■ 2006년도 학생논문 경진대회 수상작

저전력을 위한 버퍼 캐쉬 관리 기법
(Buffer Cache Management for Low Power Consumption)

이 민[†] 서 의 성[†] 이 준 원[††]
(Min Lee) (Euiseong Seo) (Joonwon Lee)

요 약 컴퓨팅 환경이 무선과 휴대용 시스템으로 변화하면서, 전력효율이 점점 중요해지고 있다. 특히 내장형 시스템일 경우에 더욱 그러한데 이중 메모리에서 소모되는 전력이 전체 전력소모의 두 번째 큰 요소가 되고 있다. 메모리 시스템에서의 전력소모를 줄이기 위해서 SDRAM의 저전력 모드를 활용할 수 있다. RDRAM의 경우 냅모드(nap mode)는 액티브 모드(active mode)의 5%이하의 전력만을 소모한다. 하지만 하드웨어 컨트롤러는 운영체제가 협조하지 않으면 이 기능을 효율적으로 활용하지 못한다. 이 논문에서는 SDRAM의 액티브 유닛(active unit)의 수를 최소화하는 방법에 초점을 맞춘다. 운영체제는 참조되지 않는 메모리를 저전력 모드에 놓음으로써 최소한의 유닛들만을 액티브 모드에 놓은 상태로 프로그램이 수행될 수 있도록 피지컬(physical) 페이지들을 할당한다. 이것은 PAVM(Power Aware Virtual Memory) 연구의 일반화된 시스템 전반에 대한 연구라고 할 수 있다. 우리는 모든 피지컬 메모리를 고려하고 있으며, 특히 평균적으로 전체 메모리의 절반을 사용하는 버퍼 캐시를 고려하고 있다. 버퍼 캐시의 용량과 그 중요성 때문에 PAVM 방식은 버퍼 캐시를 고려하지 않고는 완전한 해법이 되지 못한다. 이 논문에서 우리는 메모리의 사용처를 분석하고 저전력 페이지 할당 정책을 제안한다. 특히 프로세스의 주소공간에 매핑(mapping)된 페이지들과 버퍼 캐시가 고려된다. 이 두 종류의 페이지들간의 상호작용과 그 관계를 분석하고 저전력을 위해 이러한 관계를 이용한다.

키워드 : 에너지 관리, SDRAM, 버퍼 캐시, PAVM

Abstract As the computing environment moves to the wireless and handheld system, the power efficiency is getting more important. That is the case especially in the embedded hand-held system and the power consumed by the memory system takes the second largest portion in overall. To save energy consumed in the memory system we can utilize low power mode of SDRAM. In the case of RDRAM, nap mode consumes less than 5% of the power consumed in active or standby mode. However hardware controller itself can't use this facility efficiently unless the operating system cooperates. In this paper we focus on how to minimize the number of active units of SDRAM. The operating system allocates its physical pages so that only a few units of SDRAM need to be activated and the unnecessary SDRAM can be put into nap mode. This work can be considered as a generalized and system-wide version of PAVM (Power-Aware Virtual Memory) research. We take all the physical memory into account, especially buffer cache, which takes an half of total memory usage on average. Because of the portion of buffer cache and its importance, PAVM approach cannot be robust without taking the buffer cache into account. In this paper, we analyze the RAM usage and propose power-aware page allocation policy. Especially the pages mapped into the process' address space and the buffer cache pages are considered. The relationship and interactions of these two kinds of pages are analyzed and exploited for energy saving.

Key words : Energy Management, SDRAM, Buffer cache, PAVM

† 학생회원 : 한국과학기술원 전산학과
minlee@cc.gatech.edu
euiseong@cse.psu.edu
†† 종신회원 : 한국과학기술원 전산학과 교수
joon@kaist.ac.kr
논문접수 : 2006년 5월 18일
심사완료 : 2008년 3월 10일

1. Introduction

The memory system is the second most power-consuming part next to the processor. Also in the server system, the memory system is one of the most power-consuming parts. For an example, the 40% of the energy is consumed by the memory system in a mid-range IBM eServer machine [1]. As the memory consumes power continuously contrary to the other components, reduction in power consumption of memory system can be beneficial to the whole system. Since the memory is getting bigger this power issue gets more important. This naturally leads us to put our effort in saving the power consumption in memory system. Hai Huang and his colleagues did excellent work in this area [2,3]. However we found it need to be more system-wide. We especially apply the same idea on the buffer cache which takes big portion of the memory and successfully reduced the power consumption. In this paper we suggest power aware buffer cache which is more generalized and fine-grained approach. In this scheme, each memory region which the buffer belongs to is put into active mode only when the process which uses it is running. By this we can put as many memory regions into low-power consuming mode as possible and limit the size of the memory regions which is used for the system. By limiting the size of system region which should be put into the active mode always and minimizing the size of memory regions which the process and buffers which is used by the process, we minimize the power consumption in memory system.

2. Motivation and Related Work

For low-power computing, the effort to reduce the energy in the one component like the memory system [2-4, and 5] and the disk-drives [6] has been made. Also the cooperation between the operating system and applications for the low power consumption has been studied [7,8, and 9]. More system-wide approaches also have been studied in [10-13]. ECOSystem in [11] treats the energy as a first class resource to achieve the target battery time.

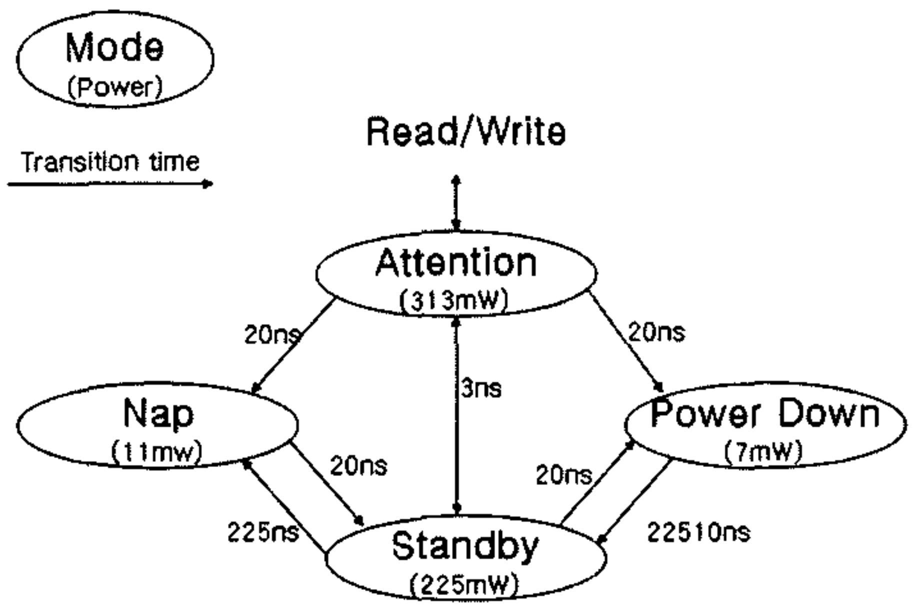


Fig. 1 RDRAM device operating modes

For the low power in memory system, the facility of lower power operating mode in the SDRAM is exploited. The smallest unit of power control in DDR(Double Data Rate) RAM memory is a rank and it's a device in the RDRAM(Rambus DRAM) memory. Although the devices in the RDRAM memory can operate in the several power states as shown in Fig. 1 [2-4, and 5], we consider only standby and nap mode and operating in each mode is said to be turned on and turned off respectively. The other power states are not of our concern because it's not cost-effective due to its resynchronization cost.

