http://dx.doi.org/10.14400/JDC.2014.12.3.227

Real-Time Scheduling Method to assign Virtual CPU in the Multocore Mobile Virtualization System

Yongho Kang, Kimoon Keum, Seongjong Kim, Kwangyoun Jin*, Jooman Kim**

Pusan National University., **R2SOFT LTD, *Kwangwon National University.

멀티코아 모바일 가상화 시스템에서 가상 CPU 할당 실시간 스케줄링 방법

강용호, 금기문**, 김선종, 진광윤*, 김주만

부산대학교 IT응용공학과, (주)알투소프트**,강원대학교 컴퓨터과학과*

Abstract Mobile virtualization is an approach to mobile device management in which two virtual platforms are installed on a single wireless device. A smartphone, a single wireless device, might have one virtual environment for business use and one for personal use. Mobile virtualization might also allow one device to run two different operating systems, allowing the same phone to run both RTOS and Android apps. In this paper, we propose the techniques to virtualize the cores of a multicore, allowing the reassign any number of vCPUs that are exposed to a OS to any subset of the pCPUs. And then we also propose the real-time scheduling method to assigning the vCPUs to the pCPU. Suggested technology in this paper solves problem that increases time of real-time process when interrupt are handled, and is able more to fast processing than previous algorithm.

Key Words : Mobile Virtualization, Real-time scheduling, Hypervisor Scheduling Algorithm, Multicore

요 약 모바일 가상화는 두 개의 가상 플랫폼을 하나의 무선 장치에 탑재하는 모바일 장치 관리의 한 접근 방법이다. 단일 무선 장치인 스마트폰은 사업용과 개인용으로의 가상 환경으로 사용될 수 있을 것이다. 모바일 가상화는 또한 동일한 장치에 두 개의 운영체제인 RTOS와 안드로이드 앱이 동시에 수행되는 환경일 수 있다. 본 논문에서는 멀티코아에서 각 코아를 가상화하고, 물리 CPU(pCPUs)에 배당된 여러 가상 CPU(vCPU)를 재 할당하는 기법을 제시하며 또한 가상 CPU들을 물리 CPU에 할당하기 위한 실시간 스케줄링 방법을 제안한다. 본 논문에서 제안된 기술은 인터럽트 처리시에 실시간 처리의 시간 지연을 해결하였고, 이전의 알고리즘보다 빠른 처리를 가능하게 한다.

주제어 : 모바일 가상화, 실시간 스케줄링, 하이퍼바이저 스케줄링 알고리즘, 멀티코아

1. Introduction

Recently, the growing number of smart phones during the last years has also led to an increased number of mobile platforms where different manufacturers provides their own platform such as

Received 3 January 2014, Revised 5 February 2014
Accepted 20 March 2014
Corresponding Author: Jooman Kim
(Pusan National University)
Email: joomkim@pusan.ac.kr

ISSN: 1738-1916

Windows Mobile, SymbianOS, Linux and IPhone OS, Android[2,3]. As performance of smartphone device has been increased, previous virtualization technology can be applied to the mobile phone with a hypervisor called virtual machine monitor(VMM)[1,2,3]. In this mobile virtualization environment, scheduling technology is required to perform a virtual CPU.

In Multi-core system equipped with two cores or more(multiple physical CPU), in order to run multiple operating systems, it was needed that a technology of virtualization scheduling which perform tasks which virtual CPU was assigned. In other words, hypervisor create multiple virtual CPU by virtualizing a physical CPU, operating system is performed by using a virtualized CPU in it. In this process, scheduler in hypervisor is needed in order to allocate efficiently virtual CPU to the operating system. Previous studies about mobile virtual scheduling are most about how to assign multiple virtualized CPU on a single physical CPU.[7][8] To these multiple virtual CPU assigned to one physical CPU require a policy which keep physical CPU resources equitably and maintain appropriately load balance. To determine how much allocate virtual CPU to each domain, it should be set the utilization ratio of physical CPU to hypervisor, scheduler assigns virtual CPU to the domain according to the utilization ratio.

