

# Implementation of PC Cluster System with Memory Mapped File by Commodity OS

Jun Kanai<sup>†</sup>, Takuro Mori<sup>†</sup>, Takeshi Araki<sup>†</sup>,  
Noboru Tanabe<sup>††</sup>, Hironori Nakajo<sup>†</sup> and Mitaro Namiki<sup>†</sup>

<sup>†</sup>Department of Computer, Information and Communication Sciences  
Tokyo University of Agriculture & Technology  
Koganei city, Tokyo, JAPAN

<sup>††</sup>Corporate Research and Development Center, Toshiba  
Kawasaki city, Kanagawa prefecture, JAPAN

**Abstract** *In this paper, we propose a programming model to implement PC cluster systems on a non-open source of the commodity OS. For this purpose, we also introduce an implementation of a Distributed Shared memory utilizing a Distributed file system and a Memory Mapped File without modifying OS. We have designed and implemented a single DFS, by using a high-speed network interface, DIMMnet-2 which has a large capacity buffer and gathers the buffers, a device driver and a library used as DSM. As a result of a matrix multiplication evaluation, it is confirmed to improve its performance up to 1.99 times higher with two nodes distributed parallel executions.*

*Keywords:* high performance computing, cluster system, system software

## 1 Introduction

The recent development in a PC cluster system with many PCs which are connected each other is remarkable in the field of High performance Computing (HPC). Most of recent PC cluster system environments are designed based on UNIX. Microsoft Windows, however, has accounted for more than 97% of the market share and has been expected to continue to play important roles in distributed processing from now on. Implementing a distributed process environment with Windows would simultaneously realizes the distributed process-

ing in the commodity OS environment without source codes, under which OS kernel cannot be modified.

High-speed network interface for HPC, DIMMnet-2 is equipped with a lower level driver to issue commands in Linux. A distributed computing environment, however, does not support a driver of DIMMnet-2 in the commodity OS environment such as Windows.

Thus, this paper proposes a method to configure cluster systems utilizing DIMMnet-2. By designing and implementing device drivers of DIMM-net for Windows, we have configured PC a cluster system that works with commodity OS and Memory Map without modifying OS.

## 2 DIMMnet-2

DIMMnet-2 hardware is plugged into a DIMM slot of a low price PC. DIMMnet-2 can reduce access latency of an approximately 1/10 compared to NIC for HPC connected to commodity bus and expand the bandwidth to the highest level. DIMMnet-2 has from several hundred MB to several GB sizes of buffers. They are available as communication buffers and areas for data backup.

If a program uses primitive command to read or write data to I/O window, the program can access to a large capacity buffer and to a DIMMnet-2 in a remote node. The I/O win-

dow consists with the Write Window (WW) of write only, the Prefetch Window (PW) of read and others. DIMMnet-2 is equipped with various extended commands in order to access to large capacity buffers located in local and remote. Extended commands include a usual sequential access primitive, a stride access primitive accessing to the data locating at even intervals, an indirect store primitive that determines where to store the data and so on in the receiver side.

### 3 The Problems of DIMMnet-2 Programming

In this section, we discuss a message passing as a method of data sharing in a usual PC cluster system and a distributed shared memory (DSM). Then, we examine the method to implement it with DIMMnet-2 and commodity OS environment.

Message passing is a method to exchange data with writing explicitly between processes. It is the most abundantly used method and MPICH[1] is an example of an implementation of the message passing system. Message passing such as MPI has an advantage of its excellent performance. However, a message passing system can be a big burden for the programmers since it requires them to write each data exchange explicitly. Bandwidth and latency are primary concerns about message passing systems. Thus, a program needs to reduce overheads caused by a call of OS as much as possible and they should access DIMMnet-2 with user land. For an existing DIMMnet-2 driver for Linux, the driver needs to map each buffer and register to user space and send primitives to DIMMnet-2 explicitly. It complicates the programming process.