The authors of [5] first studied page allocations for the low power consumption. The authors of [4] proposed a simple scheduler-based policy and the PAVM - Power-aware Virtual Memory [2,3], was successful in reducing the energy footprint by gathering the physical pages mapped in the address space into the active ranks and turning on only those active ranks. In [2], the PAVM is modified to cooperate with hardware controller so that more hints from the operating system can be exploited by the controller for the low power.

[12,14] are the studies about the buffer cache not the low power. In [14] the patterns of file I/O are categorized into sequential, looping, and others and utilize this information when replacing the buffers so that the better hit ratio is achieved. In [12], they improved the accuracy of [14] using the PCC (PC-based Classification) approach.

In [13], the garbage collection of battery-operated embedded Java system is studied so that the more memory regions can be turned off for the low power.

The DDR RAM memory system can be viewed as an array of rank which is the smallest unit of power-management. This can be a device for the RDRAM. For simplicity, we take the rank as the smallest unit of power-management in this paper. To reduce power-consumption in the memory system, each rank should be put in its appropriate power state so that power can be saved. For this, the appropriate policy is required.

In PAVM, the pages mapped into the process' address space are gathered into one or a few ranks and only those ranks are turned on during run-time. We refer these ranks to rank set. Whenever the context switch occurs, only such rank set for the process and system rank set is turned on. By this only needed ranks for the run-time can be turned on and the unnecessary ranks are turned off. However PAVM doesn't consider the memory reference from the kernel mode and as the memory the system uses increase, the size of system rank set increases. In turn it causes more power consumption because the system ranks should be turned on always. The PAVM is for the case of CPU-bound job rather than I/O-bound job because it doesn't consider system's memory usage and buffer cache.

The Fig. 2 shows the increase of system rank set size when the Linux kernel source is being compiled. The system rank set is the ranks of physical pages used for the kernel image and kernel data structures and etc. This system rank set should be turned on always regardless of which process is running but the idle process. The Fig. 2 is an example of memory system equipped with totally 8 ranks. The normal is the unmodified Linux kernel and it always turns on all 8 ranks. As Fig. 2 shows, the system rank set size increases as the time goes. This is caused mainly by the existence of buffer cache which is not mapped into the process' address space but actually is used by the process. As the buffer cache grows, eventually it invalidates the energy saving of PAVM. We should take the buffer cache into account to resolve this issue and the PABC which considers the buffer cache successfully limits the size of system rank set size as shown in the Fig. 2.

The process of heavy disk I/O especially increases the buffer cache distinctively. The diff process increases the portion of buffer cache from the 47% of boot up time up to the 77% as shown in Fig. 3. In this case, PAVM can't help putting the buffers into the system rank set and increases the size of system rank set. In turn this causes more power consumption.

Even under user's average workload, roughly the half of the memory is used for the buffer cache. These two simple experiments show that the buffer caches take roughly the half of the memory respectively on Windows machine and Linux machine under user's average workload.

Therefore we can squeeze out more power consumption through the buffer cache. By managing buffer cache out of the system rank set, the size of system rank set can be limited and more reduction in power consumption can be achieved.

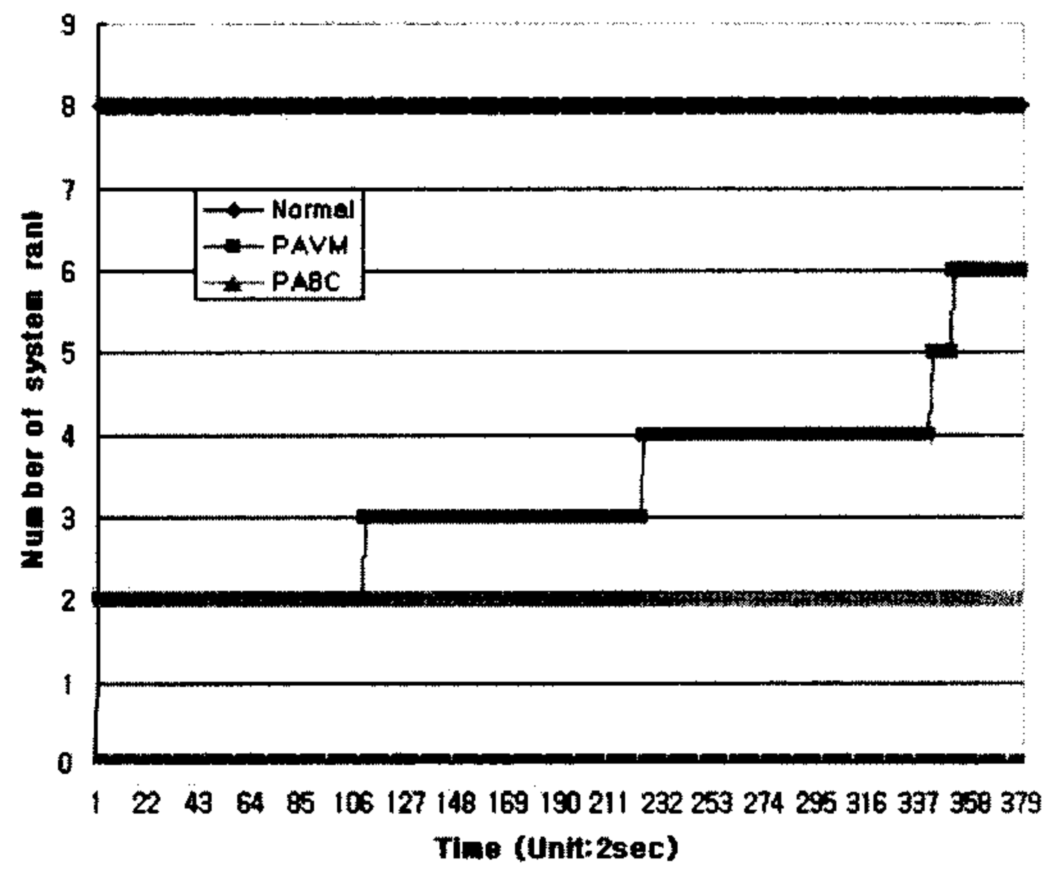


Fig. 2 The increase of system rank set size

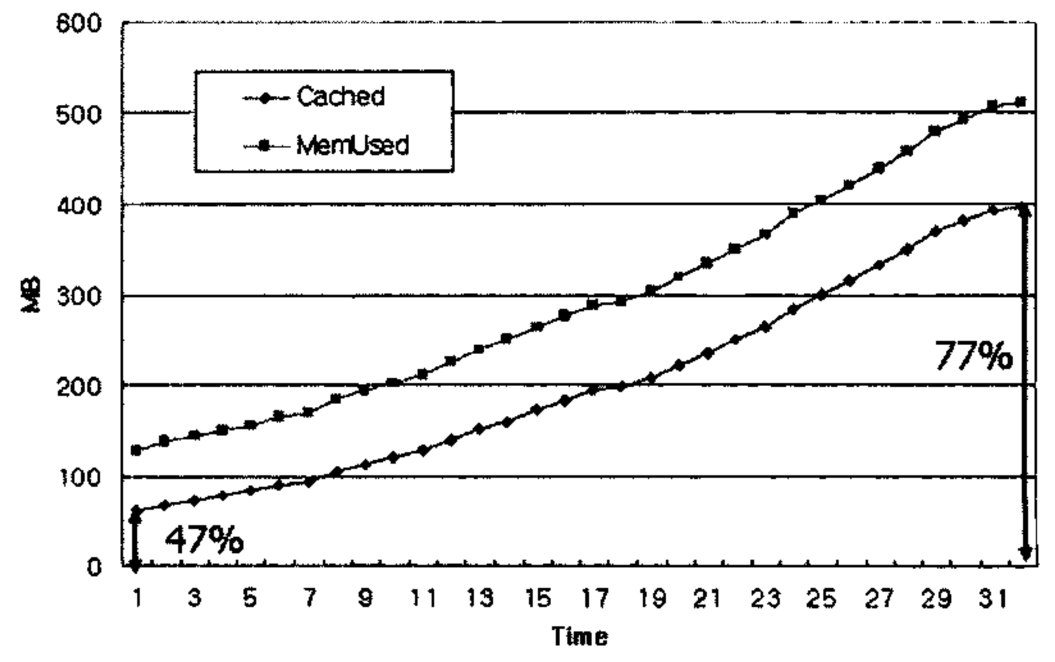


Fig. 3 the portion of buffer cache

Table 1 Memory usage in the Windows machine (unit: KB)