Thus, fast interrupt processing requires the information about load balance and fairness. According to these calculations, hypervisor select the virtual CPU which have the highest weight-value, and assign it to physical CPU. The selected virtual CPU can be interrupt or not. In this situation, because delay-time of weight-value calculation is occurred, it does not guarantee real-time process. In mobile virtualization system which uses a separate I/O driver models generally, interrupts occurred in a guest domain are performed in the domain 0 which should handle privileged mode. This brings about a late execution time, when the interrupt are occurred, because they should execute relevant interrupt in the domain 0 once again.

In this way which execute virtual CPU on a single processor, it need a twice execution time of context-switching due to pass a virtual CPU running in domain 0 to a ready queue and schedule virtual CPU for handling the interrupt. So real-time processing is difficult. However, as performance of mobile devices is developing, research by using the multi-core processors scheduling is needed for minimizing the context switching and handling interrupt fast.

Therefore, in order to ensure real-time processing, we suggest technology that virtual CPU is scheduled immediately without delay time for calculating fairness and load balance by granting highest priority on it when interrupt handles, and method for selecting and assigning the best physical CPU among multiple physical CPUs which can be assigned. Suggested technology solves the problem that the real-time processing is increased during the interrupt processing, and is able more to fast execute processing than previous algorithm.

2. Related Works

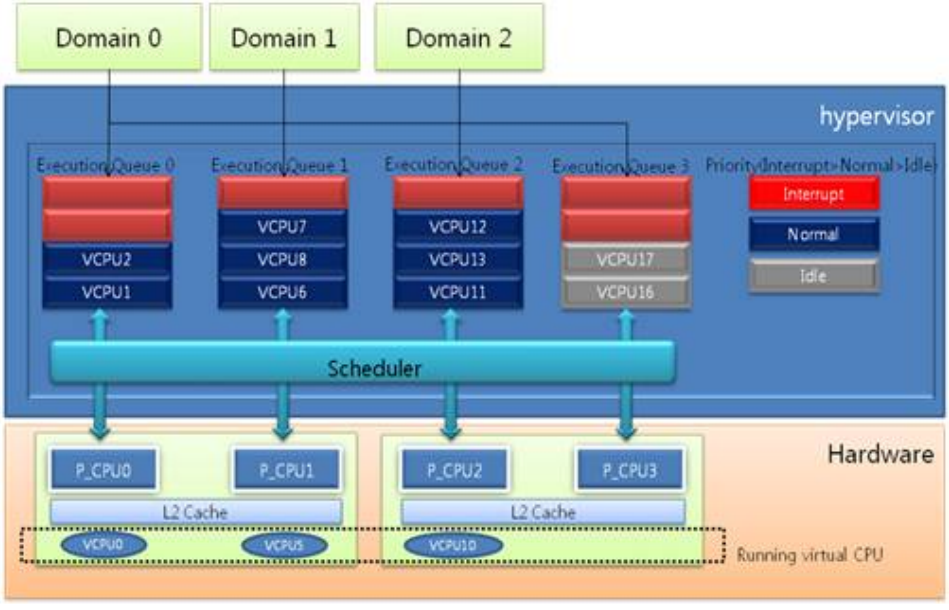
The Xen's credit-based CPU scheduler is a proportional fair share CPU scheduler built from the ground up to be work conserving on SMP hosts. It is now the default scheduler in the xen-unstable trunk. The SEDF and BVT schedulers are still optionally available but the plan of record is for them to be phased out and eventually removed. Each domain (including Host OS) is assigned a weight and a cap[1,2,3,4]. A domain with a weight of 512 will get twice as much CPU as a domain with a weight of 256 on a contended host. Legal weights range from 1 to 65535 and the default is 256. Each CPU manages a local run queue of runnable VCPUs. This queue is

sorted by VCPU priority. A VCPU's priority can be one of two value: over or under representing whether this VCPU has or hasn't yet exceeded its fair share of CPU resource in the ongoing accounting period. When inserting a VCPU onto a run queue, it is put after all other VCPUs of equal priority to it. As a VCPU runs, it consumes credits. Every so often, a system-wide accounting thread recomputes how many credits each active VM has earned and bumps the credits. Negative credits imply a priority of over. Until a VCPU consumes its alloted credits, it priority is under. On each CPU, at every scheduling decision (when a VCPU blocks, yields, completes its time slice, or is awaken), the next VCPU to run is picked off the head of the run queue. The scheduling decision is the common path of the scheduler and is therefore designed to be light weight and efficient. No accounting takes place in this code path. When a CPU doesn't find a VCPU of priority under on its local run queue, it will look on other CPUs for one. This load balancing guarantees each VM receives its fair share of CPU resources system-wide. Before a CPU goes idle, it will look on other CPUs to find any runnable VCPU. This guarantees that no CPU idles when there is runnable work in the system.

