Active Zero-copy: A performance study of non-deterministic messaging

Shinichi Yamagiwa
INESC-ID
Rua Alves Redol, 9, 1000-029, Lisboa
Portugal
yama@sips.inesc-id.pt

Keiichi Aoki and Koichi Wada
Department of Computer Science, University of Tsukuba
1-1-1 Tennodai, Tsukuba
Ibaraki, 305-8573, Japan
k1@padc.cs.tsukuba.ac.jp, wada@cs.tsukuba.ac.jp

Abstract

Zero-copy communication exchanges the messages among the buffers that are allocated and locked before the communication itself. This communication style fits into applications that the communication timings and the message sizes are known in its initialization phase. However, another application with non-deterministic messaging such as web or parallel database can not fit into the style because the sizes and timings of its messages change at every communication. This paper proposes a new zero-copy communication style for these kind of application, called active zero-copy, that receives messages without pre-allocated buffers. The performance evaluation with the active zero-copy comparing with the conventional zero-copy, when the applications with non-deterministic messaging is applied, shows its efficiency.

1 Introduction

To achieve high performance parallel computing, zero-copy communication method is applied to many large-scale problems due to allowing lower the latency and higher the throughput communication. In a physically closed environment such as cluster computers[8], zero-copy communication performs off-the-shelf performance of the respective networks.

Zero-copy communication exchanges data between the sender's memory and the receiver's memory without copy operations that were performed in conventional network communication styles, such as TCP/IP. This extracts potential performance of network hardware, thus the parallel applications are able to utilize the off-the-shelf network performance.

In the current zero-copy communication method [5][7][9], a sender sends messages to the receiver between the buffers that are allocated and locked (generally called, pindowned) before the transfer starts. This rendezvous communication style implies the buffers to receive messages must be allocated in advance to the communication itself. This communication style is suitable for the parallel applications that the communication sizes and timings are defined statically, such as matrix computation of a parallel Fast Fourier Transform (FFT) and a parallel LU decomposition[11], because the initialization part of those applications is able to determine the size of messages and the timing of the allocation of the communication buffers.

On the other hand, the applications that the message size and the transfer timings are non-deterministic at the initialization part such as web and parallel database servers that the data size of queries changes in every request eventually, can not follow the rendezvous style mentioned above. In this case, the receiver must negotiate to allocate receiving buffers with the sender in every transaction, before the query computation itself, or receive messages into several pindowned buffers that are allocated at the starting of server program. Negotiations in the former method will degrade the transaction throughput due to the buffer allocation overhead and the serialization of the allocation and the transferring of the query data. In the latter method, the receiver can not receive message when it exhausts the pre-allocated buffers. Such situation can be occurred when many messages come to the receiver at once, or when the receiver does not process the subsequent queries due to the long computing time for the current query. Thus, it degrades the transaction throughput.

This paper proposes a new technique to achieve high performance communication in applications with non-deterministic messaging, called active zero-copy communication. In the active zero-copy communication, a receiver autonomously receives the message without copy operation. This will eliminate the negotiation of receiving buffer allocation that is caused by the current zero-copy communication method.

The next section of this paper shows the background of our research and the definitions with explanation of the current zero-copy communication method and its parallel ap-

plications. After that, we will show the design and implementation of active zero-copy communication. Finally we present the performance evaluation in order to compare performance evaluation of the current zero-copy communication with the one of the proposed active zero-copy, by using the communication pattern of the applications with non-deterministic messaging.

2 Background and definitions

2.1 Zero-copy communication and Applications

Parallel application needs to achieve high performance by reducing several overheads in the system. One of the very important issues to be regarded by an application designer is inter-processor communication for exchanging distributed data among processors. Numerous researches have been performed in order to reduce the potential overhead in the network [3][10][6][13]. Especially, the zero-copy communication, which exchanges messages between the sender's application memory space to the receiver's one directly, is a great idea to achieve low latency and high throughput that are almost the peak communication performance of the network hardware.

