RI2N/UDP: High bandwidth and fault-tolerant network for a PC-cluster based on multi-link Ethernet

Takayuki Okamoto, Shin'ichi Miura, Taisuke Boku, Mitsuhisa Sato, and Daisuke Takahashi

University of Tsukuba
Graduate School of Systems and Information Engineering
1-1-1, Tennodai, Tsukuba, 305-8573 Japan.
{okamoto, miura, taisuke, msato, daisuke}@hpcs.cs.tsukuba.ac.jp

Abstract

PC-clusters with high performance/cost ratio have been one of the typical platforms for high performance computing. To lower costs, Gigabit Ethernet is often used for intercommunication networks. However, the reliability of Ethernet is limited due to hardware failures and tentative errors in the network switches. To solve this problem, we propose an interconnection network system based on multi-link Ethernet named RI2N. In this paper, we developed a user level implementation of RI2N using UDP/IP that is called RI2N/UDP. When this new system was evaluated for performance and fault tolerance, the bandwidth on a 2-link Gigabit Ethernet was 246 MB/s, and the system could remain active during network link failure to provide high system reliability.

1 Introduction

Network performance is an important issue when considering the total performance of clusters used for high performance parallel computing. System Area Networks (SAN), represented by Myrinet[13] or InfiniBand[11], are often used as the interconnect device for large scale clusters, since these systems have low latency, high bandwidth, and large scalability[17]. In fact, the low latency and high bandwidth features of SAN provide great performance for high-end cluster computing. However, despite the recent decrease in the price of InfiniBand and Myrinet, the introduction costs for both the network interface cards and switches are still relatively high, especially for small-scale clusters located in small laboratories or offices. Thus, SAN is not used, which leads to a decrease in the cost/performance ratio which is the most important reason of prosper of recent PC clusters. Therefore, a large number of small-scale clusters use Gigabit Ethernet (GbE), which is the most popular commercial technology with a high performance/cost ratio. Although most PC motherboards come equipped with one Ethernet connection, more expensive computers come with two connections.

The dependability of clusters, especially those using Ethernet switches, is heavily linked with the fault tolerance of the network. Long-term HPC applications are very sensitive to issues about the mean time between failures (MTBF) of the computing environment. Even one failure in the last phase of an application, for instance, when gathering all partial results from all the nodes, may cause a failure in the whole application. Using Ethernet, switches may suddenly become very slow at transferring packets, when heavy congestion occurs even if there is no hardware failure. Thus, the switches often need to be restarted to recover from this situation. Since the switch itself does not experience hardware failure, this situation will be called a *software failure*[1].

Thus, we propose a high bandwidth and fault-tolerant network system called Redundant Interconnection with Inexpensive Network (RI2N)[12] based on software binding of multiple Ethernet links. RI2N uses participating Ethernet-links to multiply the bandwidth of each communication channel when there is no failure. When there is a failure on a portion of the Ethernet-links or switches, the system detects which link has failed and keeps transferring the packets in the RI2N channels through the remaining links. This system works like a RAID system which has high throughput and high reliability with inexpensive disk drives. The RI2N creates high throughput and high availability communication using only commodity Ethernet and

1-4244-0910-1/07/$20.00

[1] Actually, there are several different causes including firmware failure for the error. In this paper, *software failure* is used to mean failure that cannot be attributed to the hardware.

software. A prototype system to multiply bandwidth was implemented on a TCP/IP multi-stream[12]. However, it is difficult to detect failures and to recover links using TCP/IP. Thus, the throughput significantly decreases when a link fails.

Thus, we developed a better implementation of RI2N called the RI2N/UDP, which allows for both high bandwidth and fault tolerance. In this paper, we describe this new system's theoretical basis, design, implementation, and performance evaluation. The rest of this paper is arranged as follows. Section 2 describes the designed concept of RI2N/UDP. Section 3 gives the design and technical issues for the implementation of the RI2N/UDP, while Section 4 presents the data obtained from an evaluation of a dual-link Ethernet system. Section 5 discusses related papers. Finally, Section 6 presents the conclusions and future work.

2 RI2N/UDP

2.1 Background

The concept of multi-link binding of network links for increasing bandwidth or fault tolerance is not new. For example, it has already been standardized and implemented as the IEEE802.3ad[10] and, thus, many Ethernet switches are equipped with this feature as a *network aggregation function* at the hardware level.

