Using Available Remote Memory Dynamically for Parallel Data Mining Application on ATM-Connected PC Cluster

Masato Oguchi [1,2] and Masaru Kitsuregawa [1]

[1] Institute of Industrial Science, The University of Tokyo
7-22-1 Roppongi, Minato-ku Tokyo 106-8558, Japan
[2] Informatik4, Aachen University of Technology
Ahornstr.55, D-52056 Aachen, Germany
oguchi@computer.org

Abstract

Personal computer/Workstation (PC/WS) clusters are promising candidates for future high performance computers, because of their good scalability and cost performance ratio. Data intensive applications, such as data mining and ad hoc query processing in databases, are considered very important for massively parallel processors, as well as conventional scientific calculations. Thus, investigating the feasibility of data intensive applications on a PC cluster is meaningful.

Association rule mining, one of the best-known problems in data mining, differs from conventional scientific calculations in its usage of main memory. It allocates many small data areas in main memory, and the number of those areas suddenly grows enormously during execution. As a result, the contents of memory must be swapped out if the requirement for memory space exceeds the real memory size. However, because the size of each data area is rather small and the elements are accessed almost at random, swapping out to a storage device must degrade the performance severely.

In this paper, we investigate the feasibility of using available remote nodes' memory as a swap area when application execution nodes need to swap out their real memory contents during the execution of parallel data mining on PC clusters. We report our experiments in which application execution nodes acquire extra memory dynamically from several available remote nodes through an ATM network. A method of remote memory utilization with remote update operations is proposed and evaluated. The experimental results on our PC cluster show that the proposed method is expected to be considerably better than using hard disks as a swapping device. The dynamic decision mechanism for remote memory availability and the migration operations are also evaluated.

1. Introduction

Recently data intensive applications such as data mining and data warehousing have been focused as one of the most important applications for high performance computing. As a platform, Personal computer/Workstation (PC/WS) cluster is a promising candidate for future high performance computers, from the viewpoint of good scalability and cost performance ratio. We previously developed a large scale ATM-connected PC cluster, and implemented several database applications, including parallel data mining, to evaluate their performance and the feasibility of such applications using PC clusters[1][2].

Different from the conventional scientific calculations, association rule mining, one of the best known problems in data mining, has a peculiar usage of main memory. It allocates a lot of small data on main memory, and the number of those areas multiplies to be enormous during the execution. Thus, the requirement of memory space changes dynamically and becomes extremely large. Contents of memory must be swapped out if the requirement exceeds the real memory size. However, because the size of each data element is rather small and all the elements are accessed almost at random, swapping out to a secondary storage system is likely to cause severe performance degradation. We are investigating the feasibility of using available memory in remote nodes as a swap area, when application execution nodes need to swap out their memory contents.

In this paper, we report our experimental results in which

nodes executing an application acquire extra memory dynamically from several remote nodes in the ATM-connected PC cluster. Moreover, a method using distant node's memory with remote update operations, which is expected to prevent a false-sharing problem, is proposed and evaluated. The rest of paper is organized as follows. In Section 2, the data mining application and its parallelization are explained. In Section 3, an overview of our PC cluster is presented, and an implementation of parallelized data mining on our PC cluster is described. The method of dynamic remote memory utilization for parallel data mining is explained in Section 4. In Section 5, performance results of the evaluation of proposed mechanisms are shown and analyzed. Final remarks are made in Section 6.

2. Data mining application and its parallelization

2.1. Mining of association rules

Data mining has attracted a lot of attention from both research and commercial community, for finding interesting trends hidden in large transaction logs. Data mining is a method for the efficient discovery of useful information, such as rules and previously unknown patterns existing among data items in large databases, thus allowing for more effective utilization of existing data.

Large transaction processing system logs have been accumulated because of the progress of bar-code technology. Such data was just archived and not used efficiently until recently. The advance of microprocessor and secondary storage technologies allows us to analyze vast amount of transaction log data to extract interesting customer behaviors. For very large mining operations, however, parallel processing is required to supply the necessary computational power.

One of the best known problems in data mining is mining of association rules from a database, so called ''basket analysis''[3][4][5]. Basket type transactions typically consist of a transaction identification and items bought per transaction. An example of an association rule is ''if customers buy A and B, then 90% of them also buy C''. The best known algorithm for association rule mining is the Apriori algorithm proposed by R. Agrawal of IBM Almaden Research[6][7][8].

