Technical Report UEC-IS-2005-1, Version 2005-04-01
Graduate School of Information Systems, The University of Electro-Communications

A Method to Reduce the Acknowledgement Overhead of S-DSM Systems

Kenji Kise[†], Takahiro Katagiri[†], Hiroki Honda[†], and Toshitsugu Yuba[†]
[†] Graduate School of Information Systems
The University of Electro-Communications

Abstract

We discuss the inter-process communication in software distributed shared memory (S-DSM) systems. Some S-DSM systems, such as TreadMarks and JIAJIA, adopt the user datagram protocol (UDP) which does not provide the reliable communication between the computation nodes. To detect and recover from a communication error, therefore, an acknowledgment (ACK) is used for every message transmission in the middleware layer.

In this paper, firstly, we show that an acknowledgement is not necessarily required per one message transmission in the middleware layer. Secondly, the method to reduce the acknowledgement overhead for a page request is proposed.

We implemented the proposed method in our S-DSM system Mocha. The performance of the method was measured with several benchmark programs. We show that Matrix Multiply (MM) of high page transfer frequency achieves a drastic speedup as much as 92% for a 16-node PC cluster, compared with the conventional communication method of not omitting the acknowledgment.

1 Introduction

As an environment for parallel computation, cluster systems using general-purpose personal computers (PC clusters) are becoming popular. Because a PC cluster has no shared memory, the message passing model is used to develop applications in many cases. On the other hand, the shared memory is an attractive programming model for parallel computers. Software distributed shared memory (S-DSM) has been proposed to realize virtual shared memory on a PC cluster as the middleware layer software. Since the proposal of S-DSM, several systems[1, 2, 3, 4] have been implemented.

Some S-DSM systems, such as TreadMarks and JIAJIA, adopt the user datagram protocol (UDP) which does not provide the reliable communication between the computation nodes. To detect and recover from a communication error, therefore, an acknowledgment (ACK) is used for every message transmission in the middleware layer.

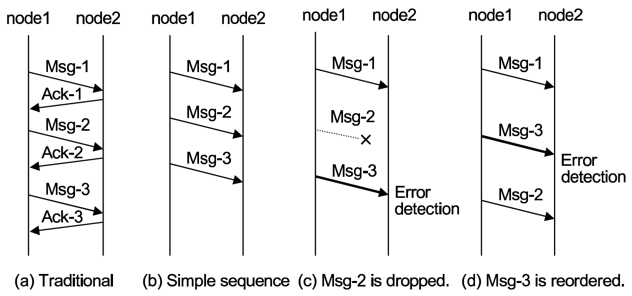

Figure 1. (a) General S-DSM communication. (b) Communication without acknowledgement. (c) Msg-2 does not reach the destination. (d) The order of Msg-2 are not guaranteed.

This paper discusses a technique to improve the S-DSM performance on a PC cluster. Since the S-DSM system using the UDP does not always need an acknowledgment for every message transmission, we propose a method of reducing the acknowledgment overhead. By implementing the proposed method on a S-DSM Mocha, which is now under development, we verify its effectiveness.

The rest of this paper is organized as follows. Section 2 describes the idea of the acknowledgement omission. Section 3 is the proposal and section 4 describes the implementation issues. Section 5 reports our quantitative evaluation results. Section 6 is a discussion and section 7 contains some concluding remarks.

2 Omission of Acknowledgment Messages

Figure 1(a) shows how three messages, Msg-1, Msg-2, and Msg-3, are sent from node 1 to node 2. This is an example of general S-DSM communication. When the messages are received, node 2 on the receiving side immediately returns acknowledgment (ACK) messages ACK-1, ACK-2, and ACK-3 respectively.

Figure 1(b) shows a communication without acknowledgment. If all transmitted messages are received without

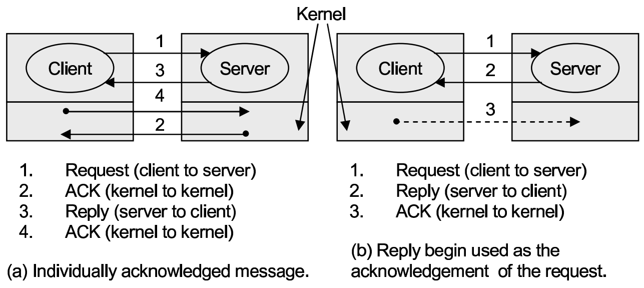
Kernel
Client
Server
1
3
4
2
Client
Server
1
2
3
1. Request (client to server)
2. ACK (kernel to kernel)
3. Reply (server to client)
4. ACK (kernel to kernel)
(a) Individually acknowledged message.
1. Request (client to server)
2. Reply (server to client)
3. ACK (kernel to kernel)
(b) Reply begin used as the acknowledgement of the request.

