AmpNet – A Highly Available Cluster Interconnection Network

Amy Apon
University of Arkansas
aapon@uark.edu

Larry Wilbur
Belobox Networks
larrywilbur@belobox.com

Abstract

One of the most important challenges facing computing clusters in the foreseeable future is providing fault tolerant, high availability cluster hardware for non-stop applications. This capability is in addition to high throughput and low latency. This paper presents the Advanced MultiProcessor Network (AmpNet), a gigabit speed cluster interconnect that was designed with these issues in mind. The AmpNet Network Interface Card (NIC) uses network-shared memory as network cache to provide a fault-tolerant, self-healing network with no data loss. Higher-level network centric services use network-shared memory to ensure high availability and continuity of service in applications. In addition, the programmable NIC, with low-latency messaging protocols and field upgradeable soft logic, provides a foundation for researchers who would like to develop additional cluster services and protocols for network centric computing. This paper describes the fault tolerant design and implementation of the AmpNet hardware architecture.

Keywords: highly available clusters, fault tolerant, rostering, network cache, real-time distributed systems

1. Introduction

Many high-performance clusters today need a fault tolerant and highly available architecture. For example, applications that require the highest level of availability include 24x7 industrial or healthcare applications, military applications, and international business applications. However, to achieve the highest levels of availability requires careful design of all of the components of the cluster, including the cluster interconnects, software protocols, system management processes, and the applications. A highly available network must be able to sustain one or more failures and continue to operate at full capacity with all services. Under severe damage conditions, a highly available network continues to function, but in a degraded mode. Ideally, faulty components are replaced during normal network operations, and the network automatically incorporates repaired components back into the network. To support this requirement, highly available systems are expected to operate correctly in the presence of multiple failures using a subset of the original components, perhaps with reduced capacity. That is, the system must be self-healing and reconstitute itself without loss of data or application services. The system must detect failures and reconfigure system operations dynamically.

The most commonly used cluster interconnection networks today have been designed first for high performance communications. The fault tolerant and self-healing aspects of the network, if any, have been designed as a secondary aspect. For example, Gigabit Ethernet is designed for high throughput on a local area network and for server access. However, the point-to-point links and spanning tree routing protocols of Ethernet do not directly address how to repair the cluster network in the event of a failure [13]. Support for a higher-level protocol that ensures reliable data transfer, such as TCP/IP, is required by the host nodes in order to ensure that the data is not lost. The use of TCP/IP in cluster networks has high overhead for most applications.

Some networks, such as Myrinet [3], Scalable Coherent Interface (SCI) [4], and Quadrics Interconnection Network (QsNet) [11], have been designed specifically for the high throughput, low latency, and low error rates expected in clusters. These networks also support low-latency protocols that operate with minimal copies to system memory and minimal operating system interaction during message sends and receives. SCI addresses the loss of routing in the event of a failure and will dynamically reconfigure the routing on affected links. In addition to low-latency protocols with SCI, efforts have been made to reduce network access time by using a CC-NUMA card that is directly plugged into the PCI slot of each cluster node and contains shared memory, network cache, and interconnection modules [2]. The use of shared memory on the network card is shown to improve the performance of applications.

With Myrinet, the GM protocol is a low-latency protocol that will automatically handle transient network errors such as dropped or corrupted packets. GM will also dynamically reconfigure the routing on any affected network links. In the case of catastrophic loss of network
connectivity, low-level cooperation of the client programs is required. If a portion of the network suddenly becomes unavailable, the return codes on the GM send calls will indicate that the send was unsuccessful. The communication layer of the application is expected to make a decision about how to handle this failure, and can choose whether to abort the send or to try to send again in a different manner [10]. Several protocols have been built over Myrinet that attempt to address the problem of reliability together with high-performance, such as BIP, Bulldog, and Trapeze [6].

QsNet uses a programmable processor in the network interface and integrates the local virtual memory into a distributed virtual shared memory [11]. The network architecture supports network fault detection and fault tolerance through the use of a fat-tree topology. QsNet shows very high throughput and low latency and can also reconfigure routing tables and automatically re-transmit packets whenever a fault is detected.