Apriori first generates so-called candidate itemsets (groups consisting of one or more items), then scans the transaction database to determine whether the candidates have the user-specified minimum support. In the first pass (pass 1), support for each item is counted by scanning the transaction database, and all items that achieve the minimum support are picked out. These items are called large 1-itemsets. In the second pass (pass 2), 2-itemsets (pairs of two items) are generated using the large 1-itemsets. These 2-itemsets are called the candidate 2-itemsets. Support for the candidate 2-itemsets is then counted by scanning the transaction database. The large 2-itemsets that achieve the minimum support are determined. The algorithm goes on to find the large 3-itemsets, the large 4-itemsets, and so on. This iterative procedure terminates when a large itemset or a candidate itemset becomes empty. Association rules that satisfy user-specified minimum confidence can be derived from these large itemsets.

2.2. Parallelization of association rule mining

In order to improve the quality of the rule, we have to analyze very large amounts of transaction data, which requires considerable computation time. We have previously studied several parallel algorithms for mining association rules[9], based on Apriori. One of these algorithms, called Hash Partitioned Apriori (HPA), is implemented and evaluated on the PC cluster. HPA partitions the candidate itemsets among processors using a hash function, like the hash join in relational databases. HPA effectively utilizes the whole memory space of all the processors, hence it works well for large scale data mining. The steps of the algorithm are as follows.

1. Generate candidate k-itemsets:

   All processors have all the large $(k-1)$-itemsets in memory when pass k starts. Each processor generates candidate k-itemsets using large $(k-1)$-itemsets, applies a hash function, and determines a destination processor ID. If the ID is the processor's own, the itemset is inserted into the hash table, otherwise it is discarded.

2. Scan the transaction database and count the support value:

   Each processor reads the transaction database from its local disk. It generates k-itemsets from those transactions and applies the same hash function used in phase 1. The processor then determines the destination processor ID and sends the k-itemsets to it.

   When a processor receives these itemsets, it searches the hash table for a match, and increments the match count.

3. Determine large k-itemsets:

   Each processor checks all the itemsets it has and determines large itemsets locally, then broadcasts them to the other processors. When this phase is finished at all processors, large itemsets are determined globally. The algorithm terminates if no large itemset is obtained.

3. ATM-connected PC cluster

3.1. Research works on PC clusters

Components of today's high performance parallel computers are evolving from proprietary parts, e.g. CPUs, disks, and memories, into commodity parts. This is because technologies for such commodity parts have matured enough to be used for high-end computer systems. While an interconnection network has not yet been commoditized thus far, ATM technology is one of the strong candidates as a de facto standard of high speed communication networks. With the progress of high performance networks, such as ATM, future parallel computer systems will undoubtedly employ commodity networks as well.

Thus, PC/WS clusters using high performance commodity networks have become an exciting research topic in the field of parallel and distributed computing. They are expected to play an important role as large-scale parallel computers in the next generation, because of their good scalability and cost performance ratio.

Initially, the processing nodes and/or networks of clusters were built from customized designs, since it was difficult to achieve good performance using only off-the-shelf products[10][11]. Such systems are interesting as research prototypes, but most failed to be accepted as common platforms. However, because of advances in workstation and high-speed network technologies, reasonably high performance WS clusters can be built using off-the-shelf workstations and high speed LANs[12].

Until recently, workstations were overwhelmingly superior to personal computers, in terms of performance as well as sophisticated software environments. However, the latest PC technology has dramatically increased CPU, main memory, and cache memory performance. While RISC processors used in today's WSs provide much better floating point performance than the microprocessors used in PCs, some applications, such as database processing, primarily require good integer performance. Since today's PCs and WSs have almost comparable integer performance, PCs have better cost performance ratios than do WSs for database operations. High-speed bus architectures, such as the PCI bus, have also improved I/O performance of PCs. Because the size of the PC market is much larger than the WS market, further improvements in the cost performance ratio are expected for PC clusters.

Various research projects, which develop and examine PC/WS clusters, have been performed until now[13][14][15][16][17][18]. However, most of them only measured basic characteristics of PCs and networks, and/or some small benchmark programs were examined. We believe that data intensive applications such as data mining and ad hoc query processing in databases are quite important for future high performance computers, in addition to the conventional scientific applications. We have developed a pilot system of ATM-connected PC cluster consists of 100 Pentium Pro PCs, and evaluated it with database applications including the TPC-D benchmark and a data mining application[1][2].

Table 1. Each node of the PC cluster

CPU	Intel 200MHz Pentium Pro
Chipset	Intel 440FX
Main memory	64Mbytes
IDE hard disk	WesternDigital Caviar32500 2.5Gbytes
SCSI hard disk	Seagate Barracuda 4.3Gbytes
OS	Solaris 2.5.1 for x86
ATM NIC	Interphase 5515 PCI Adapter

3.2. An overview of our PC cluster pilot system

100 nodes of 200MHz Pentium Pro PCs are connected with an ATM switch in our PC cluster pilot system. Each node consists of the components shown in Table 1.