Figure 2. Communication in the client-server model [5].

```
#define OP_NULL      100 /* null:                */
#define OP_EXIT      101 /* server: exit         */
#define OP_GETP      110 /* server: get page     */
#define OP_GETPGRANT 111 /* server: getp grant   */
#define OP_DIFF      112 /* server: diff         */
#define OP_DIFFGRANT 113 /* server: diff grant   */
#define OP_BARR      114 /* server: barrier      */
#define OP_BARRGRANT 115 /* server: barr grant   */
#define OP_ACQ       116 /* server: aquire       */
#define OP_ACQGRANT  117 /* server: aquiregrant  */
#define OP_WAIT      118 /* server: wait         */
#define OP_WAITGRANT 119 /* server: wait grant   */
#define OP_INV       120 /* server: invalidate   */
#define OP_WTNT      121 /* server: w-notice     */
#define OP_REL       122 /* server: release      */
#define OP_BCAST     123 /* server: broadcast    */
```

Figure 3. The list of message types used by Mocha Version 0.2.

error, the acknowledgment costs can be reduced as shown in (b). Omitting all acknowledgment messages means cutting the general send-receive message count by one-half. This is expected to enhance the communication performance, especially in applications with frequent messages. In the UDP communication, however, an error occurs by a certain frequency. The example of errors are shown in Figure (c) and (d), Msg-2 does not reach the destination node in (c), and the order of Msg-2 and Msg-3 cannot be guaranteed in (d). Despite these communication errors, S-DSM systems should be constructed to operate correctly.

As shown in Figure 1(b), this paper is aimed at improving the S-DSM performance by omitting the acknowledgment messages. Note that the idea of acknowledgment omission is not novel. For example, reference [6] discusses a technique of omitting acknowledgments in the client-server model. Figure 2 is the communication used in this discussion. An acknowledgment for the request can be omitted by using a reply as its acknowledgment. Also an acknowledgment for the reply shown as the broken line in Figure 2(b) can be omitted depending on the properties of the reply.

In this study, the concept of acknowledgment omission in the client-server model is applied to the field of S-DSM for the first time. We discuss the implementation issues and verify the performance improvement.

3 Proposal of a Method to Omit Acknowledgment

3.1 Mocha: Yet Another S-DSM System

Mocha is a S-DSM system being constructed for the following two purposes: (1) It offers a S-DSM system easy to use as a parallel processing environment. (2) It achieves good performance especially for a PC cluster of many computation nodes.

Mocha is strongly affected by JIAJIA and a Mocha application is written by using an API similar to that of JIAJIA.

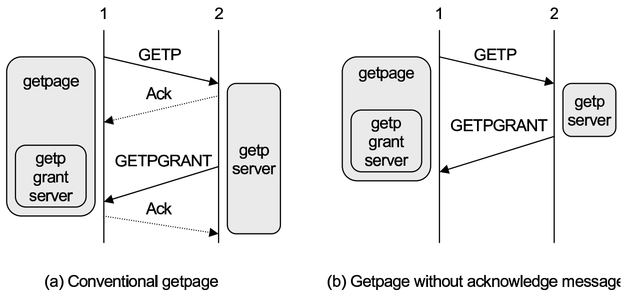

Figure 4. (a) The behavior of a page request. (b) Page request without acknowledgement.

Most JIAJIA applications run on Mocha without modification.

Mocha Version 0.2 used for evaluation in this paper manages shared memory in units of 8-KB pages. Mocha is a home based S-DSM where each page is assigned to a node (referred to a home node) according to the user specification. Home migration is not implemented in the current version of Mocha. Mocha Version 0.2 adopts the scope consistency[6].

Figure 3 is the list of message types used by Mocha Version 0.2. If a page fault occurs in referencing the non-cached shared memory, the page request (getpage) acquires the page from a node that has the necessary page. The rest of this section proposes the method of omitting the acknowledgment of message GETP and GETPGRANT for the getpage.