Several real implementations of zero-copy communication are proposed and are used in many parallel computation aspects with its supported network hardware. Well known implementation examples are GM[5], PM[9], BIP[7] over Myrinet and Infiniband[4].

GM, PM and BIP optimizes the communication with Myrinet[2] network hardware and implement the zero-copy communication. Those touch the hardware directly from the user application to bypass the thick conventional network protocol layers such as TCP. In addition the Myrinet network interface touches the user memory space directly to send or receive messages. Thus, GM, PM and BIP are able to extract almost 100% of the potential network hardware performance. To exchange data among processors with zero-copy communication, GM, PM and BIP uses a rendezvous communication way. That is, first, the sender and the receiver must allocate the buffers to place messages to be sent or received respectively. Then the sender will be allowed to transmit messages into the receiver buffer.

Infiniband defines the zero-copy communication method definitely in its specification[4]. The access layer of Infiniband allows the user application to use the zero-copy communication and the remote memory access. The zero-copy communication of Infiniband specification is following the same way as the rendezvous communication above. In addition, the remote memory access will be performed also in the rendezvous manner with a notification of the memory handle. The memory handle is a key information to allow the client to access the remote memory. For example, a server needs to send a message that includes a memory handle to the client. Then the client can access the memory space with an address space information in the memory handle. However, the server and the client sides must prepare the memory space before the memory handle exchange.

According to the concept of the zero-copy communication, the algorithm of the parallel application has to follow the rendezvous communication way, as the buffer size and the communication timings must be defined at every communication point of the computation flow. In applications that the input and output data sizes are fixed in whole the algorithm, the message size is deterministic at every communication point. Also if the algorithm is fixed for any input data size of application, the timings to allocate buffers for zero-copy communications are well known. Almost all arithmetic parallel applications such as parallel FFT, LU decomposition and other algebraic problems are suitable for the zero-copy communication mentioned above, due to its predictable message sizes and deterministic timings for the communication.

On the other hand, when the zero-copy communication is applied for the applications with irregular processing, such as web and parallel database servers, that has communications with non-deterministic message size and timings, server and client can not presume the message size before transaction because the query message size depends on the content of the query in every transaction. Therefore, the sender and the receiver must exchange one more redundant message to notify the size of the buffer allocated to the receiver from the sender before the zero-copy communication, or the receiver must allocate several small pindowned buffers before the communication and receive messages into them. In the former approach, an additional notification massage of the buffer size will be an inevitable overhead. In addition, the communication timings are not deterministic. This means the notification communications are inevitable if the receiver tries to allocate a buffer that is suitable for the query message, when those applications try to utilize the zero-copy communication's advantage to improve performance. The latter method that uses pre-allocated buffers can avoid the former additional notification message. However, the receiver can exhaust the pre-allocated buffer under irregular transactions when many query messages will come to the receiver at a same time and the receiver needs long processing time for a query. If the receiver exhausts the pre-allocated buffers, the receiver can not receive next message until some buffers are released for reusing. When the receiver runs into this situation, transaction throughput decreases. Even if the receiver prepares a lot of pindowned buffers to avoid this buffer starvation situation, OS on the receiver can become likely to be unable to assign memory pages for other purpose such as computation, because pindowned pages can not be paged out. Therefore,

this method is not the best solution for the applications with non-rendezvous communications.

2.2 Categorization of Zero-copy communication

As introduced the current zero-copy method in the previous section, we can think the availability to categorize two types of zero-copy communication.

The former applications, i.e. the ones with deterministic message size and communication timing, are suitable for the conventional zero-copy communication concept. We can say the zero-copy communication method for this type of applications is the passive zero-copy due to the pre-allocation of communication buffers by the receiver before the message transfer.

