Automatic Adaptive Page-size Control for Remote Memory Paging

Hiroko Midorikawa, Joe Uchiyama
Graduate School of Science and Technology,
Seikei University,
Tokyo, Japan
midori@st.seikei.ac.jp

***Abstract*—An automatic adaptive page size control methodology is proposed for remote memory paging. It estimates a working data set and changes page size dynamically and adaptively to each processing part of an application during it is running. It is highly effective to prevent memory server thrashing when the size of local memory is limited.**

Keywords- memory paging; memory server; page swap; page size; thrashing; page-based system

I. INTRODUCTION

To determine the most efficient size of data for transmitting over network is sometimes difficult. It is influenced by not only the network bandwidth and the latency but also the frequency and the granularity of data demands from application programs. Similarly, the page size in remote memory paging systems sometimes drastically influences the performance of application programs.

Authors already proposed a user-level remote memory paging system, DLM [1][2], which offered virtual large memory using distributed remote node memory in a cluster. User-defined page size is available for transmitting data between a local host and remote memory servers in DLM as shown in Fig.1. Generally, fewer transmitting with large page size is efficient for most of the applications under usual conditions. However, frequent transmitting with small page size sometimes achieves better performance under special conditions, where large size of working data set is required by an iterative application on very limited size of local memory.

Fig.2 shows the relationship between page size and relative execution time using remote memory with DLM compared to ordinary execution time using only local memory. In NPB BT, SP and FT [4], only 5% -10% of total data memory required by the each program resides in local memory and the remaining 90%-95 % of data resides in remote memory. In other words, the programs use 10 - 20 times larger size of memory beyond local memory. Their performances are drastically degraded as page size becomes larger. In these cases, the working data set of the iterative application overflows the small local memory and causes thrashing between local host and memory servers. On the other hand, Himeno benchmark [3], which shows a typical case, performs better as page size becomes larger. It depends on a relative relationship between absolute size of local memory and a memory access pattern that is specific to the type of applications and the size of problems. So it is difficult to determine the optimal page size before program running. Moreover, even in one application program, optimal page size may vary depending on the memory access pattern in each part of the processing that consists of the program.

This paper proposes an automatic adaptive page size control mechanism, which repeatedly estimates a working data set size and changes page size dynamically and adaptively to each processing part of an iterative application during it is running. It achieves great effectiveness for various applications.

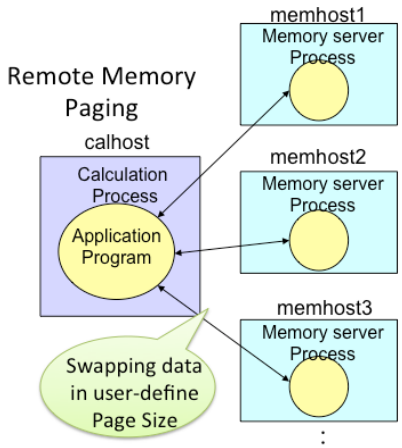

Figure 1. Remote memory paging in virtual large memory in a cluster

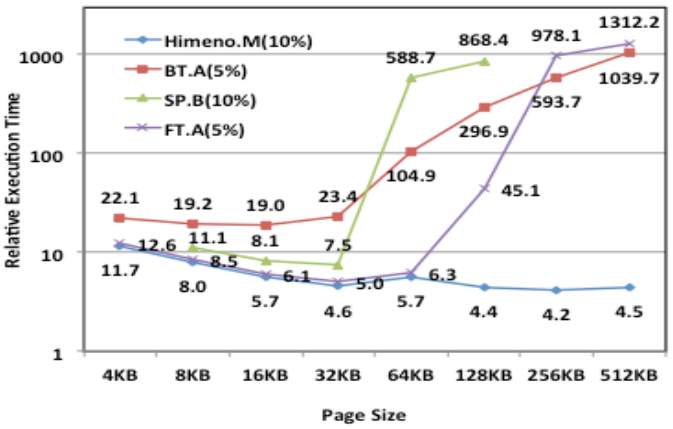

Figure 2. The worst examples of performance degradation by fixed page size for NPB and a typical normal example of Himeno Benchmark

II. AUTOMATIC ADAPTIVE PAGE SIZE CONTROL: AAPC

Basic idea of page size control is to prevent thrashing by choosing an appropriate page size, by which the working data set of an application is made to fit in the local memory as shown in Fig.3. If there is no thrashing, larger page size is usually preferable for efficient communication and data access locality.