AmpNet is designed to provide fault tolerance and high availability of applications to high performance clusters. In addition to supporting the dynamic reconfiguration of network routing in the event of a failure, AmpNet provides a fault tolerant, write through, network-shared memory, called network cache, which is accessible to user applications and can be used to recover data in the event of a failure and reconstitute system and application services. The design focus of AmpNet is a hardware implementation of a distributed shared memory architecture that is capable of supporting paradigms similar to programming in a single symmetric multiprocessor computer. Therefore, the requirements include extremely robust network hardware. Just as ECC memory technology virtually eliminates concern about the integrity of data that is read from host memory, the AmpNet network virtually eliminate any concerns about the integrity and availability of data in network cache.

Reliable, low-latency protocols are used to implement higher-layer support for the high-availability of applications. In addition, higher software layers of AmpNet have been designed to provide the extra services needed to support application migration so that the entire cluster application environment can be fault tolerant and self-healing.

Network reconfiguration and self-healing in AmpNet occur through a process called rostering. Rostering is implemented in hardware and is based on the efficient, deterministic hardware broadcast capability that is available in AmpNet. During healing and data recovery the network operates in a pure broadcast mode. Unicast switching is activated after recovery and network reconstitution. With current hardware, rostering and physical reconstitution occur in less than a millisecond. Data refresh requires 1-2 milliseconds, and application fail over can be as quick as three milliseconds from the instant a failure is detected.

The focus of this paper is on the functionality of the distributed memory capability provided by network cache and the hardware rostering algorithm and self-healing capability. Section 2 gives a general introduction to the AmpNet network architecture, including hardware components, the data link layer, and the interconnection network. Section 3 describes network cache, and Section 4 describes rostering and the reliability protocols. Section 5 is a discussion of network and memory bandwidth issues. Section 6 gives conclusions and future work.

2. AmpNet Network Architecture

The current AmpNet network has evolved from a design and development process that began in the early 1990’s. The demands of large-scale military applications required fast communication with very low latency. Gigabit networks that became available during this time period were fast but not highly available and lacked determinism. The real-time requirements of the military applications require the network to be both deterministic and highly fault and damage tolerant. Hardware support is also needed to ensure that the causal order of updates is absolutely preserved, even in the presence of failures, and a standard was developed to meet these needs. The Real-Time Fiber-Optic Communications (RTFC) Principles of Operations was published in January 1998 [8]. The concept of rostering and self-healing developed and
refined in RTFC is used in AmpNet with only minor changes.

While RTFC suits many of the needs of large-scale military applications, it has some disadvantages for cluster architectures. RTFC is only available in a VME chassis form factor and is only supported for a few operating systems commonly used in military applications, such as VxWorks. Furthermore, RTFC is a broadcast only network that does not address the efficient delivery of unicast data. Finally, RTFC sacrifices high bandwidth for low latency. AmpNet has been designed to address the deficiencies of RTFC and to provide higher-level network centric services for a fault tolerant application development environment supporting high-performance cluster applications. AmpNet is available in both a VME and PCI form factor and is supported by a variety of operating systems, including Windows 2000, Linux, AIX, Solaris, IRIX, HP-UX, and VxWorks.

AmpNet hardware components include hubs, switches, routers, and network interface cards (NICs) and are built using Field Programmable Gate Array (FPGA) technology to reduce the cost of hardware maintenance and upgrades. The components are field upgradable with new firmware and soft logic whenever new standards and technology become available. In addition, custom software and soft logic can be added to network components to meet special real-time user requirements and new designs.

The AmpNet NIC, shown in Figure 1, fits into a single PCI slot and auto-detects either a 33 or 66 MHz PCI bus, and PCI bus signaling at either 3.3 or 5.0 VDC. Figure 2 illustrates the block design of the AmpNet NIC. Each NIC has a Motorola ColdFire 5102 40MHz 32-bit microprocessor with 4 MB to 16 MB of local RAM, 1MB factory PROM, 3MB FLASH, and a multi-channel DMA controller with a 32-bit hardware checksum capability. Each NIC has a Time Of Day chip with an 8KB battery-backed SRAM, and any NIC can maintain microsecond-accurate POSIX network time for all nodes in the cluster. Multiple time strata are supported with external time inputs connected to the network, such as GPS receivers.

The AmpNet Distributed Kernel (AmpDK), a small deterministic operating system kernel, executes on the microprocessor. The AmpNet PCI NIC effectively provides an embedded real-time computer in every Windows 2000 or Linux based computer that allows messages to be transferred deterministically between all NICs and hosts on the cluster. The NICs can also host distributed multi-threaded user application processes at all nodes or at specific nodes as required.