The latter type applications, i.e. the ones with non-deterministic message size and communication timings, inevitably exchange the sizes of buffers between the receiver and the sender that are used to communicate the data related to the computation. However, at the communication point, if the receiver can autonomously allocate the receiving buffer, the redundant messaging that notify the sizes of communication buffers to the receiver or the pre-allocating buffers can be eliminated. We call this type of zero-copy communication the active zero-copy.

In the next section, we will propose the active zero-copy that addresses the overhead that occurs when the passive zero-copy communication is applied to the applications with non-rendezvous communication characteristics.

3 Active zero-copy

3.1 Techniques for implementing Active Zero-copy communication

The new technique proposed to make the current zero-copy communication be suitable for applications with non-deterministic message size and timings, called active zero-copy. Active zero-copy performs the steps as shown in the followings:

1. The sender will send a message to the receiver as well as the passive zero-copy.

2. The receiver's network interface interrupts the receiver's host processor to request an allocation of a buffer to receive the message.

3. The host processor receives the message from the buffer allocated in 2.

4. After the allocation of the communication buffer, the receiver sends an acknowledgement to the sender.

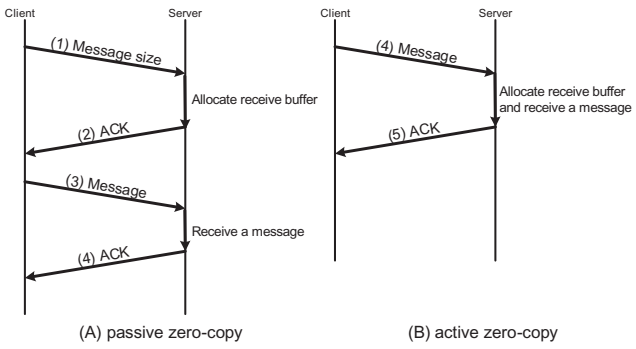

Figure 1. Comparison between the passive and the active zero-copy communication.

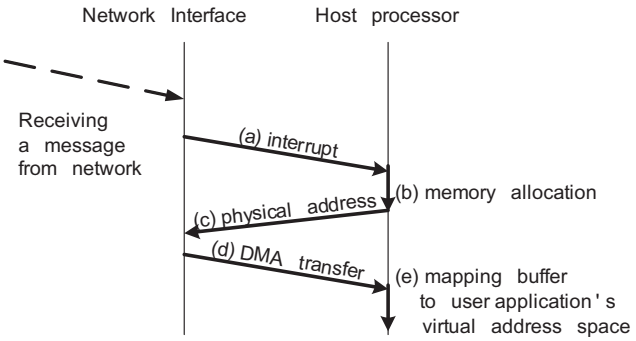

Figure 2. Receiving operation between network interface and host processor.

Fig.1 shows difference between the passive zero-copy and the active zero-copy. We call whole the transmission steps in the figure *transaction*. Fig.1 (A) performs a transaction to send a query message from a sender to a receiver with the passive zero-copy. The sender requests to allocate a communication buffer with a message that includes the size of the buffer (Fig.1 (1)). Then, the receiver sends an acknowledgement to the sender (Fig.1 (2)). Finally after receiving of the acknowledgement from the receiver, the sender can send the message to the receiver (Fig.1 (3)) and the receiver again sends an acknowledgement to the sender to signal that the receiver received the message completely. In this passive zero-copy communication flow, the transfer latency of the buffer size notification message (Fig.1 (1) and (2)) becomes inevitable for the applications with non-rendezvous communication.

On the other hand, the active zero-copy communication sends just a message to the receiver. On the receiver, the buffer that the message is received into will be allocated with an interrupt from the network interface to the host processor (Fig.1 (4)). After the message reception,

the receiver sends an acknowledgement to the sender (Fig.1 (5)). These steps do not include any explicit buffer size notification message to the receiver from the application level. Thus the setup overhead of the allocation of receiving buffers (Fig.1(A) (1) and (2)) can be eliminated. Any of the steps above does not include the communication to notify the size of buffer to the receiver. Therefore, the applications with non-deterministic messaging are able to exchange only desired messages. Moreover, because the allocation of buffer and the notification of the buffer size are performed in a single message, a serialization of those steps never occurs. Thus, the total communication performance will be improved regarding to the one obtained when the passive zero-copy communication is applied to the applications with non-rendezvous communication.

