2013 13th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing

User-level Remote Memory Paging for Multithreaded Applications

Hiroko Midorikawa, Yuichiro Suzuki, Masatoshi Iwaida
Graduate School of Science and Technology, Seikei University and JST CREST, Tokyo, Japan,
midori@st.seikei.ac.jp

***Abstract*- The new page swap mechanism is introduced to resolve an inconsistent page problem for multithreaded applications in user-level remote paging systems. According to the evaluations, its overhead is limited and it can be applicable to actual use for multithreaded applications.**

Keywords- memory server; distributed shared memory; memory paging; page swap; remote memory; virtual memory;

I. INTRODUCTION

Memory access detections by page faults and page-based data retrieving are essential mechanisms for virtual memory in OS kernel, page-based software distributed shared memory and remote memory paging. In user-level implementations, controlling such page management under concurrent page accesses by multiple threads requires a some kind of additional mechanism to preserve pages are consistent, unlike the kernel-level implementations in which a OS manages them. User-level implementations have advantages in high portability and easy use for ordinary people without root privilege. Moreover, there is a case where they gain higher performance and stable behavior than kernel-implementations [1], because they can tune and set various system parameters independent from an OS. In this paper, a simple exclusive page management mechanism is introduced in a user-level remote memory paging system for multithreaded applications and its overhead and effectiveness is evaluated.

In this experiment, a user-level remote memory paging system, DLM shown in Fig.1, is used. It offers virtual large memory using distributed node memory in a cluster [1][2]. It is highly portable to various clusters and easy to use remote memory for people without any knowledge of MPI. It was originally designed for users who want to solve a bigger-size problem that requires large amount of data beyond local memory size, using existing sequential algorithms and programs. They prefer and accept extra execution time caused by partially using remote memory instead of local memory because converting existing complex algorithms to parallel MPI programs is not easy task and requires substantial costs.

Recently, easy loop parallelizing by OpenMP and implicitly multithreaded library functions are available to accelerate the performance of existing sequential programs. Such a benefit of multicore also alleviates the slow execution of the programs accompanied with remote memory paging. The mechanism proposed here is available to such semi-parallel single-node multithreaded applications with remote memory paging, as well as full-parallel multiple-nodes multithreaded programs on page-based distributed shared memory systems. The mechanism is straightforward, but it shows effectiveness and acceptable performance to actual use in several applications.

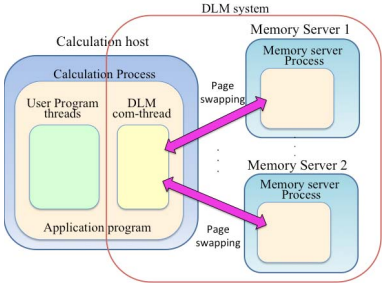

Figure 1. Remote memory paging used in virtual large memory in a cluster

II. THE PAGE SWAP FOR MULTITHREAD PROGRAMS

The inconsistent page problem caused by a multithreaded application on remote paging system is shown in Fig.2. In a calculation host, there are user application threads and a DLM *com-thread* that is automatically created when a user program starts. If user *thread-A* accesses non-local data, a page fault occurs and a handler asks the *com-thread* to retrieve the specified page from a remote memory server. The *com-thread* sends a page request to an appropriate memory server and receives the page in local host data space. The *thread-A* is suspended in the signal handler until the page receiving is finished. On the other hand, another *thread-B* has a possibility to read or write this inconsistent page area while the *com-thread* is receiving page data, because the page area is already set in read/write-possible RW state by the *com-thread.*

The basic mechanism to prevent from invalid page accessing from other threads is suspending all user threads until the page setting is correctly finished. To minimize the suspending time, another page receive buffer is provided. While a page is being received in this buffer from a memory server, the corresponding page area is still NO ACCESS state. After whole page receiving is finished, all user threads are suspended and the valid page is copied to the original page area in RW state, and after that, all threads are restarted.

To send a suspend signal to all user threads, the ID of all running threads at the moment are necessary. Multithreaded applications usually use *pthread_create()* implicitly or explicitly to create a thread. By using LD_PRELOAD, original *pthread_create()* is hooked and a modified version of

978-0-7695-4996-5/13 $26.00 © 2013 IEEE
DOI 10.1109/CCGrid.2013.63

pthread_create() which registers a thread ID in DLM system is called instead. This mechanism is effective to the programs creating threads dynamically in fork-join style like OpenMP, explicit pthread programs, and the pseudo-sequential programs using multithreaded library functions including implicit thread creations. This mechanism releases users from paying extra attentions whether their programs use threads or not.