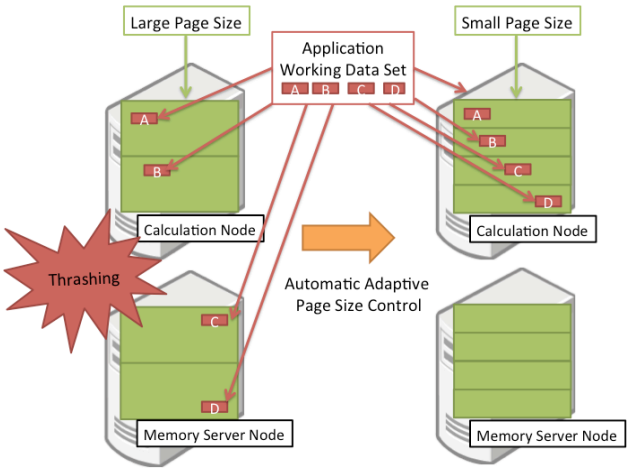

Figure 3. The automatic page size control to depress page thrashing

A. *Estimation of Working Data Set Size in Applications*

To investigate the exact working data set, ***WS***, of an application is difficult and not realistic from the point of view of measuring overhead. So we approximate the size of WS with the size of swap-in page set, ***WS page count***, during a predefined period that corresponds to each processing part of an application. The preciseness of measured WS size depends on a resolution, which is the page size used in the page count measurement during the application is running. If page size is changed, the resolution varies and the measured size of swap-in page set, ***WS page count,*** may vary. So the size of WS, ***WS page count,*** for each processing part recorded in the DLM system is updated whenever new page size is employed.

B. *Criterion for Page Size Control*

Decision of page size change is based on the comparison of WS page count explained above and Local page count, which equals local memory size divided by current page size.

(1) If ***WS Page count*** > ***Local Page count***, a thrashing occurs. Change page size small as closely as the following ***Target Size***.
Target Size = Local Memory Size / ***WS Page count***

(2) If ***WS Page count*** <= ***Local Page count***, there is no thrashing. Change page size little larger to expect more efficient communication, e.g. double the current page size.

C. *Adaptive Page Size Changing*

Adaptive page size is realized by a unified transmission of multiple basic minimum pages as shown in Fig.4 (a). It also supports a transmission of a transient fragmented large page in Fig.4 (b) generated when page size is changed from small to large. It is possible to coexistence of multiple sizes of pages in DLM. The current implementation supports 7 variations for available page size, i.e. basic size, double size, x 4 size, x8 size, x16 size, x 32 size, x 64 size. Users can set favorite basic minimum page size and initial start page size when they run their programs if they do not want to use the default values.

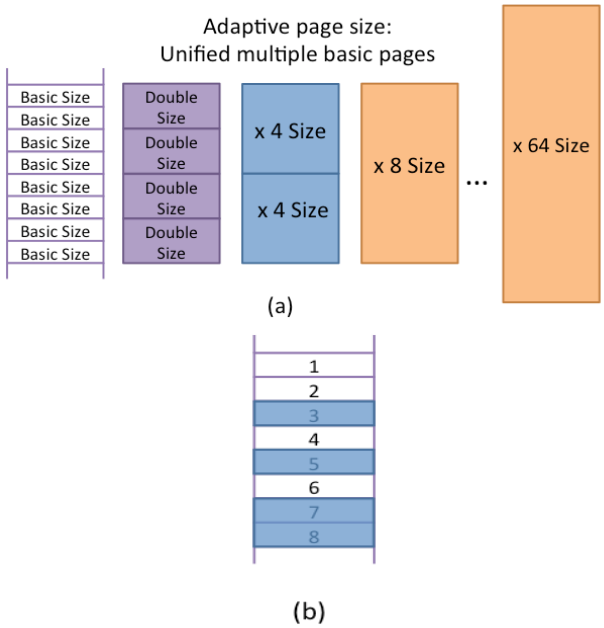

Figure 4. (a) Unifed multiple basic pages used in adaptive page size control (b) An example of transitional fragmented page after page size is changed from small to large

III. EFFECTIVENESS OF AAPC

Four benchmarks in Fig.1 are used for an evaluation of the AAPC. This experiment employs 16KB as the basic minimum page size and 1MB as the maximum page size. The initial page size when a program starts is set to 128KB. The experimental environment is shown in Table 1.

The AAPC performed satisfactorily for all programs. Fig.5 and Fig.6 show relative execution times in both cases of AAPC and fixed-size page for BT and SP respectively. They are the most effective cases with AAPC. The AAPC changes the initial page size to smaller size in the most of processing parts in BT and SP. In SP, AAPC accelerated its performance better than the best one with fixed page size of 32KB as shown in Fig.6. It was caused by the adaptive page size changing to each part of processing in SP. Fig.8 shows the execution time of each processing part of SP, and it shows SP has 2 optimal page size 16KB and 32KB depending on the processing parts. The AAPC chose the optimal page size for each processing part dynamically.

On the other hand, relative execution times in Himeno benchmark with AAPC and fixed-size page are shown in Fig.8. In this case, the performance does not change drastically between fixed-size page and the AAPC because there is no thrashing. The AAPC changes the initial page size to bigger size to increase the communication efficiency. It was confirmed that AAPC behaved appropriately for both cases and proved its effectiveness to relieve the bad impact of thrashing in the application performance in remote paging.

