Figure 3.1, when we use a virtual core emulated on a physical core to execute a certain workload, the lower part of the figure demonstrates its behavior, where the gray area stands for the execution cycles of the workload. Compared to the behavior when the same workload is executed on a physical core with the same emulated operating frequency as shown in the upper part of Figure 3.1, the delay of the response time R_d might be incurred. Therefore, the user of the virtual core might specify a *tolerable response time delay* σ that the delay of the response time on the virtual core should not exceed. In order to meet the requirement, the following lemma directs how to set the maximum budget and the replenish period of the virtual core.

Lemma 3.2 *Given a tolerable response time delay σ , the delay of the response time on the virtual core with operating frequency F_v can be no more than σ if the replenish period and the maximum budget are set according to $T \leq \frac{\sigma}{1-F_v/F_p}$ and $C \geq F_v \cdot T$, respectively, provided that the operating frequency of the physical core is F_p .*

Proof. The worst case of the response time occurs when C is equal to $F_v \cdot T$. Thus, we assume C is $F_v T$ in our proof.

For each replenish period T , the virtual core can perform C execution cycles which is the same as that can be provided on a physical core with operating frequency F_v . This leads to that there is no difference between the finished execution cycles

on the physical core and that on the virtual core. Therefore, as shown in Figure 3.1, the delay of the response time is incurred only if $\Delta = \frac{R_p}{T} - \lfloor \frac{R_p}{T} \rfloor > 0$, where W_p is the response time on the physical core. Then the delay of the response time on the virtual core R_d can be computed as follows.

$$R_d = \left(T - \frac{C}{F_p}\right) + \frac{\Delta \cdot F_v}{F_p} - \Delta$$

By substituting $x \cdot T$ for Δ , R_d becomes $T((x - 1)(\frac{F_v}{F_p} - 1))$ which is a function of x . Since the operating frequency of the virtual core F_v is no more than that on the physical core F_p , R_d has the maximal value when x approaches to 0. That is, R_d is no more than $T(1 - \frac{F_v}{F_p})$. Combined with $T \leq \frac{\sigma}{1 - F_v/F_p}$, the delay of the response time R_d could be no more than the tolerable response time delay σ .

3.2 Hypervisor Functionalities

The hypervisor is one of the most important component in virtualization systems. This section describes the functionalities that the hypervisor has to support in order to provide a power-aware virtualization system. With the virtual-core server model presented in Section 3.1, our virtualization system serves user applications in a two level hierarchical scheduling scheme [6, 7, 25]. Under the two level hierarchical scheduling scheme, the hypervisor manages the computing resource of physical cores to guarantee the needs of virtual-core servers in the lower level. In the upper level, each guest Operating System (OS) runs over virtual cores and schedules user applications to meet their timing constraints with a customized scheduler. When there are more than one virtual cores simulated on a single physical core, the schedulability of the system should be considered. Thanks to the following lemma of CBS-HR

server, the schedulability can be guaranteed under the virtual-core server model if we consider a virtual-core server with parameters (C_i, T_i, F_i) as a CBS-HR server with utilization $U_i = \frac{C_i/F_p}{T_i}$ where F_p is the operating frequency of the physical core.

Lemma 3.3 *Given a set of periodic tasks with total utilization factor U_p and a set of virtual cores with total utilization factor $U_s = \sum_i U_i$, the whole system is schedulable under the EDF scheduling algorithm if and only if $U_p + U_s \leq 1$ [31].*

In order to utilize virtual cores in the virtualization system, the hypervisor has to support three functions: *virtual core creation*, *virtual core deletion*, and *virtual core adjustment*. The virtual core creation function let users acquire a new virtual core according to their requirements. Given the required operating frequency of virtual core F_i and its timing constraint, the hypervisor derives T_i based on Lemma 3.1 or Lemma 3.2 and then creates a virtual-core server with parameters (C_i, T_i, F_i) , where C_i is $F_i \cdot T_i$. On the other hand, once a virtual core is no longer required, it can be removed from the virtualization system and release its allocated computing resource with the virtual core deletion function. Finally, with virtual core adjustment function, the virtual core in the virtualization system can be treated as a physical core with the capability of dynamic voltage scaling (DVS). More specifically, when the user specifies a new operating frequency F'_i for a virtual core with the virtual core adjustment function, the hypervisor will adjust the virtual core accordingly such that the configuration of the virtual core becomes $(F'_i T_i, T_i, F'_i)$.