Like Zhou proposed the scheduling problems in virtualized environment, and find existing CPU scheduling mechanisms do not fit for PSRT applications. Aiming at both the soft real-time constraints and synchronization problems, they present the parallel soft real-time scheduling algorithm, and implement a prototype based on Xen, named Poris. Poris introduces real-time priority, changes the time slice, and schedules all the VCPUs of a RT-VM at the same time. If there is no RT-VM in the system, Poris turns into the Credit scheduler, which can minimize the impact on nonreal- time VMs[4].

3. System Architecture

Figure 1 shows the physical CPU(PCPU0 ~ PCPU3) is managed in Hypervisor scheduler, cache(L2 Cache of P_CPU0 and P_CPU1, L2 Cache of P_CPU2 and P_CPU3) is shared by each physical CPU, the operating system (domain0~domain2) is running by being virtualized, Virtual CPU(VCPU0 ~ VCPU17)



[Fig. 1] Scheduling of Mobile Multi-Core Virtualization System

which is virtualized for processing of each domain, execution queue list. In the execution queue, a virtual CPU has been ordered according to the highest priority.

And each virtual CPU has the one priority among 'Interrupt', 'Normal', 'Idle'. If they have same priority, each in front of queue has the high priority. If the status of physical CPU is "Idle", priority of the virtual CPU which is assigned to the physical CPU is set to "Idle". When virtual CPU wake up from status of I/O operation requests and resource lock, priority of the virtual CPU is the highest("Interrupt"). Each physical CPU is assigned by virtual CPU which is lined in the execution queue of each scheduling waiting list. Sometimes, it can be migrated to virtual CPU in the 'execution queue' in other physical CPU and allocated physical CPU. At this time, the policy is needed to guarantee the Fairness and Load Balance of each virtual CPU. In this paper, as the policy to guarantee the fairness and load balance, it is maintained through the number of implemented virtual CPU in each execution queue.

In general, when interrupt occurs, the virtualized system which uses separated I/O driver model separates into guest domains (domain 1 and domain 2) on which a front-end driver is located and the domain 0 on which a back-end driver is located. I/O related interrupts are carried out in the domain 0 because the interrupts which occur in the guest domains request it to a back-end driver in the domain 0 through a front-end driver.

Therefore, if the I/O related interrupts occur in domain 1, a virtual CPU which runs in the correspondent physical CPU stops operation and is included in 'waiting queue' until the interrupt is proceeded. It requests I/O execution in domain 0.

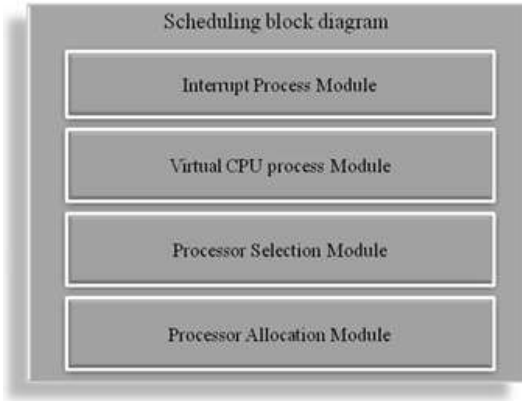
In domain 0, include virtual CPU into the execution queue of domain 0 to carry out the interrupt which occurred in domain 1.

In the existing scheduling method, these kinds of interrupt is carried by scheduler which calculates weighted value and select the highest weighted value of virtual CPU in each execution queue considering Fairness and load balance of physical CPU and then assign physical CPU. It causes time delay of weighted value calculation. It also cannot guarantee virtual CPU of interrupt execution is assigned in physical CPU. This process has a weak it cannot be used in which real time process is requested such as mobile devices and embedded devices.

