Page Replacement Algorithm using Swap-in History for Remote Memory Paging

Kazuhiro SAITO† Hiroko MIDORIKAWA† and Munenori KAI†
†Graduate School of Engineering, Seikei University,
3-3-1, Kichijoujikita-machi, Musashino-shi, Tokyo 180-8633, Japan
E-mail: †dm083406@cc.seikei.ac.jp, {midori, kai}@st.seikei.ac.jp

Abstract

The Distributed Large Memory system, DLM, was designed to provide a larger size of memory beyond that of local physical memory by using remote memory distributed over cluster nodes. The original DLM adopted a low cost page replacement algorithm which selects an evicted page in address order. In the DLM, the remote page swapping is the most critical in performance. For more efficient swap-out page selection, we propose a new page replacement algorithm which pays attention to swap-in history. The LRU and other algorithms which use the memory access history generate more overhead for user-level software to record memory accesses. On the other hand, using swap-in history generates little costs. According to our performance evaluation, the new algorithm reduces the number of the remote swapping in the maximum by 32% and gains 2.7 times higher performance in real application, Cluster3.0. In this paper, we describe the design of the new page replacement algorithm and evaluate performances in several applications, including NPB and HimenoBmk.

1. Introduction

The Distributed Large Memory system, DLM, was designed to perform a large memory for sequential application programs without restriction of a local physical memory size by using remote memory distributed over cluster nodes[1]. This system realizes a very large virtual memory by distributing a huge amount of data an application program uses to the memories on remote nodes in a cluster. If local node needs certain data in a remote node, the local node requests the page containing the data to the remote node and sends an alternative page instead. It is called “remote page swapping”. The number of remote page swapping is critical to the performance for applications running on the DLM. So it is important to employ an appropriate page replacement policy.

Many page replacement algorithms are already investigated in the long history of OS and virtual memory development. Most of them use a memory access history, read/write bit, access count and time, etc. to decrease the number of swapping. There are many well-known algorithms, such as LRU and LFU, but they are usually arranged and simplified to reduce the overhead of recording all memory access, which is necessary for a full implementation. The Linux kernel, for example, uses two types of lists, an active list and a non-active list, depending on a memory access time for a page replacement to realize a virtual memory[8].

The DLM is implemented as user-level software independently from the Linux kernel, because it is more faster and stable than incorporating it into kernel-level as a part of kernel page swapping[1]. For user-level software, like DLM, recording memory access for its page replacement causes a substantially huge overhead compared to the one in kernel-level software used the page replacement in OS kernels. So the original DLM employed a low-cost page replacement algorithm which selects a swap-out page candidate in order of an address of a page. This selection policy is expected to keep a block of pages with continuous page address, which is usually efficient for the memory management in kernel. On the other hand, there are some cases making an inefficient page selection depending on memory access sequences of applications. The page just swapped out by the page replacement algorithm is soon needed by the application. So we propose a cost effective page replacement algorithm designed for user-level software, which uses only swap-in page history with no memory read/write recording.

The page replacement policies implemented in remote memory paging are categorized into two types.

- Kernel-level implementation as a customized remote memory swap device follows the same algorithm used in OS kernel swap daemon[5][6].
- User-level software implementation uses a customized page replacement algorithm[7].

Some page replacement algorithms for OS virtual memory also use a swap history. The ARC[3] and the CAR[4] use the swap-out history as additional information of the LRU. In SEQ[2] employs the most-recently-used replacement using long sequences of the swap-in history and the LRU. They use swap information to simplify and generalize the LRU and the LFU. On the other hand, our algorithm does not use LRU or LFU which requires a high-cost memory access recording. We employ the original address-order page selection policy and use a swap-in history in additional information. In this paper, we show the design and performance of the algorithm.

2. Remote Page Swap in DLM

2.1. The DLM System

The DLM is a system which uses multi-computer in a cluster as if one computer has a large memory resource. A local node is called "calculation node" and a remote node is called "memory server" shown in **Fig. 1**. There are two threads in calculation process. One called "calculation thread" runs user programs. Another called "communication thread" communicates with memory servers. The memory servers provide their own memory by receiving and processing requests from calculation node (e.g. dynamic remote memory allocation, remote swap, etc.).

When an application accesses to data which calculation node does not have, the calculation node requests it to memory server over network. If the calculation node does not have enough local memory to allocate this data, it evicts alternative data to memory server. This is the remote swap. To receive needed data is called swap-in and to evict alternative data is called swap-out shown in **Fig. 2**. In remote swap of the DLM, the data communicates in a unit of DLM page. This remote memory paging makes large virtual memory. Also, the protocol of its communication is TCP/IP.