In addition to the functions to utilize virtual cores, the hypervisor also needs to support admission control mechanisms because of limited computing resource of physical cores. That is, at each time of the virtual core creation or the virtual

core adjustment, the admission control mechanism will be triggered to examine whether the request could be granted or not. Thus, we define $U_c = 1 - \sum_i U_i$ as the *remaining utilization* for a physical core in the virtualization system, where $\sum_i U_i$ is the total utilization of virtual-core servers on the physical core. Then, when the remaining utilization of a physical core is changed due to the virtual core creation or the virtual core adjustment, the proposed admission control mechanism will grant the corresponding request if the new remaining utilization of the physical core is no less than 0; otherwise, the request should be rejected to prevent from harming the schedulability of the system.

While the physical cores have the capability of DVS, the hypervisor of a power-aware virtualization system should be able to adjust the operating frequencies of physical cores for energy saving. In our virtualization system, we exploit the DVS capability of physical cores by integrating a DVS scheduling policy with the admission control mechanism. As a result, once our admission control mechanism is triggered, it will also invoke the DVS scheduling policy if the physical cores support DVS. Given a remaining utilization of a physical core, our DVS scheduling policy will scale down the operating frequency of the physical core to the lowest available frequency of the physical core such that the remaining utilization of the physical core is no less than 0, and will only ask the admission control mechanism to reject the request if the remaining utilization is less than 0 even if the physical core operates at the highest available frequency.

3.3 Implementation Remarks

As we only demonstrate our design of the power-aware virtualization system with a single physical platform so far in this paper, some extensions through slight modification of our design will be introduced to cope with some implementation issues in the following of this section. The extensions include (1) the support of multiple physical cores and (2) the consideration of implementation overhead.

When there is only one physical core in the system, we will map multiple virtual cores into it. We can extend the same mapping technique when there are multiple physical cores in the system. Initially, we maintain the system remaining utilization for each physical core. Then according to the required utilization of the virtual core, the hypervisor will assign each virtual core to run on a physical core based on the best-fit policy. More specifically, suppose there are n physical cores in the system. At each time we want to assign a virtual core to a physical, the hypervisor will compute the system remaining utilization of each physical core if the virtual core is assigned to it and assign the virtual core to the physical core with the minimum system remaining utilization after the assignment. In addition, the hypervisor also needs to combine the migration operation and admission control mechanism when the virtual core adjustment function is called. If the function increases the operating frequency of the virtual core and causes the system remaining utilization less than 0 when the physical core operates at the highest available operating frequency, the hypervisor will find a physical core with the enough system remaining utilization in the system and migrate the virtual core to it. Otherwise, the admission control mechanism will reject the request of the adjustment. When there exists a physical core with constant operating frequency in the system, we can replace the best-fit policy with assigning the maximum amount of workload on it.

Hence, our virtualization system is flexible for multiple physical cores platforms. We can further develop policies for virtual cores assignment and migration to achieve better system throughput or reduce the total energy consumption.

From the perspective of the implementation, the system will introduce some implementation overhead: *management overhead* and *context switch overhead*. The management overhead O_m includes the overhead resulting from the duties of the hypervisor, such as virtual core maintenance, computing power dispatching, and memory protection. The unit of O_m is a fixed utilization added into the utilization of each virtual core. The context switch overhead O_s stands for the overhead resulting from the context switch among executions of different virtual cores. The unit of O_s is a fixed number of execution cycles added into the maximum budget of each virtual core. When we consider the implementation overhead in the virtual-core server model, we need to modify the setting of CBS-HR server.

For a virtual core with parameters (C_i, T_i, F_i) , the maximum budget C_i needs to be added by extra execution cycles from implementation overhead and it will become $C'_i = C_i + O_s + O_m \cdot T_i \cdot F_i$. The replenish period T_i will keep the same value without any modification. Hence, the total utilization needed for this virtual core will increase and be affected by the granularity of the CBS-HR server which is the length of the replenish period. The user has to determine the timing constraint of virtual cores such as the tolerable response time delay with the implementation overhead consideration. It is important for providing accurate execution time in our virtualization system.

Chapter 4

Performance Evaluation

This section presents the implementation of our proposed power-aware virtualization system described in Section 3. Then we present some experimental results on it to demonstrate its ability to support virtual cores with the capability of DVS.