AmpNet is available for VME bus-based computers with the same features as the AmpNet PCI NIC. A FiberPack hardware component provides a standard Fibre Channel (FC-0 and FC-1) physical interface between the AmpNet NIC and the network fiber. Installed FiberPacks are automatically detected and configured by the NIC. The modular design allows flexibility for various types of media. FiberPack options include support for single mode and multimode fiber-optic media. FiberPacks plug onto the network module (NICs, hubs, switches or routers) using a single 160-pin high-density connector, and are detected and configured automatically.

The data link layer of AmpNet is implemented using a micropacket format [9]. The micropacket format supports up to sixteen different types. The defined micropacket types are: data, rostering, DMA, interrupt, diagnostic, and D64 atomic. Rostering micropackets are used in the rostering process, described in Section 4. Only the DMA micropacket is variable in length. When the DMA engine is used to write blocks of data into the network cache, either broadcast or unicast DMA micropackets are
The Interrupt micropacket can generate an interrupt in a specific node or in all nodes and can be used to indicate completion of a read or write operation in network cache.

Figure 3 illustrates the format of the fixed length data micropacket. Each micropacket contains four bytes of control data. One byte each is used for checksum, packet control, sender ID, and payload variable. Four bits of packet control identify the micropacket type. The remaining four bits of packet control are specific to the type of packet. Bits in the packet control also indicate whether the packet is unicast or broadcast, and if unicast, the payload variable indicates the receiver ID. The data is written to bytes in the specified network cache word using the Write Byte Select (WBS) bits, giving network cache the same capability as a host memory system.

The size of a fixed-length data micropacket is the result of a design decision that was made very early during the development of the network. An important design goal of military systems is to have deterministic message delivery times. Deterministic message delivery times support real-time applications and ensure correctness of the higher-level protocols for performing fault detection and self-healing. The size of the data payload in the data micropacket is the maximum amount that can be moved in a single CPU operation, so the amount of time to send the data is very short and predictable (sub microsecond). In addition, since small messages are delivered without having to wait behind large messages, the small packet size guarantees fine-grain, deterministic multiplexed streams of data, including real-time data, voice and video. An additional design goal of low latency led to the decision to use a fixed-length address in the packet header. With this design, a micropacket can be transmitted by the NIC in a single CPU operation.

These design decisions have tradeoffs. The addressing results in a constraint of 254 on the number of nodes that can be directly addressed in the network at the lowest level of distributed shared memory. As discussed below, larger addresses at the micropacket level would destroy the low-latency requirement for broadcast data. Also, the overhead of sending a small fixed-length micropacket is 75%, which includes checksum, control, ID, and address information as shown in Figure 3, and an additional EOFA word between each packet in a stream of fixed-length data micropackets.

The overhead of using fixed-length data micropackets results in a lower maximum user bandwidth than is possible with most other gigabit networks. To address this issue while still maintaining deterministic message delivery times, an examination was performed of the sources and causes of delay in the network hardware. The observation was made that deterministic latency can be maintained as long as a short, single arbitration bus cycle is required to send the largest packet. The larger DMA micropacket used in AmpNet has been designed to allow a data size that is equal to the amount of data that can be moved in short, single bus arbitrations. This size can be changed to suit the capabilities of the host hardware. The larger DMA micropackets and the short fixed-length data packets can be interleaved on the network while maintaining deterministic latency. DMA transfers support a sustained, aggregate broadcast bandwidth of 85 Mbytes per second and full-duplex unicast transfers of 170 Mbytes per second over 1.0625 gigabit per second media. Faster transfers are possible with higher-speed Fibre Channel components.

The basic network topology of AmpNet is a set, or segment, of nodes that is connected via one or more switches. The nodes within a segment form a physical star for switching and a logically sequenced segment for rostering and broadcast. Each node has a physical sequence number in increasing order around the segment that is automatically assigned by the switch. From one up to four redundant network paths can be configured to form a dual, triple, or quad-redundant cluster interconnection network. When a segment is configured with two or more switches, broadcast and unicast data can be directed over different paths so as to maximize the use of available bandwidth.

In broadcast mode, AmpNet is a variant of a register insertion ring and guarantees real-time, in-order, reliable delivery of data. During normal operations broadcast packets are removed from the ring by the sending node. A unique purge node in each segment removes any orphan broadcast packets that traverse it more than one time.

