DATACUBE
A Scaleable, Fault Tolerant Data Server

Noah Mendelsohn
Lotus Development / IBM
10/27/2000
Agenda

- Project History
- Goals and Hardware Overview
- Software Overview
- Fault Tolerance
- Conclusions
Project History
History

- **Milestones:**
  - 1986?: Project initiated
  - 1990: Hardware/software simulator
  - 1900-1992: Hardware/software prototype operational
  - July 1992: IBM Cambridge & LA Scientific Centers close, project ends

- **Publications**
  - ASPLOS work (not) in progress talk, fall 1992
  - Several patents

- **Most details of system remain unpublished**
(All participants were regular, part time or contract employees of IBM during their work on the Datacube project.)
Goals and Hardware Overview
Project Goals

- Investigate massively parallel *business system* architectures
- Strong focus on fault tolerance
- Investigate design & performance of required software
- Scaleable, fault tolerant, continuously available, hardware interconnect
- Focus on realistic maintenance and deployment issues
A DATACUBE NODE
The DATACUBE Parallel Data Server

A Datacube Node

Datacube Server
Datacube System Overview

- Message passing MIMD computer (shared nothing) each node has:
  - Inexpensive processor
  - RAM
  - Disk
  - Switch
  - NVRAM (optional)
  - LAN attach (optional)

- Fault tolerant, adaptive, 4-D torus, distributed switch

- All elements of system scale together
Switch Hardware

- 4 dimensional Taurus
- Distributed routing hardware (on nodes)
- Adaptive real-time path search in hardware
- 3.6 Mbyte/sec/node full duplex, approx 60 usec latency (Xilinx prototype)
- 10x improvement projected for inexpensive single chip VLSI

Remember, this was ~1988
Software Overview
The Datacube Prototype:
Software Features

- Unix kernel-based prototype
- Communications
  - Communication/disk buffer integration: zero copy disk cache update & access
  - IP packet switching
- RAID-1 (mirror) and RAID-3&5 virtual disk
  - Appears as large, common disk at all nodes
  - Optimized for 1:1 interleave...adaptive RAID 5/RAID 3
  - Faults hidden from surviving nodes
  - Distributed caching
- Distributed Unix filesystem
- Scaleable distributed reconfiguration algorithms
Datacube Software

User Program
Unix Filesystem
NFS
Logical Cache
Distributed Filesystem Coherence Enforcement

Big, shared, Fault-tolerant, Virtual Disk
Datacube Software

Diagram illustrating the interaction between various components in a distributed file system environment:

- User Program
- NFS
- Unix Filesystem
- Logical Cache
- Physical Cache
- Distributed Filesystem Coherence Enforcement
- Distributed RAID-5 Management
- Node 1
- Node N

Key features:
- Fault Tolerance
- Communication
- Performance Monitoring
Fault Tolerance
Fault tolerance model

- **Hardware**
  - Hot pluggable nodes, redundant power, etc.
  - Passive backplane (power, ground, torus wiring)
  - Hardware provides fault tolerant message routing
  - Failstop on all errors

- **Software**
  - Nodes fail and are replaced by warm standby spares
  - Distributed reconfiguration algorithms
  - Raid (1,3,5) reconstruction of disk, nvram
Reconfiguration software

- Simulates stable virtual node space
  - Spares replace failed nodes, routing tables updated
  - Nodes appear to pause for ~2 seconds on failure
  - Performance degraded during RAID reconstruction, filesystem token resync, etc.

- Anticipates realistic failure statistics
  (almost any 2 nodes at a time)

- Correctly rejects old nodes that reappear
  including after reboot

- Distributed algorithm simulated on
  thousands of nodes, tested on hardware
Dynamic node replacement

Low spare

Next lowest spare

High (replaces spare node 12)
Performance
Performance tools

- Real time displays of low level software instrumentation
- Logging of same
- Kernel event tracing...post-facto clock correlation reproduces virtual time (causality) in face of local clock drift
- Complete software emulator for switch...software stack run on emulator
- Analytical models
Realtime performance monitor

DATA CUBE PERFORMANCE DETAILS

Per Node Statistics

- CPU
- DISK BLOCKS
- SWITCH RCV
- TOKEN RING BYTES/SEC
- % DISK ACTIVE TIME
- SWITCH RCV
Software Performance

- **Message send:**
  - 2250 instruction times for full kernel to kernel RPC round-trip (1500 usec at 1.5 Mip, incl. buffer allocation, queueing, interrupts, etc.)

- **Parallel Filesystem (4K byte block size):**
  - Non cached/sequential access: 630 KBytes/sec/drive = 156 blocks/sec (drive & controller limited, same as single node system)
  - Non caching/random access: 130 Kbytes/sec/drive = 42.5 blocks/sec (drive limited, same as single node system)
  - Cache hits through filesystem & switch: 3.2 Mbyte/sec/filesys-node 800 blocks/sec (cpu limited - 89% of node's switch bandwidth!)
Model: Msg. rate vs. msg. size

Switch Speed | Driver Latency | Switch Latency | Max CPU in Switch Driver
---|---|---|---
Prototype | 3.6 MB/sec | 750 usec | 50 usec | 25 %
Product | 10 MB/sec | 375 usec | 26 usec | 20 %

Msgs per second = 1 / \( \text{MAX(Driver Latency/Max CPU, Switch Latency + Message size/Switch Speed)} \)

Bytes per second = Messages per second \( \times \) Message Size
Model: Bytes/sec vs. msg. size

Prototypes and Product

<table>
<thead>
<tr>
<th></th>
<th>Switch Speed</th>
<th>Driver Latency</th>
<th>Switch Latency</th>
<th>Max CPU in Switch Driver</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prototype</td>
<td>3.6 MB/sec</td>
<td>750 usec</td>
<td>50 usec</td>
<td>25 %</td>
</tr>
<tr>
<td>Product</td>
<td>10 MB/sec</td>
<td>375 usec</td>
<td>26 usec</td>
<td>20 %</td>
</tr>
</tbody>
</table>

msgs per second = \(\frac{1}{\text{MAX}(\text{Driver Latency/Max CPU, Switch Latency + Message size/Switch Speed})}

Bytes per second = Messages per second \(\times\) Message Size
Conclusions

- **Datacube Successes**
  - The Datacube model of fault tolerance has attractive features
  - Specialized hardware/software integrating message passing with disk cache is very effective
  - Datacube style hardware is very easy to engineer and implement
  - Datacube is both scalable and economical

- **Datacube Disadvantages**
  - Software is difficult to scale--programming these machines is difficult!
  - Assumption of uniform nodes is unrealistic
  - Specialized architecture--difficult to share hardware and software with general purpose machines
Controversial Ideas!

- Massively parallel systems must be fault tolerant
- We need software tools for parallel system development (you can't write filesystems in FORTRAN-D!)
- Designing message switch interfaces involves the same kind of hardware/software tradeoffs as designing instructions sets