3.2 Omission of Acknowledgment for Page Request

Figure 4(a) shows the behavior of a page request. Suppose that node 1 requires a page and node 2 has the page. Node 1 calls the getpage function with the referenced memory address as a argument. The getpage function creates a

```
1  void getpage(address_t addr){
2    getpwait=1;
3    generate_message(OP_GETP, addr);
4    send_message();
5    while(getpwait); /** busy wait **/
6  }
```

Figure 5. The pseudo-code of the original getpage for which the acknowledgment is not omitted.

```
1  void getpage(address_t addr){
2    getpwait=1;
3    for(i=0; i<GETPAGE_MAX_RETRY; i++){
4      generate_message(OP_GETP, addr);
5      send_message();
6      while(not_timeout() && getpwait); /*busy wait*/
7      if(getpwait==0) break;
8    }
9  }
```

Figure 6. The pseudo-code of the function getpage to be implemented in the proposed method. The while loop in Line 6 finishes when the global variable getpwait has been reset or a timeout has occurred.

GETP message corresponding to the page request and transmits it to node 2. On receiving this message, node 2 returns an acknowledgment (ACK) message as a reply. To meet the page request, node 2 then calls the getpserver function. This function packs the necessary page information in a GETPGRANT message and sends it to node 1. On receiving the GETPGRANT message, node 1 calls the corresponding getpgrantserver function. This function stores the received page information in an appropriate memory area and resets the global variable to terminate the getpage function. The getpage function waits in a busy wait state for the page to arrive. When GETPGRANT arrives, the getpage function exits from the busy wait state and continues the application processing.

The getpage function sends a GETP message and waits in a busy wait state. In this kind of page request processing flow, we propose the method of omitting acknowledgment.

The exit of the getpage function from the busy wait state guarantees that no communication errors occurred in the two messages of GETP and GETPGRANT. If the function did not receive the GETPGRANT message within the timeout limit in the busy wait state, a communication error might have occurred. In this case, the GETP message is sent again and the system waits for the GETPGRANT message. The system should be designed to have no problems, even if the same message of GETP or GETPGRANT arrives several times. Using the GETPGRANT as an acknowledgement of GETP, acknowledgment of GETP or GETPGRANT can be omitted as shown in Figure 4(b).

Figure 5 shows the pseudo-code of the original getpage for which the acknowledgment is not omitted. Line 2 sets the global variable getpwait, Line 3 generates a page request message, and Line 4 transmits the generated message. When a message corresponding to the request arrives, the received page is stored appropriately and the global variable getpwait is reset. This finishes the while loop in Line 5 and terminates the page request function getpage.

Figure 6 is the pseudo-code of the function getpage to be implemented in the proposed method. The while-loop in Line 6 finishes when the global variable getpwait has been reset or a timeout event has occurred. If the variable getpwait equals to zero in Line 7, the getpage is terminated because the requested page is assumed to have been received. Otherwise, the for-loop from Line 3 transmits the page request message again.

```
1  #define MSG_MASK 0x80  /* 10000000 */
2  void send_one_message(){
3
4    if(sendqh->op==OP_GETP ||
5       sendqh->op==OP_GETPGRANT)
6      sendqh->op = sendqh->op | MSG_MASK;
7
8    for(i=0; i<MAX_RETRY; i++){
9      ret = sendto(message);
10     if(sendqh->op & MSG_MASK) return;
11
12     while(not_timeout())
13       if(FD_ISSET(fds[serverproc], &fds)!=0){
14         recvfrom(message);
15       }
16   }
17 }
```

Figure 7. The pseudo-code of the message transmission function send_one_message.

4 Implementation

This section discusses the implementation of the acknowledgment omission method discussed in the previous section.

To identify a message type, the S-DSM system Mocha uses a char-type variable of 8 bits. As summarized in Figure 3, however, not all of the 8 bits are used because the message types are not more than 20. Therefore, the low-order 7 bits are used to indicate an message type and the highest-order bit is used as a flag to indicate whether the message requires acknowledgment. This bit is called the reliable_msg_flag. The system sends an acknowledgment only when reliable_msg_flag is set.

