High Performance Communication Subsystem for Clustering Standard High-Volume Servers Using Gigabit Ethernet*

Wenzhang Zhu, David Lee, Cho-Li Wang
Department of Computer Science and Information Systems
The University of Hong Kong
Pokfulam, Hong Kong
{wzzhu+cmlee+clwang}@csis.hku.hk

Abstract

This paper presents an efficient communication subsystem, DP-II, for clustering standard high-volume (SHV) servers using Gigabit Ethernet. The DP-II employs several light-weight messaging mechanisms to achieve low-latency and high-bandwidth communication. The test shows an 18.32 us single-trip latency and 72.8 MB/s bandwidth on a Gigabit Ethernet network for connecting two Dell PowerEdge 6300 Quad Xeon SMP servers running Linux. To improve the programmability of the DP-II communication subsystem, the development of DP-II was based on a concise yet powerful abstract communication model, Directed Point Model, which can be conveniently used to depict the inter-process communication pattern of a parallel task in the cluster environment. In addition, the API of DP-II preserves the syntax and semantics of traditional UNIX I/O operations, which make it easy to use.

Keywords: *Gigabit Ethernet, SMP server, low-latency communication, cluster*

1. Introduction

The recent advances in improved microprocessor performance and high speed networks are making clusters an appealing vehicle for cost effective parallel computing [1]. Particularly, the emerging of chipset technology in supporting symmetric multiprocessing (SMP) servers has proved successful in making Standard High-Volume (SHV) servers, such as the 4-way or 8-way x86-based SMP servers, for high-speed computing. In addition, Gigabit Ethernet has recently becomes an ideal system area network (SAN) for SHV clusters because of its reliability, simplicity and lower cost. With the supports of such high performance interconnection networks, multiple SHV servers can be connected to form a powerful supercomputing environment.

In the past, various fast messaging mechanisms for clusters have been proposed, such as AM [4], FM [5], U-Net [6], VMMC [13], and BIP [7]. These mechanisms have been ported on Fast Ethernet, ATM or Myrinet. Recently several prototype cluster communication systems using Gigabit networking have been built. For example, Berkeley's Linux VIA [9] is a high-performance implementation of the Virtual Interface Architecture [14] over Myrinet LanAI 4.x boards and Linux 2.0.x. It can achieve a round-trip latency of 70 us for small messages, and 53.1 MB/s bandwidth for large messages. Packet Engines' implementation of VIA for Linux, M-VIA [8], has optimized drivers for Packet Engines' G-NIC II Gigabit Ethernet cards (Hamachi). M-VIA provides a factor of 2 to 3 latency improvement over the Linux TCP performance. GigaE PM [3] can achieve 48.3 us round-trip latency and 56.7 MB/s bandwidth on Essential Gigabit Ethernet NIC using Pentium II 400 MHz processor. GigaE PM II [2] has been implemented on Packet Engines G-NIC II for connecting Compaq XP-1000 workstations, each with 64-bit Alpha 21264 processor running at 500 MHz. The performance results show a 44.6 us round-trip time for an eight-byte message. XP1000's four 64-bit CPU data buses, which support a 2.6 GB/s aggregate bandwidth, help GigaE PM II achieve 98.2 MB/s bandwidth for message length 1,468 byte.

Even thought these communication systems achieved good performance results. However, these communication subsystems are always lack of good programmability as well as high resource utilization. Indeed, the design of high-speed networking for clustering SHV servers demands higher standards than connecting low-end PCs or workstations for general-purpose parallel computing. In this paper, we focus on the following design issues of the communication subsystems for clustering SHV servers.

* The research was supported by Hong Kong Research Grants Council grant 10201701 and HKU CRGC grant 10203009.