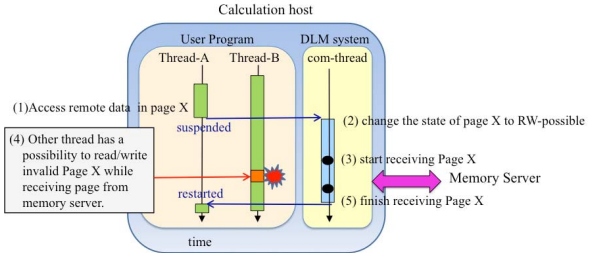

Figure 2. The inconsistent page problem caused by a multithreaded user program on remote paging system with no mechanism

III. PERFORMANCE EVALUATION

Three benchmarks including two OpenMP programs, matrix multiplication (MM) and stencil, one pseudo-sequential FFT program using fftw multithreaded library are evaluated in three systems, T2K cluster (16 cores/node, MPICH-MX, Myrinet-10G x2), FDA cluster (12 cores/node, MPICH, IB 4xQDR, IPonIB) and CREST cluster (16cores/node, MVAPICH, IB 4xFDR).

A part of the evaluation is shown here. Fig.3 to Fig5 show relative performances to an ordinary execution in which one thread using only local memory. They are evaluated with 1 to 16 threads in various *local memory ratios* (called LMR) of 5%-100% in T2K and FDA clusters. LMR is the ratio of data size in local memory to total data size used in an application. LMR 100% stands for using local memory only and LMR 5 % stands for using 5% of data in the program is in local memory and the remaining 95% of data is in remote memory. LMR 5% means that the program uses 20 times bigger size of memory compare to the local memory size.

MM in Fig.3 shows a little degradation at lower LMR, because it gains a memory access locality by blocking. It achieves 6.6 with 12 threads at LMR 10%. In 3D FFT in Fig.4, thread effectiveness is influenced by cluster's local memory bandwidth. T2K relative performance looks good because of a little degradation at lower LMR. It is caused by the saturated performance at LMR 100% with low memory bandwidth. In contrast, FD cluster gains the highest thread effectiveness at LMR 100% by its high memory bandwidth. Fig.5 shows the overhead caused by the all-thread suspension proposed here in the stencil program on FD cluster. The left is a valid case with the thread suspension and the right is the case without it that causes invalid results. It is the case that has the biggest difference. The performances with 6 to 12 threads are influenced by this thread suspension, but it is acceptable level. The overhead becomes obvious in calculation dominant applications and it hides in data access dominant applications, where threads have more chance to stop calculation to wait data. In the former case, rich calculation with rare memory accesses causes high thread effectiveness even in lower LMR.

To resolve an inconsistent page problem for multithreaded applications in user-level remote paging systems, an implicit thread ID catching and a thread suspension scheme are introduced. According to the evaluations, its overhead is limited and it can be applicable to actual use for multithreaded applications with a certain level of memory access locality and calculations. It may become one of the key technologies for multi-node multithreaded applications on page-based SDSM.

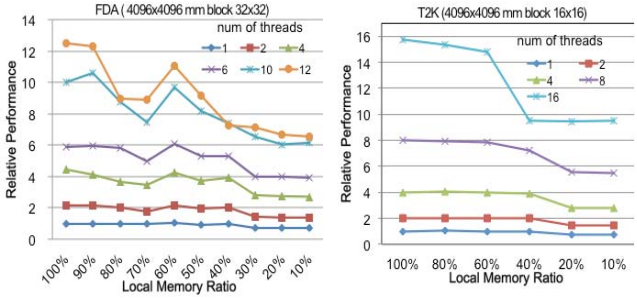

Figure 3. Matrix multiplication (4096x4096) on FDA and T2K clusters

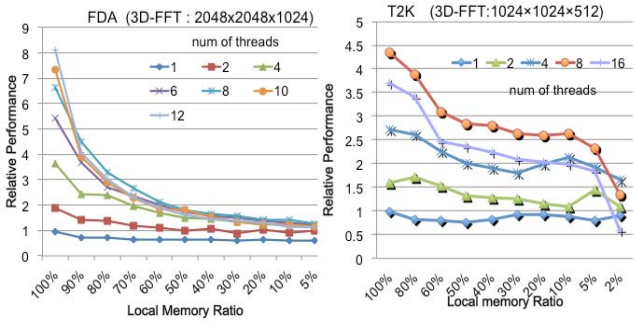

Figure 4. 3D- FFT program using fftw multithreaded library on FDA and T2K clusters

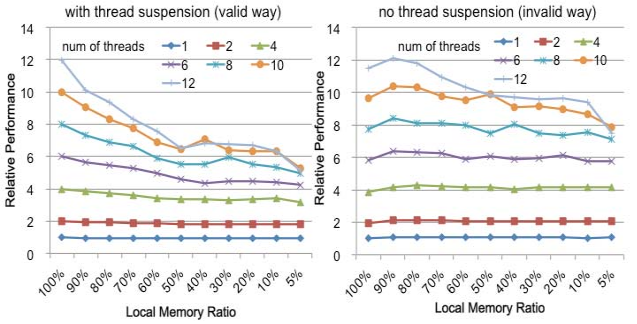

Figure 5. Thread suspension overhead in stencil program in FDA cluster (8192x8192, 15x15 mask)

REFERENCES.

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

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