Figure 7 shows the pseudo-code of the message transmission function send_one_message. If the message is GETP or GETPGRANT (Line 4 and 5), the reliable_msg_flag is set. in Line 6. The sendto in Line 9 transmits the message. If the reliable_msg_flag is set, the return in Line 10 can terminate the transmission function immediately and start the next processing because there is no need to wait for the acknowledgment. Otherwise, it is necessary to wait for the acknowledgment message. The while loop from Line 12 to Line 15 waits for the acknowledgment and then terminates the send_one_message function.

The message receiving section checks the reliable_msg_flag of the received message. Like the conventional system, the system transmits an acknowledgment message only when this flag is set.

We use the code shown in Figure 6 as getpage function to transmit a page request message (GETP). As a result of the parameter adjustment, the timeout interval is set to 50 ms. Therefore, if a page request message is sent again by a communication error, the overhead of 50ms is imposed on a system.

5 Evaluation of the Proposed Method

This section evaluates the performance of the proposed method by using the S-DSM system called *Mocha*, whose implementation was discussed in Section 4. A Mocha system that does not use the proposed method is called the *Mocha base* here.

5.1 Evaluation Environment

For the evaluation, we use a 16-node PC cluster where 16 personal computers are connected with a gigabit ethernet switch. Each node is an SMP (symmetric multi-processor) type computer with two processors of Intel Pentium 4 Xeon (2.8 GHz) and 1 GB memory. The system software of the cluster is SCore 5.6.1 constructed on RedHat Linux 7.3.

Although each node is an SMP computer, the data shown in this paper was obtained with the configuration running one process per each node. The execution time of each benchmark program is calculated by the arithmetic mean of three measurements.

5.2 Benchmark Programs

As the benchmark programs, we use N-queens[7], LU (parallel dense blocked LU factorization, no pivoting), Water (N-body molecular simulation), SOR (Red-Black Successive Over-Relaxation) and MM (Matrix Multiply). The binary of the benchmark programs is generated using GCC version 2.96 compiler with the optimization option O2.

As a parameter of N-queens, the problem size N=17 and the task allocation size of 8 is used. The elapsed time of the sequential version is 55 second.

As a parameter of LU, the matrix size of 1024×1024, and the block size of 8 is used. The element of the matrix is double precision. The elapsed time of the sequential version is 267 second.

As a parameter of Water, the number of particles is 1000. The elapsed time of the sequential version is 7.7 second.

As a parameter of SOR, the matrix size of M=4096 and N=4096, iterations=400 is used. The elapsed time of the sequential version is 98.1 second.

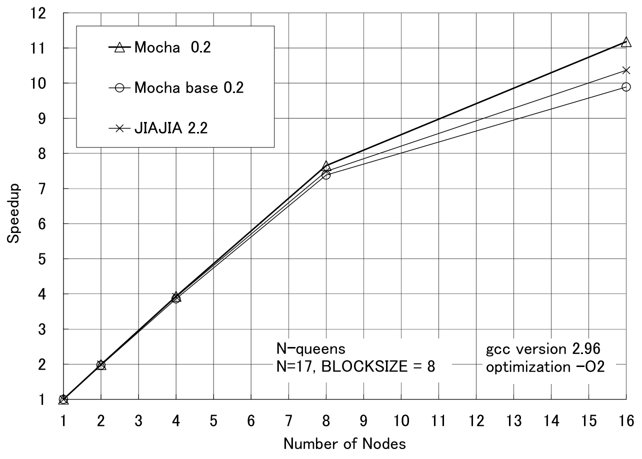

Figure 8. The performance comparison of the S-DSM systems. Benchmark is N-queens. The speedup is normalized by the elapsed time of JIAJIA single node.

As a parameter of MM, the matrix size of 2048×2048 is used. The element of the matrix is double precision. The elapsed time of the sequential version is 39.7 second.

5.3 Performance Improvement

Figures 8 to 12 summarize the results with the number of nodes on the x-axis and the speedup on the y-axis. The speedup is normalized by the elapsed time of JIAJIA single node. There are results for JIAJIA Version 2.2, TreadMarks Version 1.0.3.3-BETA (in Figures 11 and 12 only), and Mocha using the proposed method of omitting the acknowledgment (Mocha) and not omitting the acknowledgment (Mocha base).

The N-queens benchmark result is shown in Figure 8. The speedup on the 16-node configuration is 10.3 for JIAJIA and 11.1 for Mocha. The mocha is 13% faster than the JIAJIA on the 16-node configuration.