We need to pay attention to the receiving space for messages when we use the active zero-copy communication in the case the receiver does not have enough space to receive it. If the receiver can not receive messages, due to the starvation of memory space, it must respond with an error message to the sender. Regarding the error reporting, we can select two ways. One is an immediate NACK that returns a response just after an error occurs at receiving of message. The other is timeout that occurs when a timer of the sender side expires. The former will be a realistic way to report errors to the sender, because the application needs to process queries as fast as possible. Therefore, we will implement the active zero-copy with the immediate NACK.

3.2 Implementation

Now, we describe an implementation of the active zero-copy mechanism.

3.2.1 Requirements for system

To implement the active zero-copy, special functions are required on the receiver's network interface and the host processor's software.

Because the receiver must receive and transfer messages into its host memory, the active zero-copy requires the following functions:

1. Interrupt the host processor to receive messages

   The network interface hardware needs a function to interrupt to ask to the host processor for the allocation of the communication buffers to the host processor. In addition, the OS needs a function to respond to the request. For these steps, the network interface adapted to the host processor needs to exchange the size of message with the host processor. And also, the OS, or device driver for the network interface, that is running on the host processor needs to allocate the buffer for the message.

2. Mapping message buffers to a user application space

   The memory space that the network interface can access (i.e. physical address) and the one that the application can access (i.e. virtual address) are different address space. After receiving message in the receiver side, the host processor needs to map the buffer address space that allocated at function 1 into the user application space. The OS needs to map the buffer from the kernel memory space to the user virtual memory space.

3. Error detection when no more host memory space is available for reception of the subsequent messages

   To send a NACK to the sender, the host processor needs to respond with the status of allocation operation to the network interface hardware. The network hardware will send the NACK to the sender side when the host processor could not allocate the buffer due to the starvation of resources.

These special functions are not usual in commodity network interfaces. However, we can implement them if we use network interfaces that the functionality of network interface can be changed by the firmware, such as the Myrinet and some Gigabit Ethernet.

3.2.2 Maestro2 Network and MMP

We will utilize the Maestro2 network to implement the required functionalities for the active zero-copy mentioned above. The Maestro2 network [1][12] is an intelligent network that has been developed to address the overheads of interconnection in cluster computers. Over the Maestro2 network, a message passing library was implemented, called MMP, which has the functions of the active zero-copy communication.

Maestro2 network is composed of network interface cards and one or more switch boxes. Network interface cards are connected to each host processor via the 64bit 66MHz PCI bus, transfer messages from/to each host processor and are connected to Switch box via full-duplex LVDS of 6.4Gbps peak bandwidth.

The network interface is composed of a 300MHz PowerPC603, 64MByte SDRAM, PCI interface, LVDS physical layer, and MLX link layer controller. The switch box is composed of message analyzers and MLX link layer controllers for eight communication ports, a 300MHz PowerPC603 and a switch controller.