HITACHI's AN1000-20, which has 128 port 155Mbps UTP-5, is used as an ATM switch. Since this switch has 128 port, all nodes can be connected directly with each other, forming a star topology rather than a cascade configuration. All nodes of the cluster are connected by a 155Mbps ATM LAN as well as by an Ethernet. An RFC-1483 PVC driver, which supports LLC/SNAP encapsulation for IP over ATM[19][20], is used. Only UBR traffic class is supported in this driver. TCP/IP is used as a communication protocol. An overview of the PC cluster is shown in Figure 1.

3.3. Implementation of parallelized association rule mining

HPA program described in Section 2 was implemented on the PC cluster pilot system. Each node of the cluster has a transaction data file on its own hard disk. Transaction data was produced using a data generation program developed by Agrawal, designating some parameters, such as the number of the transaction, the number of different items, and so on[8]. The produced data was divided by the number of nodes, and copied to each node's hard disk. WesternDigital Caviar32500 IDE disks are used for this purpose.

At each node, two processes are created and executed. One process makes candidate itemsets from previous large itemsets, and sends them to the other process, which puts the data into a hash table. In the data counting phase, one

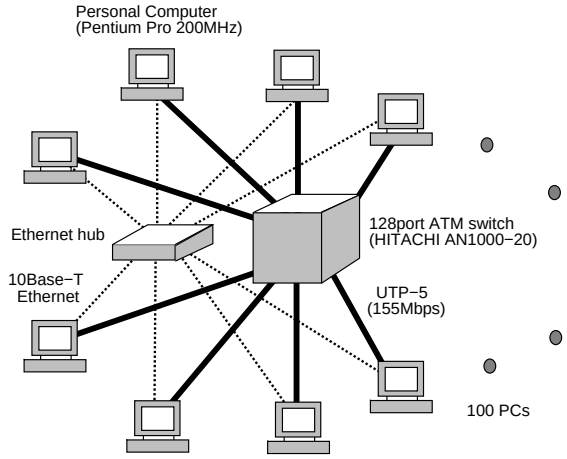

Figure 1. An overview of the PC cluster

process generates itemsets by scanning the transaction data file, and sends them to the other processes on the nodes selected by the hash function. The target processes check and increment their hash table values accordingly.

Solaris Transport Layer Interface (TLI) system calls are used for the inter-process communication. All processes are connected with each other by TLI transport endpoints, thus forming a mesh topology. /dev/tcp is used as a transport layer protocol. It is a two-way connection based byte stream. On the ATM level, Permanent Virtual Channel (PVC) switching is used, since data is transferred continuously between all processes.

During the execution of HPA, itemsets are kept in memory as linked structures that are classified by a hash function. That is to say, all itemsets having the same hash value are assigned to the same hash line on the same node, and their structures are connected with each other to form a list.

As an example, a result of HPA is shown in Table 2. In this execution, the number of transactions is 10,000,000, the number of different items is 5,000, and the minimum support is 0.7%. When the PC cluster using 100 PCs is employed for this problem, reasonably good performance improvement is achieved[2][21].

It is known that the number of candidate itemsets in pass 2 is very much larger than in other passes, as can be seen in the table. This often happens in association rule mining.

Table 2. The number of candidate and large itemsets at each step

C	Number of candidate itemsets
L	Number of large itemsets

pass	C	L
pass 1		1023
pass 2	522753	32
pass 3	19	19
pass 4	7	7
pass 5	1	0

4. Using available remote memory dynamically on the PC cluster

4.1. Swapping operation during the execution of data mining

As mentioned in Section 3, the number of candidate itemsets in pass 2 is very much larger than other passes in association rule mining. The number of itemsets is strongly dependent on user-specified conditions, such as minimum support value, and it is difficult to predict before execution how large the number will be before execution. Therefore, it may happen that the number of candidate itemsets increases dramatically in this step so that the requirement of memory becomes extremely large. When the required memory is larger than the real memory size, part of the memory contents must be swapped out. However, because the size of each data is rather small and all the data is accessed almost at random, swapping out to a storage device is expected to degrade performance severely in this case. In the case of large scale clusters, a large fraction of the memory of total nodes is not in active use on average[22]. In the rest of this paper, several methods are discussed in which available memory in remote nodes is used as a swap area, when huge memory is dynamically required during the execution of parallel data mining on PC clusters.

4.2. Dynamic decision mechanism for remote memory availability

Remote nodes, whose memories are available for application execution nodes, are found dynamically during the execution. We call them ‘‘memory available nodes’’. The mechanism to decide the availability of remote nodes is shown in Figure 2. On memory availble nodes, a process is running to monitor the amount of avilable memory periodically. ‘‘netstat -k’’ command provided by Solaris

operating system is used to get memory information from the kernel statistics structure. [1] Each time the process gets the information, the process broadcasts it to all application execution nodes.