The remote memory is managed by DLM page table in a unit of DLM page. The DLM page size can be set freely in multiples of OS page size when the system is built. An optimal value of its parameter depends on the network speed. Also, users can set used memory sizes and order of priority of each node by DLM config file specified command argument when DLM programs run. Besides, programmers can specify DLM data in their programs by calling the function of DLM library, and DLM data is allocated in order of these using nodes. If data allocation size exceeds an upper bound which a node is set by the user, the data over it is allocated the next node (memory server).

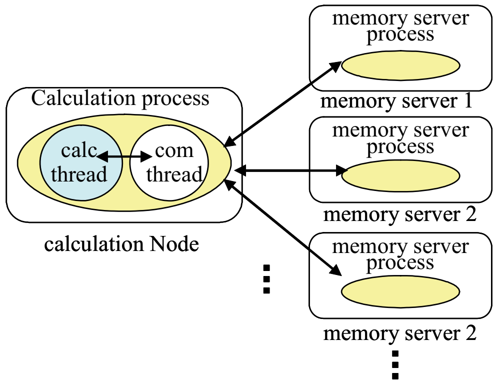

Fig. 1: Machine constitution of DLM

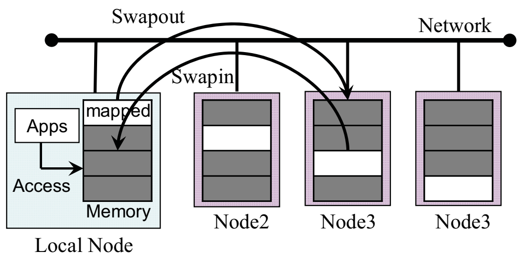

Fig. 2: Remote swap

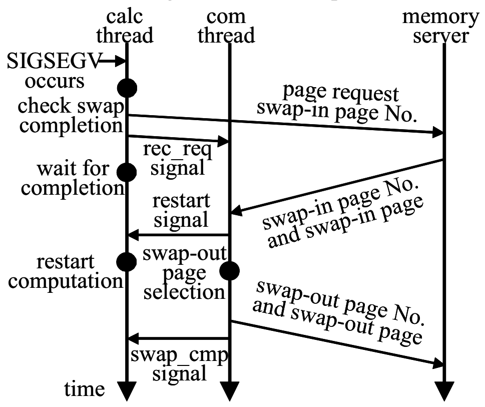

Fig. 3: Sequence of remote swap of DLM

When user program (calculation thread) accesses DLM data which calculation node does not have, the remote swap is handled according to **Fig. 3**. First, a calculation thread catches SIGSEGV and sends needed DLM page (swap-in page) number, which includes the access data, to memory server allocated this DLM page. Second, a communication thread receives the DLM page which the memory server sends and the calculation thread restarts the user program. Finally, the communication thread selects an alternative DLM page (swap-out page), and sends it to the memory server. If SIGSEGV occurs again before remote swap completes, the calculation thread suspends new page request until receiving remote swap complete signal.

2.2. DLM Page Replacement Algorithm

In the original DLM, a swap-out page which is sent in remote swap from a calculation node to a memory server is selected by cost effective page replacement algorithm. First, this algorithm searches the DLM page allocated in calculation node in order of a DLM address. Then, it is sent to the memory server as a swap-out page. Also, the start page of this algorithm is a next page of previous swap-out.

3. Proposed Page Replacement Algorithm

In the original DLM page replacement algorithm, there is a possibility of selecting ineffective swap-out DLM page which is accessed in the next time according to timing of memory access. **Fig. 4**, which is the one example in an application: Cluster3.0 (The details is in Chap. 5.1), shows a part of time series of the swap-in and swap-out DLM page number. In this case, swap-out page is selected swap-in page soon.

To avoid this inefficient selection and to archive low cost process in user-level software, we invent a new page replacement algorithm which uses a swap-in history based on the original DLM algorithm. It is based on the assumption that a swap-in page which swap-in occurs in after previous swap-in of its page is likely needed the current swap-in soon again when swap-in occurs in a certain page. For this, this algorithm records the swap-in page history and looks at this history not to select pages which may access next as swap-in page. It calls "reflecting history" how many pages is reflected in swap-in history.