On the other hand, a distributed shared memory system (Disturbed Shared Memory: DSM) is a distributed memory type of machine and a technique to show the shared memory virtually. Since a programmer does not need to write a data exchange explicitly in the distributed shared memory system, the program-

ming is not complicated. Nevertheless, due to problems of a page fault, the number of communication, and a load of the consistency control, the distributed shared memory system often performs worse than the message passing type does. In addition, the distributed shared memory system has an issue of the consistency control when it uses a cache. The distributed shared memory system such as TreadMarks[2] usually implement a distributed shared memory using a page fault mechanism in a Memory Management Unit (MMU) of a processor. However, UDSM[3] is an example that the distributed shared memory is implemented in userlevel software.

The DSM system using a usual MMU cannot implement DSM in the environment where OS cannot be modified. In addition, the method to realize DSM only by the user mode software may bring overheads due to an inability to use the MMU. Therefore, this paper proposes the method to realize DSM utilizing MMU indirectly in the environment which does not allow to modify OS with the support from a memory mapped file and Distributed File System (DFS). A memory mapped file is a method to map a file to a virtual address space and implemented with using MMU. A program can utilize MMU without modifying OS by allocating shared data on a distributed file system to utilize memory mapped file. When allocating the shared memory, a memory mapped mechanism loads only necessary pages automatically and writes only dirty pages. Thus, this method can optimize a frequency of communication and reduce overheads caused by DSM management software. In addition, since this method has a cache in local node, page fault occurs in this method for the first time. In short, the method has an advantage of its speed. When implemented with this method, its implementation load is relatively small because DSM needs only file systems in the driver level. If DSM is implemented with this method, we only need to implement a distributed file system at a driver level, which is considered as an easy implementation. DIMMnet-2 has a large scale of buffer by SD-DIMM, which enables DIMMnet-2 ex-

change data located in SO-DIMM with high bandwidth and low latency with network functions. It is the distributed file system which moves the buffer of DIMMnet-2 to the memory area of DFS and DSM that is suitable for this method.

## 4 The Goal of this Research

The goal of this research is to configure a practical cluster system using Windows, a memory map and DIMMnet-2. First of all, a storage Driver, AT-dRAM which uses a large capacity buffer of DIMMnet-2 as RAM disk area for RAM disk is invented. Then, we realize DSM and evaluate it. Next, we have investigated whether DIMMnet-2 and AT-dRAM instead of a main memory which can maintain practical speed. Then, we have implemented a high-speed message passing system and developed and evaluated a driver MT-dNET for the expanded function of DIMMnet-2.

## 5 The Concepts of Device Drivers for a Cluster System by Windows

Based on consideration discussed in the previous section, we will discuss the design and the configuration of a RAM disk driver, AT-dRAM with which DIMMnet-2 is utilized as DFS and DSM and a driver MT-dNET with which DIMMnet-2 is accessed by user process directly. Figure 1 is an entire configuration of a PC cluster system using DIMMnet-2 and Windows we developed this time.

AT-dRAM is a RAM disk driver using a large capacity buffer of DIMMnet-2 as a disk area. AT-dRAM hides a complicated indirect access system from programmers. It also implements a distributed shared memory and a distributed file system used in a cluster system. AT-dRAM virtualizes multiple DIMMnet-2 into a single disk and conceals an indirect access system using a window. The conventional file access technique makes AT-

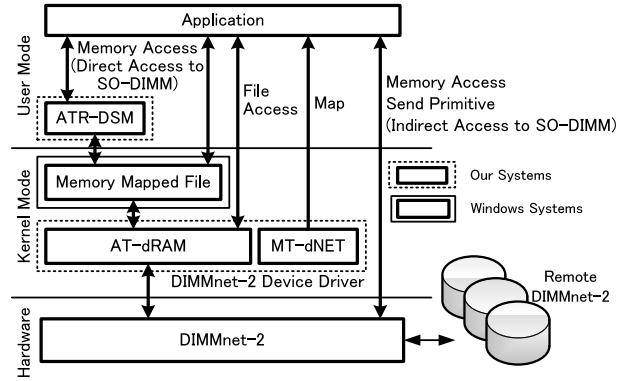


Figure 1: System Architecture

dRAM accessible to a large capacity of buffer in a local and remote DIMMnet-2. AT-dRAM is available as a distributed file system when drivers are loaded on multiple nodes.