In multi-core processor, one cache(L2 Cache) is shared in every two cores, in general. This shared cache is used to overcome the speed difference with other devices by previously storing data which CPU needs while the CPU is processing data. As the two cores store frequently used data in L2 Cache, performance can be improved.

Figure 2 shows scheduling block diagram of mobile multi-core virtual system suggested in this paper. In this figure, scheduler of mobile multi-core virtual system includes Interrupt Process Module, Virtual CPU Process Module, Processor Selection Module, and Processor Allocation Module.

Interrupt Process Module investigates and detects interrupts occurred when each virtual CPU is processed and judge correspondent interrupt type. Virtual CPU Process Module moves virtual CPU to waiting queue and execution queue and give priority



[Figure 2] The scheduling block diagram of mobile virtualization system

<Table 1> Information of Physical CPU

CPU	Interrupt	Recently used Virtual CPU	Status	Shared Cache	Domain	Execution Queue
P_CPU0	No	VCPU0	running	0	0	0
P_CPU1	Yes	VCPU5	waiting	0	1	1
P_CPU2	No	VCPU10	running	1	2	2
P_CPU3	No	-	Idle	1	0	3

to process the interrupt. Processor Selection Module decides which physical CPU should be allocated to allocate selected virtual CPU in each physical CPU. For the decision, the Information of Physical CPU can be used as Table 1.

Table 1 shows physical CPU information to determine what the physical CPU will choice for assigning selected virtual CPU to them., CPU field is the name of the CPU, an interrupt field contains information about if the interrupt has been bring. Recently used virtual CPU field contains virtual CPU information recently performed at each physical CPU. The Status field has the current status information of the physical CPU(running, waiting, Idle value). Field of Shared cache is cache information shared at each physical CPU, domain field is for information of the domain which each physical CPU is processing. Execution queue field means number of queue.

In the Table 1, P_CPU0 with value 0 and P_CPU1 share the information inside of cache. domain field is the information of domain where each physical CPU is processing. P_CPU0 and P_CPU3 execute virtual CPU about domain 0. Execution queue field indicates the waiting queue number to execute in each physical CPU.

In the physical CPU selection, it can happen in several cases.

In the first case, allocable physical CPU is one. The physical CPU which was executing virtual CPU stops execution and moves to waiting queue if interrupt occurs. To carry out the interrupt, the physical CPU exists in the state of holding for a moment. Therefore, the physical CPU is waiting to get allocated a new virtual CPU. This is the case that interrupt process virtual CPU is allocated in the physical CPU.

The second case is that the number of allocable physical CPU is more than two. For example, physical CPU which was executing virtual CPU with interrupt and a physical CPU which is in the Idle state can execute new virtual CPU. Even though a physical CPU, which is in the Idle state, temporarily stops execution, it can use the physical CPU through changing of correspondent domain.