- ***Low Resource Consumption:*** The design of communication subsystems for clustering high-end servers should minimize the resource usage of the host machines, including CPU and memory. Some high-speed networking solutions achieve low latency by pinning down a large area of memory but with low utilization. Trading memory space for shorter communication latency is not proper design when clustering the SHV.
- ***High Availability Communication Channel:*** Any failure in communication may result in unrecoverable loss. The communication subsystem should be reliable and support certain level of fault tolerance since servers are usually serving crucial tasks. In addition the communication subsystem of the server cluster should allow server nodes to be added or removed at any time without stopping its routine work.
- ***Good Programmability:*** The inter-process communication patterns should be easily expressed based on the abstraction model and coded using the provided API without long learning period.
- ***Multi-protocol Support:*** Each server node not only connects to some other server nodes but also connects to the external world using standard communication protocols. The new communication subsystem should coexist with traditional networking protocols (e.g. TCP/IP) running on the same network.

For the rest of the paper, we first introduce the Directed Point communication model in Section 2. Then we discuss the architecture of DP-II communication subsystems in Section 3. The light-weight messaging techniques are discussed in Section 4. In Section 5, we describe the implementation details and the performance measurement using a 4-phase model. Finally, the conclusions are given in Section 6.

2. Directed Point Abstraction Model

The communication traffic in a cluster is caused by the inter-process communication between a group of cooperating processes, which reside on different nodes to solve a single task. Various communication patterns are usually used in algorithm design, such as point-to-point, pair-wise data exchange, broadcast tree, total-exchange, etc. A communication abstraction model can be used to describe the inter-process communication patterns during the algorithm design stage, as well as a guide to implement the primitive messaging operations or API for the underlying communication subsystem.

The Directed Point abstraction provides programmers with a virtual network topology among a group of communicating processes. Directed Point abstraction model is based on a Directed Point graph (DPG). It allows users to statically depict the communication pattern and provide some schemes to dynamically modify the pattern during the execution time. All inter-process communication patterns can be described by a directed graph, with a directed edge connecting two endpoints representing a uni-directional communication channel between a source and a destination processes. A formal definition of DPG is given below:

Let DPG = (N, EP, NID, P, E), where N, EP, NID, P and E are:

N (Node set): A subset of integer set, representing the nodes in a cluster.

EP (Endpoint set): A subset of integer set, representing endpoints of the directed edges.

P (Process set): The power set of *EP*, each element in *P* represents all endpoints created by a communicating process in a cluster. For example, P_i represents all the endpoints created by process *i*. A process in DPG is usually shown as a circle; while the endpoint is shown as a vertex in the circle.

NID (Node Identification function): *NID* is a function from *P* to *N*, representing the node in a cluster where a process resides. For simplicity, we write $NID(P_i)$ as NID_i. The restriction on *NID* is that $\forall P_i\,,\ P_j \in P$: $NID_i = NID_j \rightarrow P_i \cap P_j = \varnothing$. This property ensures that no two processes in the same node will share the same endpoints.

E (Edge set): $E=\{<i,m,j,n> \mid i \in P_a$ and $j \in P_b$ and $NID_a = m$ and $NID_b = n$ and $a \neq b$ where *i, j, m, n, a,* and *b* are all integers, P_a and $P_b \in P\}$. We use the notation $<i,m> \rightarrow <j,n>$ to represent an edge $<i,m,j,n>$ in *E*, which is a communication channel for sending messages from the endpoint *i* of process *a* to an endpoint *j* of process *b*.

The proposed model supports not only the point-to-point communication but also other types of group operations. For examples, an endpoint can be used as the root of a broadcast tree or a destination point for a reduce operation. Below is a simple example to illustrate the usage of the DP abstraction model.