However, these switches only allow link binding between two switches directly connected by the multiple links, which implies that the network can recover from a general link failure, but not from a failure in the switch itself. If this concept is extended to multiple channels that consist of different switches, the entire communication can be salvaged even if several switches fail.

A study was performed on the implementation of IEEE802.3ad function using software on a PC node side[4], where multiple network links on a PC server were connected to a switch equipped with IEEE802.3ad. However, the performance of this implementation was poor as we could not exploit the potential bandwidth provided by multiple Ethernet links. PM/Ethernet[16] also binds several Ethernet links by software, but it does not provide a fault tolerance feature.

Based on the above results, we developed a new network system that realizes the same concept with a more flexible network configuration, a higher bandwidth, and a higher portability.

2.2 Overview

RI2N/UDP is an implementation of RI2N on UDP/IP to provide high bandwidth and fault tolerant communication channels for user-applications based on multi-link Ethernet. Dividing the transferred data into small packets on the source node, the RI2N/UDP transfers them through individual multiple Ethernet links (or networks) and then combines the received packets to rebuild the original data of the source node. If the multiple links are used efficiently, the increase in bandwidth can be used to increase network performance. While some failures can occur, RI2N/UDP restricts the links for transfer and resends dropped packets to keep the communication channels available. A way to provide fault tolerance is to use a fault tolerant feature similar to that used in RAID, that is, to send a data chunk always with a *parity link* for redundancy[12]. However, in our case, the number of links to be bound is not all that large, 2 or at most 4, so that this implementation would not be efficient for the total bandwidth.

2.3 Implementation level

There are 2 implementation levels that need to be considered: the system level with an implementation using device drivers, and the user level with an implementation using libraries with socket programming. System level implementation can provide greater portability for applications through a virtual network device with lower overhead than user level implementation. However, low level implementation may depend on the hardware, lower level drivers, and the operating system kernel, as well as a possible increase in development and porting costs. On the other hand, a user level implementation can be ported to various environments with slight modification such as changing of several system-calls to which the operating system supports, changing of thread library, and tuning several parameters; it can be developed at a lower cost. Therefore, we selected a user level implementation for the RI2N/UDP prototype with an effort to minimize the overhead.

2.4 Utilization of UDP/IP

On most user-programmable communication layers in parallel computing on clusters, a stream communication that guarantees reliability at the lower layer is required. Generally, TCP/IP is used to provide stream communication on Ethernet and is also used for HPC applications executed on a PC-cluster. However, TCP/IP is a connection-oriented protocol that creates static channels so several system calls such as *select()* may take a long time to return back from them when failure occurs on the communication link.[2] In this situation, the system calls take approximately a few minute, then, they return an error value.

[2]It may depend on the operating system, especially the version of kernel, but we observed a case where this problem occurs.

This problem was shown in the prototype implementation of RI2N based on the TCP/IP multi-stream. It increases the difficulty of implementing the desired fault tolerant features as an extension of the prototype. Therefore, we developed another implementation of the RI2N on UDP/IP from the scratch. UDP/IP is a connectionless oriented protocol that does not provide the function of retransmission so that the socket APIs are independent from hardware failures. If the packet-transmit functions for the UDP/IP socket are called when a link fails, the failure cannot be detected based on the return value of the system call. The only difference is that all the packets transmitted through the faulty link cannot be received at the remote side. The most important condition to create a fault-tolerant system is that the lower layer must be free from the possibility of hang-ups when hardware failure occurs. UDP/IP avoids this condition if the failure occurs out of the Network Interface Card (NIC), such as a failure at the switches or in the cables. To support the recovery from the failure of the NIC itself, the dynamic hardware plug-on/off feature must be supported both by the motherboard and the operating system. Therefore, this system does not support NIC or hardware failures at the node. This implies that the system is sensitive to the failure of cables or any of the intermediate switches between the source and destination nodes.

