A Middleware-Level Parallel Transfer Technique over Multiple Network Interfaces

Nader Mohamed, Jameela Al-Jaroodi, Hong Jiang, and David Swanson

Presented By: Nader Mohamed
nmohamed@cse.unl.edu
Coming Up...

- Introduction
- Related Work
- Proposed Solution: MuniSocket
- Design and Implementation
- Message Partitioning
- Performance
- Concluding Remarks
Introduction (1)

- Cluster and grid infrastructures provide opportunities to accommodate compute and data intensive applications.
- Current configurations provide multiple network interfaces per node and multiple interconnections between nodes.
- Current network protocols, software, and sockets are designed for a single physical network interface.
- To increase bandwidth, we need protocols, and software services that seamlessly support multiple physical network interfaces and provide transparent and efficient utilization of these interfaces.
Introduction (2)

- Parallel data transfer over multiple network interfaces
- What can we achieve with parallel transfer?
  * Similar to parallel processing where parallelism benefits mostly medium to large size computational problems but not small problems, parallel transfer can be utilized to enhance the performance of medium and large messages. Similarly, just as the performance of a small computational problem can only be enhanced by upgrading the processor, the performance of transferring small messages can best be enhanced by upgrading the network.
Challenges!

- Heterogeneous resources and operating environments
  * Different operating environments (e.g. Windows, Linux, Unix, etc.)
  * Different network equipment (e.g. Fast Ethernet, Gigabit Ethernet, Myrinet, etc.)

- Some of the current approaches are too specific and inflexible
  * Channel bonding
Related Work (1)

- Two approaches to increase bandwidth and/or throughput
  1. Virtual TCP Connections
     - Multiple TCP Connections
     - Multiple Streams
     - Parallel Socket
  * Used in networks with high \( \text{delay} \times \text{bandwidth} \) product such as the Internet to avoid the limitations of the TCP window size.
  * Multiple threads are used at the socket or the transport layers.
Related Work (2)

1. Lower Level Striping
   - Example: Ethernet Channel Bonding in Linux
   - Provides good performance but
     - System dependent solution
     - All NICs on a node must have the same MAC address.
     - Separate switches required
     - Non-bonded nodes to bonded nodes communication is too slow
Characteristics of Proposed Model

- Middleware solution
- Located between applications and transport layer
- Can support heterogeneous environments
- Portable
Middleware Solution

Provides the following:

* Hides lower-level technical details
* Is portable and can support heterogeneous environments
* Can be used for local and wide area networks
* Can support single or multiple switched networks
TCP-Based MuniSocket

- Advantages of using TCP
  * Readily available for all operating systems and majority types of networks.
  * Part of the Internet infrastructure, thus, MuniSocket can be used on nodes connected over the Internet.
  * Provides efficient multiplexing among multiple logical communication streams, which is needed by any transport or middleware layer protocol.

- Disadvantage of using TCP
  * Has high overhead
Design and Implementation

Solution

Fragment large message into small fragments and send them in parallel through available network interfaces.

Implementation Issues

* Out-of-order arrival of fragments.
* Dynamic load on the networks.
* Varying latency and bandwidth parameters of the networks used.
* Utilization of existing resources such as multiple network interfaces
* Providing a reliable transfer
MuniSocket Architecture

Sender components

Receiving
Thread 1

Receiving
Thread 2

Receiver components

Sending
Thread 1

Sending
Thread 2

Connected by two physical networks.
Message Partitioning Techniques

Two message partitioning techniques

1. Equal Sizes
   - For dedicated networks or networks with similar constant workloads.
   - No extra header is needed.

2. Small Fragments
   - For networks with different workloads.
   - To achieve load balancing among networks.
   - Sequence numbers is added as a striping header.
   - A counter is needed at sender to provide fragment sequence numbers.
Experiments

All experiments executed on Sandhills, a 24 dual processor node cluster. Each node contains two AthlonMP 1.2GHz processors with 256KB cache per processor and a total of 1GB RAM per node. Two nodes equipped with two Fast Ethernet cards and two Gigabit Ethernet cards.