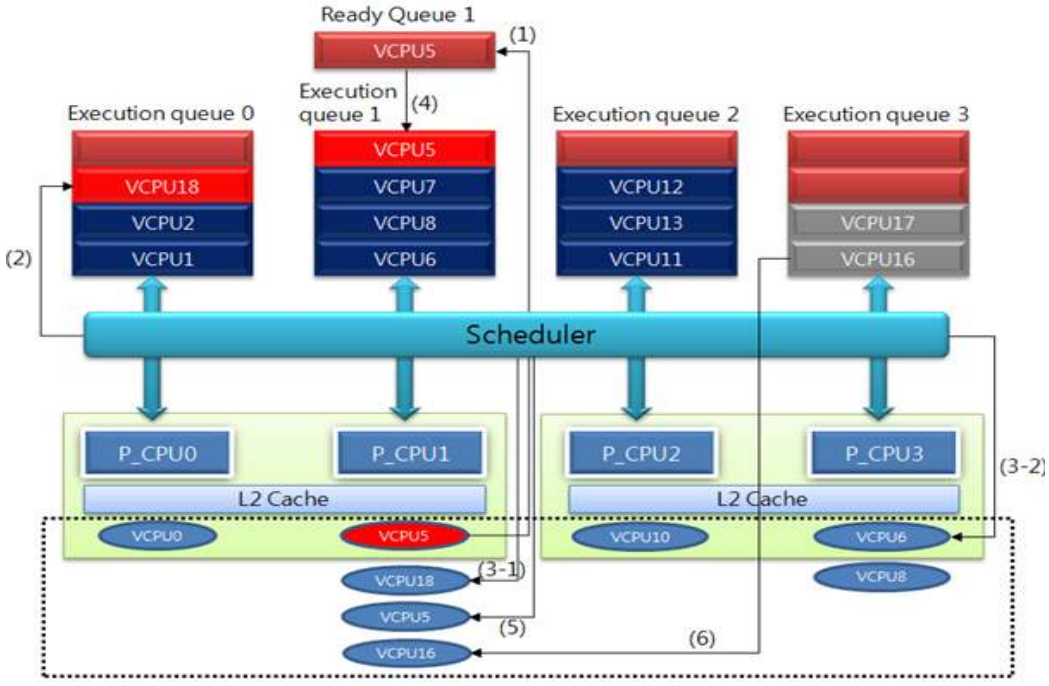
For example in the Table 1, according to the information of the physical CPU, the physical CPU with interrupt is P_CPU1 and P_CPU3 is in the Idle state. Therefore, currently available physical CPU is P_CPU1 which is in waiting state and P_CPU3 which is in Idle state. When we consider shared cache and the information of recently available virtual CPU, it is the best to allocate in P_CPU1. After that, even though the state of P_CPU3 is Idle, the target is about domain 0. Because P_CPU3 can use shared cache such as P_CPU0 of domain 0, the process of virtual CPU(VCPU6), which is waiting in P_CPU1, changes into P_CPU3 and the process of virtual CPU, which is waiting in P_CPU3, changes to be processed in P_CPU1 and the information of physical CPU is also changed.

The following is the information after the selection of each physical CPU is performed in the above Table.

For the information of considering processor selection, physical CPU in the Idle state, physical CPU with interrupt, physical CPU with the least Time Slice, and/or physical CPU with the most time slice.

<Table 2> Information of Physical CPU

CPU	Interrupt	Recently used Virtual CPU	Status	Shared Cache	Domain	Execution Queue
P_CPU0	No	VCPU0	running	0	0	0
P_CPU1	No	VCPU5	waiting	0	0	3
P_CPU2	No	VCPU10	running	1	2	2
P_CPU3	No	VCPU6	running	1	1	1



[Fig. 3] Result of scheduling according to interrupt occurrence

processor allocation module(140) allocates virtual CPU which is selected in selected processor. It allocates virtual CPU which was selected in the Virtual CPU process module(120) into the physical CPUs which was selected in the Processor Selection Module(130).

4. Operational Verification

The Figure 3 shows the series of process upon the interrupt when the virtual CPU(VCPU5) is executed in the physical CPU(P_CPU1) based on the physical CPU information of the Table 1. In order to handle the interrupt, the virtual CPU(VCPU5) is moved to virtual queue (1) and insert the virtual CPU(VCPU18), which is related to interrupt handling process, into execution queue of the physical CPU(P_CPU0), which is executing domain 0 (2). The scheduler immediately searches available physical CPU using the information of the physical CPU in the domain 3 to allocate the virtual CPU(VCPU18) into the physical CPU. The scheduler allocates VCPU18 into P_CPU1 according to the selection result (3-1).

In addition, even though P_CPU3 is in the Idle state, it is the CPU allocated by domain 0. In order to increase its availability, P_CPU3 allocates VCPU6, which is not allocated because of interrupt handling of VCPU5, into P_CPU3 (3-2). At this time, the scheduler

changes the information of physical CPU to allocate the virtual CPU, which exists in the execution queue of P_CPU1, into P_CPU1, and to allocate the virtual CPU, which exists in the execution queue of PCPU3, into P_CPU1.

After handling the interrupt, if response interrupt about VCPU5 occurs, VCPU5 in the waiting queue is inserted into execution queue (4). At this time, VCPU5 has the highest priority (Interrupt) and it is instantly executed by the scheduler. The process is the same as the process above. Instead, because the available physical CPU is only the P_CPU1, allocated physical CPU becomes P_CPU1. Finally, the virtual CPU in the execution queue 3 is allocated in P_CPU1 by the information of the physical CPU when the Idle state is withdrawn (6).

