

Bipolar Disorder in Cluster Networking

Ron Minnich

Cluster Research Lab

Advanced Computing

Los Alamos National Laboratory

(www.acl.lanl.gov/cluster)

Presentation Courtesy MagicPoint (www.mew.org/mgp)

LAUR 01-5425

Overview

- Definition
- The Five Step Program
- The problem
- Possible fixes
- Conclusions

Bipolar Disorder in Cluster Networking

- We think about cluster interconnect two ways
- Reliable but slow
- Unreliable but fast
- Hence the bipolar disorder
- Of course you might think otherwise but
 - That means you haven't done the 5 step program

The Five Step Program for Recovering NIC Designers

- Idealism: This hardware eliminates all errors
- Optimism: This hardware eliminates all errors
 - you might encounter
- Disappointment: We have found that
 - On certain machines
 - On certain days
 - In certain circumstances
 - No data for you!
- Hope: V++ (hardware/software) will fix it

Realism: Software checksum and ARQ is really neat, huh?

Some examples ...

▷ Type Ether ATM H800 Myrinet Quadrics

Idealism 1981 1988 1994 1994 199?

Optimism 1982 1990 199? 199? 199?

Disappointment 1986 1991 1998 1999 2001

Realism 1988 1995 1999 1999 2001

□ Hope is always with us

Ethernet

- Idealism: we can run memory ops over this memory-bandwidth network
- Optimism: Who needs UDP checksums for NFS?
- Disappointment: Can't edit/ftp certain files
- Hope: Find and fix the hardware problems
- Realism: Edit kernel binary to turn on UDP checksums

ATM

- Switched, End-to-end CRC, flow control, QOS
- Idealism: Who needs IP?
- Optimism: Run IP but you'll never really use it
- Disappointment: It runs HOW SLOWLY?
- Hope: perpetual
- Realism: Who needs ATM?
 - TCP over SONET
 - Cells in Frame

HIPPI 800

- Switched, end-end CRC
- Idealism: Errors "can't happen"
- Optimism: Errors "won't happen to you"
- Disappointment: 1200 interfaces, 48 machines, 15 minutes
 - Every 8640000000000000 bits or so, something BAD happens
 - Actual failure is not clear since undetected errors are undetected
- Hope: H800++ (GSN) will fix it!
- Realism: Build software reliability into user library
 - ULM

Myrinet (LANAI 7)

- Switched, flow-controlled, end-end CRC
- Idealism: Just pump data through, it will work
- Disappointment: 128 interfaces, run full bore, NAS FFT won't finish
 - Data is getting corrupted and not detected
 - Actual failure is not clear since undetected errors are undetected
- Hope: Software fix will do it!
- Realism: We're porting reliable messaging to Myrinet

128 node ES45 cluster, Quadrics

- Switched, Flow Control, end-end CRC, built-in ARQ
- Idealism: Just pump data through, it will work
- Disappointment: 128 interfaces, run full bore, undetected errors
 - Main symptom is unexplained hangs (packet loss?)
 - Actual failure is not clear since undetected errors are ...
- Hope: The Elan 4 will be better
- Realism: We're porting reliable messaging to Quadrics

What's going on?

- In the ideal case the interfaces provide:
 - Flow Control
 - End-End CRCxx (CRC32 for Myrinet, CRC16 for Quadrics)
 - ARQ (request-resend) in hardware

- How can this not work?

- It DOES work

- Unless it fails

- And if it fails, how would you know?
 - You won't
 - Unless the applications writers tell you

Applications-based error checking

- Typical scenario:
- App fails
- Vendor says
 - "Can't be us. We're reliable"
 - Because your software doesn't detect errors
 - "Our Diags run to completion with no errors detected!"
 - Diags only find trivial errors
 - "You're doing something wrong"
 - But it's your own demo code!
 - "We haven't seen this"
 - Of course not, you don't even HAVE a cluster

How do you know? (cont)

- Programmer works hard and sooner or later finds
 - It's the network

- Programmer moves to TCP/IP
 - "It may be slow, but it's reliable"
 - This happened on our Rockhopper cluster
 - Many commercial Myrinet users run IP over Myrinet

- Programmer recreates IP reliability in user mode
 - This happened to us on our 48x128 CPU SGI cluster
 - This also happened on our Rockhopper cluster
 - It is happening on our ES45 cluster

Wait a minute!