4.1 System Implementation

In order to provide accurate timing information in our power-aware virtualization system, we choose the L4 μ -kernel [28] as the hypervisor to implement our virtual-core server model. The L4 μ -kernel is a kind of pre-virtualization system [26] and expands its domain towards embedded systems. The guest operating systems upon the μ -kernel run as user-level applications and have a comparable performance to the native operating systems [9]. Currently, there are numerous implementations of L4 μ -kernels. In this paper, we choose the DROPS (Dresden Real-Time Operating Systems Project) [16] project as the fundamental virtualization system to implement

our system.

In the DROPS project, the hypervisor is composed of the L4:Fiasco μ -kernel [17] and L4Env [18]. Our implementation consists of the virtual-core server model supported in L4:Fiasco and an extension of application programming interface (API) for the virtual core functions in L4env. The following figure shows the system architecture of our virtual-core server model in DROPS.

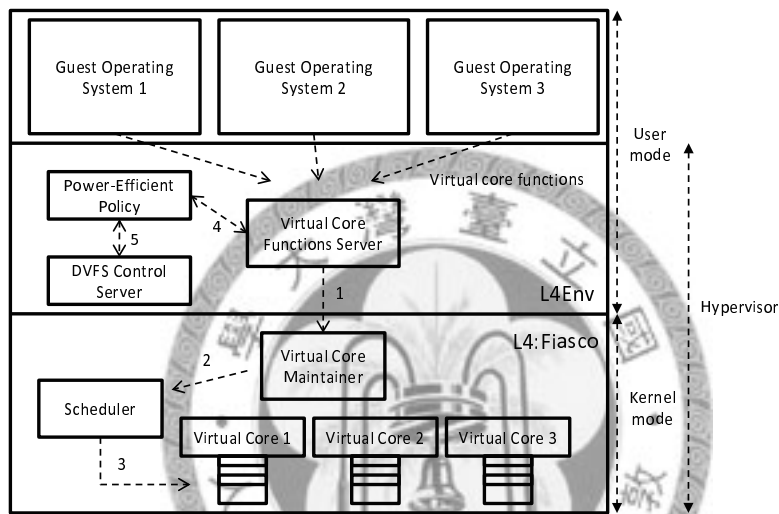


Figure 4.1: The Architecture of Our Virtual Core Model.

In Figure 4.1, there are three virtual cores maintained in the system and each of them has a single guest operating system running on it. Because the minimum execution element of L4:Fiasco is a thread with a priority and the scheduler is a priority-driven scheduler with $O(1)$ complexity, we will group several threads into a virtual core and modify the scheduling scheme of L4:Fiasco to schedule each virtual core according to their deadlines. We implement a *Virtual Core Maintainer* inside L4:Fiasco which maintains the behavior of each virtual core and a *Virtual Core Functions Server* inside L4Env which receives the requests of virtual core functions from the guest operating systems. When the virtual core functions server receives a request and accepts it according to the admission control mechanism, it will send the

virtual core control message to L4:Fiasco, i.e., the message labeled 1 in Figure 4.1. In L4:Fiasco, the virtual core maintainer then receives the virtual core control message. It is an abstraction which support the functionality of an EDF scheduler which gives a higher priority to a virtual core according with earlier absolute deadline. The virtual core maintainer maintains the ready queue for each virtual core which contains all ready threads of that virtual core and sends the index of the virtual core with highest priority to the scheduler of L4:Fiasco with the message labeled 2 in Figure 4.1. The scheduler will schedule the thread with the highest priority in the ready queue of that virtual core through the control message labeled 3 in Figure 4.1.

With the implementation of the virtual-core server model, we then build the *Energy-Efficient Policy* and the *DVS Control Server* inside L4Env. The DVS control server is a device driver used to adjust the DVS setting of the physical core in the system. The energy-efficient policy will determine whether to change the DVS setting according to the requests in virtual core functions server. The messages labeled 4 and 5 in Figure 4.1 show the interactions among the energy-efficient policy, the DVS control server and the virtual core functions server.

4.2 Experimental Setup

We conducted all experiments on a Davinci evaluation board from Texas Instruments [14] including an ARM926ejs core with 256 MBytes of memory. We counted the number of processor cycles using a 64-bit hardware timer on the board. For power measurement, there exists 2 test pins on the board which can be applied to measure the power consists of ARM side, DSP side and so on. We used the device *Agilent 34970A* to get the power trace in a millisecond granularity. The capability of DVS

Frequency(MHz)	Power(mW)
297	32.40
283.5	30.93
270	29.45
256.5	27.98
243	26.51
229.5	25.04
216	23.56
202.5	22.09

Table 4.1: DVS Level Support in Davinci on ARM side.