5. Conclusion

In this paper, we proposed the real-time scheduling technique that aim to increase a performance of mobile virtualization by high-priority interrupt-driven method in the mobile environment. immediately without delay time for calculating fairness and load balance by granting highest priority on it when interrupt handles, and method for selecting and assigning the best physical CPU among multiple physical CPUs which can be assigned. Suggested technology in this paper solves problem that increases time of real-time process when interrupt are handled, and is able more to fast execute processing than previous algorithm.

REFERENCES

[1] Paul Barham, Boris Dragovic,etc., "Xen and the Art of Virtualization", SOSP'03, 2003

[2] Joo-Young Hwang, Sang-Bum Suh, etc., "Xen on ARM: System Virtualization using Xen Hypervisor for ARM-based Secure Mobile Phones", CCNC, pp257-261, 2008

[3] Henrik Andersson, Joakim Svensson, "VIRTUALIZATION IN A MOBILE ENVIRONMENT AN INTRODUCTION TO PARA-VIRTUALIZATION WITH XEN-ARM", Department of Electrical and Information Technology Lund University

[4] http://wiki.xen.org/wiki/Credit_Scheduler

[5] M. Lee, A. S. Krishnakumar, P. Krishnan, N. Singh, and S. Yajnik, "Supporting soft real-time tasks in the xen hypervisor," in Proc. VEE'10, 2010, pp. 97 - 108.

[6] N. Nishiguchi, "Evaluation and consideration of the credit scheduler for client virtualization," Xen Summit Asia, 2008.

[7] D. Patnaik, A. S. Krishnakumar, P. Krishnan, N. Singh, and S. Yajnik, "Performance implications of hosting enterprise telephony applications on virtualized multi-core platforms," in Proc. IPTComm'09, 2009.

[8] L. Cherkasova, D. Gupta, and A. Vahdat, "Comparison of the three cpu schedulers in xen," SIGMETRICS Perform. Eval. Rev., vol. 35, no. 2, pp. 42 - 51, 2007.

[9] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the art of virtualization," in Proc. SOSP'03, 2003, pp. 164 - 177.

[10] Kernel-based Virtual Machine (KVM) for Linux. http://www.linux-kvm.org.

[11] V. Uhlig, J. LeVasseur, E. Skoglund, and U. Dannowski, "Towards scalable multiprocessor virtual machines," in Proc. VM'04, 2004, pp.43 - 56.

[12] H. Kim, J. Jeong, J. Hwang, J. Lee, and S. Maeng, "Scheduler support for video-oriented multimedia on client-side virtualization," in Proc. MMSys'12, 2012, pp. 65 - 76.

[13] S. Xi, J. Wilson, C. Lu, and C. Gill, "Rt-xen: Towards real-time hypervisor scheduling in xen," in Proc. EMSOFT'11, 2011, pp. 39 - 48.

[14] O. Sukwong and H. S. Kim, “Is co-scheduling too expensive for smp vms?” in Proc. EuroSys’11, 2011, pp. 257 - 272.

[15] C. Weng, Q. Liu, L. Yu, and M. Li, “Dynamic adaptive scheduling for virtual machines,” in Proc. HPDC’11, 2011, pp. 239 - 250.

[16] C. Weng, Z. Wang, M. Li, and X. Lu, “The hybrid scheduling framework for virtual machine systems,” in Proc. VEE’09, 2009, pp. 111 - 120.

[17] I. Molnar, “Linux cfs scheduler,” http://kerneltrap.org/node/11737.

[18] C. Xu, S. Gamage, P. N. Rao, A. Kangarlou, R. R. Kompella, and D. Xu, “vslicer: latency-aware virtual machine scheduling via differentiated frequency cpu slicing,” in Proc. HPDC’12, 2012, pp. 3 - 14.

[19] H. Chen, H. Jin, K. Hu, and J. Huang, “Dynamic switching-frequency scaling: scheduling overcommitted domains in xen vmm,” in Proc. ICPP’10, 2010, pp. 287 - 296.

[20] J. Hwang and T. Wood, “Adaptive dynamic priority scheduling for virtual desktop infrastructures,” in Proc. IWQoS’12, 2012.

[21] D. G. Feitelson and L. Rudolph, “Gang scheduling performance benefits for fine-grain synchronization,” J. Parallel Distrib. Comput., vol. 16, no. 4, pp. 306 - 318, 1992.