Also, a programmer used to need to consider an indirect access system of DIMMnet-2 in usual DIMMnet-2 programming. DIMMnet-2 programming becomes easier by using a memory mapped file because an algorithm that uses a usual main memory can be applied without modifications of Windows. Moreover, DIMMnet-2 can be used as DSM when mapping the same file from multiple nodes.

In order to use DSM for a distributed processing, the system needs to have allocating shared memory area and functions and barrier synchronization and so on. Therefore, we have developed library ATR-DSM which supports these functions. In order to build a distributed shared memory rigorously, the library needs consistency control. However, because we regard its performance as most important, the programmer explicitly implements data flush and update instead a consistency control by a library.

On the other hand, a programmer sometimes wants to handle DIMMnet-2 explicitly with fewer overheads by OS in order to implement a message passing system and to use various extended commands such as a stride command. Thus, we have developed Driver MT-dNET that makes DIMMnet-2 possible operating explicitly only with a user land from application programs.

```

mat_a = (CAST)dn_malloc(MATSIZE);
mat_b = (CAST)dn_malloc(MATSIZE);
mat_c = (CAST)dn_malloc(MATSIZE);
//a conventional method uses a main memory
for (i = 0; i < SIZE; i++){
    for (j = 0; j < SIZE; j++){
        for (k = 0; k < SIZE; k++){
            (*mat_c)[i][j] += (*mat_a)[i][k] * (*mat_b)[k][j];
        }
    }
}

```

Figure 2: Example for AT-dRAM Programming

```

for (i = 0; i < SIZE; i++){
    //read a column with a continuous command
    VL(mat_a, i * step, step);
    for (j = 0; j < SIZE; j++){
        //read a row with a stride command
        VLS(mat_b, MATSIZE + j * szdbl, 3, SIZE, step);
        for (k = 0; k < SIZE; k++){
            mat_c[k] += mat_a[k] * mat_b[k];
        }
    }
    VS(mat_c, MATSIZE * 2 + i * step, step);
}

```

Figure 3: Example for MT-dNET Programming

MT-dNET maps windows and register in DIMMnet-2 to a user space of a process. It makes a program accessible to a large capacity of buffer and possible to complete DIMMnet-2 access by remote node in user land.

This method enables to ease overheads by OS and to reduce communication latency as much as possible. A program needs to access data explicitly by using a library executed in a user mode in order to access DIMMnet-2.

## 6 Programming Examples

In this section, we introduce examples of the programming developed with drivers and libraries and discuss its usefulness.

Figure 2 and 3 illustrate a programming of a matrix multiplication with a local programming at a local node with AT-dRAM and a user mode library. A matrix multiplication with a conventional method uses a main memory (with AT-dRAM and a memory mapped file). A matrix multiplication with a column read command with continuous read and row read with a stride command (explicit issue with MT-dNET and library).

As the figures indicate, a programmer no longer needs to use an indirect access mechanism of DIMMnet-2 and utilize almost the same algorithm as regular one with AT-dRAM.

In other words, the programmer can easily carry out DIMMnet-2 programming. Moreover, AT-dRAM uses a file system cache, which enables AT-dRAM to minimize access to DIMMnet-2 and improve its performance. On the other hand, the programmer needs to code the data exchange with DIMMnet-2 explicitly when using MT-dNET and stride commands. However, a stride command enables a programming with effective cache. In our example, we use a library for a primitive issuing. However, the programmer has to check a status register due to the consideration of DIMMnet-2’s window size even with only MT-dNET. Thus, it makes program more complicated.

## 7 Performance Evaluation

### 7.1 Overheads of the OS

Since AT-dRAM is implemented as a storage driver, an AT-dRAM brings overheads such as a driver call. We have applied the following formula and measured the OS overheads when accessing files of the explicitly described API.  $T_{os}$  here is an overhead by OS,  $N_{call}$  is the number of OS call (actual measurement value),  $N_{size}$  is the size of the overheads (actual measurement value), and  $T_{transspd}$  is time for transfer (actual measurement value). We have calculated the cost for one time of the OS call by calculating the differences in total time ( $\Delta T_{all}$ ) when changing the size of the overheads to be transferred at one time without changing the entire transferred size and also changing the number of an API call ( $\Delta N_{call}$ ).