Guest OS	Utilization	The Behavior
L4Linux	50%	Booting sequences
μ C/OSII	25%	Mathematics program
μ C/OSII	25%	Sorting program

Table 4.2: The Behavior of Virtual Cores.

supported on the ARM side in the Davinci evaluation board is shown in Table 4.1.

The hypervisor we adopted is L4:Fiasco 1.2 and L4Env V0.2. We revised L4:Fiasco to support the virtual-core server model as shown in Figure 4.1. The guest operating systems we adopted for demonstration are L4Linux 2.6.29-14 [19] and μ C/OS-II where L4Linux is a port of the Linux kernel to the L4 μ -kernel and μ C/OS-II is revised to the pre-virtualization version for running on the L4 μ -kernel.

4.2.1 Experimental Results

Our experiments measured the power consumption of the physical core when we apply an energy-efficient policy in the system. There are three virtual cores which running a L4Linux and two μ C/OS-II in the system. Their behavior is shown in Table 4.2. In L4Linux, the booting sequence includes the load of Linux kernel until the shell prompt gets ready. In μ C/OS-II, the mathematics program is a set of

programs which calculate several numbers with addition and subtraction operations. The sorting program sorts 1000000 numbers with bubble sort.

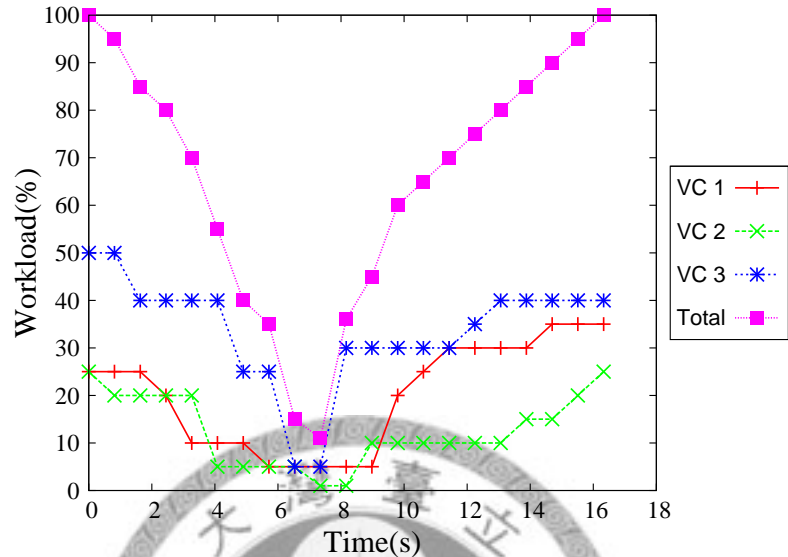


Figure 4.2: The Trend of Virtual Cores in The System.

During the execution of the programs, we used virtual core adjustment function to change the utilization of each virtual cores. The trend of our workload adjustment is shown in Figure 4.2. The system remaining utilization (the total workload in Figure 4.2) will decrease in the beginning then increase. Our energy-efficient policy checked the system remaining utilization and scaled the physical operating frequency to the one which satisfied the user requirement as presented in Section 3.2. It will be triggered at each time where the virtual core adjustment function is invoked.

As shown in the power trace in Figure 4.3, with our virtual core model, our proposed system can scale the physical operating frequency according to the system remaining utilization to reduce the power consumption. The physical power consumption will decrease from 0.65 to 0.48 mW in the beginning and increase at 12 second. It had no changes during 8 to 12 second because the physical core already