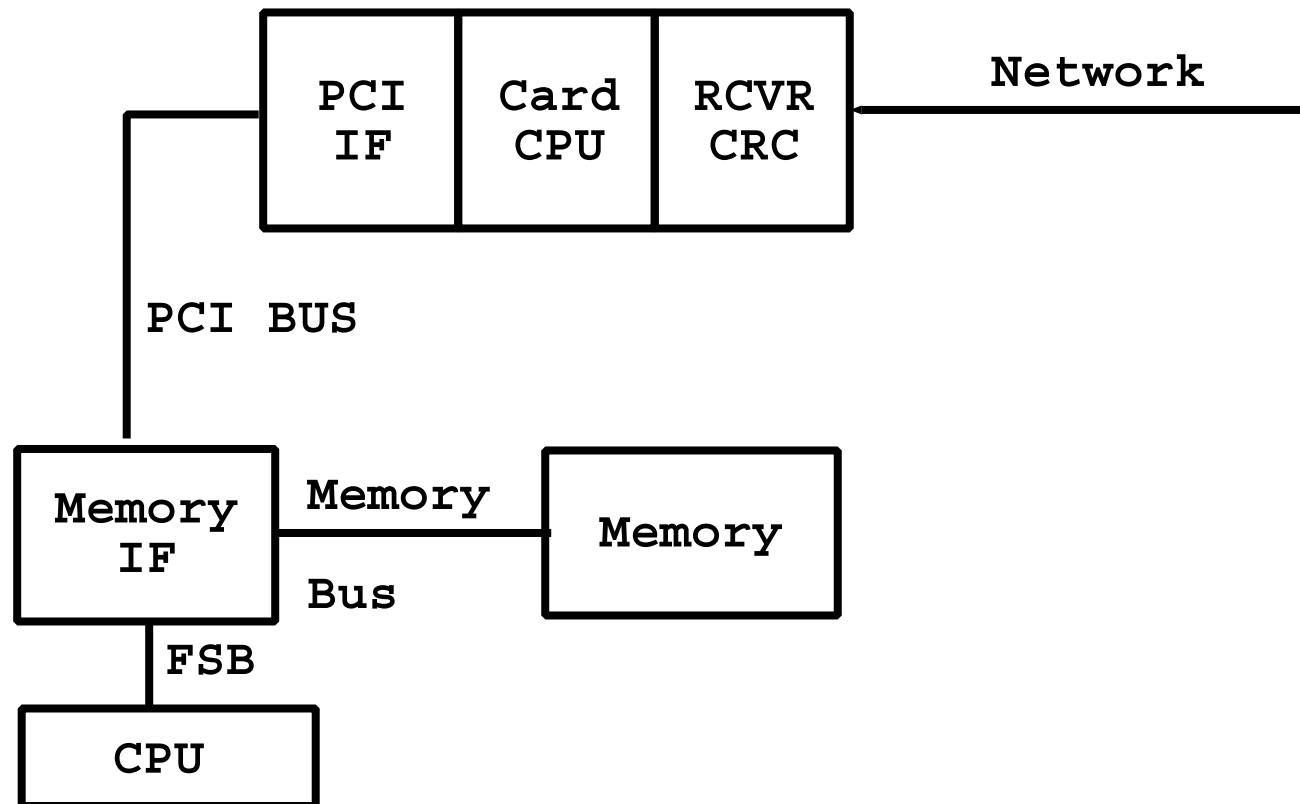
- The card-card checking is high-quality CRC
- Some cards have end-to-end flow control
- Some cards even have automatic request-resend
- Some cards have error counters to detect packet errors
- Again: how can these cards not work given built-in end-to-end protocols?