$$T_{all} = T_{os}N_{call} + N_{size}T_{transspd} \quad (1)$$

$$T_{os} = \Delta T_{all} / \Delta N_{call} \quad (2)$$

Next, we have measured overheads of OS when executing a file access by a memory mapped file. For reading, when a page fault occurs in each page (4KB), a storage driver is called and necessary data of the page is read from the disk.

$$T_{mmap} = (T_{all} - T_{transspd}N_{size}) / N_{page} \quad (3)$$

The time for reading from the disk is calculated with formula (1). We have calculated OS overheads of a memory mapped file per page using formula (3).  $T_{transspd}$  describes the time for reading per sector from the disk (actual measurement value),  $N_{size}$  shows total number of times for the reading from disks (actual measurement value) and  $N_{page}$  shows the number of the pages which are already read (actual measurement value). When a memory mapped file is applied, OS automatically writes the updated data on a disk. However, a programmer can code to write on the disk explicitly. In ATR-DSM, a programmer often codes to write on the disk explicitly. We calculate the overheads with the formula (2). Table 1 shows the overheads. The overheads are larger when a memory mapped file is used than when ReadFileAPI is used. When a memory mapped file is used, these overheads occur only the first time access. Therefore, the performance is higher when a memory mapped file is used compared to the case when read files are called in several times explicitly.

## 7.2 Transfer Rate between SO-DIMM and Host Memory

As shown in Figure 4, we have measured MT-dNET and measure its transfer rate between a SO-DIMM buffer of DIMMnet-2 and host memory without OS calls. Figure 4 illustrates its performance of the transfer. The current hardware of a DIMMnet has a problem. Therefore, the cache attribute of the window of DIMMnet-2 is Uncached. Since a cache attribute to a write window becomes Write Combine and a cache attribute become Cached in the future, its transfer rate will be improved. A combination of these cache attributes is called DriverDefault.

Table 1: OS Overhead for Data Access

	time[us]
Read(ReadFile)	4.27
Write(WriteFile)	3.75
Read(MemoryMappedFile)	9.87
Write(MemoryMappedFile)	4.39

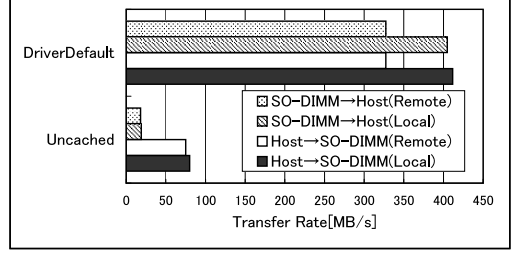


Figure 4: Transfer Rate between DIMMnet-2 SO-DIMM and Host Memory(MT-dNET)

## 7.3 The DFS Performance

We have estimated a distributed file system performance in the case of the cache attributes of Uncached and a disk performance  $T_{DD}$  in a case of DriverDefault with transfer performance between a SD-DIMM buffer and a host memory with the following formulas.

$$T_{DD} = T_{os} + T_{DDD} \quad (4)$$

$$= (T_{all} - T_{DUN}) + T_{DDD} \quad (5)$$

$T_{os}$  shows overheads by OS call,  $T_{DDD}$  shows transfer time to SO-DIMM when OS is not used in DriverDefault.  $T_{all}$  shows total amount of file access time in Uncached with OS.  $T_{DUN}$  shows transfer time to SO-DIMM in Uncached without OS. The result is shown in Figure 9. The transfer performance in DriverDefault decreases its transfer performance up to 9% compare to one handling DIMMnet-2 explicitly from a user program. This performance degradation is caused by a file system management and overheads by OS. However, the transfer performance is still up to twelve times more than its performance limit by 30MB/s with a file sharing using SMB. In other words, AT-dRAM will dramatically improve its performance as a distributed file system.