Right after boot-up, 169MB used as system cache	Total 522988, System Cache 169664
After a little bit of use, 174MB used as system cache	Total 522988, System Cache 174752
During copy big files, 379MB used as system cache	Total 522988, System Cache 379688
After the copy, still the system cache lingers	Total 522988, System Cache 381116

Table 2 Memory usage in the Linux machine (unit: MB)

Workload	Used memory	Buffer cache	Ratio
Right after boot-up, serveral X-terms	196	84	42%
Additionally run Opera	265	117	44%
Additionally run Openoffice Writer	335	176	52%
Additionally run Openoffice Spreadsheet	346	185	53%

3. Power Aware Buffer Cache

3.1 Overview

The memory system can be viewed as an array of rank which is the smallest unit of power-management and each rank covers physical address of its size sequentially as shown in the Fig. 4. Therefore a rank consists of thousands of consecutive physical pages.

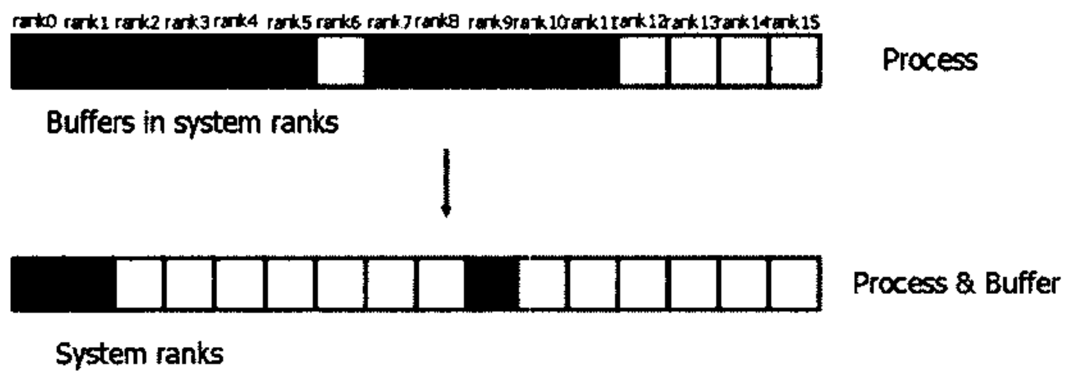

rank0 rank1 rank2 rank3 rank4 rank5 rank6 rank7 rank8 rank9 rank10 rank11 rank12 rank13 rank14 rank15
Process
Buffers in system ranks
Process & Buffer
System ranks

Fig. 4 Basic idea

The basic idea is to gather not only the process' pages which is mapped into address space but also the pages used for buffer cache for each file the process uses. Moreover we want these two kinds of pages to coincide so that only minimal number of ranks should be turned on. Low power can be achieved by turning on only needed ranks as above.

3.2 Basic design

PABC defines a rank set as the below. A rank set is an basic structure to control the memory allocation and account the usage of pages.

```
Struct rankest {
    Spinlock_t              ranks_lock
    Struct ranklist         ranks;
    Struct rank_account     account;
    Int                     ranks_type;
    Int                     ranks_user;
}
```

The 'ranks' field which is the list of ranks which the pages spans. In other words, it contains the ranks which the pages belongs to. The field 'account' is used to account the pages. This is a list of same size of the field 'ranks' and counts how many pages each ranks have. The preceding element of the list of the 'ranks' field has the priority over the following element on the allocation.

Each instance of this structure falls into three categories by the 'ranks_type' field as the below and these rank sets control the allocation for the corresponding pages and get involved in freeing the pages.

α : rank set for a process
β : rank set for the buffers of a file
S : system rank set

We denote the actual instance of each rank set as the below.

$\alpha(p)$: rank set for the process p
$\beta(i)$: rank set for the buffers of the file of inode i
S : system rank set

$\alpha(p)$ represents the process' pages which are the anonymous pages and the page tables of the process and consists of the ranks these pages span. $\beta(i)$ represents the buffers for the file of inode i and consists of the ranks those pages span. The system rank set represents the rest of the pages which are used by the system such as kernel image and data structures. This system rank set is always turned on unless the system is idle and there is only one instance of system rank set unlike the rank sets for a process or the buffers.

This system rank set is initialized as the first two ranks of the memory when boot-up initialization time due to the DMA. Since the DMA references the physical pages without CPU's intervention, such pages should be allocated from the system rank set and the rank 0 should be included in the system rank set because the ISA-DMA uses lower 16MB of the physical memory. Therefore virtually the second rank, rank 1, is the system rank and rank 0 is primarily used for the DMA. For this reason, actual implementation of PABC has the first element of the rank set is rank 1 and the second element of the rank set is rank 0. We may optimize this system rank set a little bit more here, however it's a trivial issue.

For the rank set of type α or β, the initial rank which is the rank the first page is allocated from is very important. Because all the pages are actually allocated from the one rank, once this initial rank is decided, the page allocation afterwards is done from this rank so it is hard to make a change to this rank set.

These rank sets can be expanded when the page is inevitably allocated from the other rank than those in the rank set. However this increase the number of ranks which should be turned on during run-time, this should be allowed by the proper policy prudently. On the contrary, the rank set can shrink when all the pages in the rank are freed. The first and second rank of the system rank set is exceptional as the above.

The decision of which rank should be chosen for the initial rank or expansion of rank set is one of the major policy. Basically we may use worst fit policy that we always choose the emptiest rank.

The Fig. 5 shows how the rank sets are related with the other kenel data structures. Since the α(p) manages the physical pages, it's associated with the address space rather than the process. Unlike the β(i) is associated with the one inode object so there's one to one correspondence relationship, several address spaces which share the pages share