3.1. Data Structure for replacement

There are two main necessary data structures in **Fig. 5**. One is a cyclic array of swap-in history. This is to record the swap-in DLM page number sequentially when the remote swap is generated. When the cyclic array is written in final element, a swap-in page number generated next is written in first element. The size of this cyclic array is set to an amount of all DLM page set at runtime. This size can accommodate sequential access through all used memory.

Another is a variable in each DLM page table entry. This variable is stored a swap-in history's index which indicates its own previous swap-in.

3.2. Implementation into DLM

In this chapter, we show an implementation of this page replacement algorithm into the DLM. It does at swap page selection part in **Fig. 3**.

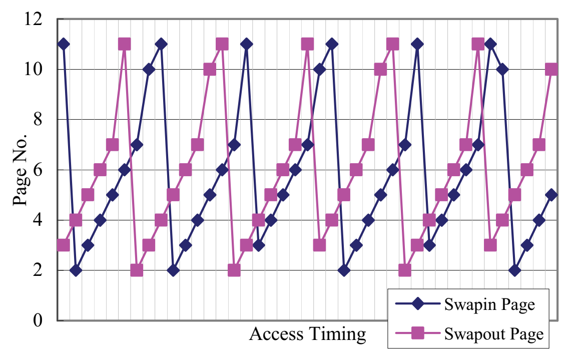

Fig. 4: Time series of swap page

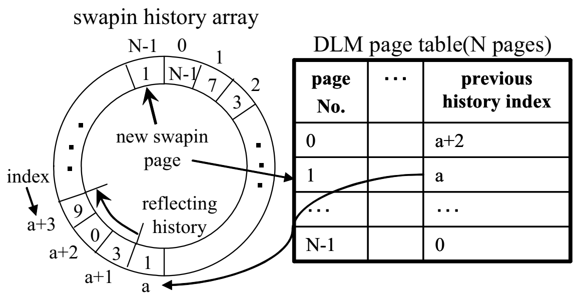

Fig. 5: Swap-in history and DLM page table
(It doesn't swap-out at page 0, 3, 9 in 3 reflecting histories)

When a swap-out DLM page is selected, DLM pages in the previous swap-in history of the current swap-in DLM page for reflecting history are excluded from the swap-out DLM page. The previous swap-in history is led from current swap-in DLM page's previous history index entry in DLM page table. A swap-out DLM page is selected a DLM page which does not match such excluded DLM pages and is found allocated in the calculation node by original DLM algorithm. After swap-out DLM page selection, a current swap-in DLM page number is registered into the swap-in history. Therefore, its array index is registered into its previous history index entry in DLM page table.

In addition, there is little overhead in this page selection because of running parallel with the calculation thread.

4. Performance Evaluation

We evaluate the performance of the new page replacement algorithm proposed in this paper in some applications. The experiments are done in two environments. One is two nodes over 1GbitEthernet cluster (1GbpsCL) and another is the supercomputer at Tokyo University: T2K (10GbpsCL) in **Table 1**. The most of experiments are run in 10GbpsCL except for one application, Cluster3.0, which is done in 1GbpsCL.

First, we measure performance of the original DLM

Table 1: Cluster environment

	Calculation Node	Memory Server	T2K HA8000
machine	HP ML150 G2	HP ML150 G3	HITACHI HA8000-tc/RS425
CPU	Xeon 2.8GHz x 2CPU HyperThread	Xeon E5310 1.6GHz QuadCore x 2CPU	AMD Opteron 8356 2.3GHz QuadCore x 4CPU
memory	1GB	8GB	32GB
Cache	L2 : 1MB/CPU	L2 : 4MB/CPU	L2 : 2MB/CPU(512KB/Core) L3 : 2MB/CPU
OS	Linux kernel2.6.20-1.2320.fc5 x86_64	Linux kernel2.6.23.17-88.fc7 x86_64	Linux kernel2.6.18-53.1.19.el5 x86_64
NIC	Broadcom 5721 PCI-Express Gigabit NIC	NC7781 OnBoard Gigabit NIC	
Network	1GbitEthernet		IP on Myrinet-10G

on several *local memory ratios* (the ratio of use memory size in calculation node against one of the entire program). Next, we measure performance of the new DLM with page replacement algorithm using swap-in history on the several local memory ratios and on several *reflecting history ratios* (the ratio of reflecting history count against the size of swap-in history array). In this chapter, the y-axis of the graph is represented the ratio compared from the original DLM result. In Chapter 4.3, we evaluated the performance in fixed *reflecting history ratio* which we expected the results would be better than other ratio.