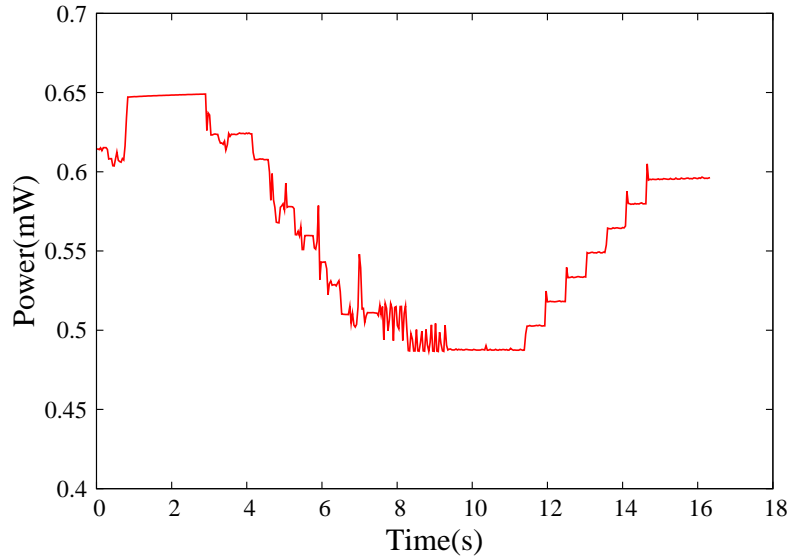


Figure 4.3: The Power Trace.

operated at the lowest operating frequency. Compared to the system without using energy-efficient policy, we can find that the energy consumption is reduced by 13.47% (,i.e, from 10.58mJ to 9.16mJ).

L4:Fiasco Type	Original	Virtual Core Model
Execution Time(ms)	8536.08	8948.85

Table 4.3: Overhead of Virtual-Core Server Implementation.

Function Type	Creation	Deletion	Adjustment
Execution Time(ms)	6.61	6.60	7.13

Table 4.4: Overhead of Virtual Core Functions.

The implementation of our virtual-core server model will introduce some extra computing cycles in the virtualization system. We measured the performance overhead for the execution time in the millisecond scale. We used the same setting in Table 4.2 and measured the execution time until the L4Linux finished its booting sequence. Table 4.3 shows the overhead which is only 412.77 milliseconds. So the implementation overhead is less than 5% in terms of the system performance. In

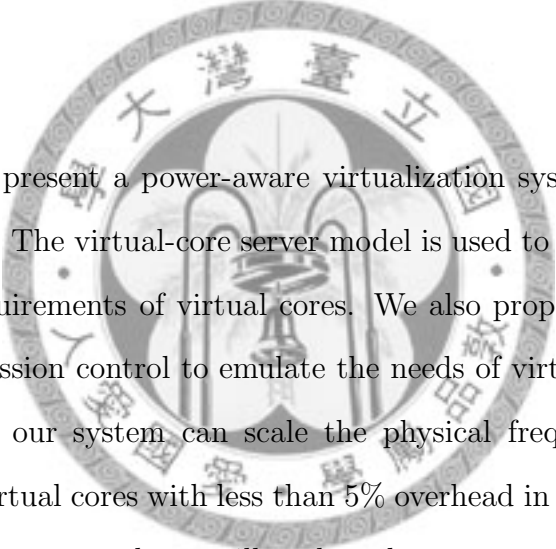
addition, we also measured the overhead of each virtual core function. As shown in Table 4.4, the required execution times for virtual core creation/deletion function and virtual core adjustment function are about 6ms and 7ms, respectively.





Chapter 5

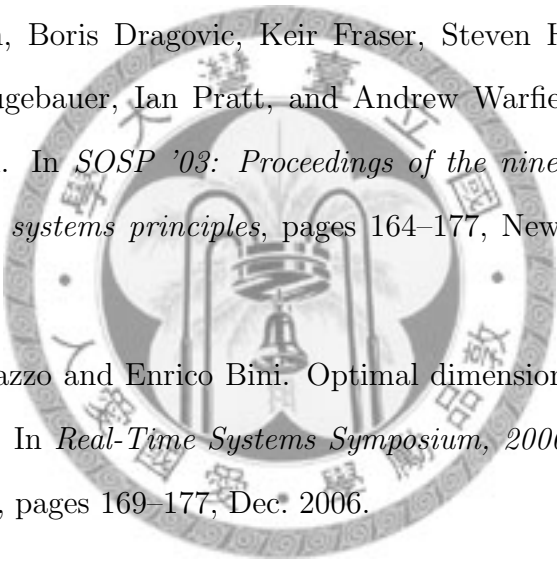
Conclusions



In this paper, we present a power-aware virtualization system based on a virtual-core server model. The virtual-core server model is used to map the DVS needs and guarantee the requirements of virtual cores. We also propose a scheduling mechanism and an admission control to emulate the needs of virtual cores. Experimental results show that our system can scale the physical frequency according to the requirements of virtual cores with less than 5% overhead in terms of the system performance. For future research, we will explore the energy management mechanism of other hardware components for virtualization, such as main memory. Moreover, in the multi-core extension, we will also study the management and resource allocation designs that map a single virtual core to multiple physical cores.