The design decision was made to use a sequenced segment, rather than a bus, tree, or other topology for several reasons. The system requirements of gigabit data transmission in a large distributed system necessitate the use of unidirectional fiber optic links. As other gigabit networks have demonstrated, a bus topology is impractical for the speeds and distances required. Secondly, since

<table>
<thead>
<tr>
<th>Word 0</th>
<th>Word 1</th>
<th>Word 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Checksum</td>
<td>4-bit WBS and 28-bit Network Cache Word Address</td>
<td>32-bit Network Cache Data</td>
</tr>
<tr>
<td>Packet Control</td>
<td>Sender ID</td>
<td>Payload Variable</td>
</tr>
<tr>
<td>23-16</td>
<td>15-08</td>
<td>07-00</td>
</tr>
</tbody>
</table>

Figure 3: Fixed-Length Data Micropacket Format
network access is always fair and there are no hot spots in
the network, the use of a sequenced segment allows the
network to gradually saturate as traffic is added, rather
than saturating quickly. A very important third factor is
that rostering time scales linearly both in the number of
nodes and in the length of the connections. The segment
can be automatically reconfigured in linear time using an
algorithm that is simple enough to implement in hardware.

The latency for broadcast transmissions can be
determined as follows. AmpNet uses 62.5-micron multimode fiber that supports a maximum distance of 300
meters between the node and the switch. Data propagation
time on the fiber, which can be included as appropriate, is
approximately 5 nsec per meter. The measured packet
processing time at the NIC is fixed at 875 nsec per packet.
In addition, the AmpNet switch in broadcast mode
introduces a 450 nsec processing time per packet between
nodes. Therefore, for broadcast transmissions each node
adds approximately 1.325 microseconds to the
transmission latency. This time includes updating network
cache in the node, but assumes no memory interference
at the node. The worst case memory interference is the cost
of one memory cycle, or 50 nsec. A latency of 1.4
microseconds per node is used in calculations to account
for some practical memory interference.

Some distributed real-time systems, such as nuclear
craft carriers, use the maximum distance possible
between nodes and switches and maximum separation
between switches to survive physical damage from missile
hits. However, users of these systems still want to
maintain acceptable low latency less than 300
microseconds. This is accomplished by limiting the
number of nodes in such clusters to 64. Assuming the
maximum distance of 300 meters to every node and a
maximum of 64 nodes gives a worst-case latency of 281.6
microseconds (300x2x64x.005 + 64x1.4). The first
deployed carrier systems have 16 nodes with a maximum
latency of 70.4 microseconds. For an equivalent 64-node
cluster that is tightly packaged, the worst-case glass
latency is 92.8 microseconds (5x2x64x.005 + 64x1.4),
based on using an average distance of 5 meters to each
node. For a maximum mini-cluster with 254 nodes, the
latency is 368.3 microseconds (5x2x254x.005 + 254x1.4).

The current algorithm for flow control is designed for
broadcast traffic and is performed using local node
observations [12]. Each NIC has an input and output
FIFO. If the output FIFO exceeds a threshold, currently
set at half full, then a hardware delay is introduced that
slows the rate at which micropackets are put onto the
network by that host. Since all nodes see this traffic, each
node can modify its own contributions as needed to keep
the segment from becoming saturated. That is, all nodes
participate in the delay of new micropackets equally, and
the total traffic on the network is controlled.

The designers of AmpNet recognize that a tradeoff of
using a sequenced segment topology for all traffic is the
high latency that is introduced for unicast messages as the
size of the segment increases. AmpNet allows for cut-
through switching of unicast messages. With cut-through
switching a unicast message is sent from the source node
to the switch, and then switched directly to the receiving
node. The receiving node purges the micropacket, and it
does not traverse the segment. However, the effectiveness
of the technique of using local node observations for flow
control is diminished by the use of cut-through switching for
unicast micropackets. Effective techniques for the
incorporation of efficient cut-through switching for unicast
packets are an active area of research.

3. Network Cache

The AmpNet PCI NIC features from 2MB up to 12MB
of network cache, as shown in Figure 2. Network cache is
a special-purpose memory that enables the NIC to provide
a hardware-implemented distributed shared memory. A
write to network cache results in data being sent on the
AmpNet network.