RI2N/UDP provides reliable stream communication channels to applications on UDP/IP that does not guarantee communication reliability. To realize this system, we implemented the receipt-acknowledgement and the retransmission for lost packets functions. In an ordinary single channel communication system, a time-out watcher is used to detect a channel failure because there is no way to reply to the acknowledgment. However, in RI2N/UDP, the acknowledgment packet can be broadcasted to all communication links, even on a link failure. This greatly helps to reduce the implementation overhead, as well as the time wasted while waiting for time out. In the RI2N/UDP, we combined the implementation of the functions for reliability and fault tolerance so as to reduce the overhead.

3 Design and Implementation

In this paper, the following terms are defined:

1. *Channel* refers to the application level of the communication entity, regardless of how many links are used to support it. A channel consists of one or more links.

2. *Link* refers to the communication network from a network interface of the source node to the destination node, including the NICs on both sides, cables and switches that provide IP-level reachability. A link consists of two NICs with the intermediate cables and one or more switches.

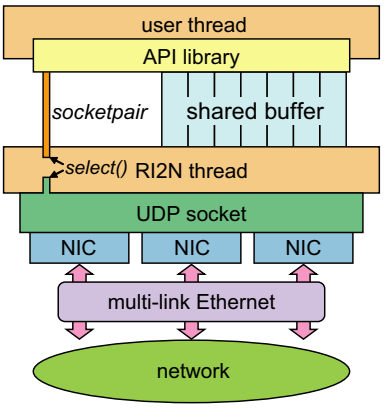

Figure 1. Construction of RI2N/UDP

3. *Cable* refers to the physical cable used to connect NICs and switches.

3.1 System construction

RI2N/UDP provides the function for retransmission and management of connections similar to TCP/IP. These functions must be processed asynchronously and concurrently with the user application. Therefore, a new thread for communication management is created to provide these functions for all RI2N connections. Figure 1 shows the construction of the RI2N/UDP. We call the thread processing the application as the *user thread*, and the new thread processing the communication as the *RI2N thread*. Only the RI2N thread can directly accesses the UDP socket. Section 3.4 will consider the inter-thread communication details.

3.2 Network construction

Figure 2 shows the network construction that we assumed as the environment to apply RI2N/UDP. Each node has multiple NICs connected with switches separated by IP network addresses. One of the IP addresses is bound to the NIC as the *primary address* that is used as the IP address visible to the user's application. Redundant switches provide a wider bandwidth and fault tolerance than a single switch, since in this case a single switch is not the cause of a single point-of-failure.

RI2N/UDP does not rely on MAC addresses and other hardware information to select the link to send a packet, because RI2N/UDP has access to the NIC through the UDP socket. RI2N/UDP uses IP routing in the operating system, which is available through the socket APIs. RI2N/UDP selects a remote side IP address to send the packet, and then the packet is transmitted through the NIC associated with that network address.

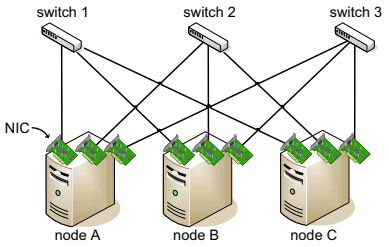

Figure 2. Target network construction

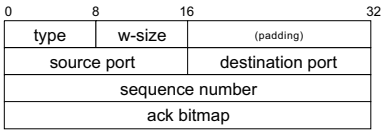

Figure 3. Structure of the packet for selective acknowledgement

This method requires redundant IP addresses to manage multiple NICs. On an ordinary IP network system, this causes a problem, as the address space needs to be increased in order to allow network reachability between the source and destination nodes. Since it is assumed that RI2N/UDP is used as an internal network to construct a cluster system, this is not considered to be a serious limitation. Furthermore, we also assume that RI2N/UDP is used on a high end cluster where each node is dedicated to a single job. Thus, RI2N/UDP uses a light-weight flow control algorithm discussed in Section 3.3.2.

3.3 RI2N protocol

The RI2N protocol is a unique protocol on UDP/IP designed for communication between nodes. It is similar to TCP over UDP/IP[5, 9] although it includes the fault-tolerant function and is tuned for using multi-link Ethernet.