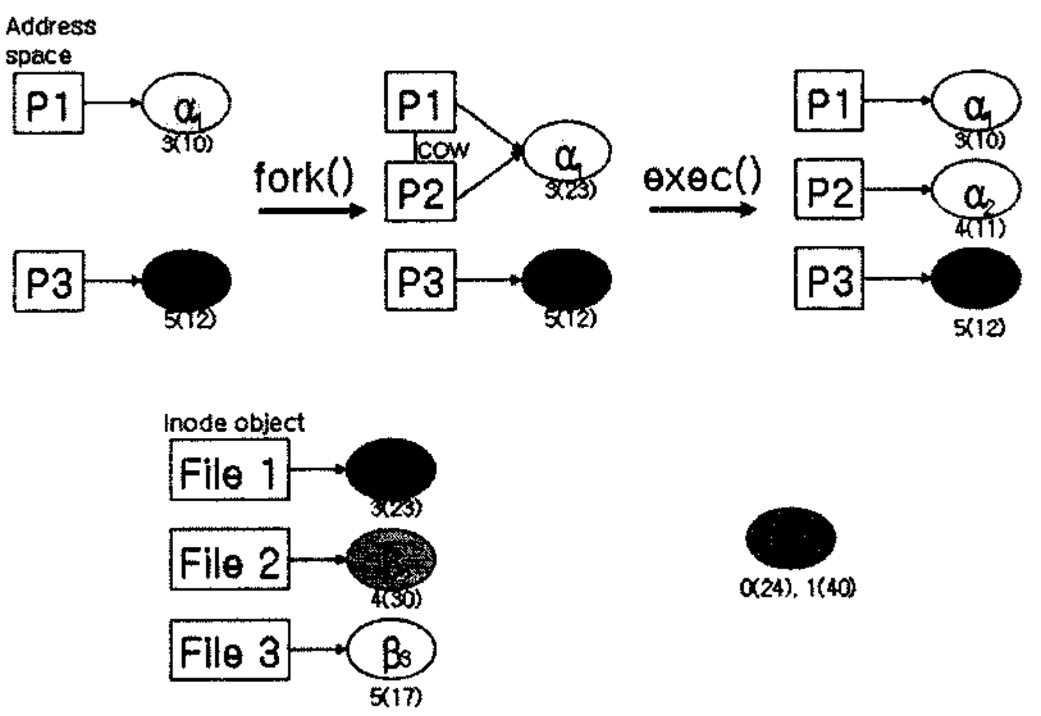


Fig. 5 The relationship of the rank sets

α due to COW(copy on write) because although the new process is forked, physical pages are shared by COW. In other words, any address spaces which share at least one physical page by the COW share the rank set. Therefore α is inherited across the fork system call and new α is allocated only by the creation of new address space through the exec system call. The field 'ranks_user' has an meaning when it's an α, this field count the number of address spaces which use the rank set and the rank set is freed only when the field equals zero. The β is allocated when the first buffer is allocated for the file and is freed when all the buffers are freed. The S is allocated at the system boot up, and freed at the shutdown of the system.

The Fig. 6 shows the actual memory usage through the rank sets. After the P1 forked P2, the memory for P2 is allocated against the α1. Afterwards, P2 discards its address space by exec() system call and some pages of α1 are freed. As the new address space which doesn't share any physical page with any other address space is created, new rank set, α2, is allocated and afterwards the memory for P2 is allocated against the α2.

The kernel maintains the three types of rank sets - α, β, and S and the pointer to the appropriate rank set is passed to the kernel memory allocator as an parameter for the every page allocation. The kernel memory allocator tries to allocate the page from the designated ranks by the rank set. If the ranks are full and fails to allocate, allocate from the another rank which is not in the rank set and expand the rank set to include the rank which the page is allocated from. If the allocator succeeds in

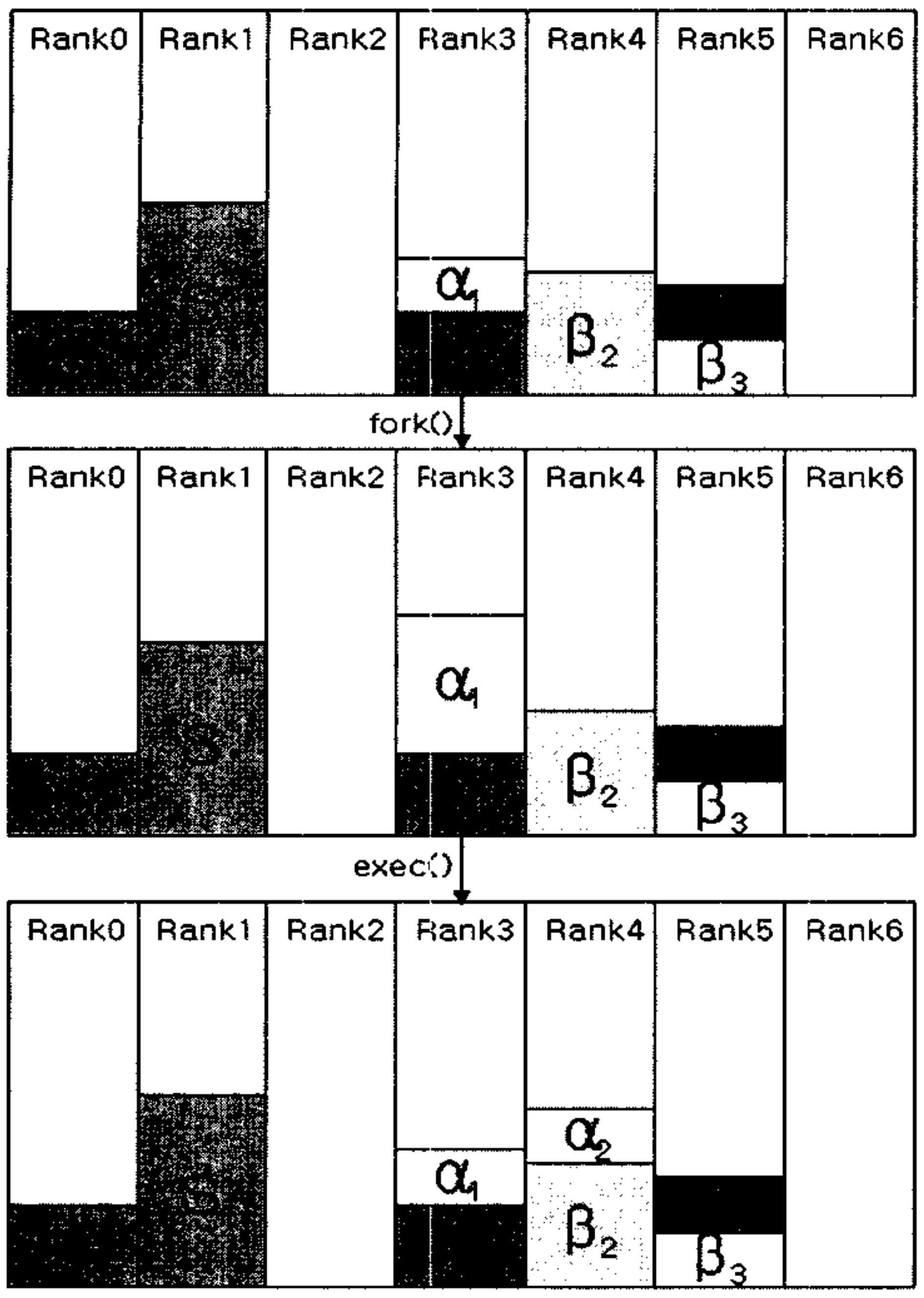


Fig. 6 Actual memory usage through the rank sets

allocation, it leaves the pointer to the rank set in the allocated page's page descriptor for the sake of accounting. This information is used to decrease the count value of the rank set which the page is allocated from. By this, physical pages for the process and buffers for each file are gathered at the allocation time. Also the accounting for each rank set is possible at the low cost.