Network cache is divided into a system partition and a
user partition. As an example of how a programmer might
access the user partition of network cache memory,
suppose that two processes of a distributed application are
executing on nodes A and B in the cluster. In SPMD
fashion, each process might allocate a portion of the user
partition on network cache to local user memory, as:

```c
int x;
int *pNetCache;
int sizeOfNetCache;
...
an_ncMap(&pNetCache, &sizeOfNetCache);
...
x = pNetCache[4] // read fifth word
pNetCache[5] = x; //write sixth word
```

Thereafter, any valid references using the pointer
variable pNetCache cause a read or write of user
network cache. This virtual memory region that is
mapped onto user network cache is flagged by the
operating system as a non-cached memory region in the
host processor. Execution of the above read statement
causes a transfer of the fifth data word from user network
cache on the NIC across the system bus and into the local
host variable x. Execution of the above write operation
causes a transfer of the local variable x from across the
system bus to the sixth word of user network cache, and
the network cache write-though operation generates a
broadcast data micropacket containing the variable x.

A read by any process is a local host operation only and
does not cause any traffic on the network. However, any
writes to memory-mapped network cache in this fashion
by any process in the cluster cause data to be copied to the
corresponding network cache address of other nodes in the cluster. For example, suppose the process on Node A writes to address $p\text{NetCache}[5]$. The AmpNet NIC receives the data into network cache. At the same time a broadcast micropacket is constructed containing the data and sent onto the AmpNet network. In this case the data is a single integer and fits into a single micropacket. Since the micropacket has a broadcast packet type and a network cache word address that corresponds to a physical NIC address in each NIC, the data is copied into the corresponding physical location at every node. The next read of location $p\text{NetCache}[5]$ by a process on Node B will retrieve the new data. Note that the memory location $p\text{NetCache}[5]$ is not cached at Node B, which ensures that the newest data is always read from the NIC.

An interrupt micropacket can be sent from Node A to Node B to notify the receiving process that the data is available. Suppose, for example, that the data to be transferred is a large array or structure of items rather than a single integer. In this case, assignments would be made to the memory region as above. For large blocks, the AmpNet NIC uses DMA transfers. After the move, the process on Node A can send an interrupt micropacket to the process on Node B to notify it that the new data is available.

Because of the sequenced segment architecture, all host processes in the network are guaranteed to see the same order of writes to the same network cache address when the writes are performed by a single process. However, the memory in network cache is subject to race conditions if both the process on Node A and the process on Node B attempt to write to the same network cache address at the same time. To protect against race conditions AmpNet provides network-wide semaphores, which are currently implemented in software. The development of an expedited semaphore micropacket is an active area of research.

4. Rostering and Reliability Protocols

Fault tolerance is supported through the rostering process, which selects the best paths from many possible physical routes [8]. A node in an AmpNet network can be configured to have from one up to four transmitters and receivers for quad-redundant fault-tolerance. The two goals of rostering are: 1) to determine, at each node, which receiver to receive on, and 2) to determine the unique purge node in the segment that is responsible for removing orphan packets from the segment. Under extreme damage conditions the inclusion of all nodes may not be possible, so the goal in the selection of the receivers at the nodes is to determine the largest segment that can be built with the remaining accessible hardware. Then, data recovery in network cache is performed through collaboration by programs executing under the distributed micro kernel that executes in the onboard microprocessor in each NIC. Host computers are never involved in data recovery or the rostering process.

Rostering and automatic data recovery in network cache occur when any network component is abruptly powered off, destroyed, or catastrophically fails. If the NIC detects the fault, a Fiber Channel Link Reset is generated on all links to all switches. Then, each switch generates a Link Reset on all links to all nodes. Simultaneously, all hardware FIFOs in the switches are flushed, and all hardware FIFOs in the NICs are flushed and held reset during the entire rostering process. Any data written to network cache by processors or DMA engines continues to be successfully written to network cache, but no new traffic is generated on the network during rostering, which makes 100% of the network bandwidth available for the rostering process. This ongoing activity combined with the loss of data from flushing the glass and all FIFOs, results in network cache inconsistency and requires a cache refresh phase at the completion of rostering.