## 7.4 An Evaluation by a Distributed Processing

We have evaluated some distributed processing by using AT-dRAM and ATR-DSM. As mentioned earlier, the current cache attributes of each window of DIMMnet-2 is Uncached. We estimate the performance with the formula be-

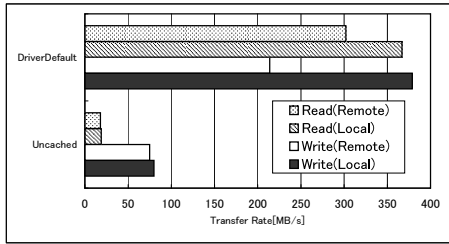


Figure 5: Transfer Rate of DIMMnet-2 RAM Disk(AT-dRAM)

low by assuming that the problems in hardware were solved.

The number of times of reading in the disk  $Nr_i$ , the number of times of writing  $Nw_i$  and reading time  $Tr_i$  in case of Uncached and writing time  $Tw_i$  in case of Unchahed are measurement value. AT-dRAM now limits re-sending when it is remote transfer. Currently, when AT-dRAM is in a remote transfer, software has a control over re-sending. When  $i$  is more than 1,  $i$  means time required for re-sending and the number of times re-sending. The time required to re-send increases as the number of times to resend. Because AT-dRAM is no longer limited re-sent in DIMMnet-2,  $Nr_i$  and  $Nw_i$  become zero when  $i$  is more than 1.  $Tw_i$ ,  $Tw_i$  and total transfer time  $T_{trans}$  when the cache attribute is a Driver Default are estimated in Figure 4.  $T_{os}$  is an overhead by a page fault and data flush OS calls. OS calls by a page fault is computed. The former is computed by the product of calculated overheads per page fault  $T_{mmap}$  and calculated page numbers  $N_{page}$ . Overheads by data flush are computed by a product of calculated overheads per data flush  $T_{flush}$  and calculated the number of flush.  $N_{flush}$ . Either  $T_{mmap}$  or  $N_{flush}$  does not include transfer time.  $T_{barrier}$  is a sum of required time for barrier synchronization. It has been proved by a measurement that it costs at least 1 [ms.]  $T_{barrier}$  is parameter and the numerical value which makes each  $T_{all}$  in each node equivalent.  $T_{calc}$  is total sum of required time for computation except for the time required for OS overheads, disk transfer time and barrier synchronization.

$$T_{all} = T_{os} + T_{barrier} + T_{trans} + T_{calc} \quad (6)$$

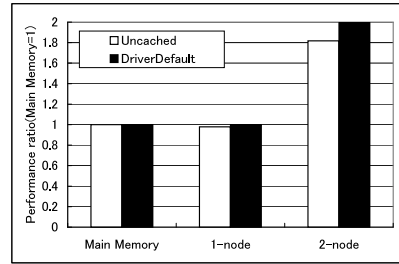


Figure 6: A Result of Distributed Matrix Multiplication

$$T_{os} = T_{mmap}N_{page} + T_{flush}N_{flush} \quad (7)$$

$$T_{trans} = \sum_{i=0}^N Tr_i Nr_i + \sum_{i=0}^N Tw_i Nw_i \quad (8)$$

$$T_{barrier} = T_{all} - \max(T_{os_i} + T_{trans_i} + T_{calc_i}) \quad (9)$$

### (1) A Matrix Multiplication

We have evaluated a matrix multiplication by a simple distributed processing. The program divides a required matrix into  $P$  toward a column and has measure a matrix product. Figure 6 shows the result. Disk accessing time in total processing time for a matrix multiplication is short. Moreover, each node has the same amount of computing and there are few needs for synchronization. Therefore, a matrix multiplication can be parallelized very effectively. The multiplication computing by two nodes has become 1.99 times faster.

### (2) Wisconsin Benchmark

Wisconsin Benchmark is a benchmark to measure a database performance. We have coded 6 queries included in Wisconsin Benchmark in Figure 7 explicitly and measure their performance with C language. Assigned data storage by a benchmark is either SO-DIMM of DIMMnet-2 or a hard disk. Figure 8 shows the result of a distributed processing of Wisconsin Benchmark.

```
(Q1) select * from tenk1 where (unique2 > 301) and (unique2 < 402)
(Q2) select * from tenk1 where (unique1 > 647) and (unique1 < 65648)
(Q3) select * from tenk1 where unique2 = 2001
(Q4) select * from tenk1 t1,tenk1 t2 where (t1.unique2 = t2.unique2)
and (t2.unique2 < 1000)
(Q6) select t1.*,o.* from onek o,tenk1 t1,tenk1 t2 where (o.unique2 =
t1.unique2) and (t1.unique2 = t2.unique2) and (t1.unique2 != 1000) and
(t2.unique2 != 1000)
(Q7) select MIN(unique2) from tenk1
```