Given a DPG = (*N, EP, NID, P, E*), where

$N \quad = \{1, 2, 3, 4\}$

$EP \quad = \{1, 2, \dots, 256\}$

$P \quad = \{P_1, P_2, P_3\}$

$NID_1 = 1$, $NID_2 = 1$, $NID_3 = 2$

$P_1 = \{1, 2, 3\}$, $P_2 = \{5, 6\}$, $P_3 = \{1, 2, 7, 8\}$

$E = \{ <1,1,5,1>,\ <3,1,6,1>,\ <2,1,2,2>, <2,1,6,1> \}$

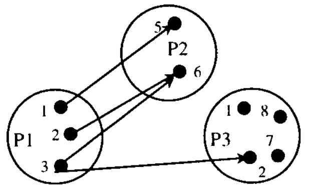

Figure 1. A Simple Example of DP Graph

Figure 1 shows the diagram to represent the DP graph of the given example. From the function NID, we know that process 1 and process 2 are executed in node 1. There are four communication channels between these processes. For example, the channel <1,1> → <5,1> is from the endpoint 1 of process 1 to the endpoint 5 of process 2. The endpoint 2 in P_1 is used to connect with P_2 and P_3.

DP graph provides a snap shot of the process-to-process communication. The inter-process communication pattern can evolve by adding a new endpoint within a process, adding a new edge between two distinct endpoints in different processes, deleting an endpoint as well as the edges linked to it, or deleting an edge between different endpoints. With these operations, any run-time inter-process communication patterns can be modeled.

3. DP-II Architecture

Based on the DP abstraction model, we design DP-II communication subsystem. DP-II consists of three main layers: (1) API Layer (2) Service Layer (3) Network Interface Layer. Figure 2 shows an overview of the DP architecture.

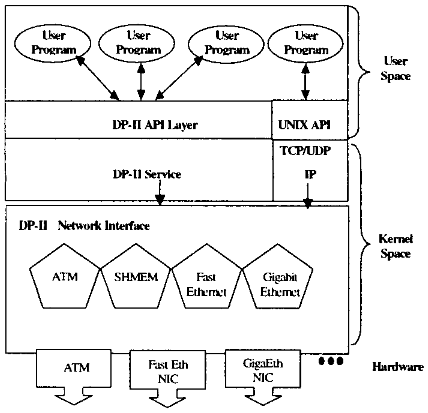

Figure 2. The Architecture of DP-II

The API Layer implements the operations for users to program their communication codes. To provide better programmability, DP-II API preserves the syntax and semantics of traditional UNIX I/O interface by associating each DP endpoint with a *file descriptor*, which was generated when a DP endpoint is created. All messaging operations can only access through the *file descriptor* for sending or receiving messages. The communication endpoint is released by closing the file descriptor. With the file descriptor, a process can access the communication system via traditional I/O system calls. This kind of interface has been widely used in traditional UNIX I/O, such as Socket, which can reduce the burden of learning new API.

The DP-II Service Layer realizes the DP abstraction model and is hardware independent. It is built by different components to provide services for passing message from user space to network hardware and to deliver incoming packets to the buffer of the receiving process.

The DP-II Network Interface Layer consists of network driver modules. Most of the driver modules are hardware dependent. Each of them is an individual kernel module that can be loaded to and unloaded from the system. Multiple network interfaces can be loaded at the same time. Currently, network driver modules supported in DP-II include Digital DEC 21140A Fast Ethernet, Hamachi Gigabit Ethernet, and FORE PCA-200E ATM. We have also developed DP SHMEM module to support intra-node communication through shared memory. Modular design makes DP-II implementation need not recompile the whole kernel source tree while adding new drivers.

4. Light-Weight Messaging Techniques

DP-II is designed with the goals to achieve low communication latency and high bandwidth as well as minimizing the resource usage. We propose various techniques, namely, *directed message*, *token buffer pool*, and *light-weight messaging call*. They reduce protocol processing overheads, network buffer management overhead and process-kernel space transition overhead.

In DP-II, we use Hamachi Gigabit Ethernet NIC as the network interface. The Hamachi Gigabit Ethernet NIC uses a typical descriptor-based bus-master architecture [11]. Two statically allocated fixed-size descriptor rings, namely, the transmit and receive descriptor rings.