3.3.1 Receipt acknowledgement and retransmission

The RI2N thread selects the destination IP address not by the user specified address but in a round-robin fashion from the addresses associated with the available NICs so as to increase the bandwidth for each channel. In this manner, the packets may arrive more frequently than using only a single link Ethernet, and there is a high possibility of miss-ordered packet arrival. In the standard TCP protocol, a packet retransmission is invoked after the first lost packet is found. However, if we apply the same strategy on a RI2N/UDP, the number of packets that is retransmitted may greatly increase since there is the possibility that a given packet may be assumed to be mistransmitted, when in fact the given packet is being sent on a different link. This causes too much wasted retransmission, followed by performance degradation. To improve the throughput on a multi-link Ethernet, we implemented a selective acknowledgement method as the receipt acknowledgement function by allowing a greater time delay.

In order to handle the acknowledgment and retransmission efficiently, two kinds of receive buffers are provided. One is a ring buffer of pointers that re-arranges the order of the packets, and the other is a ring buffer that passes the received data to the user thread.

Figure 3 shows the construction of a packet for selective acknowledgement[1]. The packet in the RI2N protocol does not have a dedicated field for the acknowledgement number to indicate the sequence number from which the system should retransmit the packet. Instead of this, the field for the sequence number is also used for the acknowledgement number. The ack-bitmap field indicates the packets that have already been successfully received at the remote node beyond that acknowledgement number. In this case, the size of the ack-bitmap field is 32 bits; thus, this field can indicate at most 32 consequent packets after the acknowledgement number. Since we assume that the number of links to be used is less than 10, this size for the field is reasonable.

3.3.2 Flow control

On a Wide Area Network (WAN), there are many unexpected congestions caused by anonymous users and tasks. This leads to low throughput. TCP utilizes a sophisticated flow-control algorithm[3] that predicts how congestions occur and adjusts the window size to improve the throughput on a WAN. On the other hand, if all the traffic is caused by the application itself, then we can use the whole network bandwidth just for the application in the LAN environment of a closed cluster network. Under such a condition, the commonly used TCP-like flow-control algorithm is too complicated and heavy. In fact, it may lead to performance degradation known as *slow starter problem of TCP*[2]. Based on these reasons, we decided to introduce a simple window control on the RI2N/UDP with a *w-size* field as shown in Figure 3. The W-size field indicates the window size, that is, the remaining size of the receive buffer. The sender node of the RI2N/UDP restricts the amount of data sent to the remote node by changing the window size.

Current RI2N/UDP uses a simple and light weight flow control algorithm. However, when we port RI2N/UDP to support multiple applications in time-sharing manner, it is required to implement more sophisticated algorithm which restricts the transmission pace to reduce the packet loss ratio.

3.3.3 Failure and recovery detection

RI2N/UDP provides fault tolerance for network failure by retransmitting lost packets. However, the packet loss may occur on Ethernet even if there is no failure in any of the links. We have already discussed the retransmission function under normal conditions in Section 3.3.1. Using the round-robin method for packet transmission, the lost packet can be retransmitted successfully on any of the remaining active links. However, the success rate of retransmission is reduced because the library may try to send the packet using the failed link again. This causes another packet loss. Therefore, we need the function to explicitly omit the failed link from the target links to be used for round-robin packet transmission.

In the RI2N protocol, the packets received at each link on the receiver node are counted. Since the sender nodes distribute the packets to all links in a round-robin manner, there should be no difference between traffic on the individual links. Therefore, the receiver node can detect the link malfunction when the count of received packets for a particular link decreases rapidly compared with the others. By sending back this information to the sender node, the sender node, it can subsequently recognize the link as failed and stops transmitting packets through it. It is important to use the packet reaching rate as the indicator of link failure instead of the heartbeat packet, since then the *soft error* in a link or switch where the packet transmission function can be determined. Sometimes, the Ethernet switch may drop into a condition where the packet transmit throughput is largely degraded but the switch itself is still active. We consider this situation as a switch failure that can be detected using the above algorithm.

The detection of system recovery is also important, as well as the detection of system failure. We assume that most of the failure is due to the Ethernet cables and switches, which can only be fixed manually by reconnecting the link, by restarting the switch, or by replacing the given switch. Therefore, a concrete and not a time-consuming method should be used to detect system recovery. The simplest and most efficient way is to use a heartbeat packet with a low frequency, since we aim to detect a recovery from a long-term failure as described above. As a result, the RI2N/UDP library keeps sending heartbeat packets to all links at an interval of several seconds. If any packet is received at the link marked as failed, then it is recognized as recovered from the failure. It is then marked as a healthy link. This information is shared between the pair of sender and receiver nodes in a manner similar to that used for failure detection.