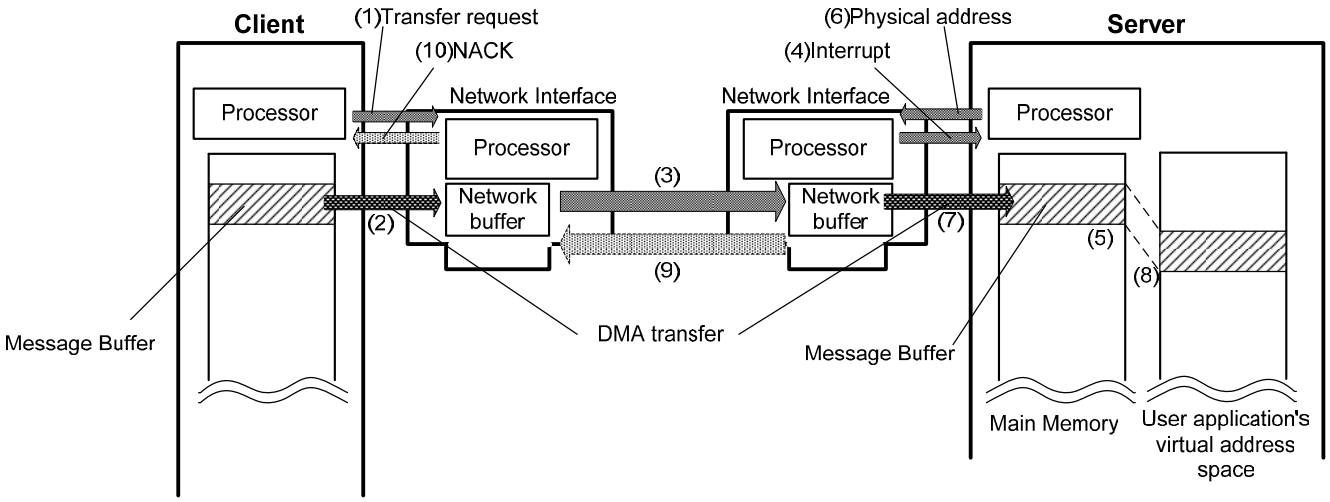

Figure 3. Communication flow example of the active zero-copy.

MMP implements the passive and the active zero-copy mechanism on Linux OS over Maestro2 network with non-blocking communication interface functions (`MMP_send()`, `MMP_send_wait()`, `MMP_receive()` and `MMP_wait()`). When "ANY" keyword is passed as the argument to specify the message size, or the source processor ID, or the message tag into `MMP_receive()` function, MMP will try to receive messages in the active zero-copy manner. The messages are received by the device driver of the network interface and passed into the user application software when the conditions passed into `MMP_receive()` function corresponds to the `MMP_wait()` function.

Fig.2 shows a behavior of the receiver side in the active zero-copy. First, when the network interface receives a message, it interrupts the processor to request the allocation of the buffer for the message with the message size (Fig.2 (a)). The device driver in the host processor allocates the buffer in its memory (Fig.2 (b)) and returns its physical addresses to the network interface (Fig.2 (c)), then the network interface transfers message by DMA (Fig.2 (d)). Finally, the device driver of the host processor maps the buffer to the user virtual address space to allow the user application to touch the message data when the mmap system call in `MMP_wait()` function is called (Fig.2 (e)).

3.2.3 Communication flow example

Fig.3 shows the flow of the communication when a sender machine transmits a message to the receiver machine using the active zero-copy. The sender's user application waits a request to its network interface that indicates a transfer of message (Fig.3 (1)) by a invocation of `MMP_send()` function. The network interface receives the request and transfers the message to the network buffer on the network interface (Fig.3 (2)). Then the network interface sends away a message to the receiver (Fig.3 (3)). When a message arrives at the receiver, its network interface interrupts the host processor and notifies the message size to the host processor(Fig.3 (4)). The receiver allocates a buffer in its memory by the device driver's interrupt handler (Fig.3 (5)) and passes the physical address to the network interface (Fig.3 (6)). The network interface transfers the message into the buffer by DMA (Fig.3 (7)). The user application on the receiver requests the device driver to map the receiving buffer into its virtual memory space by the `MMP_wait()` function(Fig.3 (8)).

If the receiver fails to allocate a receiving buffer in the case the receiver exhausted its main memory, it rejects the receiving message and returns an NACK to the sender (Fig.3 (9)). When the sender network interface receives the NACK, it notifies the NACK to the host processor (Fig.3 (10)). Then the user application will receive the NACK as a return status of the `MMP_Wait()` function and re-transmits the failed message to the receiver.