On application execution nodes, a client process is running and waiting for the information sent from the memory monitoring processes running on memory available nodes. The client process has a memory area which can be shared with application processes, and the received information about the amount of memory at each node is written on the shared memory. The application processes can read this information at anytime, to decide the policy of swapping operations. For example, when a memory available node does not have enough memory space, the shortage of memory is detected by application processes, so that another node is chosen as a swapping destination afterward.

On the other hand, if some other processes begin their execution on a memory available node which already accepted swapping operations, the swapped out data must be migrated to other memory available nodes to make space on its memory for the new processes. This mechanism works as follows. First, the memory shortage is monitored on a memory available node, then this information is broadcast to client processes of all application execution nodes. When the application execution nodes detect this memory shortage, they check a memory management table which shows where each entry currently exists. If they find that part of their memory contents has been swapped out to this memory available node, they sends a migration direction to this node to tell to which node these entries should be migrated. The memory available node migrates its contents to other memory available nodes, according to the direction.

4.3. Dynamic remote memory acquisition with simple swapping

As a method of experiments, a limit value for memory usage of candidate itemsets is set at each node. When the amount of memory usage exceeds this value during the execution of HPA program, part of contents is swapped out to available remote nodes' memory. That is to say, the application execution node acquires memory area dynamically from one of memory available nodes when it is needed.

When the number of candidate itemsets becomes extremely large in pass 2 and the amount of memory usage exceeds a specified value, the node sends some of its memory contents to destination memory available nodes. The unit of swapping operation is a hash line, which is a listed structures as explained in Section 3. The hash line swapped out is selected using a LRU algorithm. At the memory available node, the received contents are allocated and written in its main memory. Each memory available node may receive swapped out data from several application execution nodes.

A pagefault occurs, on the other hand, when an application execution node accesses an item that had been swapped out. It sends a request to a memory available node which holds the data, and the memory available node sends back the requested hash line. After the application execution node receives the contents, they are allocated and written on main memory again, then the execution of application resumes. Replacements of data are decided by LRU manner. The maximum number of nodes used as memory available nodes can be changed as required from one to many in the experiment.

The basic behavior of this approach has something in common with distributed shared memory systems (See[24], for example). If data structures inside applications are considered in distributed shared memory, almost the same effect can be expected. That is to say, it is possible to program almost the same mechanism using some types of distributed shared memory systems. Thus, our mechanism might be regarded as equivalent to a case of distributed shared memory optimized for a particular application.

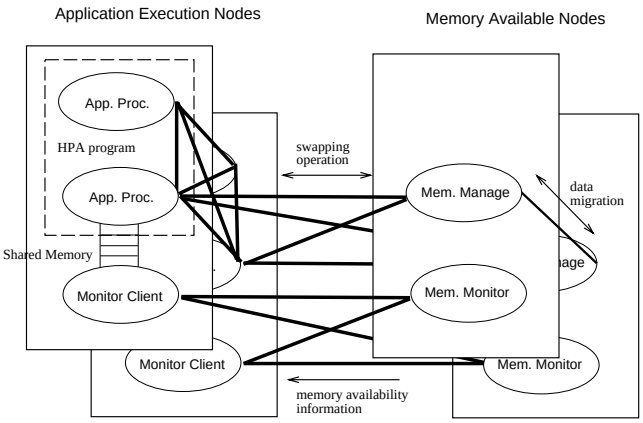

Figure 2. Dynamic decision mechanism for remote memory availability

4.4. Remote memory update operation

Because most of memory contents are accessed repeatedly, the number of pagefaults is considered to become very high when the memory usage limit is small. A kind of false-sharing may happen in such a case. In order to prevent this phenomenon, we investigate a method to restrict swapping operations.

When usage of memory reaches to the limit value at an application execution node, it acquires remote memory and

[1] The ''-k'' option to netstat command is not documented on the manual pages. See [23] for more information.

swaps out part of its memory contents. The contents will be swapped in again if this data is accessed later. Instead of swapping, it is sometimes better to send update information to the remote memory when a pagefault occurs. That is to say, once some contents are swapped out to memory in a remote node, they are fixed at there and accessed only through a remote memory access interface provided by library functions. We apply this policy to the itemsets counting phase, in which the memory access operation is simple -- to compare itemsets with the contents of the hash table and update the table.

5. Performance analysis of dynamic remote memory utilization

5.1. Implementation of the mechanism on PC cluster

The proposed mechanism has been implemented on the PC cluster pilot system. The parameters used in the experiment are as follows: The number of transactions is 1,000,000, the number of different items is 5000, and the minimum support is 0.1%. The size of the transaction data is about 80Mbytes in total. The message block size is set to be 4Kbytes, and the disk I/O block size is 64Kbytes.