Based on this information the kernel turns on the active set defined as the below and turns off all the other ranks at the every context switch to process p. Our goal is to minimize the size of this active set.

$$\text{Active set size} = |S \cup \alpha(p) \cup \Sigma\beta(i)|$$

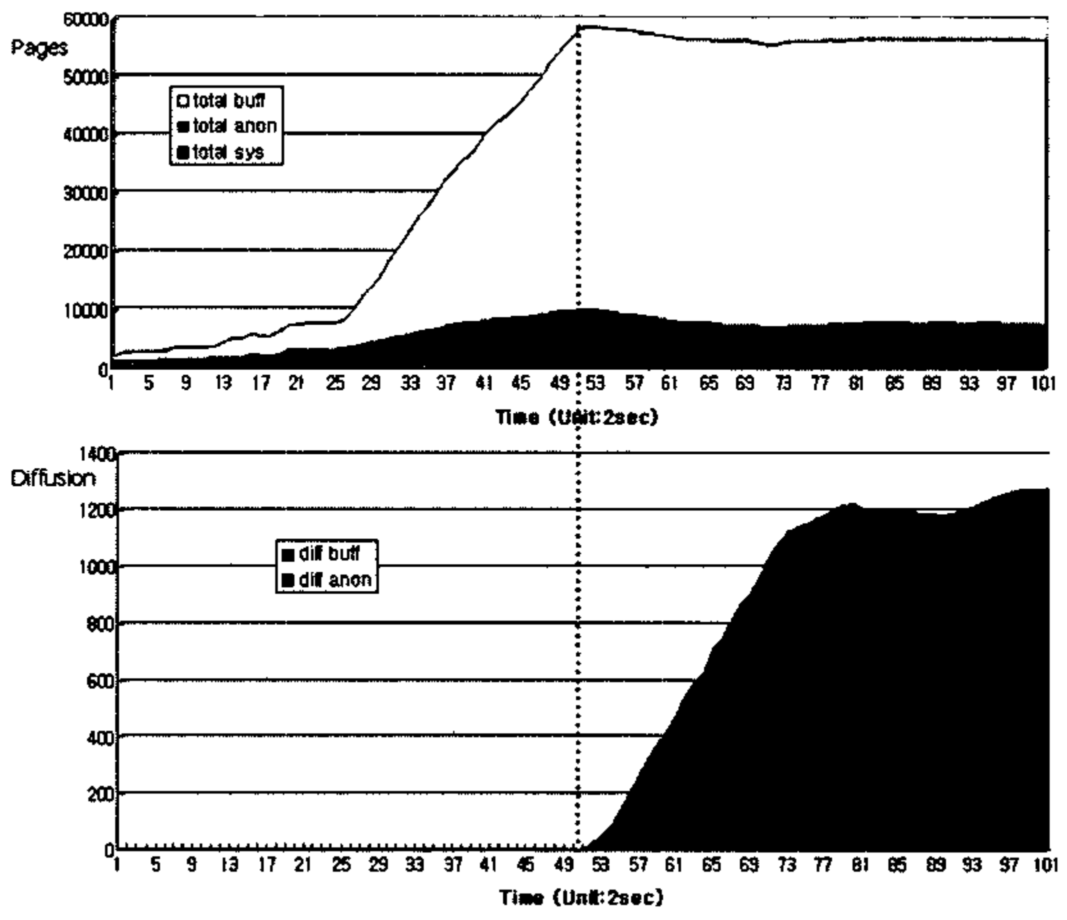
The summation Σ means the union set of all the rank sets for the opened files of the process p. This active set denotes the ranks of potential memory reference, so we turn on only these ranks during run time. Besides this, the non-anonymous pages which are mapped into the address space are discussed in the PAVM [2, 3] excellently. For the simplicity of discuss, in this paper we focus on the buffer cache.

3.3 Compaction

Ideally if the size of the rank set is 1, we say the rank set is compact. As the memory runs out, each rank set starts to expand so that the size of the rank set gets larger. We define the diffusion as the below to analyze this.

$$\text{Diffusion} = \text{Rank set size} - 1$$

Let the total diffusion to be the sum of the diffusions for all the rank sets. If this total diffusion is zero, it means that all the sizes of the rank sets are 1, that is, perfect compaction is achieved. On the other hand, as this total diffusion increase, we have to turn on many ranks during run time. The Fig. 7 shows the increase of the total diffusion value as the memory runs out. To achieve the compaction, we defer the expansion of the rank set and reclaim the pages from the rank set first. We can achieve a good compaction by this strategy. We define two policies for this strategy by its timing of page reclaiming.



Diff_anon : total diffusion for α rank sets
Diff_buff : total diffusion for β rank sets
Total_anon : memory used for α (process' memory)
Total_buff : memory used for β (buffer cache)
Total_sys : memory used for S (system memory)

Fig. 7 the increase of total diffusion as the memory runs out

PABC : The expansion of the rank set is allowed always

PABC-mempol1 : Allow the expansion of the rank set right before the call to oom killer(out-of-memory killer)

PABC-mempol2 : Allow the expansion of the rank set right before the invocation of disk I/O

PABC-mempol1 and PABC-mempol2 reclaims the pages from the rank set before the expansion as the above. In either case of two policies, a good compaction is achieved.

3.4 Coincidence of the process and buffers

To minimize the active set size, we need to coincide α(p) ,the rank set of the process, and Σβ(i), the buffers this process uses. For this purpose we set the initial rank of the β to be the first element of the rank set of the process which caused the allocation for the first buffer. We coincide α(p) and Σβ(i) by this strategy and this minimize the active set size.

4. Evaluation

4.1 Experimental setup and metric

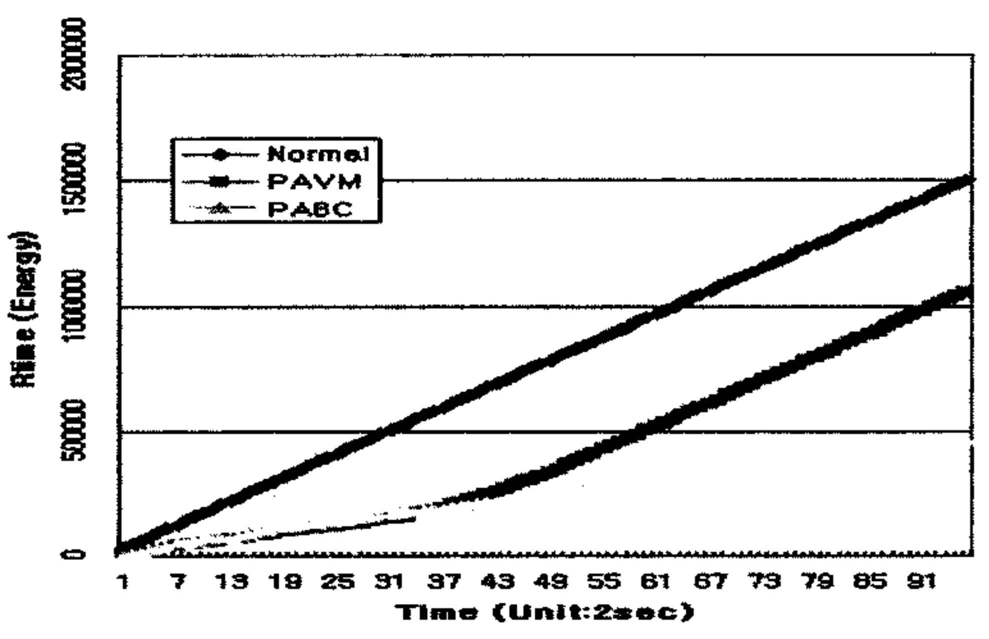
All the experiments in this paper are done on the machine equipped with 256MB DDR SDRAM (Samsung M3 68L3223ETM-CCC), Pentium 4 2.8GHz CPU. A rank is defined as a chunk of memory of 32MB so this machine has totally 8 ranks and has up to 8192 pages in one rank except the first and last rank because of memory holes. We implemented PABC on the Linux 2.6.10 in Fedora Core 3. As the metric for the measurement we define the rtime, the rank time, as the follow.