As shown above, the allocation of buffers and the transfer of messages from the receiver's network interface to its host memory are performed both at the same machine, without any negotiation communications via physical network before the transmission of the messages that are used to be exchanged in the passive zero-copy communication. Thus, for the non-rendezvous applications, the active zero-copy will achieve higher communication performance than the conventional passive style.

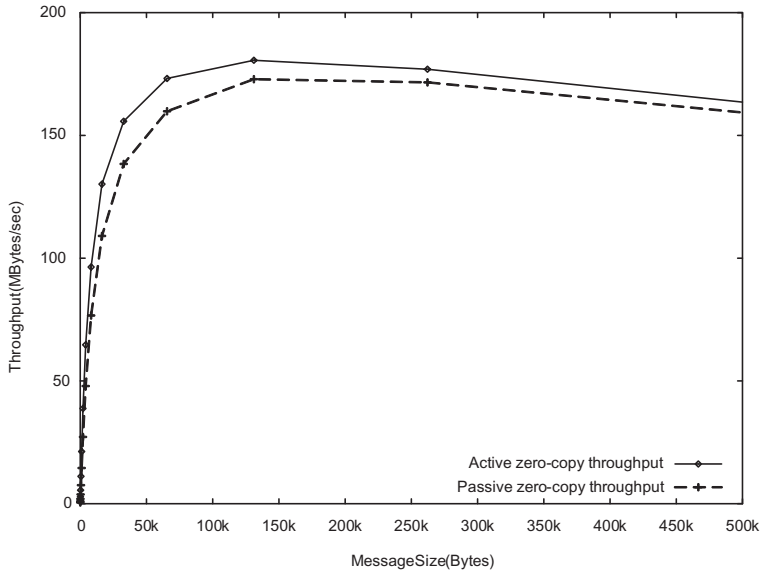

Figure 4. Throughput comparison between active and passive zero-copy.

4 Experimental evaluation

Now let us evaluate the performance impact of the active zero-copy. To compare the performance between the passive zero-copy and the active zero-copy, we use non-rendezvous application models to perform two evaluations: 1) throughput between the server and the client and 2) transactions per second. For the former evaluation, we will use a pingpong communication that the sender sends a message to the receiver, and then the receiver will return the message to the sender. Note that the communications in the active or passive zero copy will follow the ways depicted in Fig.1. The one way throughput will be calculated from the latency between the beginning of sending and the end of the receiving in the sender side. This will show the overhead of allocating the receiving buffer for every communication. For the latter evaluation, we will measure the number of transactions per second when the sender sends random sized query messages up to a limit size continuously to the receiver. This experiment will show the dynamic performance effect of active zero-copy communication. On the both evaluations, we will compare the performance results of the passive and active zero-copy. We will use the experimental environment shown in Table 1, with the Maestro2 network.

Table 1. The experimental environment

Host PCs for the server and the client	Dual PentiumIII 1GHz Serverworks HE-SL chipset PC133 SDRAM 512MByte
OS	Linux 2.4.7 SMP

4.1 Basic comparison between active and passive zero-copy

Fig.4 shows throughput comparison between the active zero-copy and the passive zero-copy. The maximum throughput of the passive zero-copy is up to 172 MByte/sec. This number increases when the active zero-copy is used up to 181 MByte/sec. We confirmed that active zero-copy achieves about 5% higher throughput than the passive zero-copy.

The lager the message size is, more overhead is introduced for the allocating the buffer (pindowning the physical pages) in the receiver side. That's why the throughput degrades when the message size is larger than 150KByte in this experience.