The number of application execution node is 8 in this evaluation. The number of hash line for candidate itemsets is 800,000 in total, hence about 100,000 hash lines are assigned to each node during the execution. The unit of swapping operation is a hash line, which can be contained in one message block.

With the above conditions, the number of candidate itemsets in pass 2 was 4,871,881 in total. These candidate 2-itemsets are assigned to each node using a hash function. The numbers of candidate 2-itemsets at each node are shown in Table 3. Although the itemsets are assigned using a hash function, the numbers at each node are not equal. This frequently happens in the execution of HPA, because some amount of skew usually exists in transaction data in association rule mining. We have also developed a method to treat it, which can be found in another paper[25].

Since each candidate itemset occupies 24bytes in total(structure area + data area), approximately 14-15Mbytes of memory are filled with these candidate itemsets at each node.

Table 3. The number of candidate 2-itemsets at each node

node 1	node 2	node 3	node 4
602559	641243	582149	614412
node 5	node 6	node 7	node 8
604851	596359	622679	607629

5.2. Dynamic remote memory acquisition with simple swapping

First, a method using available remote nodes' memory with simple swapping is examined. The maximum number of nodes used as memory available nodes is changed from 1 to 16. In this experiment, all memory available nodes are assumed to have enough memory space to accept requests of swapping operations. In such a case, all the memory available nodes are used for swapping operations throughout the execution of the program and therefore the number of memory available nodes is constant during the experiment. The execution time of pass 2 in HPA program, when the number of memory available nodes changes from 1 to 16, is shown in Figure 3. In this figure, the result of 5 different cases are shown. The upper 4 lines are the cases of memory usage for candidate itemsets being limited as 12[MB], 13[MB], 14[MB], and 15[MB], respectively. The lowest line is the case with no memory usage limit.

When the number of memory available nodes is small, the execution time is quite long especially when the memory usage limit size is smaller. Apparently memory available node(s) become bottleneck in these cases. This bottleneck is resolved when the number of memory available nodes is 8 - 16 in this experiment.

The execution time becomes longer as the memory usage limit size becomes small, since the number of swap out increases in such cases. When memory usage is limited, the execution time is quite longer than that of the no memory limit case. This is because the number of swap out is extremely large.

We can calculate the execution time of each pagefault as follows. We will focus on the case when the number of memory available nodes is 16, in which memory available nodes are not considered to be bottleneck. In the case of memory usage limit being 13[MB], for example, the execution time of the program is 4674.0[s], and the difference of the execution time between this and the no memory limit case is 4427.0[s]. The total execution time is decided by the busiest node that does the most swapping operations. In this case, the maximum number of pagefaults in the busiest node was 1,896,226. Thus, the execution time of each pagefault can be obtained by dividing the difference in execution times by the maximum number of pagefaults, 2.33[ms] in this example. The other cases are also calculated in Table 4.

The execution time of each pagefault consists of round trip delay time, data transmission time, and memory allocation and/or search time at memory available nodes. The point-to-point round trip time on our PC cluster is approx-

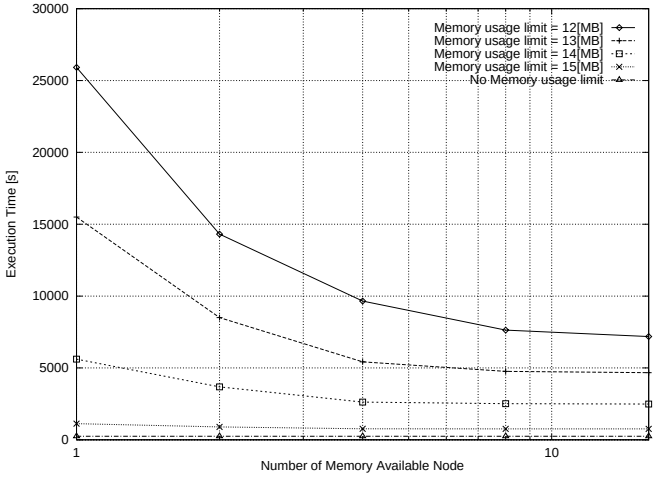

Figure 3. Execution time of HPA program (pass 2)

imately 0.5[ms] as shown in [2]. Since the point-to-point throughput is about 120[Mbps] on our cluster, and each pagefault data is contained in one message block (4Kbytes), the data transmission time can be calculated from these values as approximately 0.3[ms]. The rest of time is considered to be swapping operations cost in memory available nodes.

