Efficient Swap Protocol of Remote Memory Paging for Out-of-Core Multi-Thread Applications

Hiroko Midorikawa, Kenji Kitagawa, Hikari Ohura
Department of Computer and Information Science
Seikei University, JST CREST
Tokyo, Japan
midori@st.seikei.ac.jp

***Abstract*— A new page swap protocol is proposed for a user-level remote memory paging system to accelerate the performance of out-of-core processing with multi-thread user programs and libraries written in OpenMP and pthread. The original swap protocol has a bottle-neck in efficient page swapping which is requested by multiple threads in a user program, because all MPI communications to memory servers and page swap managements are allocated to one system thread. Though, this protocol has a benefit that it is widely available anywhere even if MPI thread-support-level is not so high. The new protocol uses two independent system threads, one for page swapping and the other for managing user thread requests. Although it requires the highest MPI thread-support-level (multiple) which is usually considered to degrade MPI communication performance compared to than in lower MPI thread-support-level, the new protocol achieves higher performance than the original one in micro benchmark, Stream benchmark, matrix multiplication, and stencil computation.**

Keywords—out-of-core, memory paging, page swap, remote memory, memory server, remote paging, swap protocol, MPI

I. Introduction

A user-level remote memory paging system, DLM (Distributed Large Memory), offers virtual large memory using distributed node memories 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 designed for users who want to solve a large-size problem that requires large amount of data beyond local memory size, using existing algorithms and programs originally designed for shared memory model. 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. DLM supports multi-thread programs and libraries written in OpenMP and pthread[1].

The current DLM page swap protocol, *protocol r3*, is shown in Fig. 1. The numbers, from (1) to (11), in the figure correspond to the steps of one page swap procedure. All page requests from user threads are managed by single DLM system thread, *com-thread*, one by one. Thus, many page requests are queueing in an internal request queue in Fig. 1. It is considered as a bottle-neck in efficient page swapping. In this centralized control system, internal system data, such as DLM page table and user thread ID table, are accessed by only *com-thread* without mutual exclusion. Moreover, this protocol has a benefit that it is widely available anywhere even if MPI thread-support-level is not so high.

II. Page Swap Protocol using Two System Threads

A new protocol, *protocol r77*, introduces an additional system thread, *rec-thread*, as shown in Fig. 2. *Com-thread* in this protocol is responsible for only managing user thread requests in (3) and (4) steps in Fig. 2. *Rec-thread* is responsible for page swapping from (5) to (11) steps in Fig. 2. Two threads communicate memory servers independently, i.e. send page requests by *com-thread,* and receive swap-in-page / send swap-out-page by *rec-thread*. Both threads access the internal system data with mutual exclusion. It also requires the highest MPI thread-support-level, MPI_THREAD_MULTIPLE, which is considered to have inferior MPI communication performance compared to default support-level, such as FUNNELED or SERIALIZED.

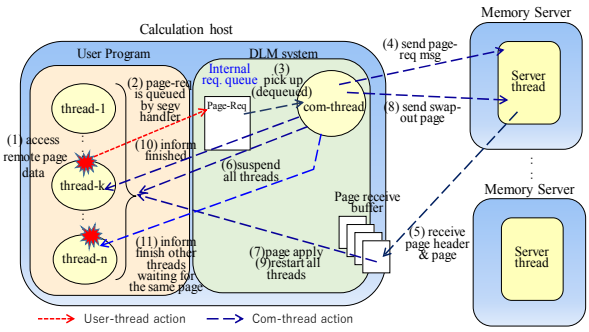

Fig.1 Page swap protocol r3 with single thread centralized control

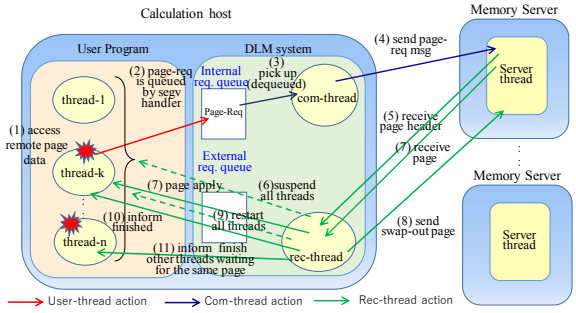

Fig. 2 Revised page swap protocol r77 with two system threads: *com-thread* for page requests and *rec-thread* for page swapping

JST CREST.

2168-9253/17 $31.00 © 2017 IEEE
DOI 10.1109/CLUSTER.2017.55

III. PERFORMANCE EVALUATION

To evaluate the new protocol, four OpenMP programs are used with Xeon cluster (64 GiB-128 GiB memory, Xeon E5-2687, E5-2687v3, FDR-InfiniBand).