The LU benchmark result is shown in Figure 9. Since the traffic in LU is small compared with the calculation, an ideal speedup is achieved by increasing the number of nodes in every S-DSM system.

The Water benchmark result is shown in Figure 10. The speedup on the 16-node configuration is 6.0 for JIAJIA and 8.9 for Mocha. The Mocha is 49% faster than the JIAJIA on the 16-node configuration. The Mocha base is slower than JIAJIA. The Mocha is 54% faster than the Mocha base on the 16-node configuration. Mocha using the proposed method is advantageous especially where there are many nodes,

The SOR benchmark result is shown in Figure 11. The speedup on the 16-node configuration is 10.3 for JIAJIA and 12.3 for Mocha. The Mocha is 19% faster than the JIAJIA on the 16-node configuration.

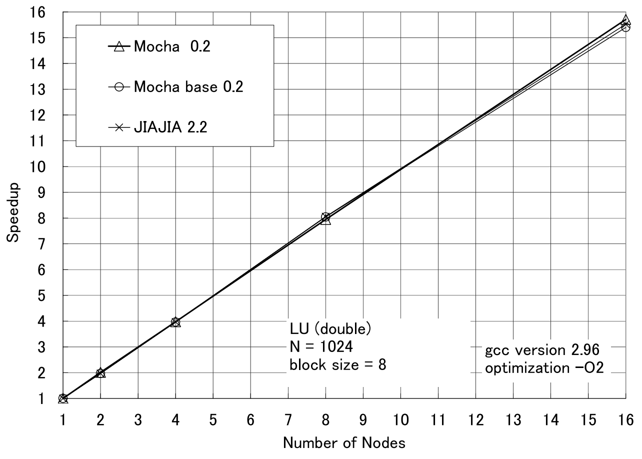

Figure 9. The performance comparison of the S-DSM systems. Benchmark is LU. The speedup is normalized by the elapsed time of JIAJIA single node.

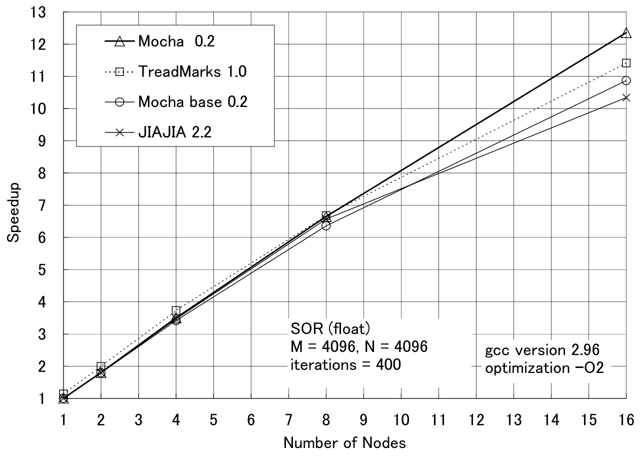

Figure 11. The performance comparison of the S-DSM systems. Benchmark is SOR. The speedup is normalized by the elapsed time of JIAJIA single node.

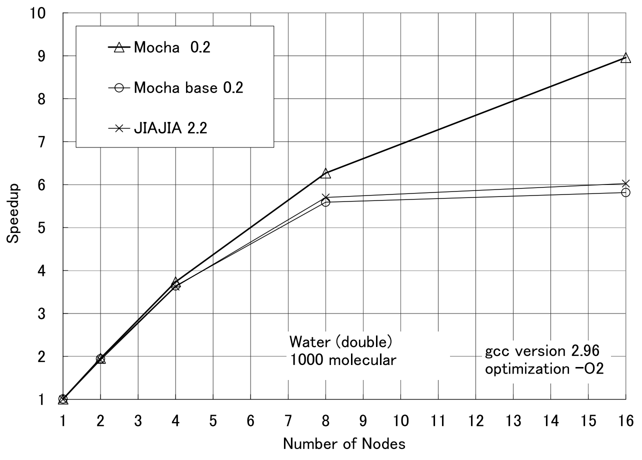

Figure 10. The performance comparison of the S-DSM systems. Benchmark is Water. The speedup is normalized by the elapsed time of JIAJIA single node.