We can compare the pagefault execution time with the access time of hard disks. According to a state-of-art specification of SCSI hard disks, Seagate Barracuda 7,200[rpm] disks for example, the average seek time for read is about 8.8[ms] and the avarage rotation waiting time is about 4.2[ms]. In the case of latest fast hard disks such as HITACHI DK3E1T 12,000[rpm] disks, the average seek time for read is about 5[ms] and the avarage rotation waiting time is about 2.5[ms]. Therefore, it takes at least 13.0[ms] in average to read data from 7,200[rpm] hard disks, and 7.5[ms] even with the fastest 12,000[rpm] hard disks.

5.3. Using remote memory with remote update operation

The access interface function is developed to realize the remote update operations. In this experiment also, all memory available nodes are assumed to have enough memory space to accept swapping requests, so that the number of memory available nodes is constant during the experiment.

The execution time using this method is shown in Figure 4. This figure shows the execution time of pass 2 of HPA program, when the number of memory available nodes is 16. The execution times of dynamic remote memory acquisition

Table 4. The execution time for each pagefault (The number of memory available nodes is 16)

$Exec$	execution time of pass 2 in HPA [sec]
$Diff$	difference in execution time between this and the no memory limit case [sec]
Max	maximum number of pagefaults [times]
PF	execution time of each pagefault [msec]

Usage limit	$Exec$	$Diff$	Max	PF
12[MB]	7183.1	6936.1	2925243	2.37
13[MB]	4674.0	4427.0	1896226	2.33
14[MB]	2489.7	2242.7	1003757	2.22
15[MB]	757.3	510.3	268093	1.90

using remote update operations and using simple swapping are compared in the figure. The execution time using hard disks as a swapping device is also shown, for comparison. Seagate Barracuda 7,200[rpm] SCSI hard disk is used for this purpose. Other conditions are the same with the case of dynamic remote memory acquisition.

The execution time using hard disks as swapping devices is very long especially when the memory usage limit is small, because each access time to a hard disk is much longer than that of remote memory through the network. The execution time of dynamic remote memory acquisition with simple swapping is better than for swapping out to hard disks. It increases, however, when the memory usage limit is small, since the number of pagefaults becomes extremely large in such a case.

The execution time of dynamic remote memory acquisition with remote update operations is quite short compared to these results, even when the memory usage limit is small. It seems to be effective to provide a simple remote access interface for the itemsets counting phase, because the number of swapping operations during this phase is extremely large. According to these results, performance of the proposed remote memory utilization with remote update operations is considerably better than other methods.

5.4. Dynamic memory migration on memory available nodes

In the previous experiments, all memory available nodes are assumed to have enough memory space to accept requests of swapping operations. In such cases, the number of memory available nodes is constant during the experiment, while the amount of memory is monitored periodically and their availability is checked when they are accessed. As explained in Section 4, when a destination memory available

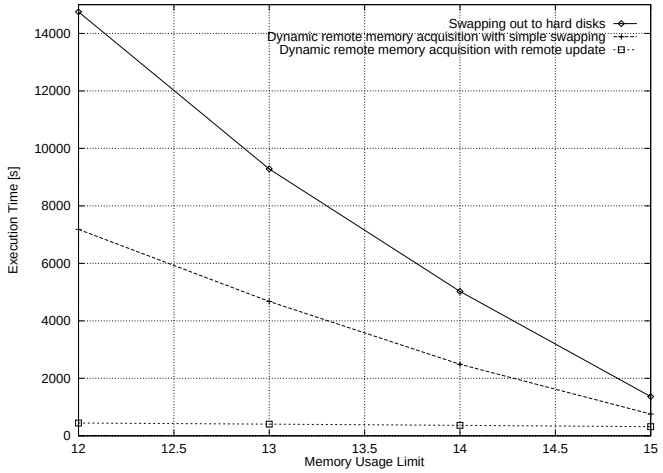

Figure 4. Comparison of proposed methods

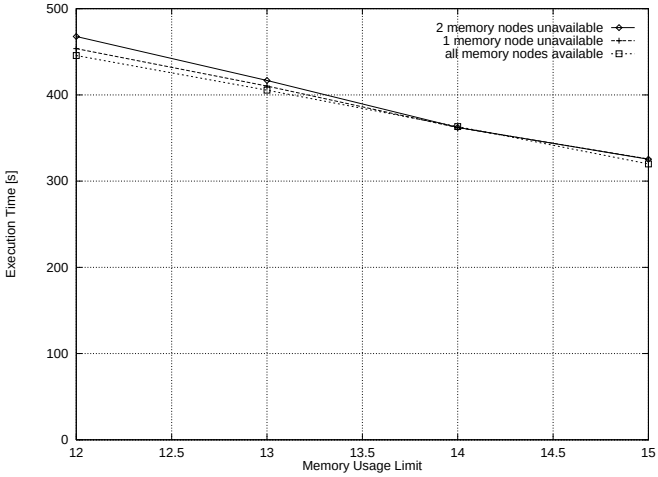

Figure 5. Dynamic memory migration on memory available nodes

node does not have enough memory space and the shortage is detected on the application execution node, another node is chosen as a swapping destination. If some other processes begin to run on a memory available node which already accepted swapped out data and therefore it must make space on its memory for the new processes, the swapped out data is migrated to another memory available node.