The worst-case time for the first step in the recovery operation is $2 \times 300$ meters x 5 nsec/meter, or 3 microseconds. However, it was empirically determined that it is best to wait 150 microseconds before proceeding with the rostering process. This period is referred to as the glass settling time that always follows a fault or damage event. Therefore, 150 microseconds after the detection of a fault, all nodes initiate a roster micropacket on all links and enter the rostering algorithm. Rostering uses a modified flooding algorithm in which the nodes forward roster packets and are responsible for filtering the packets to avoid overloading the network. A switch remains in pure broadcast mode during rostering. All nodes receive on all receivers and transmit on all transmitters during the rostering process. A node begins the rostering algorithm by sending a rostering packet on all outgoing links. The rostering packet is a fixed-length micropacket with a format similar to the format of the data micropacket as shown in Figure 3, except that the roster micropacket contains no data or network cache address and does contain entries for a wrap flag, the last transmitter ID, a node count, and a sender ID. The roster packets initiated by a node have the wrap flag set to FALSE, last transmitter ID set to the node’s ID, node count set to 0, and sender ID set to the node’s ID. The node also initializes a vector, MaxCount, to zero. MaxCount contains an entry for each node, and is used to hold the maximum count seen in any roster packet with the corresponding node ID. Additionally, each port has a wrap flag which is initially set to FALSE. When a node receives a roster packet it updates the last transmitter ID to its own ID and increments the node count. If a node receives a roster packet in which the sender ID equals its
Figure 4: Roster Packet Process in a Node

Is NODE ID= ORIGIN ID? Is this pkt from the present node?

A potential pass-along: if not filtered out, for population control.

Is NODE ID > ORIGIN ID?

Node got its own roster packet back.

YES NO

NO

NODEID < ORIGINID

Is NODEID > ORIGINID?

YES

YES

MAXCNT [ORIGINID] = NODE COUNT
Save input port # from which the present roster packet came.

NO

NODEID < ORIGINID

Is NODEID > ORIGINID?

NO

YES

MAXCNT (ORIGIN) > NODECOUNT?

Is Packet's wrap bit = true?

NO

Set pkt's wrap bit to "true".

YES

Set current input port's wrap flag to "true".

Return: Discard packet.

YES

Is Packet's wrap bit = true?

NO

Return: Discard packet.

YES

Is NODEID > MAXCNT [ORIGINID]?

MAXCNT [ORIGINID] = NODE COUNT
Save input port # from which the present roster packet came.

YES

Return: Forward packet.

NO

RETURN

Is NODEID > ORIGINID?

YES

NO

MAXCNT [ORIGINID] = NODECOUNT

RETURN

Discard packet.
own ID then the node has received its own roster packet. In that case, it does not forward the packet. If the node count in the packet is higher than the previous value of MaxCount[node ID], then the node sets the input receiver to the port that it has received the roster packet on and updates the value of MaxCount[node ID].

The wrap flag in the roster micropacket and the node with the lowest node ID are used by the nodes to filter packets that may travel a route that is shorter than the largest possible sequenced segment that can be built and to filter the total number of rostering packets. Additionally, if a switch does not receive a signal from a node, then it bypasses that node. To cover the case when the switch to node link is broken, the nodes follow the rule that darkness must be answered with darkness. Figure 4 illustrates the complete logic used by nodes for the rostering algorithm.

The network was designed to permit all components to be turned off and on in a live network as required for maintenance, expansion, or physical reconfiguration. The design also permits fiber-optic cables to be disconnected and reconnected on a live network. Therefore, chattering must be expected. This can also occur on a failing link that jumps in and out of sync or if the detection of light is intermittent. Chattering is dealt with by requiring a link to be stable for two seconds before entering the network. Built-in hardware diagnostics are used to validate links in an isolated loop back mode with the switch before entering the network as a resource.

If a fault or damage event is detected anytime during the roster process, the roster process is immediately restarted. There is no limit on the number of times rostering can be restarted. Extreme testing has shown that when faults stop occurring for approximately 500 microseconds, and given that the minimal components to form a network survived, the surviving components always form a network.

The rostering process completes in slightly less than two segment-tour times. Because of the deterministic nature of the segment, the roster process is declared complete by a node when a sufficient amount of time has passed since the beginning of rostering. Current implementations of AmpNet wait 500 microseconds from the last fault before releasing the FIFOs on the NICs and entering the cache refresh phase that follows the rostering process.