Figure 3 shows the messaging flow with respect to different components in DP-II using such descriptor based network interface controller. The transmission unit of DP-II is called Directed Message (DM). DM packet consists of a header and a data portion called *container*. The header is constructed at DP service layer. It consists of three fields: target NID, target DPID, and the length of the container. The simplicity of DM packet only requires very small of packet processing time comparing to other complex protocols. The NART (Network Address Resolution Table) is used for the header construction in the transmission.

Buffer management affects the communication performance. On the receive side, we maintain a *token buffer pool* (TBP). It is a fixed-size physical memory area dedicated to a single communication endpoint. It is allocated when the communication endpoint is opened and freed when the endpoint is closed. The unit of storage in TBP is called token buffer. It is a variable-length storage unit for storing the incoming DM packet to reduce the memory usage as compared to the fixed length buffer

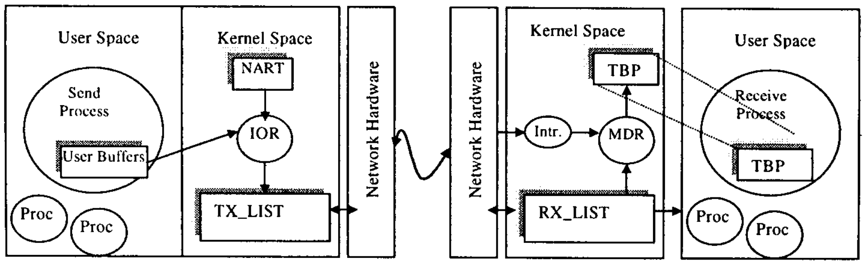

Figure 3. The Messaging Flow in DP-II

used in other implementations. Each token buffer has a control header, called token. It is a data structure containing the length and linkage information to next chained token buffer.

The TBP is directly accessible by kernel and user processes. Thus, incoming message can be directly used by user program. When the packet arrives, interrupt signal is triggered by the network interface. The interrupt handler calls MDR (Message Dispatcher Routine) to examine the header of packet, locate the buffer at TBP to store the incoming message based on the information stored in DP-II, and copy the incoming message to TBP. Since TBP is accessible by both kernel and user processes, no extra memory copy is needed to bring message up to the user space.

DP-II allocates one TBP whenever a new DP endpoint is opened. It requires no common dedicated system buffers for storing incoming messages. Thus, the memory resource in a server can be efficiently utilized. The amount of memory needed depends on the number of endpoints created in the applications.

To reduce the overheads while crossing kernel and user space, both send and receive operations in DP-II are using *light-weight messaging calls* (LMC). LMC provides fast switch from user space to kernel space. It is implemented using Intel x86 *call gate*. The use of LMC can eliminate the cost of possible process rescheduling, context switching, and bottom-half operations after return from a system call.

5. Performance Analysis

DP-II has been implemented to connect four Dell PowerEdge 6300 SMP servers, using Packet Engines' G-NIC II Gigabit Ethernet adapters. Each server consists of four Pentium III Xeon processors sharing 1 GB memory and 18 GB hard disk. Each processor consists of 512 KB L2 cache and operates at 500 MHz. All servers are installed with Linux 2.2.5 Kernel. Two G-NIC II Gigabit Ethernet adapters are used in each server to connect to the PowerRail 2200 Gigabit Ethernet Switch for the purpose of fault tolerance. Each server also has one Fast Ethernet connection to the campus LAN for external access. The PowerRail 2200 switch can achieve backplane capacity 22 Gbps.

5.1. Latency and Bandwidth Tests

We evaluated the performance of the single-trip latency of the communication system for various message sizes. In all benchmark routines, source and destination buffers were page-aligned for steady performance. The benchmark routines used hardware time-stamp counters in the Intel processor, with resolution within 100 ns, to time the operations. The round-trip latency test measured the ping-pong time of two communicating processes and repeated two hundreds iterations. The first and last 10% (in terms of execution time) were neglected. Only the middle 80% of the timings was used to calculate the average. Single-trip latency is defined as the average round-trip time divided by 2.