The following experiment is performed to evaluate the memory migration mechanism. First, the HPA program starts using the mechanism of the dynamic memory acquisition with remote update operation. During the execution of the program, a signal is sent to the amount of memory monitoring process on one of memory available nodes. After the process receives this signal, it begins to pretend to have no available memory space anymore as if other new processes begin to use the whole memory on this machine, and broadcasts the information to its client processes on application execution nodes. When application execution nodes detect the memory shortage on the memory available node, they sends a migration direction to this memory available node for entries which was belonging to them, to which node the entries should be migrated. Then the memory available node migrates its contents to other memory available nodes. After this procedure, the application program resumes, while the number of memory available nodes reduces by one. It is possible to send a signal more than two memory available nodes and migrate memory contens from them.

The execution time of pass 2 in HPA program in this experiment is shown in Figure 5. The maximum number of nodes used as memory available nodes is 16, and the interval of monitoring the amount of available memory is 3[s]. This figure shows three lines. The lower line is the case in which all 16 memory available nodes are used for swapping operations throughout the execution of the program. The middle line is the case in which one of memory available nodes receives a signal during the execution of the program, and it migrates contents of its memory to other memory available nodes. The upper line is the case in which two of memory available nodes migrate their memory contents to other nodes.

As shown on the figure, the execution time did not change significantly from case to case. According to the result, the overhead of memory contents migration is almost negligible in this experiment. The results are not significantly changed either when the interval of monitoring the amount of available memory is a little shorter, e.g. 1[s]. Too short interval, such as shorter than 1[s], degrades the system performance because of the monitoring and communication overhead. Such a short interval is expected to be unnecessary in most cases.

6. Conclusion

At present, judging by the number of installation sites, high performance parallel computers are becoming more popular in business applications than in scientific research. Data mining and *ad hoc* query processing are considered two of the most important applications for parallel processing. Since the PC cluster is a promising platform for future high performance computers from the cost/performance aspect, the feasibility of implementing parallel data mining application over a PC cluster was examined.

In contrast to conventional scientific calculations, association rule mining, one of the best-known problems in data

mining, has a peculiar feature in its usage of main memory: it needs many small data elements in main memory at each node, and the numbers of those areas suddenly increase greatly during execution.

In this paper, we show and discuss experimental results in which application execution nodes acquire extra memory dynamically from available remote nodes in an ATM-connected PC cluster. The experimental results show this method is considerably better than using hard disks as a swapping device. A method of updating remote memory in order to prevent false-sharing was proposed and examined. This method achieves the best performance. The dynamic decision mechanism for remote memory availability and the migration operations were also evaluated. The overhead of memory contents migration is almost negligible in this experiment, unless the interval of monitoring the amount of available memory is too short.

Acknowledgment

This project is partly supported by the Japan Society for the Promotion of Science (JSPS) RFTF Program and New Energy and Industrial Technology Development Organization (NEDO). HITACHI, Ltd. gave us extensive technical help with ATM-related issues. We would like to thank Prof. Spaniol in Aachen University of Technology for supporting our research.

References

[1] T. Tamura, M. Oguchi, and M. Kitsuregawa: ‘‘Parallel Database Processing on a 100 Node PC Cluster: Cases for Decision Support Query Processing and Data Mining’’, *Proceedings of SC97: High Performance Networking and Computing (SuperComputing ’97)*, November 1997.

[2] M. Oguchi, T. Shintani, T. Tamura, and M. Kitsuregawa: ‘‘Optimizing Protocol Parameters to Large Scale PC Cluster and Evaluation of its Effectiveness with Parallel Data Mining’’, *Proceedings of the Seventh IEEE International Symposium on High Performance Distributed Computing*, pp.34-41, July 1998.

[3] U. M. Fayyad, G. P. Shapiro, P. Smyth, and R. Uthurusamy: ‘‘Advances in Knowledge Discovery and Data Mining’’, *The MIT Press*, 1996.

[4] V. Ganti, J. Gehrke, and R. Ramakrishnan: ‘‘Mining Very Large Databases’’, *IEEE Computer*, Vol.32, No.8, pp.38-45, August 1999.

[5] M. J. Zaki: ‘‘Parallel and Distributed Association Mining: A Survey’’, *IEEE Concurrency*, Vol.7, No.4, pp.14-25, 1999.

[6] R. Agrawal, T. Imielinski, and A. Swami: ‘‘Mining Association Rules between Sets of Items in Large Databases’’, *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, pp.207-216, May 1993.