3.4 Communication between threads

RI2N/UDP is a user level library using the *pthread* library. RI2N/UDP uses two different methods for communication between the user thread and the RI2N thread (as shown in Figure 1), where *socket pair* is used for event notification and as the shared buffer for data transfer. One of the common synchronization methods between threads is to use the *condition flag* and the *mutex* functions. However, the RI2N thread has to constantly monitor the network condition of any packet that it has sent or received, as well as to process any requests from the user thread. To perform everything in an efficient way, the RI2N thread handles all events through the *select()* system call. The conditional signal and mutex are not suitable to be combined with the *select()* system call. Thus, the request from the user thread must also be notified through a file descriptor to allow the RI2N thread to just watch all the sockets on a single *select()* system call. However, this causes too much overhead; all the communication data must pass through the socket pair since several data copies are required for both the user and kernel spaces. So, the theoretical bandwidth of socketpair is limited to only a half of the memory bandwidth. Therefore, the actual data to be transmitted or received is copied through the shared buffer between the RI2N and user threads to provide high bandwidth, while the notification of data ready to be processed is sent using the socketpair. To provide a higher throughput with lower CPU load, this combined-method is better for data transfer between threads.

Table 1. Environment

item	specification
CPU	Intel Xeon 3.0 GHz 1-way (Hyper-Threading)
memory	DDR2 1024 MB
kernel	linux 2.6.17
NIC	Intel PRO1000MT dual port 1000base-T (PCI-X 64bit/100 MHz)
NIC driver	Intel PRO/1000 Network Driver 7.0.33
switch	Dell Power Connect 5224 (24 ports GbE switch)

4 Evaluation and Discussion

In this section, we evaluate the performance of RI2N/UDP on a dual-link Gigabit Ethernet. We measured the throughput on a single-sided burst data transfer and the latency on the ping-pong communication. Table 1 shows the environment for this evaluation. In all measurements, the network MTU was set to 6,000 bytes, and the packet size of RI2N/UDP was set to 5,950 bytes.

4.1 Throughput

To evaluate the bandwidth and the correctness of the fault tolerance function, we measured the throughput when all links were available and when some links had failed. At

first, we made a single-side burst transfer on RI2N/UDP from one node to another, and measured the average throughput every 100 milliseconds. As we continued to measure the throughput, we unplugged and then reconnected the Ethernet cables to simulate failure and recovery. The threshold to detect link failure was set to 1:50, which implies that a link is recognized as failed if the packet receiving frequency of the link becomes less than 2% of the others. The heartbeat interval to detect link recovery was set to 3 seconds. We observed the communication throughput for a single-sided burst data transfer between two nodes as the number of available links is reduced from 2 to 0.

Figure 4 shows the result. The horizontal axis represents the time after starting the measurement, while the vertical axis represents the throughput at that time. Two bars under the graph represent the condition of each link, *plugged* or *unplugged*, to imitate link failure. Since the unplugging and reconnecting of the cables was performed manually, the graph does not show the precise timing of the change, as its resolution was 10 milliseconds. Nevertheless, we can observe the approximate behavior of the system and how the throughput varies.

Without any link failure, the RI2N/UDP throughput is 246 MB/s where 98% of the theoretical peak with a dual Gigabit Ethernet link (250 MB/s) is achieved. At time t_1, we unplugged link #1 to cause a single link failure, which degraded the throughput by half (123 MB/s), but the system still kept 98% of the theoretical peak with a single Gigabit Ethernet link. Thus, it can be concluded that the RI2N/UDP provides higher bandwidth. Furthermore, the throughput did not decrease below 100 MB/s from time t_1 to time t_2. As described in Section 3.3.3, we implemented the failure detection algorithm based on the arrival packet ratio. We also confirmed that the throughput with a single link failure drops to approximately 10 MB/s without a failure detection algorithm. Since UDP access does not return any error condition even with a link failure, it is possible to send a packet on a *dead* link. However, in this implementation, the fault is correctly detected, and we can achieve the same performance as the standard TCP/IP that only uses the healthy link.