How can the cards not work?

- They can "not work" if the protocol is not end-to-end
- And it's not ... it's card to card!

Card-to-card Protocol

- CPU runs complex software
 - In fact the memory footprint is larger than V6 Unix
- Result: for these cards, think of PCI bus as a network
 - Unreliable packet network, in fact
- Protocol is hence NOT end to end



Failure modes

- Most frequently observed failure is not corruption
- It is a NIC failure that causes loss of data
 - Blocks of Zeros
 - Missing Word (PCI Abort bug)
 - Lost packets (even on cards with ARQ)
- Caused by
 - Resource exhaustion on the NIC
 - Race conditions
 - NIC vs. CPU "Version Skew"
 - Programming errors
 - PCI problems
- All these are "Can't happen" errors
 - No need to detect "Can't happen"
 - So undetected errors are undetected

What do we do?

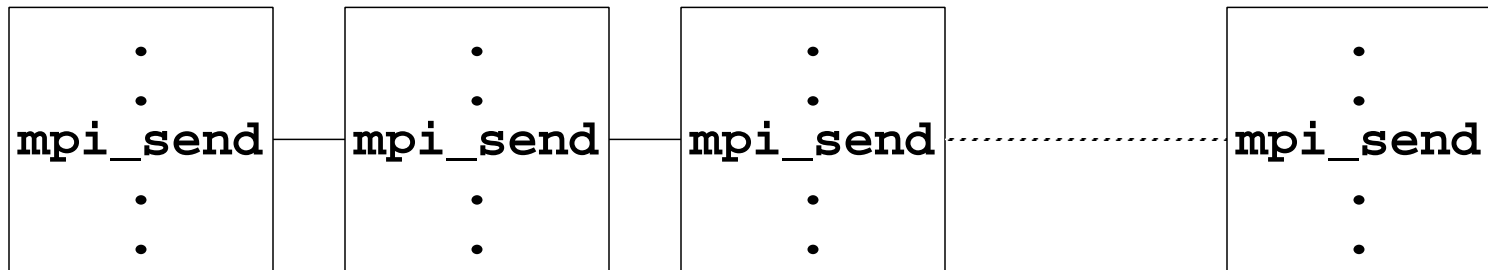
- Run TCP/IP for everything
 - A frequently-used solution
- Recreate TCP in user mode?
 - Frequently used in HPC

Why do we use these interfaces?

- We need bandwidth
 - And will need more
 - Problem data sets only grow larger as time goes by
 - IPC increases as problem sizes grow
- We need low latency
 - Or do we?

Do we really need low latency?

- Is it inherent to applications
- Or is it an artifact of MPP programming style?
- Lock step SPMD
- Requires low latency, scheduling tricks
- Doesn't tolerate node outage at all



What if we could relax the latency requirement?

- Asynchronous rather than synchronous communication
 - Allow latency to go back up to ~50 microseconds
 - Find ways to use "Task bags" for traditional applications
 - Paul Woodward is experimenting with weather model
 - Can tolerate very high latency (milliseconds) for tasks

- What if we go to a hierarchical latency model?
 - Low latency "in group"
 - Higher latency "between groups"
 - Analogous to using "Fat SMPs" with fast interconnect

If we can relax the global latency constraint

- We could focus on bandwidth and worry less about latency
- Could try using TCP-like protocols optimized for small area
 - e.g. Plan 9 IL
- Once we remove global latency requirements
 - We can use kernel-level protocols for fast, reliable comms
 - We can eliminate unreliable, bug-prone user-level "TCP"
 - We can build simple, reliable, high quality NICs

Conclusion

- I only promised questions and a challenge
- The quest for low latency is producing
 - complex
 - high cost
 - unreliable
 - low quality
- NICS
- Need to develop latency-tolerant programming models for clusters
 - Also requires new kernel interfaces (a la CLIC)
- Result: we can use simpler, commodity NICS