Figure 7: Queries in Wisconsin Benchmark

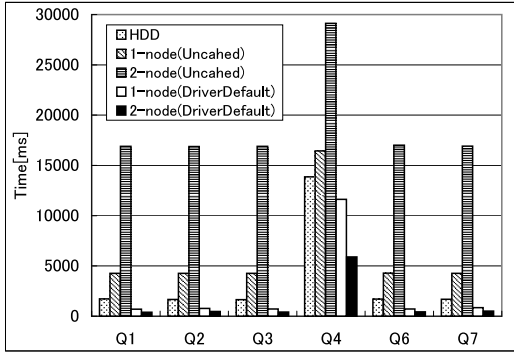


Figure 8: A Result of Wisconsin Benchmark

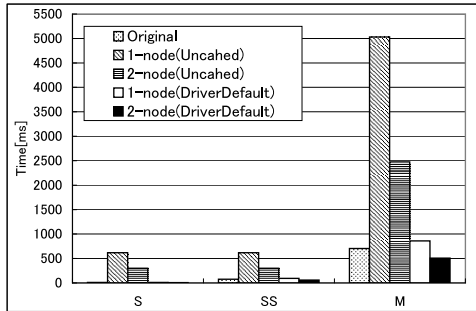


Figure 9: A Result of Himeno Benchmark

The disk performance has significant influence in queries of Wisconsin Benchmark without using simple join commands. While performance of AT-dRAM has been improved 2.49 times better than that of HDD in Q1, a distributed processing is useful in queries with a join (Q4,Q6). A processing with two nodes improves its performance 1.97 times better than that of a single node in Q4. AT-dRAM and DIMMnet-2 are also effective in the field required to store enormous data such as database.

### (3) Himeno Benchmark

Himeno Benchmark is invented in order to evaluate the performance of non-compressed fluid analysis codes and is measured by speed of a main loop in the case of Poisson equation by Jacobi iteration. We have measured the performance of Himeno Benchmark by using a new edition which is based on the program of OpenMP edition and revised to use AT-dRAM and ATR-DSM and a single node edition (an original edition) which uses main memory.

The ratio of disk access time out of a total time is large in Himeno Benchmark. Its performance declines 0.82 times as low in the case of

one node using AT-dRAM as an original version using main memory. In addition, a distributed processing in two nodes has only been improved 1.38 times more than the originals. However, a two node distributed processing using ATR-DSM and AT-dRAM has improved its performance 1.69 more than a single node. It proves that ATR-DSM and AT-dRAM are effective even to Himeno Benchmark.

## 8 Conclusions

We have developed a driver of DIMMnet-2 for Windows and evaluated its distribution system by configuring a cluster system in this study. As a result, we have implemented a distributed processing environment with DIMMnet-2 in Windows. In our experimental evaluation, the distributed processing has become up to 1.99 times more effective and therefore, we have proved its usefulness as a cluster system by our method.

Although we have discussed mainly DIMMnet-2 and Windows in this paper, the application of our method is not limited to them. Moreover, this method implies a possibility of a distributed processing by a device driver and a memory mapped file in the commodity OS environment.

## References

- [1] W. Gropp, E. Lusk: A High-Performance, Portable Implementation of the MPI Message Passing Interface Standard, *Parallel Computing*, Vol. 22, No. 6, pp. 789–828, (1996.9)
- [2] P. Keleher, S. Dwarkadas, A.L. Cox, and W. Zwaenepoel: TreadMarks: Distributed Shared Memory on Standard Workstations and Operating Systems, *Proc. of the Winter 94 Usenix Conf.*, pp. 115–131 (1994.1).
- [3] T. Inagaki, J. Niwa, T. Matsumoto and K. Hiraki: Supporting Software Distributed Shared Memory with an Optimizing Compiler, *Proc. of 1998 Int. Conf. on Parallel Processing (ICPP'98)*, pp.225–234 (1999.10).