During cache refresh, any new transmissions continue simultaneously in the normal write through cache method of transmission. A high priority thread executing in the microprocessor onboard the NIC manages cache refresh for buffers that need to be refreshed. The NIC at each node is responsible for maintaining a circular log of every block access to network cache. The circular log maintains a pointer to the memory block, size, and a timestamp indicating when the block of memory was last updated. This circular log is also maintained for network cache updated during the rostering process. Whenever a fault occurs, the time of the fault is recorded. After rostering completes, if no data has been written to network cache during rostering and the millisecond prior to the fault, then no network cache refresh is required. However, if any data has been written to network cache, then the circular log will contain these entries. During network cache refresh, the high priority thread writes the network cache memory address and size of each log entry, as required, to a special set of cache refresh registers, and special cache refresh hardware generates DMA block mode micropackets to refresh this area of network cache.

At the detection of a fault, configuration options in the network permit a freeze condition to be enforced on all transmissions during the cache refresh process. The 500-microsecond time for the rostering process to complete allows the host driver to take control of the host computer and freeze all processing. If this option is selected then all host computers lose one to two milliseconds of computation time by being held in check by a very high priority Interrupt Service Routine in the host driver. Clearly, there is a story why such a capability exists.

Cache refresh and synchronization typically takes one to two milliseconds, no data is lost, and the process is transparent to all host computers unless the freeze option is selected. Once cache refresh completes, collaboration can occur. Each node is responsible for posting in its area of system network cache all information that pertains to the state of network components, the fact that it is present on the network, and its operational state. Higher-level protocol layers use this information to reconfigure middleware and application processes.

The network always maintains exactly one timekeeper node that maintains microsecond accurate and synchronized POSIX time in network cache. Any node can be timekeeper. However, the network supports multiple time strata and an algorithm that guarantees that some node in the highest valid time strata is timekeeper. Consequently, the timekeeper node is also elected to make the final configuration decisions, such as, “Does sufficient hardware remain to support both a switched path and a logical broadcast path between all nodes?” If not, the network can still continue in a degraded mode with all data multiplexed over a single path. In a multi-path network, if all configured nodes are also successfully connected to a common switch that is not used to form the broadcast path, then a separate unicast path is opened to that switch.

Upon completion of these final decisions and actions the network returns to a production state. Typical time for the entire process is 1-3 milliseconds depending on how much data must be refreshed.
5. Network and Memory Bandwidth Issues

Using only the small fixed size data micropacket designed to support low latency CPU direct operand write cycles to network cache, the broadcast bandwidth is 26.5625 MB/sec per gigabit of fiber-optic bandwidth. Using only the larger DMA micropacket, designed to support DMA block transfers to network cache, the broadcast bandwidth is 85 MB/sec per gigabit of fiber-optic bandwidth. The same data rates apply for each pair of point-to-point cut-through unicast transfers through a switch, limited by the switching technology and flow control. All broadcast, multicast, and unicast transfers can be either CPU or DMA initiated.

The current network was designed with a small replaceable FiberPack that supports the 1.0625-gigabit fiber-optic links. Development is proceeding on new Fiber Packs that operate at 1.25 and 2.5-gigabit links, with expected availability in Q2 2003. The data rates possible with the new Fiber Packs are 31.25 MB/sec (using the CPU for data transfer) and 100 MB/sec (using DMA for data transfer) at 1.25 gigabits/sec and 62.5 MB/sec (CPU) and 200 MB/sec (DMA) at 2.5 gigabits/sec.

Since this is a distributed shared memory architecture designed to exchange, process and/or store data for later processing, the bandwidth of the memory systems used for network cache is critical. Also the network objective is to support two active network links at each node, one for broadcast/multicast traffic and one for unicast traffic. (Multicast is implemented as broadcast with filtering in the NIC.) Heavy broadcast/multicast data equally impacts all nodes in a segment, even those not concerned with this information. A node must be capable of receiving bursts of unicast data on a second link and responding simultaneously with full bandwidth broadcast traffic. In addition, network cache must be accessible by one or more CPUs for reading the data directly as operands and potentially updating data directly in network cache as operand stores. Furthermore, network cache bandwidth must be sufficient to support DMA activity simultaneously. Finally, the onboard microprocessor also has direct access to all network cache; however, like the host, it has a private memory for most of its activity.

Because network cache can be used as file buffers in the host operating system, the SCSI DMA capability of the host computer can be used to move data directly between network cache and RAID storage. Consequently, the data passes over the PCI bus only once, never enters the host’s local memory, and, therefore, doesn’t interfere with host processors accessing their local memory.