A micro benchmark using 2GiB-data (Fig.3) is executed on a calculation host using 800 MB local memory and 1.2 GiB of remote memory. It allocates data by dlm_alloc function, writes sequentially, and read one data element per DLM page, 1 MB in this case. It generates very heavy page requests by multiple threads. Fig. 4 shows execution time comparison between r3 and r77 protocols for various number of threads. The new protocol r77 achieves better performance than r3.

In Stream benchmark, new protocol r77 achieves 1600 MB/s remote memory access bandwidth in contrast with previous protocol r3 achieves 1000 MB/s, as shown in Fig.5. The benchmark is evaluated on a server with 64 GiB memory.

In 7-point stencil computation with temporal-blocking algorithm [4], problems using 64 GiB – 512 GiB data are evaluated on calculation host with 128 GiB memory. Fig. 6 shows comparison of effective Mflops of r3 and r77 protocols. 512 GiB-problem performance achieves 79% of the performance in 64 GiB-problem using only local memory, even though 75% of its program data are in remote memory. Fig. 7 shows execution times of 32 K x 32 K matrix multiplication that requires 24 GiB memory. In this experiment, local memory size is limited from 120 GiB to 3 GiB, which corresponds to from 100% to 12% of 24 GiB program data. In other words, remote memory usage ratio is from 0% to 88%. Even in the case whose remote memory usage ratio is 88%, its time grows 24% of the time using only local memory. In both programs, r77 gains more advantage in larger-size problem than r3.

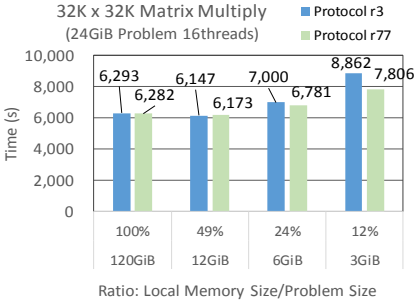

Fig.7 Matrix Multiply Time (DLM local memory 128GiB)

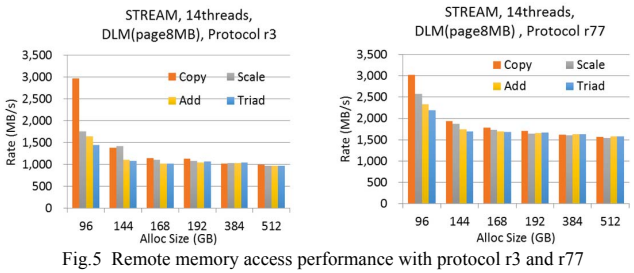

Fig.5 Remote memory access performance with protocol r3 and r77 Stream benchmark (DLM local memory 64GiB, E5-2687)

```
#define ENUM ((unsigned long ) (1L<<28))   //256M Elements
int main( )
{
  dlm_startup(&argc, &argv);
 // 2GiB array allocation
  array = (unsigned long *) dlm_alloc(sizeof(unsigned long) * ENUM);
// Initialization, array parallel write
#pragma omp parallel for
  for ( i = 0; i < ENUM; i++ ) array[i] = i;
// Array  parallel read per 1MB DLM page
#pragma omp parallel for
  for ( i = 0;  i < ENUM; i+=(1L<<17) ) // data read per 1MB
      if (array[i] != i )  return 1;
  dlm_shutdown();
}
```

Fig. 3 Micro benchmark using remote memory

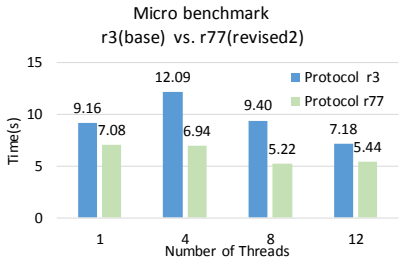

Fig. 4 Micro benchmark time: protocol r3 vs. protocol r77 (DLM local memory 800MB, E5-2687)

In all cases, proposed protocol r77 achieves higher performance by using two system threads that are responsible for page swap and user thread request management individually and communicate to memory servers independently.

Reference

[1] H. Midorikawa, Y, Suzuki, M.Iwaida: "User-level Remote Memory Paging for Multithreaded Applications", IEEE/ACM International Symp. on Cluster, Cloud and the Grid Computing CCGrid2013, pp.196-197, 2013

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

[3] Stream Benchmark https://www.cs.virginia.edu/stream/

[4] H. Midorikawa, H.Tan, T.Endo,"An Evaluation of the Potential of Flash SSD as Large and Slow Memory for Stencil Computations", IEEE The 12th International Conf. on High Performance Computing & Simulation,

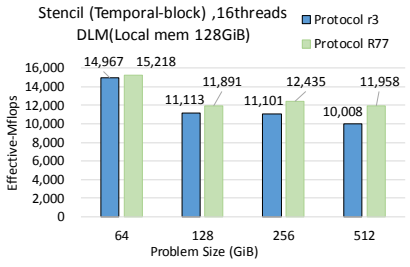

Fig.6 Temporal-block stencil computation performance (DLM local memory 128GiB, E5-2687v3)