At time t_2, link #2 was also unplugged, and no packets were transferred until time t_3, but the system itself was still active. This means that there was no fatal fault on the communication system from the viewpoint of the user thread; it was waiting for system recovery. At time t_3, link #1 was reconnected. Since the heartbeat interval was set to 3 seconds, it takes a few seconds before the system recovers from the full link failure mode. As expected, throughput rose again with a single link after several additional overhead. It was assumed that this overhead was caused by the delay in the Ethernet hardware link and the auto-negotiation that takes a few seconds after reconnecting the cable. We can reduce

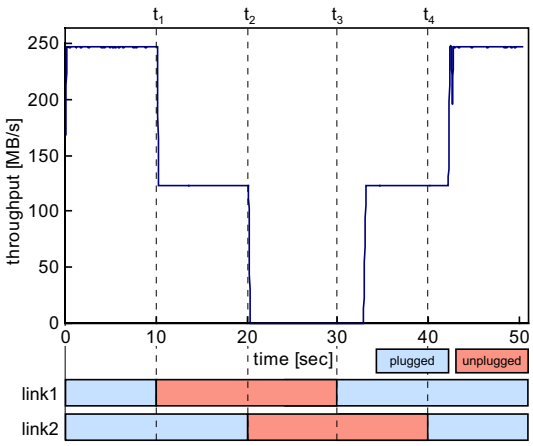

Figure 4. Throughput with varying link conditions

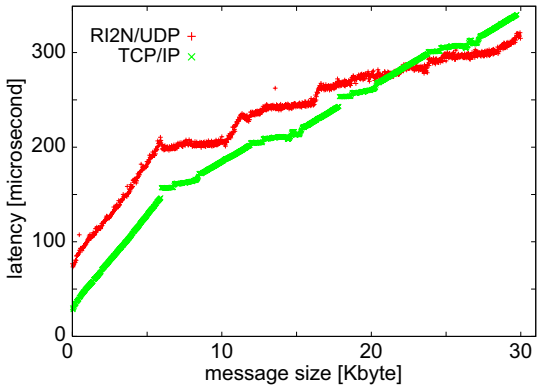

Figure 5. Result of the latency measurement

the time for recovery detection by shortening the heartbeat interval. However, since the time from failure to recovery is basically much longer than the time for recovery detection, we think that the current implementation is reasonable.

Finally at time t_4, link #2 was reconnected, and the system throughput recovered to the original 246 MB/s. As a result, we can conclude that the implementation of RI2N/UDP provides enough throughput in both the dual and single link cases, as well as correct behavior during link failure to avoid a fatal error in the user application.

4.2 Latency

Next, we evaluated the latency of the RI2N/UDP communication layer. To measure the latency, we performed ping-pong communication between 2 nodes 10 times and then determined the average time for one-way ping-pong.

Figure 5 shows the comparison between RI2N/UDP and standard TCP. Each point shows the latency calculated from

the ping-pong communication. The horizontal axis represents the message size for each ping-pong communication, and the vertical axis represents the latency for the given message size. The shortest latency for RI2N/UDP is 72 μs, while that of TCP/IP is 28 μs.

RI2N/UDP uses socket pairs to communicate with the user thread for the request handling, which takes approximately 35 μs in our environment. A single side transfer through a socket pair takes about 12 μs including the costs of the *write()* and *read()* system calls. However, RI2N/UDP must reply to the user thread request, so the RI2N thread calls the *write()* system call to determine the data size to be passed through the shared buffer with both threads. This system call takes approximately 5 μs on only the sender side. Furthermore, the RI2N thread uses the *select()* system call to detect the requests, which takes approximately 1 μs, and the processing after calling *write()* and before switching the context takes approximately 2 μs.

Three of these four overhead reasons, 12 μs, 1μs and 2 μs, respectively, appear on both the sender side and receiver side; therefore, the total cost for communication between threads is $(12+2+1) \times 2 + 5 = 35$ μs. On the other hand, the cost in the lower layer of RI2N/UDP or the latency of UDP/IP is about 30 μs including the overhead of the *select()* system calls. Furthermore, the analysis and reordering of the received packets takes approximately 4 μs. Therefore, the total latency of the RI2N/UDP should be approximately $35 + 30 + 4 = 69$ μs. It is almost same as the observed latency (72 us); there are still a small fraction of unknown 3 us.