4.2 Transaction comparison between active and passive zero-copy

Fig.5 compares the random transaction rate between the active zero-copy and the passive zero-copy. The horizontal axis shows maximum query message size. The queries up to this maximum size will be transferred from the sender to the receiver continuously. We count the transactions as shown in Fig.1 as a single transaction for each zero-copy communication style. The vertical axis on the left side shows available number of transactions per second that corresponds to the bars. The line shows the ratio of the number of transac-

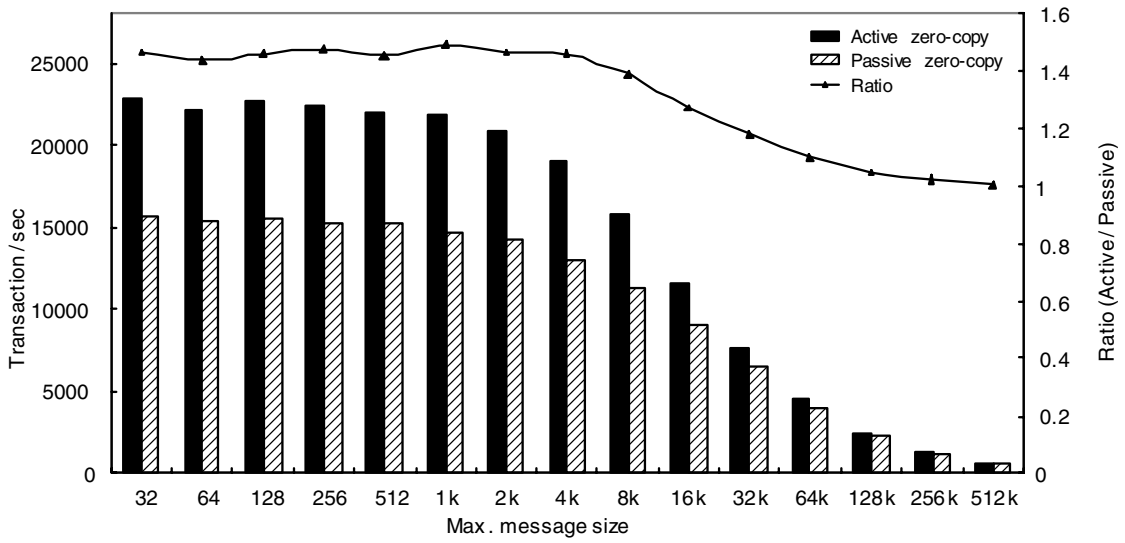

Figure 5. Comparison of random transaction rate.

tions for the active zero-copy versus the passive zero-copy.

With the increase of the maximum size of query message, the number of transactions per second will decrease due to increase of the message transfer latency via the physical network. We confirmed that the active zero-copy achieves about 1.5 times larger number of transactions per second than the one of the passive zero-copy. However, the difference of the number of transactions between two zero-copy communications is getting closer as the maximal size of the query message size increases because the latency to allocate the buffers (pindown operation) and the one to transfer the messages into the host memory via DMA become larger. From this result, we confirmed that the latency to pindown buffers in host memory becomes much larger, when the buffer size is more than 4KByte (that is, a page size).

4.3 Analysis of the results

The two experimental evaluations above show that the active zero-copy achieves higher performance than the current passive zero-copy for the throughput and the random transaction, especially when message size is small. In the throughput result, the impact on the performance is getting smaller when the query message size is greater than 150Kbyte, and in the random transaction result, the effect of the active zero-copy decreases when the message size is greater than 4KByte.

According to both evaluation results, we confirmed the performance difference between the active and the passive zero-copy styles is getting closer because the whole communication time increases as the message size becomes larger. Therefore, we can conclude that the active zero-copy is effective for small-scale communication, when the message size is smaller than 4Kbyte, that is a page size of memory management by the host processor.

5 Conclusion

We discussed a new solution for the problem of the overhead imposed by the current zero-copy communication when it is applied to applications with non-deterministic messaging timings and message size such as web and parallel database servers.

We categorized the zero-copy communication styles with two types. One is the passive zero-copy communication that is suitable for applications that have deterministic communication timings and its sizes. The other is the active zero-copy that we proposed in this paper. The active zero-copy is able to reduce redundant messages that occur in applications with non-deterministic messaging timings and message sizes.