For processing large contiguous data blocks, the data can be moved with the NIC’s DMA engine directly between network cache and the local processor memory. When a copy of the data is in local host memory the CPUs can access it more efficiently because it can be cached in the local processors. Therefore, the bandwidth of network cache is a key factor in this networking technology and the type of memory technology selected is critical to performance.

The bandwidth of network cache must be at least two times faster than the transmission rate to keep up with network traffic and preferably three times faster than the transmission rate. Current hardware is implemented using 32-bit SRAM. The real-time customer base drove this decision. SRAM is the most deterministic and has no refresh cycles. Current AmpNet hardware uses inexpensive SRAM with a 50 nsec cycle time, which yields an 85 MB/sec bandwidth. With more expensive SRAM a 25 nsec cycle time is possible, yielding a 170 MB/sec bandwidth. A 64-bit wide SRAM the memory can support a 340 MB/sec bandwidth. This memory architecture is more than adequate for all networks at 1.25 gigabits/sec, but it is not capable of supporting dual active paths at 2.5 gigabits/sec. There is also a significant cost factor, but SRAM CPU performance is needed for some classes of customers.

AmpNet products also include NICs that can use 64-bit SDRAM. The characteristics of SDRAM are quite different from SRAM. For SRAM all data can be randomly spread all over memory, and the performance is the same as a block transfer. However, for SDRAM individual random cycles and block transfers are quite different. The use of 10 nsec SDRAM yields 100 megacycles in an 8-byte wide memory or 800 MB/sec. However, 15% or more is lost to refresh cycles and transfer setup. Therefore, a realistic maximum is 650 MB/sec. A practical range is 500-650 MB/sec for block transfers. However, compared to 170 MB/sec for 32-bit SRAM this is a three to four times improvement, and gives the type of bandwidth that is required for supporting dual active paths at 2.5 gigabits/sec. But, there is a tradeoff when CPU operations are used for data transfer. For 32-bit random memory word operations the worst-case cycle time is approximately 50 nsec less refresh or about 58os per random word access as compared to 50 nsec for slow SRAM and 25 nsec for fast SRAM. This is the nature of CPU operand cycles and SRAM is a clear winner. However, SDRAM has its biggest advantage by providing one or more gigabytes of network cache at the same price of a few megabytes of SRAM.

The conclusion is that both SRAM and SDRAM are needed. For systems doing a lot of CPU direct access to distributed shared memory, SRAM is a better solution. Also note that SRAM works well for network speeds up to 1.25 gigabits/sec. For systems doing a lot of DMA transfers, SDRAM is a much better solution and is required for 2.5-gigabit networks and networks requiring large caches. It is important to note that network speeds beyond 2.5 gigabits/sec will require additional memory
bandwidth by building wider memory arrays and/or new memory technology. The focus of AmpNet is to build a working, highly available network at 1.25 gigabits/sec with SRAM, and then at 2.5 gigabits/sec using SDRAM.

6. Conclusions and Future Work

The AmpNet network has been designed and developed to support high-performance and highly available cluster computing. Network protocols at the data link layer guarantee that the network is reconfigured dynamically without loss of data when a link, hub, or switch is lost in a redundant path configuration. Higher-level layers can support application fail over using network semaphores and cache coherent distributed shared memory provided by the NIC. Higher-level AmpNet application programming interfaces provide libraries for developing fault tolerant applications. In addition, the programmable NIC, reliable low-latency messaging protocols, and field upgradeable firmware and soft logic provide a foundation for researchers who would like to develop additional services and protocols for network centric computing.

The final objective of AmpNet is to build a high performance, self-healing cluster with thousands of nodes. The first major step is an efficient mini-cluster with a maximum of 254 nodes. The design of the cluster includes provisions for building larger clusters using up to 254 mini-clusters for a total of 64,516 nodes.

AmpNet supports standard message passing libraries, including TCP/IP. Applications can be written using message passing interfaces such as MPI [7] or PVM [5], or applications can be written using the available AmpDC higher-level software tools. MPI currently executes using TCP/IP over the AmpNet IP driver. Efforts are planned to build an interface to the MPICH version of MPI that will execute directly over the low-latency AmpNet drivers for more efficiency. AmpNet is available for PCI interfaces and is fully supported on Windows 2000 and Linux platforms.

References