In addition, Figure 5 shows the crossover point of the communication cost on the RI2N/UDP and TCP/IP. Based on our measurements, the cost of TCP/IP equals that of RI2N/UDP at a message size of approximately 20 Kb. Thus, it can be concluded that the performance of the RI2N/UDP layer is better than the TCP/IP for messages larger than 20 Kb. This seems to be a large message for some applications. However, for scientific applications, a message size of 20 Kb represents less than 2500 double precision values. Thus, for many scientific applications that require HPC PC cluster systems, this message size is not so large, and the RI2N/UDP communication layer would be appropriate.

4.3 Implementation improvement

RI2N/UDP uses socket pairs and a shared buffer to communicate between threads as described in Section 3.4, which was assumed to increase the communication throughput on the application. Thus, the RI2N/UDP realizes a doubling throughput with a dual link Gigabit Ethernet. Furthermore, we are interested in how the communication throughput is degraded if we use the socketpairs not only for the synchronization but also for the data transfer between threads. If we only use the socketpairs, the time required for memory copying may increase, and the throughput may be decreased. However, it is also expected that this would reduce the communication latency for short messages, because a single system call of *read()* or *write()* would perform both the synchronization and data transfer at once without any additional data copying on the shared buffer. There is yet another reason to only use socket pairs, since, for that user thread, the socketpair becomes the single contact point with the RI2N thread. Thus, we can apply many of the native socket operations, such as *select(), poll(), write()*, and *read()*, that can be applied to ordinary communication sockets. In the current implementation of RI2N/UDP, there are several requirements for the modification of the user application. Even though they are simple and easy, we can greatly reduce the effort to develop or port a user application to the environment of RI2N/UDP from standard TCP. For example, it may be easy to apply RI2N/UDP to a commonly used socket-ready MPI implementation, such as the ch_p4 device of MPICH[8]. To reduce the latency and to simplify the interface between the user thread and the RI2N thread, we should apply the above implementation and compare the performance.

5 Related work

There are many works related to a multi-link interconnection for a cluster network. PM/Ethernet[16] uses a multi-link Ethernet to improve the throughput under the special environment of a SCore[15] cluster middleware. PM/Ethernet also provides a lower latency than UDP/IP or TCP/IP with specific APIs and driver level optimization that bypasses the deep TCP stacks. However, the fault tolerant feature has not been provided for this layer. PM/Ethernet cannot be utilized without Score cluster middleware, while RI2N/UDP requires only UDP/IP socket and pthread so it can be utilized on most Linux clusters without any additional modules or kernel modifications.

VMI[14] is a communication layer supporting heterogeneous interfaces, such as using both GbE and Myrinet. This can be used not only in a cluster but also in a grid environment. The APIs are provided by VMI 2.0 on the socket layer as like as by RI2N/UDP. VMI 2.0 provide fault tolerance feature, but it cannot stripe a data transfer to increase the bandwidth, while RI2N/UDP provides both fault tolerance and high bandwidth.

Link Aggregation[10] is a technology that provides load distribution and fault tolerant features using multiple Ethernet links with IEEE 802.3ad compliant switches. However, in Link Aggregation, all links must be connected to the same switch. Therefore, Link Aggregation supports only NIC or cable failure but not switch failure, and the number of supporting nodes is limited by the number of ports on the

switch connected with all nodes. RI2N/UDP can be applied to multiple links connected with different switches where the communication system is fully duplicated in the NIC, cables, and switches. In such a network system, RI2N/UDP can survive even with the failure of one of the switches. We tested such a situation with a cluster of 64 nodes. We confirmed that the RI2N/UDP system works after a complete loss of power to one of the switches. As well as, there is a complete, correct recovery after the switch is rebooted.

LA-MPI[7] and new OpenMPI[6] are MPI implementation which can utilize a multi-link interconnection for both fault tolerance and high bandwidth. They are implemented in one layer for whole features including MPI operation, fault tolerance and high bandwidth. Thus, they may cause lower overhead than a MPI implementation on RI2N/UDP. However, they provide only the MPI interface but RI2N/UDP provides more versatile socket API.