[7] R. Agrawal, T. Imielinski, and A. Swami: ‘‘Database Mining: A Performance Perspective’’, *IEEE Transactions on Knowledge and Data Engineering*, Vol.5, No.6, pp.914-925, December 1993.

[8] R. Agrawal and R. Srikant: ‘‘Fast Algorithms for Mining Association Rules’’, *Proceedings of the Twentieth International Conference on Very Large Data Bases*, pp.487-499, September 1994.

[9] T. Shintani and M. Kitsuregawa: ‘‘Hash Based Parallel Algorithms for Mining Association Rules’’, *Proceedings of the Fourth IEEE International Conference on Parallel and Distributed Information Systems*, pp.19-30, December 1996.

[10] R. S. Nikhil, G. M. Papadopoulos, and Arvind: ‘‘*T: A Multithreaded Massively Parallel Architecture’’, *Proceedings of the Nineteenth International Symposium on Computer Architecture*, pp.156-167, May 1992.

[11] M. Blumrich, K. Li, R. Alpert, C. Dubnicki, E. Felten, and J. Sandberg: ‘‘Virtual Memory Mapped Network Interface for the SHRIMP Multicomputer’’, *Proceedings of the Twenty-First International Symposium on Computer Architecture*, pp.142-153, April 1994.

[12] C. Huang and P. K. McKinley: ‘‘Communication Issues in Parallel Computing Across ATM Networks’’, *IEEE Parallel and Distributed Technology*, Vol.2, No.4, pp.73-86, 1994.

[13] T. Sterling, D. Saverese, D. J. Becker, B. Fryxell, and K. Olson: ‘‘Communication Overhead for Space Science Applications on the Beowulf Parallel Workstation’’, *Proceedings of the Fourth IEEE International Symposium on High Performance Distributed Computing*, pp.23-30, August 1995.

[14] R. Carter and J. Laroco: ‘‘Commodity Clusters: Performance Comparison Between PC’s and Workstations’’, *Proceedings of the Fifth IEEE International Symposium on High Performance Distributed Computing*, pp.292-304, August 1996.

[15] D. E. Culler, A. A. Dusseau, R. A. Dusseau, B. Chun, S. Lumetta, A. Mainwaring, R. Martin, C. Yoshikawa, and F. Wong: ‘‘Parallel Computing on the Berkeley NOW’’, *Proceedings of the 1997 Joint Symposium on Parallel Processing(JSPP ’97)*, pp.237-247, May 1997.

[16] A. Barak and O. La'adan: ''Performance of the MOSIX Parallel System for a Cluster of PC's'', *Proceedings of the HPCN Europe 1997*, pp.624-635, April 1997.

[17] H. Tezuka, A. Hori, Y. Ishikawa, and M. Sato: ''PM: An Operating System Coordinated High Performance Communication Library'', *Proceedings of the HPCN Europe 1997*, pp.708-717, April 1997.

[18] M. Oguchi, T. Shintani, T. Tamura, and M. Kitsuregawa: ''Characteristics of a Parallel Data Mining Application Implemented on an ATM Connected PC Cluster'', *Proceedings of the HPCN Europe 1997*, pp.303-317, April 1997.

[19] J. Heinanen: ''Multiprotocol Encapsulation over ATM Adaptation Layer 5'', *RFC1483*, July 1993.

[20] M. Laubach: ''Classical IP and ARP over ATM'', *RFC1577*, January 1994.

[21] M. Oguchi, T. Tamura, T. Shintani, and M. Kitsuregawa: ''Implementation of Parallel Data Mining on an ATM Connected PC Cluster and Performance Analysis of TCP Retransmission Mechanisms'', *The Transactions of the Institute of Electronics, Information and Communication Engineers*, Vol.J81-B-I, No.8, pp.461-472, August 1998.

[22] A. Acharya and S. Setia: ''Availability and Utility of Idle Memory in Workstation Clusters'', *Proceedings of the 1999 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, pp.207-216, May 1993.

[23] A. Cockcroft: ''How Much RAM is Enough?'', Sun World Online, http://www.sunworld.com/sunworldonline/swol-05-1996/swol-05-perf.html, May 1996.

[24] C. Amza, A. L. Cox, S. Dwarkadas, P. Keleher, H. Lu, R. Rajamony, W. Yu, and W. Zwaenepoel: ''TreadMarks: Shared Memory Computing on Networks of Workstations'', *IEEE Computer*, Vol.29, No.2, pp.18-28, February 1996.

[25] T. Shintani and M. Kitsuregawa: ''Parallel Mining Algorithms for Generalized Association Rules with Classification Hierarchy'', *Proceedings of the 1998 ACM SIGMOD International Conference on Management of Data*, June 1998.