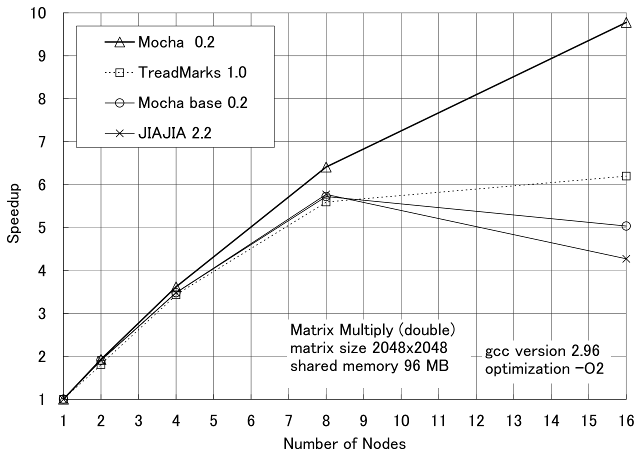

Figure 12. The performance comparison of the S-DSM systems. Benchmark is MM. The speedup is normalized by the elapsed time of JIAJIA single node.

Table 1. The number of communication errors per second of a PC cluster.

benchmark	2 node	4 node	8 node	16 node
N-queens	0.0	0.0	0.0	0.0
LU	0.0	0.0	0.0	0.0
Water	0.0	0.0	0.0	0.0
SOR	0.0	111	692	1,360
MM	0.0	0.0	0.0	0.0

The MM benchmark result is shown in Figure 12. In MM execution, the message GETP and GETPGRANT account for most of the elapsed time of all communications. Therefore, the proposed method of reducing the overhead for the page request produces remarkable effects. The conventional system, JIAJIA 2.2, and the Mocha base do not show performance improvement where the configuration of more than 8 nodes. TreadMarks makes the slower performance improvement. Even on the 16-node configuration, however, Mocha using the proposed method keeps the highest performance improvement and achieves a speedup as large as 9.7. Compared with the 16-node Mocha base, the 16-node Mocha achieves a speedup as large as 92%.

From the evaluation results in this section, the following conclusion can be obtained. Mocha using the proposed method achieves high performance in all benchmark programs except for LU, which had already attained the ideal speedup. Especially in a benchmark of high page transfer frequency, such as MM, Mocha achieves a drastic speedup as much as 92% on the 16-node configuration, compared with the conventional communication method of not omitting the acknowledgement.

5.4 Communication Error Frequency

Table 1 summarizes the number of errors in all communications on the S-DSM system of the Mocha base, which does not use the proposed method. The sum of the communication errors at all nodes was divided by the benchmark elapsed time to calculate the number of errors of the entire PC cluster per second (the average of three measurements).

From Table 1, we see that communication errors occur only in the SOR benchmark. In SOR, the communication errors become more frequent as the number of nodes increases. On the 16-node configuration, 1,360 errors occur per second.

When a similar measurement was made with Mocha using the proposed method, the number of communication error was zero in all benchmark including SOR. The proposed method reduces the communication of the acknowledgment. This might have eased traffic and reduced the communication errors.

We measured the communication error frequency of a S-DSM system on a PC cluster. We clarified that the communication error frequency was very low and that using the proposed method could greatly reduce the communication errors.

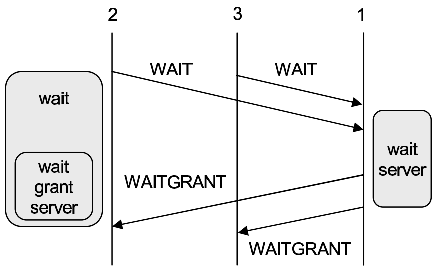

Figure 13. Omission of acknowledgment for wait. Node 1 is the server that manages the wait.

6 Discussion

6.1 Omission of Acknowledgment for the Wait

We discussed a method of omitting acknowledgment for page request and its implementation. As another type of communication processing, this section discusses a method of omitting acknowledgment in the wait process. Unlike barrier process, the wait does not include the process of the memory consistency.

Figure 13 shows the wait process at three nodes from node 1 to node 3. To simplify the figure, the wait and waitgrantserver functions of node 3 are omitted. In this figure, node 1 is the server that manages the wait. Node 2 and node 3 send a WAIT message to the server. The server counts the number of the WAIT arrivals. If the server receives WAITs from every node, it end the wait and broadcasts WAITGRANT to continue the application processing. The nodes other than the server send a WAIT and wait for WAITGRANT in a busy wait state. On receiving WAITGRANT, each node calls the waitgrantserver function, exits from the busy wait state, and continues the application processing.