Bibliography

- 
- [1] L. Abeni and G. Buttazzo. Integrating multimedia applications in hard real-time systems. *Real-Time Systems Symposium, IEEE International*, 0:4, 1998.
- [2] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. Xen and the art of virtualization. In *SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles*, pages 164–177, New York, NY, USA, 2003. ACM.
- [3] Giorgio Buttazzo and Enrico Bini. Optimal dimensioning of a constant bandwidth server. In *Real-Time Systems Symposium, 2006. RTSS '06. 27th IEEE International*, pages 169–177, Dec. 2006.
- [4] Karim Chine. Biocep, towards a federative, collaborative, user-centric, grid-enabled and cloud-ready computational open platform. In *ESCIENCE '08: Proceedings of the 2008 Fourth IEEE International Conference on eScience*, pages 321–322, 2008.
- [5] John Criswell, Andrew Lenharth, Dinakar Dhurjati, and Vikram Adve. Secure virtual architecture: a safe execution environment for commodity operating systems. In *SOSP '07: Proceedings of twenty-first ACM SIGOPS symposium on Operating systems principles*, pages 351–366, New York, NY, USA, 2007. ACM.

- [6] Z. Deng and J. W.-S. Liu. Scheduling real-time applications in an open environment. In *RTSS '97: Proceedings of the 18th IEEE Real-Time Systems Symposium*, page 308, Washington, DC, USA, 1997. IEEE Computer Society.
- [7] Z. Deng, J.W.-S. Liu, and J. Sun. A scheme for scheduling hard real-time applications in open system environment. In *Real-Time Systems, 1997. Proceedings., Ninth Euromicro Workshop on*, pages 191–199, Jun 1997.
- [8] Pawel Garbacki and Vijay K. Naik. Efficient resource virtualization and sharing strategies for heterogeneous grid environments. In *Integrated Network Management*, pages 40–49, 2007.
- [9] Hermann Härtig, Michael Hohmuth, Jochen Liedtke, Jean Wolter, and Sebastian Schönberg. The performance of μ -kernel-based systems. In *SOSP '97: Proceedings of the sixteenth ACM symposium on Operating systems principles*, pages 66–77, New York, NY, USA, 1997. ACM.
- [10] G. Heiser. Hypervisors for consumer electronics. In *Consumer Communications and Networking Conference, 2009. CCNC 2009. 6th IEEE*, pages 1–5, Jan. 2009.
- [11] Gernot Heiser. The role of virtualization in embedded systems. In *IIES '08: Proceedings of the 1st workshop on Isolation and integration in embedded systems*, pages 11–16, New York, NY, USA, 2008. ACM.
- [12] Gernot Heiser, Kevin Elphinstone, Ihor Kuz, Gerwin Klein, and Stefan M. Petters. Towards trustworthy computing systems: taking microkernels to the next level. *SIGOPS Oper. Syst. Rev.*, 41(4):3–11, 2007.
- [13] <http://ertos.nicta.com.au/research/sel4/>. Secure microkernel project.

- [14] <http://focus.ti.com/docs/toolsw/folders/print/tmdxevm6446.html>. Dm6446 digital video evaluation module - tmdxevm6446.
- [15] <http://java-virtual-machine.net/>. Java virtual machine - all about jvms.
- [16] <http://os.inf.tu-dresden.de/drops/>. Drops - the dresden real-time operating system project.
- [17] <http://os.inf.tu-dresden.de/fiasco/>. The fiasco microkernel.
- [18] <http://os.inf.tu-dresden.de/l4env/>. L4 environment.
- [19] <http://os.inf.tu-dresden.de/L4/LinuxOnL4/>. L4linux - running linux on top of l4.
- [20] <http://os.inf.tu-dresden.de/vfiasco/>. the vfiasco project.
- [21] <http://www.microsoft.com/windows/virtual-pc/>. Windows virtual pc.
- [22] <http://www.ok-labs.com/products/okl4>. Okl4 : Open kernel labs.
- [23] <http://www.virtualbox.org/>. Virtualbox.
- [24] <http://www.virtuallogix.com/>. Virtuallogix - real-time virtualization for connected devices.
- [25] Tei-Wei Kuo and Ching-Hui Li. A fixed-priority-driven open environment for real-time applications. In *Real-Time Systems Symposium, 1999. Proceedings. The 20th IEEE*, pages 256–267, 1999.
- [26] Joshua LeVasseur, Volkmar Uhlig, Matthew Chapman, Peter Chubb, Ben Leslie, and Gernot Heiser. Pre-virtualization: soft layering for virtual machines. Technical Report 2006-15, Fakultät für Informatik, Universität Karlsruhe (TH), July 2006.