The bandwidth test measured the time to transmit 4 MBytes data from one process to another process, plus the time for the receive process to send back a 4-bytes acknowledgement. The time measured was then subtracted by a single-trip latency time for a 4-byte message. Thus, the bandwidth was calculated as the number of bytes transferred in the test divided by the calculated time.

Figure 4 shows the latency results. The DP-II can achieve single-trip latency 18.35 us for send 1-byte message with back-to-back Gigabit Ethernet connection. The switch causes at lease extra 21 us delay while performs in a store-and-forward mode for data transmission.

Figure 5 shows the bandwidth results of TCP/IP and DP-II with back-to-back connection. The DP-II can achieve sustained bandwidth 72.8 MB/s at message size 1504 bytes; while TCP/IP can only achieve 36 MB/s.

5.2. Performance Breakdowns

To help understand the performance results, we examine the communication cost of a single-trip data transfer using a 4-phase model. The 4-phase model consists of the following parameters:

l : The length of the message.

L_t : The single-trip communication latency.

$T_{startup}$: The start-up time of a send operation. It includes the time for the API wrapper, the time to switch from user-space to kernel space and the time to prepare the frame header.

T_{send} : The time spent in DP-II I/O operations (IOR) on the sender's side to copy the user space buffer to the NIC's DMA buffer and set up its descriptor for the NIC.

T_{net} : The network delay and the OS overhead. The network delays include the time to copy the data from host memory to the NIC on the sender's side and from the NIC to the host memory on the receiver's side. Generally, different parts in the network delay time may overlap. The OS overhead includes the execution time of interrupt handler in the OS kernel.

$T_{delivery}$: The message delivery time. It is the time to deliver an incoming message to the destination memory at receiving process, which is mainly the execution time of Message Dispatch Routines.

Thus, to transmit a message of size l, the single-trip latency time can be expressed by the following equation:

$$L_t(l) = T_{startup} + T_{send}(l) + T_{net}(l) + T_{delivery}(l)$$

Figure 6 shows the latency breakdown on sending 1 byte message. Performance breakdowns on various x86-based PCs connected by 32-bit PCI Fast Ethernet NICs were reported for the purpose of comparison. For all testing cases, DP-II shows small overheads in handling the communication protocol and the delay occurred in starting up the PCI bus and NIC. On Gigabit Ethernet, the $T_{startup}$, T_{send}, T_{net}, and $T_{delivery}$ time for transmitting 1 byte message are 0.44, 0.7, 16.82, and 0.36 us respectively. All machines achieved nearly the same performance at the network delay (T_{net}). The G-NIC II network interface didn't cause long delay in its more complex hardware. The K6-2 shows the largest delay in T_{net} which could be caused by its special Socket-7 motherboard architecture. For startup, send, and deliver phases, faster CPU can always achieve shorter latency for handling the communication protocol. The K6-2 featured by its larger L1 cache and 1MB on-board L2 cache can handle protocol execution faster than other INTEL x86-based PCs on Fast Ethernet. Overall, the faster 500 MHz Pentium III Xeon processor, efficient PCI bus design, and faster system bus on the PowerEdge help in achieving much smaller overheads in startup, send and delivery phases.