The DLM page size can be changeable according to the network speed used in the cluster. In this experiment, 128KB and 1MB are used in 1GbpsCL and 10GbpsCL respectively for DLM page size.

4.1. Cluster3.0

Cluster3.0 processes a variety of clustering for the gene analysis[12]. It takes considerable time because of using a great deal of data, and it is different from the application of numeric computation in that it repeats dynamic memory allocation and release. In this paper, we measure execution time and remote swap count by the gene analysis using the hierarchical clustering in two environments. Also, we use a small size sample program: demo.txt (use memory size is 26MB) as a gene data file, for making the evaluation time shorter.

4.1.1. Execution in 1GbpsCL. We run Cluster3.0 at 1GbpsCL and measure its execution time and remote swap count. This result is shown in **Fig. 6** and **Fig.7**. In this program, an amount of all DLM page is 207 pages. **Fig. 6** shows remote swap count ratios in the new DLM against one of the original DLM. **Fig.7** also shows execution time ratios.

In **Fig.6**, it is found that the remote swap count is reduced by enlarging the reflecting history ratio. Especially, the remote swap count reduced to 32% at 77% of *local memory ratio* and 92% of *reflecting history ratio*. However, the *local memory ratio* is too high or too low to make the performance worse. A cause in the high ratio case is the large range of selection, and the cause in the low ratio case is the small range of selection. As a result, a page except for the other page around swap-in history is selected by page selection before reflecting history. In **Fig. 7**, the best reduction ratio of execution time is 37% (2.7 times as high speed as original). This value is smaller than reduction ratio of the remote swap count because this execution time contains a time except for remote swap time, e.g. calculation time.

4.1.2. Execution in 10GbpsCL. We run Cluster 3.0 at 10GbpsCL as well as Chapter 4.1.1. An amount of all DLM page is 26 pages. **Fig. 8** shows remote swap count ratios, and **Fig. 9** shows execution time ratios.

Both remote swap counts and execution times are also reduced as well as at 1GbpsCL. However, the reduction ratios of execution time in 1Gbps are better than the ones in 10Gbps. The best of the reduction ratio of remote swap count is 50% in 58% of *local memory ratio*, but the one of execution time is 65%. As a result, it is found that the speed-up of execution time in low speed network is bigger than high speed one because the overhead of remote swap in high speed network is only a little.

4.2. Reflecting history ratio versus Swap count

We measure performances of seven applications, Cluster3.0, NAS Parallel Benchmark (NPB2.3-omni-C)[9][10] and Himeno benchmark[11]. In NPB, we use IS, FT, SP, CG and MG, and each applications are set class C. In Himeno benchmark, we use its C program version and select ELARGE size (513x513x1025, 14GB). **Table 2** represents the used data of these seven applications and remote swap count which implies memory access frequency of each application. More remote swap counts any application occurs, higher memory access frequency it is.

Fig.10 shows relative count of remote swap compared to the one in the original. The effectiveness of the algorithm varies on applications and local memory ratios. In NPB CG.C, the remote swap count is reduced up to 26% of original one when *reflecting history ratio* is 32% and *local memory ratio* is 36%. Generally, the larger the *reflecting history ratio* is, the smaller the remote swap count is. On the other hand, in NPB SP.C and FT.C with 71% and 46% of local memory ratio respectively, the remote swap count increase by enlarging the reflecting history ratio. **Fig.10** shows that the most of the applications decrease swap counts and gain better performance when *reflecting history ratio* is 20%. So, we employ the value 20% for *reflecting history ratio* as DLM built-in parameter.

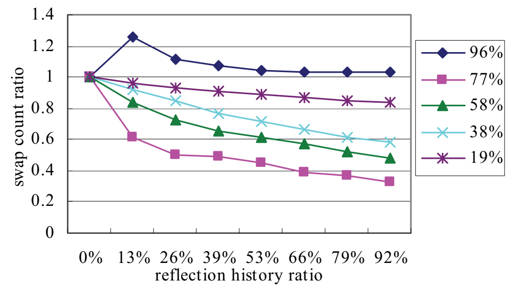

Fig. 6: Cluster3.0's remote swap count ratios by the local memory ratio in the 1GbpsCL