Rtime : the energy consumed of the one activated rank for one tick

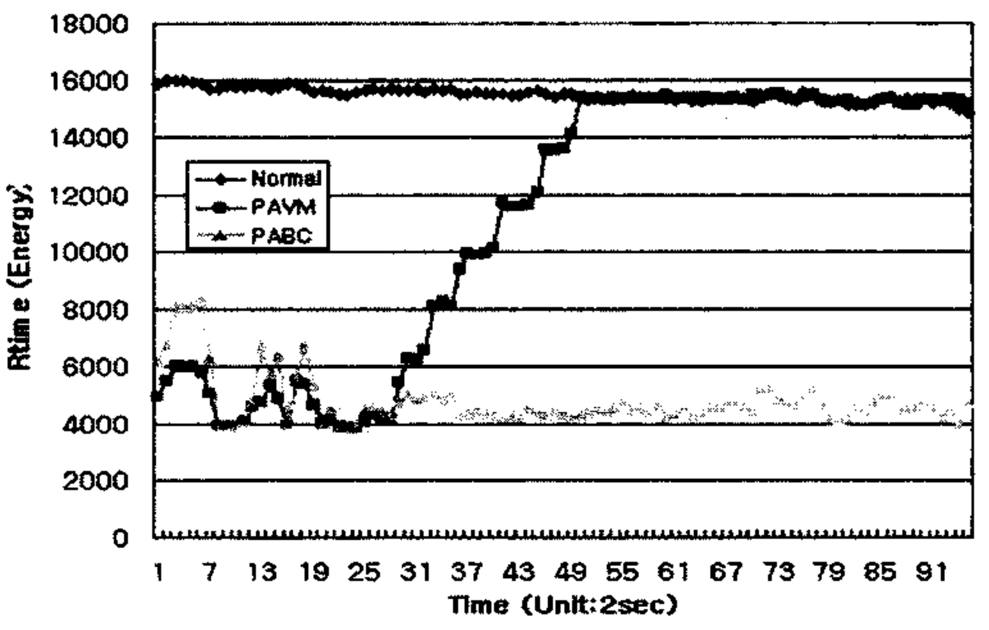
In Linux 2.6 on i386, one tick is 1/1000th second. This rtime is an energy unit by the definition and is measured as the below because each process used the activated ranks of the size of its active set for the ticks the process spent.

Rtime += the ticks the process spent * size of active set

4.2 Energy saving



(a)



(b)

Fig. 8 Energy consumption of the diff process (a) Total energy (b) power

The Fig. 8 shows energy and power consumed by the single diff process on the two Linux kernel source trees. The most power consuming is the normal case which is an unmodified kernel and the PAVM saves the power for a minute after the boot up but soon it follows the normal case while the PABC continues the energy savings. The Fig. 8 describes the behavior of PABC well in the case of single disk I/O bound job.

In the Fig. 9 and Fig. 10 the practical workload

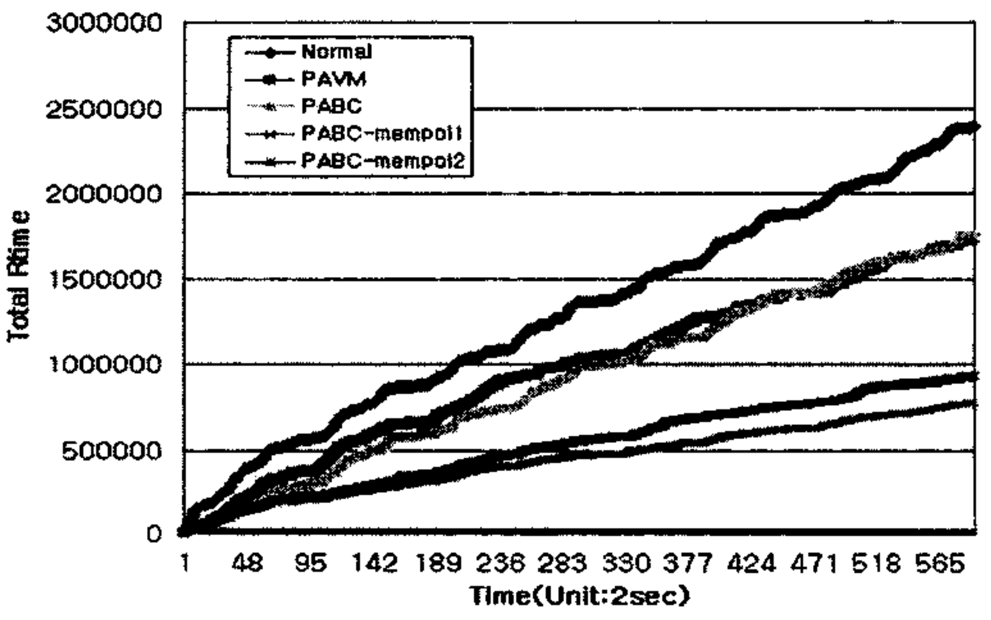


Fig. 9 Energy consumption under the user's average workload

is used. This workload is automated by the expect script program to mimic the user's average workload. The script surf the several internet web sites with firebox 1.0 and at the same time uses gcalc, gedit, gimp for an image manipulation, evolution as an email client, and gpdf for pdf viewer. This also uses open office's oowriter, oomath, ooimpress, oodraw and oocalc. Each task is invoked at the inter-

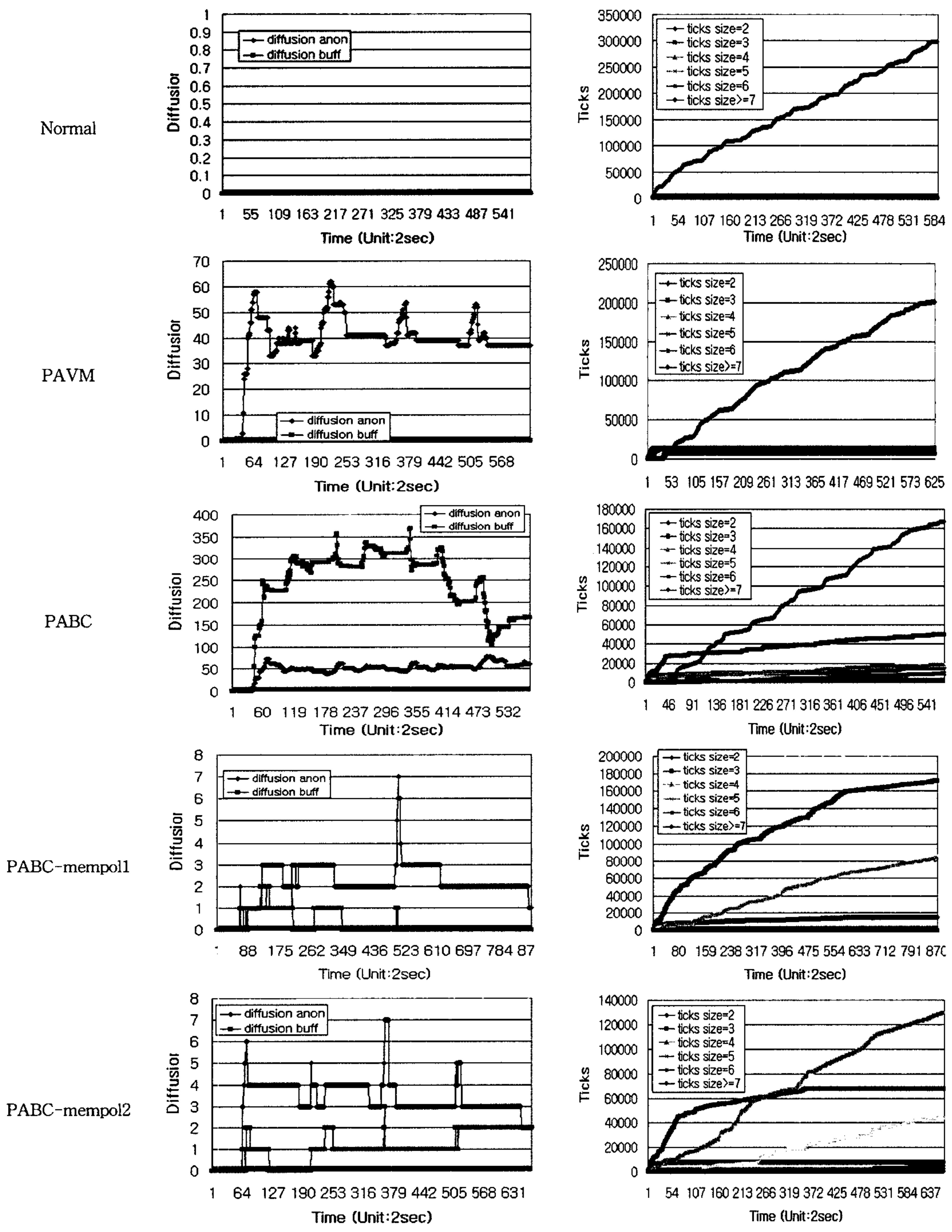


Fig. 10 Details under the user's average workload

val of 10 seconds and repeated 4 times so the total elapsed time is roughly 20 minutes.