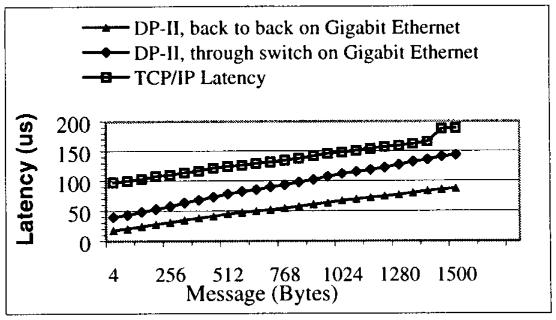

Figure 4. Single-trip Latency Performance

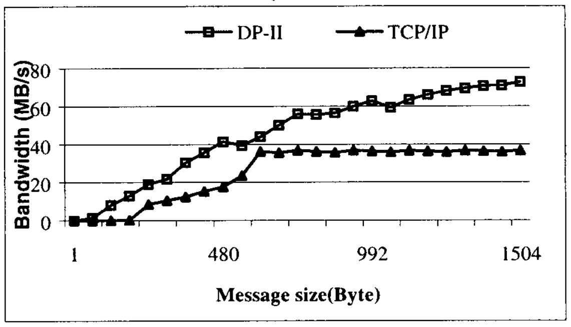

Figure 5. Bandwidth Performance

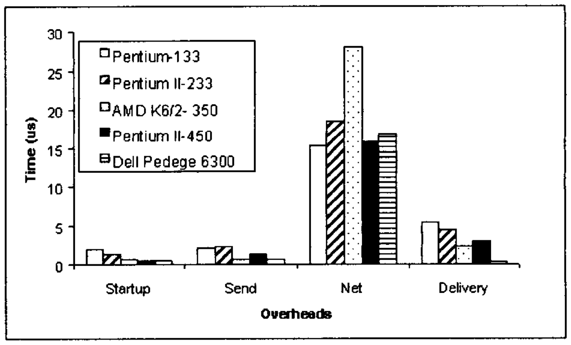

Figure 6. Single-trip Latency Breakdown on Sending 1-byte Message

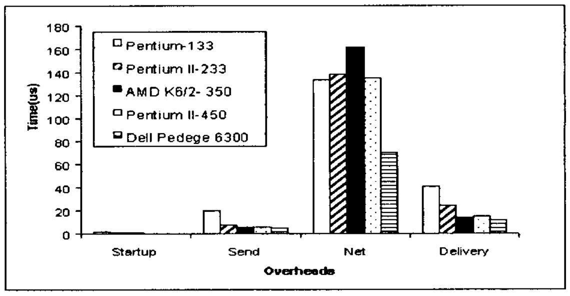

Figure 7. Single-trip Latency Breakdown on Sending 1504-byte Message

Figure 7 shows the latency breakdown on sending standard Ethernet packet of payload 1504 bytes. With the support of the 64-bit PCI bus and the efficient INTEL 82450NX chipset [12] in the PowerEdge server, T_{net} was significantly reduced as compared with the rest of PCs connected by Fast Ethernet since data can be moved between memory and Hamachi NIC on a wider PCI bus. The measured $T_{startup}$, T_{send}, T_{net}, and $T_{delivery}$ time for transmitting 1500 byte message are 0.44, 5.23, 69.96, and 12.21 us respectively. $T_{startup}$, T_{send}, and $T_{delivery}$ involve host node processor. All together they contribute 20.3 % total messaging time for sending 1504 bytes. Major delay was still contributed by the host PCI and the Hamachi NIC. In the 64-bit 33 MHz PCI server, the speed of PCI bus with its overhead seems slower than the full-duplex Gigabit Ethernet line rate.

6. Conclusions

In this paper, we present a high-performance communication subsystem DP-II on Gigabit Ethernet based on the Directed Point Model. We emphasize both on high-performance communication as well as good programmability. With the performance breakdowns, we have shown that the DP-II has greatly reduced the software overheads. Our light-weight messaging mechanisms can reduce the CPU involvement while performing data communication on the SHV server. However, while Gigabit network media is able to transfer data in low latency and high bandwidth, the network delay (T_{net}) still contributes major portion of the communication time in sending both short and long message. We conclude that the current bottleneck in Gigabit Ethernet networking is the interface between CPU and NIC. The move from a 100 MHz PC system bus to a higher clock rate bus, as well as the move from a 64-bit 33 MHz PCI bus to a 64-bit 66 MHz PCI interface could greatly improve the communication performance in the future.