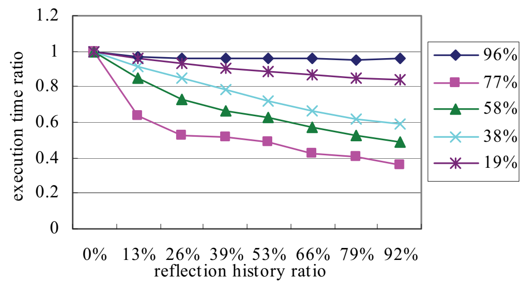

Fig. 7: Cluster3.0's execution time ratios by the local memory ratio in the 1GbpsCL

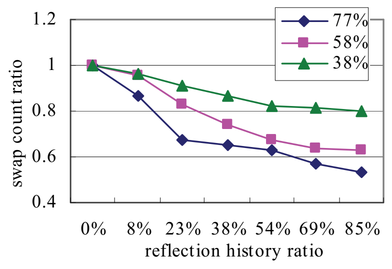

Fig. 8: Cluster3.0's remote swap count ratios by the local memory ratio in the 10GbpsCL

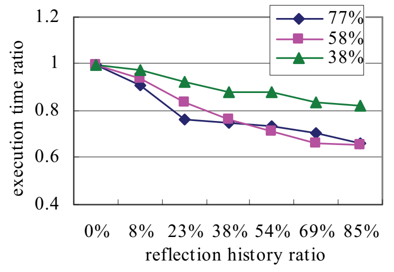

Fig. 9: Cluster3.0's execution time ratios by the local memory ratio in the 10GbpsCL

4.3. Evaluation of built-in parameter

With fixed built-in 20% *reflecting history ratio,* we investigate relative swap counts and relative execution times to the original algorithm for seven benchmarks with various *local memory ratio*s on 10GbpsCL.

Table 2: Experimental parameters for benchmarks

Application	Use memory	All DLM page	Swap count (Local memory ratio)
Cluster3.0(1G)	26MB	207	249764 (38%)
Cluster3.0(10G)		26	41319 (38%)
Himeno	15.1GB	14406	16411 (7%)
NPB IS.C	1.6GB	1537	11784 (26%)
NPB FT.C	7.0GB	6691	143812 (46%)
NPB SP.C	1.3GB	1256	113800 (40%)
NPB CG.C	1.1GB	1104	111091 (9%)
NPB MG.C	3.6GB	3417	346868 (15%)

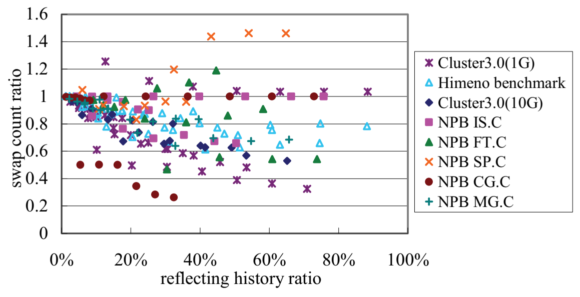

Fig. 10: Remote swap count ratio of seven applications

Experimental result reveals that all benchmarks gains better or equal performance compared to the ones in original algorithm. As shown **Fig. 11,** SP.C, which causes bad performance at higher *reflecting history ratio* in Fig.10, does not deteriorate. The performance of Cluster3.0 in **Fig. 12** gains comparatively good value. If users want to get better performance in a certain application, they can tune the *reflecting history ratio*.

4.4. The Cost of Swap out Page Selection

To evaluate new page replacement protocol, we measured times of the fractions in page swap handling with Himeno benchmark. **Fig. 13** shows typical time fractions in swap-out processing. One swap-out process consists of a page selection, a munmap function call (release memory mapping) and a socket write function call (send swap-out page to a memory server). The result shows that a page selection time to a munmap function time is about 1/14, and a page selection time to a write function time is about 1/109. So the page selection time is relatively so small to other processes.

Table 3 shows swap count and page selection time in millisecond. The multi-column of page selection time consists of total time in all swap-out, max time in and mean time of one page selection. The max time is long compared to the mean time because it is the case that page selection search go around to the cyclic page array to find appropriate swap out page which does not appear in swap-in history. In such case, the calculation thread waits for long time to restart. The mean time is so short because the above case hardly happens.