- [27] Joshua Levasseur, Volkmar Uhlig, Jan Stoess, and Stefan Götz. Unmodified device driver reuse and improved system dependability via virtual machines. In *Proceedings of the 6th Symposium on Operating Systems Design and Implementation*, pages 17–30, 2004.
- [28] J. Liedtke. On micro-kernel construction. *SIGOPS Oper. Syst. Rev.*, 29(5):237–250, 1995.
- [29] C. L. Liu and James W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM*, 20(1):46–61, 1973.
- [30] Antonio Mancina, Giuseppe Lipari, Jorrit N. Herder, Ben Gras, and Andrew S. Tanenbaum. Enhancing a dependable multiserver operating system with temporal protection via resource reservations. In *Proc. 16th International Conference on Real-Time and Network Systems (RTNS'08)*, Rennes, France, oct 2008.
- [31] L. Marzario, G. Lipari, P. Balbastre, and A. Crespo. Iris: a new reclaiming algorithm for server-based real-time systems. In *Real-Time and Embedded Technology and Applications Symposium, 2004. Proceedings. RTAS 2004. 10th IEEE*, pages 211–218, May 2004.
- [32] Yu Murata, Wataru Kanda, Kensuke Hanaoka, Hiroo Ishikawa, and Tatsuo Nakajima. A study on asymmetric operating systems on symmetric multiprocessors. In *EUC*, pages 182–195, 2007.
- [33] Ripal Nathuji and Karsten Schwan. Virtualpower: coordinated power management in virtualized enterprise systems. In *SOSP '07: Proceedings of twenty-first ACM SIGOPS symposium on Operating systems principles*, pages 265–278, New York, NY, USA, 2007. ACM Press.

- [34] Sergio Ruocco National and Sergio Ruocco. Real-time programming and 14 microkernels. In *In Proceedings of the 2006 Workshop on Operating System Platforms for Embedded Real-Time Applications*, 2006.
- [35] Raj Rajkumar, Kanaka Juvva, Anastasio Molano, and Shuichi Oikawa. Resource kernels: A resource-centric approach to real-time and multimedia systems. In *In Proceedings of the SPIE/ACM Conference on Multimedia Computing and Networking*, pages 150–164, 1998.
- [36] Ashley Saulsbury. Ultrasparc virtual machine specification, 2008.
- [37] Arvind Seshadri, Mark Luk, Ning Qu, and Adrian Perrig. Secvisor: a tiny hypervisor to provide lifetime kernel code integrity for commodity oses. In *SOSP '07: Proceedings of twenty-first ACM SIGOPS symposium on Operating systems principles*, pages 335–350, New York, NY, USA, 2007. ACM.
- [38] A.A. Soror, A. Aboulnaga, and K. Salem. Database virtualization: A new frontier for database tuning and physical design. In *Data Engineering Workshop, 2007 IEEE 23rd International Conference on*, pages 388–394, April 2007.
- [39] Jan Stoess, Christian Lang, and Frank Bellosa. Energy management for hypervisor-based virtual machines. In *ATC'07: 2007 USENIX Annual Technical Conference on Proceedings of the USENIX Annual Technical Conference*, pages 1–14, Berkeley, CA, USA, 2007. USENIX Association.
- [40] Proceedings Of The, Jeremy Sugarman, Ganesh Venkitachalam, Beng hong Lim, and VMware Inc. Virtualizing i/o devices on vmware workstation's hosted virtual machine monitor, 2001.
- [41] VMWare Inc. *VMWare ESX Server Performance and Resource Management for CPU-Intensive Workloads*, 2005.

- [42] Yefu Wang, Xiaorui Wang, Ming Chen, and Xiaoyun Zhu. Power-efficient response time guarantees for virtualized enterprise servers. In *Real-Time Systems Symposium, 2008*, pages 303–312, 30 2008-Dec. 3 2008.