Comparing with these works the novelties of RI2N/UDP are shown as follows;

- It is implemented in a user level library.
- It is independent from hardware devices and allows some redundant switches.
- It may provide both high bandwidth and fault tolerance.
- It requires only UDP/IP and pthread library but not any special middleware or kernel.

6 Conclusions

This paper describes the design, implementation, and performance evaluation of a RI2N/UDP that is a user level implementation of RI2N, a high-bandwidth, fault-tolerant network for PC clusters. For the throughput, the RI2N/UDP provided 246 MB/s with dual-link Gigabit Ethernet. This implies that we can use the doubled communication bandwidth with only a 2% increase in overhead. For the fault-tolerant feature, RI2N/UDP keeps the communication channel open, and the user application does not stop even if the link fails. The latency of RI2N/UDP is measured at 72 μs in our PC cluster, while that of TCP/IP is 28 μs. Based on an analysis of this overhead, it was determined that it would be difficult to shorten it. However, this latency is acceptable for large scale HPC applications. Furthermore, the performance of RI2N/UDP overtakes that of TCP/IP when the average message size is larger than 20 Kb.

Future work includes: improving the communication method between the user thread and RI2N thread so as to increase latency performance and application portability; improving the flow control algorithm to increase the throughput in a congested situation; developing an MPI implementation on our system; comparing the performance of this system with those previously reported; and evaluating the system using actual HPC applications.

Acknowledgment

This research work is partly supported by Core Research for Evolutional Science and Technology program of Japan Science and Technology Agency (JST-CREST), "Computational platform for embedded system with low-power and dependability" in the category of "Dependable Embedded Operating Systems for Practical Use" and the Grant in Aid of Ministry of Education, Culture, Sports, Science and Technology in Japan (C-17500031).

References

[1] TCP Selective Acknowledgement Options. RFC 2018 (Proposed Standard), Oct. 1996.

[2] TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms. *RFC200I*, 1997.

[3] TCP Congestion Control . RFC 2581 (Proposed Standard), Apr. 1999. Updated by RFC 3390.

[4] T. Davis. Linux Ethernet Bonding Driver.

[5] T. Dunigan and F. Fowler. A tcp-over-udp test harness, 2002. Technical report, Oak Ridge National Laboratory, Oak Ridge, ORNL/TM-2002/76.

[6] E. Gabriel, G. Fagg, G. Bosilca, T. Angskun, J. Dongarra, J. Squyres, V. Sahay, P. Kambadur, B. Barrett, A. Lumsdaine, et al. Open MPI: Goals, concept, and design of a next generation MPI implementation. *Proceedings, 11th European PVM/MPI Users' Group Meeting*, pages 97–104, 2004.

[7] R. Graham, S. Choi, D. Daniel, N. Desai, R. Minnich, C. Rasmussen, L. Risinger, and M. Sukalski. A Network-Failure-Tolerant Message-Passing System for Terascale Clusters. *International Journal of Parallel Programming*, 31(4):285–303, 2003.

[8] W. Gropp, E. Lusk, N. Doss, , and A. Skjellum. A high-performance, portable implementation of the mpi message-passing interface standard. *Parallel Computing*, 22(6):789–828, 1996.

[9] S. Horman. iproxy: Running tcp services over udp.

[10] IEEE. IEEE 802.3ad "Link Aggregation", 2000.

[11] InfiniBand Trade Association. Infiniband. http://www.infinibandta.org/.

[12] S. Miura, T. Boku, M. Sato, and D. Takahashi. RI2N - Interconnection Network System for Clusters with Wide-Bandwidth and Fault-Tolerancy Based on Multiple Links. In *ISHPC*, pages 342–351, 2003.

[13] Myricom. Myrinet. http://www.myri.com/.

[14] S. Pakin and A. Pant. VMI 2.0: A dynamically reconfigurable messaging layer for availability, usability, and management. *SAN-1 Workshop (in conjunction with HPCA)*.

[15] PC Cluster Consortium. SCore Cluster System Software. http://www.pccluster.org/.

[16] S. Sumimoto and K. Kumon. PM/Ehernet-kRMA: A High Performance Remote Memory Access Facility Using Multiple Gigabit Ethernet Cards. In *CCGrid 2003*, pages 326–333, 2003.

[17] A. J. van der Steen and J. J. Dongarra. Overview of recent supercomputers. http://www.top500.org/orsc/.