We implemented the active zero-copy using Maestro2 and its message passing library MMP. Moreover, we performed experimental evaluations to see the performance impacts of the active zero-copy. From those evaluations, we concluded that the active zero-copy achieves better performance than the passive zero-copy when it is applied to applications with non-deterministic messaging timing and message sizes.

For the future works, because we found the active zero-copy works well when small messages (up to page size of operating system) are exchanged between the sender and

the receiver, we will evaluate the effect when the application program scatters a message into pages. And also we will implement the active zero-copy into other operating system. However, we can not implement the active zero-copy in an OS with high security policy that does not allow the device driver to allocate memory pages and to map it into a user process, such as Windows. In this case, we need to have a virtual network device that is in charge of an intermediate between the network interface hardware and the user application space, which manages the buffer allocation for the autonomous message receiving from the upper layer in the OS.

References

[1] K. Aoki, S. Yamagiwa, K. Ferreira, L. M. Campos, M. Ono, K. Wada, and L. Sousa. Maestro2: High speed network technology for high performance computing. In *Proc. of 2004 IEEE International Conference on Communication (ICC2004)*, number HS01-8, June 20-24 2004.

[2] N. J. Boden, D. Cohen, R. E. Felderman, A. E. Kulawik, C. L. Seitz, J. N. Seizovic, and W.-K. Su. Myrinet – a gigabit-per-second local-area network. *IEEE Micro*, Vol.15, No.1:29–35, 1995.

[3] C. Dubnicki, L. Iftode, E. W. Felten, and K. Li. Software support for virtual memory mapped communication. In *Proc. of the 10th Int'l Parallel Processing Symp. (IPPS'96)*, 1996.

[4] Infiniband Trade Association. *InfiniBand Architecture Specification, Release1.0*, October 2000.

[5] Myricom, Inc. *The GM Message Passing System*, 1999.

[6] S. Pakin, V. Karamcheti, and A. Chien. Fast messages(FM): Efficient, portable communication for waorkstation clusters and massively-parallel processors. *IEEE Concurrentcy*, 5(2):60–73, 1997.

[7] L. Prylli, B. Tourancheau, and R. Westrelin. Modeling of a high speed network to maximize throughput performance: the experience of BIP over myrinet. In *Proc. of Parallel and Distributed Processing Techniques and Applications (PDPTA'98)*, 1998.

[8] Rajkumar Buyya ed. *High Performance Cluster Computing: Systems and Architectures*, volume 1. Prentice Hall, 1999.

[9] H. Tezuka, A. Hori, Y. Ishikawa, and M. Sato. PM: An operating system coordinated high performance communication library. In *High-Performance Computing and Networking*, volume 1225 of *Lecture Notes in Computer Science*, pages 708–717. Springer-Verlag, 1997.

[10] V.Karamcheti and A.Chien. Software overhead in messaging layers: Where does the time go? In *Proceedings of the Sixth Symposium on Architectural Support of Programming Languages and Operating Systems (ASPLOS-VI)*, pages 51–60, 1994.

[11] F. Wong, R. P. Martin, R. H. Arpaci-Dusseau, and D. Culler. Architectural requirements and scalability of the NAS parallel benchmarks. In *Proc. of the 1999 ACM/IEEE Conference on Supercomputing*, November 1999.

[12] S. Yamagiwa, K. Ferreira, L. M. Campos, K. Aoki, M. Ono, K. Wada, M. Fukuda, and L. Sousa. On the performance of maestro2 high performance network equipment, using new improvement techniques. In *Proc. of IEEE International Performance, Computing, and Communications Conference (IPCCC2004)*, April 2004.

[13] S. Yamagiwa, M. Fukuda, and K. Wada. Design and performance of maestro cluster network. In *IEEE International Conference on Cluster Computing (CLUSTER2000)*, November 2000.