Under this practical workload, PAVM and PABC both suffer from performance degradation. This is because of the growth of system rank size for a PAVM and because of the diffusion or expansion of rank sets for the PABC. The PABC-mempol1 and PABC-mempol2 which we applied the policy to prevent the diffusion shows impressive drops in energy consumption.

The Fig. 10 shows the details on this situation. The figures on the right side shows the ticks spent for each rank set size. Here we can see that the number of ticks of rank set size equals or above 7 overwhelm the others in PAVM and PABC. However we see that most of the ticks are spent for the rank set size of 3, 4 or 5 in the case of PABC-mempol1 and PABC-mempol2. This reveals how the energy savings are earned.

The system rank set is always turned on unless the system is idle so it is expensive rank set and we should limit its size. Fortunately as shown in the Fig. 11 two ranks of size 64MB suffice for the system rank set for the all the experiments above. The size of the system rank size is limited successfully and this makes big contributions to energy saving.

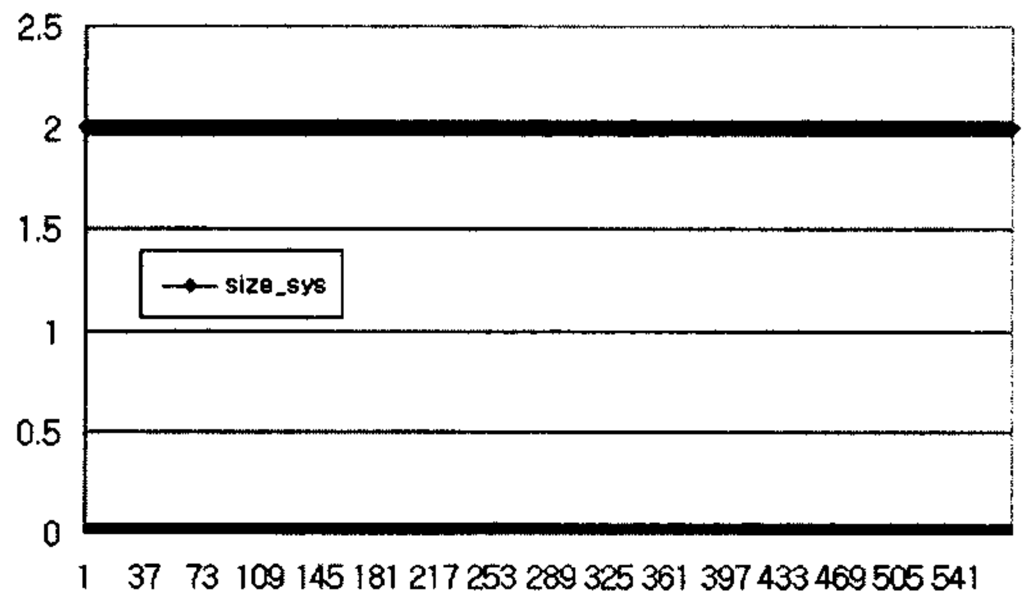


Fig. 11 System rank size

4.3 Compaction

The Fig. 12 shows the perfect compaction of zero diffusion. Both PABC-mempol1 and PABC-mempol2 shows the same result as in the Fig. 12. If we compare the Fig. 12 with the Fig. 7, we see the compaction strategy works. Although this is the simple example, as shown in the figures of left