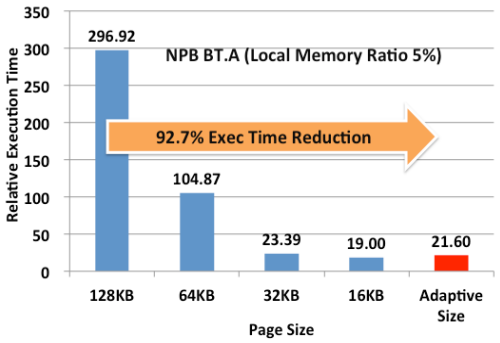

Figure 5. Relative execution times with AAPC and fixed-page size for NPB BT (Class A, Local Memory Ratio 5%)

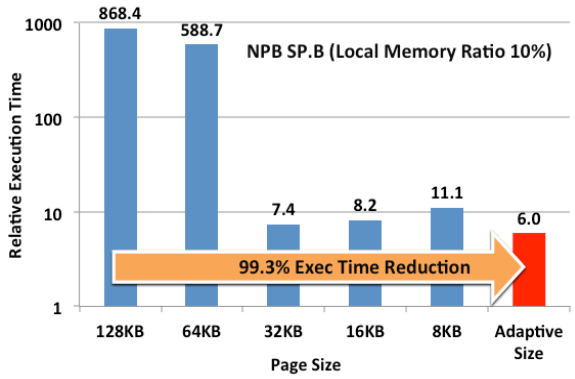

Figure 6. Relative execution times with AAPC and fixed-page size ifor NPB SP (Class B, Local Memory Ratio 10%)

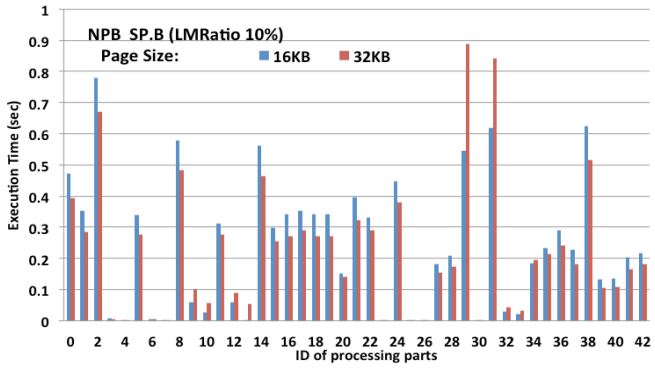

Figure 7. The execution time of individual processing parts for SP.B (Local Memory Ratio 10%) with different page sizes, 16KB and 32KB

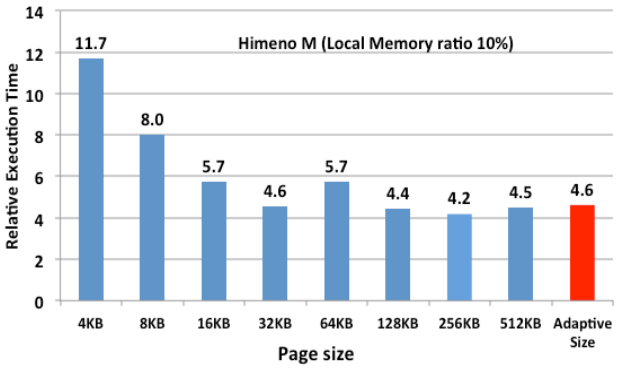

Figure 8. Relative execution times with AAPC and fixed-page size for Himeno Benchmark (Medium Size, Local Memory Ratio 10%)

TABLE I. T2K –TOKYO OPEN SUPER COMPUTER

	T2K Open Supercomputer, HA8000
CPU	AMD QuadCore Opteron 8356(2.3GHz)4CPU/ node
Memory	32GB/node (936 nodes), 128GB/node (16nodes)
Cache	L2 : 2MB/CPU (512KB/Core), L3 : 2MB/CPU
Network	Myrinet-10G x 4, (5GB/s full-duplex) bonding2 Myrinet-10G x 2, (2.5GB/s full-duplex) bonding4
OS	Linux kernel 2.6.18-53.1.19.el5 x86_64
Compiler	gcc version 4.1.2 20070626, Hitachi Optimizing C mpicc for 1.2.7
MPI Lib	MPICH-MX (MPI 1.2)

IV. CONCLUSION

The methodology proposed here is very simple but so effective that it will be applicable for various page-based memory accessing system, like distributed shared memory and general paging systems, especially for applications with various memory access patterns.

ACKNOWLEDGMENT

This work was supported by JSPS KAKENHI 21500062 and MEXT Grant-in-Aid for Building Strategic Research Infrastructures.

REFERENCES

[1] H. Midorikawa, K.Saito, M.Sato, T.Boku, “Using a Cluster as a Memory Resource: A Fast and Large Virtual Memory on MPI,” Proc. IEEE International Conf. of Cluster Computing (Cluster2009), pp.1-10, 2009

[2] H. Midorikawa, M.Kurokawa, R.Himeno, M.Sato: "DLM: A Distributed Large Memory System using Remote Memory Swapping over Cluster Nodes,” Proc. IEEE International Conf. of Cluster Computing (Cluster2008), pp.268-273, 2008

[3] Himeno Benchmark, URL [Online] 2012-3 http://accc.riken.jp/HPC_e/himenobmt_e.html,

[4] NPB, NAS Parallel Benchmark, URL [Online] 2012-3 http://www.nas.nasa.gov/publications/npb.html