References

[1] Mark Baker, "Cluster Computing White Paper", http://www.dcs.port.ac.uk/~mab/tfcc/WhitePaper/WhitePaper.htm

[2] Shinji Sumimoto, Atsushi Hori, Hiroshi Tezuka, Hiroshi Harada, Toshiyuki Takahashi and Yutaka Ishikawa, "GigaE PM II: Design of High Performance Communication Library using Gigabit Ethernet", http://pdswww.rwcp.or.jp/db/paper-J/1999/swopp99/sumi/sumi.html

[3] Shinji Sumimoto, Hiroshi Tezuka, Atsuhi Hori, Hiroshi Harada, Toshiyuki Takahashi, and Yutaka Ishikawa, "The Design and Evaluation of High Performance Communication using a Gigabit Ethernet", *International Conference on Supercomputing* '99, June 1999, pp. 243-250.

[4] T. Von Eicken, D. E. Culler, S. C. Goldstein and K.E. Schauser, "Active Messages: a Mechanism for Integrated Communication and Computation". *The 19th Annual International Symposium on Computer Architecture*, Gold Coast, Qld., Australia, May 1992, pp.256-266.

[5] S. Pakin, V. Karamcheti, A. A. Chien. "Fast Messages: Efficient, Portable Communication for Workstation Clusters and MPPs", *IEEE Concurrency*, vol.5, (no.2), April-June 1997, pp.60-72.

[6] T. von Eicken, Anindya Basu, Vineet Buch and Werner Vogels, "U-Net: A User-level Network Interface of Parallel and Distributed Computing", *Proc. of the 15th ACM Symposium of Operating Systems Principles*, vol. 29, (no.5), Dec. 1995, pp. 40-53.

[7] L. Prylli and B. Tourancheau. "BIP: a New Protocol Designed for High Performance Networking on Myrinet", *Workshop PC-NOW, IPPS/SPDP98*, Orlando, USA, 1998.

[8] M-VIA: A High Performance Modular VIA for Linux, National Energy Research Supercomputer Center at Lawrence Livermore National Laboratory, http://www.nersc.gov/research/FTG/via/

[9] P. Buonadonna, A. Geweke, D. Culler, "An Implementation and Analysis of the Virtual Interface Architecture", *Proc. of Supercomputing '98*, Orlando, Florida, November 1998.

[10] Alan Heirich, David Garcia, Michael Knowles & Robert Horst, "ServerNet-II: a Reliable Interconnect for Scalable High Performance Cluster Computing", Compaq Computer Corporation, Tandem Division, http://www.servernet.com/flat/public/brfs_wps/snetii/snetii.pdf

[11] Donald Becker, "A Packet Engines GNIC-II Gigabit Ethernet Driver for Linux", http://beowulf.gsfc.nasa.gov/linux/drivers/yellowfin.html

[12] Intel ADC450NX Pentium II Xeon Processor Server Dynameasure/Massaging Performance Report, version 1.0, Server Performance Lab., INTEL Corporation, June 1998.

[13] E. Felten, R. Alpert, A. Bilas, M. Blumrich, D. Clark, S. Damianakis, C. Dubnicki, L. Ifode, and K. Li, "Early Experience with Message-passing on the Shrimp Multicomputer", *Proc. of the 23rd Annual Symposium on Computer Architecture*, 1996.

[14] "Virtual Interface Architecture Specification V1.0", ", *Compaq, Intel and Microsoft Corporations*, Dec. 16, 1997, http://www.viarch.org/

[15] Raymond Wong and Cho-Li Wang. "Push-Pull Messaging: A High-Performance Communication Mechanism for Commodity SMP Clusters", *Proc. of International Conference on Parallel Processing*, Fukushima, Japan, September 1999, pp. 12-19.

[16] Anthony Tam and Cho-Li Wang, "Realistic Communication Model for Parallel Computing on Cluster," *the First International Workshop on Cluster Computing*, August 11, 1999, pp. 92-101.