The experiments were designed to measure the round trip time (RTT) so that the effective bandwidth for the single network Socket and for MuniSocket, can be derived indirectly from the RTT as follows:

\[ \text{Bandwidth (Mbps)} = \frac{8 \times \text{Message size/10}^6}{\text{RTT/2}} \]
Equal Sizes Partitioning - Fast Ethernet

Over 90 percent efficiency for messages larger than 32KB and reaching up to 99.4 percent for a 2MB message
TCP 1000 is better than MuniSocket 2*1000 for small messages, but it gets better with messages larger than 32KB.
The proposed solution does not do extra data copies other than those made by TCP for both sender and receiver.

CPU utilization increases from 12.5% to 20% with Fast Ethernet.

CPU utilization increases from 37.8% to 48.4% with Gigabit Ethernet.

To reduce the overhead, light-weight interfaces can be plugged into MuniSocket. Example:

* GMSOCKS: Direct Socket Implementation on Myrinet
* SOVIA: User-level Socket Layer over Virtual Interface Architecture
* MyVIA: Design and Implementation of the High Performance Virtual Interface Architecture
* High Performance User Level Sockets over Gigabit Ethernet
Enhancing MuniSocket Performance for Equal Sizes Partitioning (1)

- **Problem**: The performance of sending small messages on single network is better than multiple networks

- **Solution**: Small messages should be sent on a single network while large messages should be sent through multiple networks

- **Solution Requirement**: We need to find a cut-off point between single and dual networks, where we can send messages of size $M$ on a single interface (when $M < \text{cut-off}$) or use two interfaces (when $M \geq \text{cut-off}$).
Enhancing MuniSocket Performance for Equal Sizes Partitioning (2)

- We need to find
  \[ \text{RTT}_1(\text{cut-off}) = \text{RTT}_2(\text{cut-off}) \]

- Cut-off point can be found at startup by:
  * Mathematical models such as curve fitting
  * Experimental search methods

![Graph showing RTT of TCP and MuniSocket on 100 and 1000Mbps Ethernet]
Load Balancing Solution

- The counter provides sequence number between 0 and $\left\lfloor \frac{M}{F} \right\rfloor - 1$. $M$ is message size and $F$ is fragment size.

- Each sending thread performs the following:
  * takes a fragment number, $i$, from the counter
  * Sends fragment $i$ followed by the data for fragment $i$.
  * A sending thread is blocked if it tries to place a new packet in the TCP send buffer when it is full due to packets yet to be delivered.

- Each receiving thread performs the following:
  * accepts the fragments
  * based on the sequence number, receive the data fragments to their proper place in the receiving buffer provided by the user.
MuniSocket Utilizes the flow and congestion controls of the transport protocol to achieve load balancing.

Load balancing is achieved by having threads connected to unloaded networks process more fragments while other threads connected to loaded networks are blocked for longer periods by the flow and congestion control mechanisms.
Load Balancing Performance

An average of 64.21 Mbps can be obtained using either network and standard socket. However, using enhanced MuniSocket, we can achieve an average of 118.95 Mbps using both interfaces simultaneously.
Fault Tolerance (1)

- The fragment status vectors (FSV) maintain information about the packets status.
- On the sender side, a fragment can be either ready, in-transit, or acknowledged.

One thread is sending the fragment

Ready → In-transit
Error in network

In-transit → Ready
Send completed successfully

Acknowledged
Fault Tolerance (2)

- Fault Tolerance is achieved by allowing any sender thread connected to a stable network to continue sending fragments while making any sender thread connected to an unstable or disconnected network monitor the network status.

- When a network connection fails, the associated sender thread marks the fragment in the state in-send as ready such that another sending thread can process the fragment. The result will be a continuing connection, but with a lower capacity.

- The sender thread (on a disconnected network) probes the interface to see if it is back online. When an interface is reconnected, the sender thread resumes normal operation and starts processing more fragments.
Conclusion

- MuniSocket utilizes existing multiple network interfaces between nodes
- The model provides expandability, high bandwidth, fault tolerance, and load balancing among the available interfaces
- Message fragmentation, transmission and reconstruction are performed in parallel and transparent from the user applications
- TCP protocol was used in MuniSocket, but other lightweight protocols can also be plugged in MuniSocket
- We are working on different optimization techniques for enhancing the middleware-level parallel transfer.
Please refer to
http://cse.unl.edu/~nmohamed
for other publications and more information