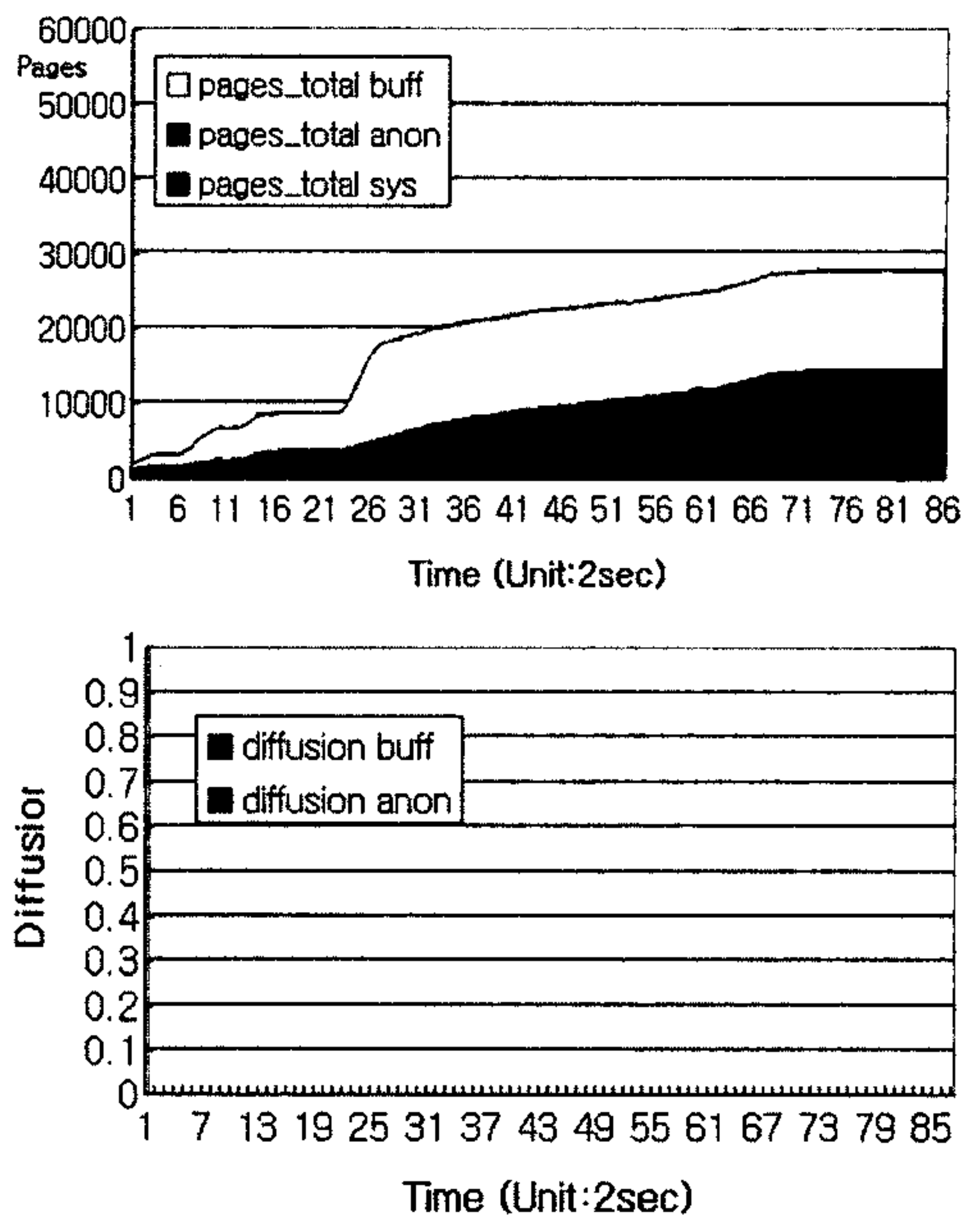


Fig. 12 Diffusion value with compaction

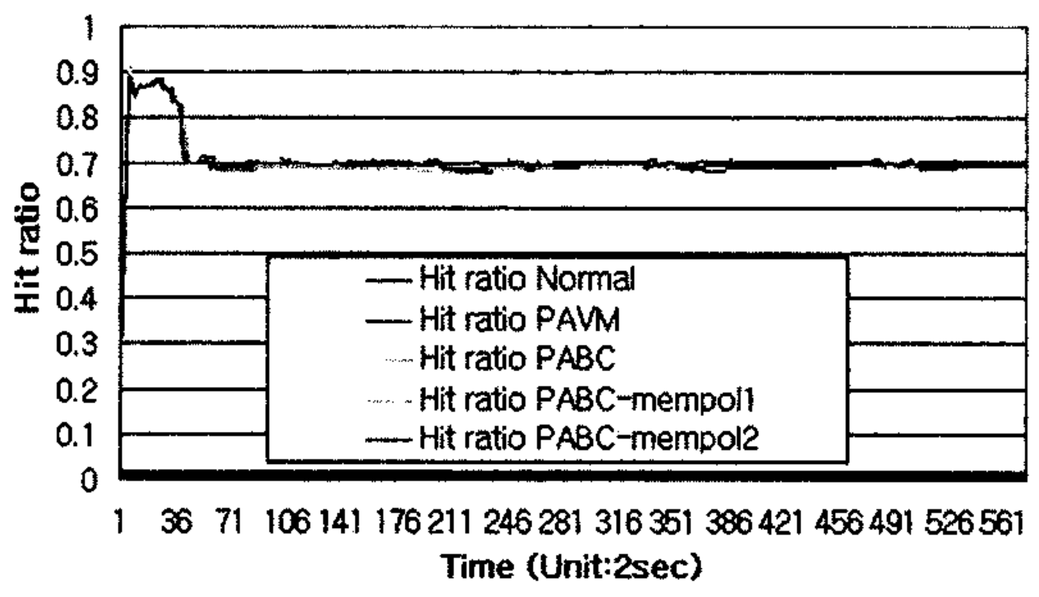


Fig. 13: Impact on the hit ratio

side of the Fig. 10, this strategy works well under the practical workload as we see the diffusion value below 7.

The cost for the compaction is the impact on the hit ratio. We measured the hit ratio as in the Fig. 13 to see the impact on the hit ratio. To avoid the difficulties of comparison and unfairness of too high hit ratio due to the read-ahead facility, we measured the hit ratio without read-ahead facility. Contrary to the intuitive expectation, the impact on the hit ratio turns out to be insignificant as shown in the Fig. 13.

4.4 Overhead

The major runtime overhead is to turn on and turn off the rank sets at the every context switch. However as discussed in [2], this latency is overlapped with the context switching time so virtually it is not an overhead. Although there are some other runtime overheads such as accounting, allocation and freeing of the rank sets, these are all done in the constant time so they are of little significance.

The major spatial overheads are the rank set structures and the pointer to the rank set in the page descriptor. However these are in the acceptable range and takes small amount of memory.

5. Conclusion

We generalized the PAVM's approach to the system-wide scale and designed the PABC scheme especially for the buffer cache. To limit the system rank set size we take the buffer caches out of the system rank set and allocate buffers against the rank set of the process which uses them. Successfully system rank set size is limited. As the memory runs out the expansions of the rank sets degraded the performance of PABC and we remedied this by modification to the page reclaim policy. Good compaction was achieved by new page reclaim policies and as the result under the user's average workload, 67% energy reduction than the normal, unmodified kernel and 61% energy reduction than the PAVM were achieved.

References

[1] C. Lefurgy, K. Rajamani, F.Rawson, W. Felter, M. Kistler, and Tom Keller.: Energy management for commercial servers. In IEEE Computer, pages 39-48, Dec 2003.

[2] Hai Huang, Padmanabhan Pillai, Kang G. Shin.: Design and Implementation of Power-Aware Virtual Memory.

[3] Hai Huang, Kang G. Shin.: Cooperative Software-Hardware Power Management for Main Memory.

[4] Delaluz and et al.: Scheduler-based DRAM energy management. In Design Automation Conference 39, 2002.

[5] Alvin R. Lebeck and et al. Power aware page allocation. In Architectural Support for Programming Languages and Operating Systems, pages 105-116, 2000.

[6] Andreas Weissel, Björn Beutel, Frank Bellosa. Cooperative I/O-A Novel I/O Semantics for Energy-Aware Applications.

[7] Jason Flinn and M. Satyanarayanan. Energy-aware adaptation for mobile applications.

[8] Brian D. Noble, Morgan Price, and M. Satyanarayanan. A Programming Interface for Application-Aware Adaptation in Mobile Computing.

[9] Brian D. Noble, M. Satyanarayanan, Dushyanth Narayanan, James Eric Tilton, Jason Flinn, Kevin R. Walker. Agile Application-Aware Adaptation for Mobility.

[10] Carla Schlatter Ellis. The Case for Higher-Level Power Management.

[11] Heng Zeng, Carla S. Ellis, Alvin R. Lebeck, Amin Vahdat. ECOSystem: Managing Energy as a First Class Operating System.

[12] Chris Gniady, Ali R. Butt, and Y. Charlie Hu. Program-Counter-Based Pattern Classification in Buffer Caching.

[13] G. Chen, M. Kandemir, N. Vijaykrishnan, M. J. Irwin, M.Wolczko. Adaptive Garbage Collection for Battery-Operated Environments.

[14] J.M. Kim, J. Choi, J. Kim, S. H. Noh, S. L. Min, Y. Cho, and C. S. Kim. A Low-Overhead High-Performance Unified Buffer Management Scheme that Exploits Sequential and Looping References. In Proc. OSDI, October 2000.

[15] X. Fan, C. S. Ellis, and A. R. Lebeck. Memory controller policies for dram power management. In *International Symposium on Low Power Electronics and Design*, 2001.

[16] X. Fan, C. S. Ellis, and A. R. Lebeck. Modeling of dram power control policies using deterministic and stochastic petri nets. In *Workshop on Power-Aware Computer Systems*, 2002.

이 민

이민은 2004년 연세대학교에서 학사 학위를 받고 2006년 한국과학기술원에서 석사 학위를 받았다. 현재는 미국 Georgia Institute of Technology에서 박사 학위 과정에 있다. 그는 운영체계, 가상 머신, 컴퓨터 구조 및 인공 지능에 대한 관심을 갖고 있다.

서 의 성

서의성은 한국과학기술원에서 2000년 학사 학위, 2002년 석사 학위 그리고 2007년 박사 학위를 받았다. 현재 미국 Penn State University에서 연구원으로 활동하고 있다. 그는 저전력 컴퓨팅, 리얼타임 시스템, 가상 머신에 대한 연구를 수행한 바 있다.

이 준 원

이준원 교수는 1983년 서울대학교에서 학사 학위를 받고 1990년과 1991년 각각 석사 학위와 박사 학위를 미국 Georgia Institute of Technology에서 받았다. 1992년 이후로 한국과학기술원의 교수로 재직하고 있다. 주요 연구 분야는 저전력 컴퓨팅, 임베디드 시스템과 가상 머신이다.