The page request proceeds with the communication between one client and one server. In contrast, the wait is different in that more than one client accesses a single server.

The wait process can be realized in the collection of client-server communications. In the implementation, however, WAITGRANT may be received a long time after WAIT is sent because of the processing dispersion between nodes. It is then difficult to distinguish a timeout in waiting for WAITGRANT and a timeout by a communication error. Sending WAIT several times not for a communication error produces new communication overhead. Therefore, the method of omitting the acknowledgment in the wait and other communication requires a detailed study that considers the tradeoff of the new communication overhead.

6.2 Other Methods of Omitting the Acknowledgment

By using the characteristic that the page request can be handled as server-client processing as shown in Figure 2, we proposed and evaluated a method of omitting the acknowledgment for a page request in S-DSM.

In general, the message receiving side checks a sequential number on a message. If the number is different from the expected value, a communication error is detected and error recovery is attempted. This simple error detection method, using a sequential number, may cause a great discrepancy between the error occurrence time and the error detection time and thus makes error recovery difficult.

Another method is to transmit one acknowledgment message for n messages received. By increasing the value of n, most of the acknowledgment overhead can be eliminated. Even when this method is used, however, it is difficult to solve the problem of the great discrepancy between the error occurrence time and the error detection time.

As a result of studying these candidates, we selected and proposed the method comparatively effective and easy to implement.

7 Conclusions

This paper discussed a technique to improve the S-DSM performance on a PC cluster. For communication between computation nodes in a PC cluster, TreadMarks, JIAJIA, JUMP and several other S-DSM systems use the UDP which does not provide the reliable communication between the nodes. To detect and recover from a communication error, therefore, an acknowledgment is used for every message transmission in the middleware layer.

Since the S-DSM system using the UDP does not always need an acknowledgment for every message transmission, we proposed a method of reducing the acknowledgment overhead for a page request and discussed its implementation.

We implemented the proposed method on out S-DSM system Mocha. The performance was measured with several benchmark programs. From the evaluation results in this section, the following conclusion can be obtained. Mocha using the proposed method achieves high performance in all benchmark programs except for LU, which had already attained the ideal speedup. Especially in a benchmark of high page transfer frequency, such as MM, Mocha achieves a drastic speedup as much as 92% on the 16-node configuration, compared with the conventional communication method of not omitting the acknowledgement.

References

[1] Kai Li. IVY: A Shared Virtual Memory System for Parallel Computing. In *Proceedings of the International Conference on Parallel Processing (ICPP'88)*, volume 2, pages 94–101, 1988.

[2] Pete Keleher, Alan L. Cox, Sandhya Dwarkadas, and Willy Zwaenepoel. TreadMarks: Distributed Shared Memory on Standard Workstations and Operating Systems. In *Proceedings of the Winter 94 Usenix Conference*, pages 115–131, 1994.

[3] M. Rasit Eskicioglu, T. Anthony Marsland, Weiwu Hu, and Weisong Shi. Evaluation of the JIAJIA Software DSM System on High Performance Computer Architectures. In *Proceedings of the Thirty-second Annual Hawaii International Conference on System Sciences-Volume 8*, page 8012. IEEE Computer Society, 1999.

[4] Benny Wang-Leung Cheung, Cho-Li Wang, and Kai Hwang. Migrating-Home Protocol for Implementing Scope Consistency Model on a Cluster of Workstations. In *International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'99)*, 1999.

[5] Andrew S. Tanenbaum. *Modern Operating Systems*. Prentice-Hall International Editions, 1992.

[6] Liviu Iftode, Jaswinder Pal Singh, and Kai Li. Scope Consistency: A Bridge between Release Consistency and Entry Consistency. In *Proceedings of the eighth annual ACM symposium on Parallel algorithms and architectures*, pages 277–287, 1996.

[7] Kenji Kise, Takahiro Katagiri, Hiroki Honda, and Toshitsugu Yuba. Solving the 24-queens Problem using MPI on a PC Cluster. Technical Report UEC-IS-2004-6, Graduate School of Information Systems, The University of Electro-Communications, June 2004.