[22] R. McDougall, “Filebench: Application level file system benchmark,” http://sourceforge.net/apps/mediawiki/filebench/index.php.

[23] MPlayer. http://www.mplayerhq.hu/.

[24] C. Bienia, “Benchmarking modern multiprocessors,” Ph.D. dissertation, Princeton University, January 2011.

[25] J. Katcher, “Postmark: A new file system benchmark,” Technical Report TR3022, Network Appliance Inc., Tech. Rep., 1997.

[26] Darwin Streaming Server. http://dss.macosforge.org/.

[27] J. H. Anderson and J. M. Calandrino, “Parallel real-time task scheduling on multicore platforms,” in Proc. RTSS’06, 2006, pp. 89 - 100.

[28] S. Kato and Y. Ishikawa, “Gang edf scheduling of parallel task systems,” in Proc. RTSS’09, 2009, pp. 459 - 468.

[29] K. Lakshmanan, S. Kato, and R. Rajkumar, “Scheduling parallel realtime tasks on multi-core processors,” in Proc. RTSS’10, 2010, pp. 259 - 268.

[30] C. Liu and J. Anderson, “Supporting soft real-time dag-based systems on multiprocessors with no utilization loss,” in Proc. RTSS’10, 2010, pp. 3 - 13.

[31 A. Saifullah, K. Agrawal, C. Lu, and C. Gill, “Multi-core real-time scheduling for generalized parallel task models,” in Proc. RTSS’11, 2011, pp. 217 - 226.

강 용 호(Yongho Kang)

- 1994년 2월 : 충남대학교 컴퓨터공학(공학사)
- 1997년 2월 : 충남대학교 컴퓨터공학(공학석사)
- 2000년 2월 : 충남대학교 공학박사 수료
- 2012년 3월 ~ 현재 : 부산대 IT응용공학과 박사과정
- 관심분야 : 클러스터 컴퓨팅, 모바일 클라우드컴퓨팅, 모바일 가상화 및 보안
- E-Mail : kang@r2soft.co.kr

금 기 문(Kimun Keum)

- 1994년 2월 : 충남대학교 컴퓨터공학과(공학사)
- 1996년 2월 : 충남대학교 컴퓨터공학과(공학석사)
- 2004년 1월 : 충남대학교 대학원 전문연구요원
- 2012년 11월 ~ 현재 : ㈜ 알투소프트 이사
- 관심분야 : 가상화, 네트워크, 패턴인식
- E-Mail : kmkeum@gmail.com

김 선 종(Seong-Jong Kim)

- 1989년 2월 : 경북대학교 전자공학(공학사)
- 1991년 2월 : 경북대학교 전자공학(공학석사)
- 1996년 2월 : 경북대학교 전자공학(공학박사)
- 1997년 3월 ~ 현재 : 부산대학교 IT응용공학과 교수
- 관심분야 : 디지털 시스템 설계, 이미지 처리, 패턴인식, 스마트장치 응용
- E-Mail : ksj329@pusan.ac.kr

진 광 윤(Kwangyoun Jin)

- 1984년 2월 : 서울과기대 컴퓨터공학과(공학사)
- 1987년 2월 : 건국대학교 컴퓨터공학과(공학석사)
- 2005년 7월 : 경남대학교 컴퓨터공학과(공학박사)
- 1990년 3월 ~ 현재 : 강원대학교 컴퓨터공학과 교수
- 관심분야 : 정보보안, 임베디드 시스템
- E-Mail : kyjin@kangwon.ac.kr

김 주 만(Jooman Kim)

- 1984년 2월 : 숭실대학교 전자계산학(공학사)
- 1998년 8월 : 충남대학교 컴퓨터공학(공학사)
- 2003년 2월 : 충남대학교 컴퓨터공학(공학박사)
- 1985년 1월 ~ 2000년 2월 : ETRI OS팀장(책임연구원)
- 1995년 7월 ~ 1996년 6월 : 미국 Novell사 객원연구원
- 2000년 3월 ~ 현재 : 부산대학교 IT응용공학과 교수
- 관심분야 : 임베디드 시스템, 실시간 시스템, 클러스터 컴퓨팅, 병렬분산 시스템
- E-Mail : joomkim@pusan.ac.kr