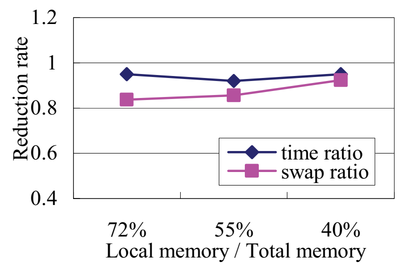

Fig. 11: NPB SP.C results in 20% reflecting history ratio

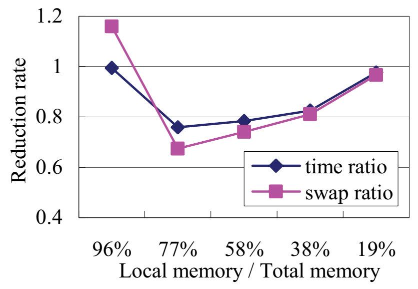

Fig. 12: Cluster3.0 results in 20% reflecting history ratio

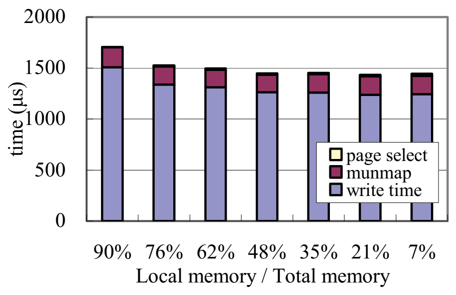

Fig. 13: each processing mean time of one swap-out (µs)

Table 3: Swap count and page selection time

local memory	swap count	page select time(ms)		
		total	max	average
90%	14547	43.90	3.429	0.003
76%	30319	331.23	3.611	0.011
62%	51761	622.73	2.846	0.012
48%	72494	952.53	1.022	0.013
35%	83203	1192.66	1.851	0.014
21%	86003	1293.61	2.625	0.015
7%	91701	1854.56	4.461	0.020

5. Conclusions and Future Works

We propose the new page replacement algorithm using swap-in history and incorporate it into the DLM. As the evaluation in Cluster3.0, it gains the best performance with about 50% of the *local memory ratio*. Generally, the higher *reflecting history ratio* is used, the better performance is achieved. Additionally, the algorithm is more beneficial in slow network such as 1Gbit/s network. For the evaluation in seven programs, the new algorithm reduces the swap count to 26% at a maximum in NPB CG.C with 30% of *reflecting history ratio* and 36% of *local memory ratio*. Moreover, it is found that the best value of the ratios is different according to the application. With incorporating 20% *reflecting history ratio* in the DLM as built-in parameter, all programs used here gains better or equal performance compared to the one in the original DLM .

Future works include the comparison of the page replacement algorithm using swap-in history to others realized in user-level software. According our experimental result, the page selection time is relatively small compared to other fractions of the process in swapping, so we can incorporate more sophisticated method into the DLM to gain better performance.

Acknowledgment

The part of this work was supported by JSPS KAKENHI 21500062.

References

[1] H. Midorikawa, K. Motoyoshi, R. Himeno and M. Sato, “DLM: A Distributed Large Memory System using Remote Memory Swapping over Cluster Nodes”, In IEEE International Cluster Computing, Sept. 2008, pp. 268-273
[2] G. Glass and P. Cao, “Adaptive Page Replacement Based on Memory Reference Behavior”, ACM SIGMETRICS, Feb. 1997
[3] N. Megiddo and D. S. Modha, “ARC: a self-tuning, low overhead replacement cache”, 2nd USENIX FAST, 2003, pp. 115-130
[4] S. Bansal et al., “CAR: Clock with Adaptive Replacement”, 3rd USENIX FAST, Mar. 31, 2004
[5] S. Liang, R. Noronha, and D. K. Panda, “Swapping to Remote Memory over InfiniBand: An Approach using a High Performance Network Block Device”, In IEEE Cluster Computing, Sept. 2005
[6] Tia Newhall et al., “Nswap: A Network swapping Module for Linux Clusters”, EuroPar03, 2003
[7] Scott Pakin and Greg Johnson, “Performance Analysis of a User-level Memory Server”, In IEEE International Cluster Computing, Sept. 2007, pp. 249-258
[8] Daniel P. Bovet and Marco Cesati, “Understanding the LINUX KERNEL”, O’Reilly Media Inc., 2006
[9] (2008)NAS Parallel Benchmarks, web site [Online]. http://www.nas.nasa.gov/Resources/Software/npb.html
[10] (2008) NPB2.3-omni-C web site [Online]. http://phase.hpcc.jp/Omni/benchmarks/NPB/index.html
[11] (2009) Himeno Benchmark web site [Online]. http://accc.riken.jp/HPC/HimenoBMT/index.html
[12] (2009) Cluster3.0 web site [Online]. http://bonsai.ims.u-tokyo.ac.jp/~mdehoon/software/cluster/