Hive: Fault Containment for Shared-Memory Multiprocessors.

Chapin95: John Chapin, Mendel Rosenblum, Scott Devine, Tirthankar Lahiri, Dan Teodosiu, Anoop Gupta, ACM Symp. on Operating Systems Principles, 1995.

Failures in Large Systems

- As systems get larger, P(failure) grows
- So?
  - Failure containment: limit effect
    - This paper (HW or software partitioning)
  - Failure masking/tolerance: keep going
    - Checkpoint/restart - coarse or fine-grained
    - Paxos, BFT, {restarts, Rinard, etc.}
    - Tandem / 3 voters (NASA)
    - Multicore approaches
      - Suspending cores (heat, too!)
      - Log-based architectures

Why failures?

- Software (bugs, etc.)
  - App
  - OS
- Hardware
  - Overheating
  - Particle strikes cause memory bit-flips/transient errors
  - Wear-out
  - Residual error rate
- Human...

Component Reliability

- Why not just make {CPUs, mem, etc.} more reliable??
- Tension between perf/density and reliability
  - Consider resistance to cosmic rays
    - Space-hardened procs many generations back (bigger paths / higher voltage -> effect of one particle strike smaller, doesn't change signal)
      - Modern flash mem uses ~20 electrons/bit. That's not a lot of high-energy particles to totally change value.
    - Lead is expensive and heavy...
    - Mem can spend ECC to do so (~10% more mem + ECC circuitry)

Reliability already barrier

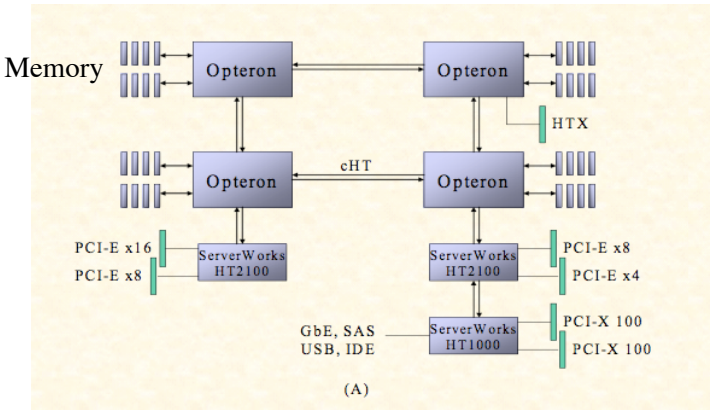
- What determines "rated" speed of CPU?
  - Process variation & marketing
    - If it can't run stably at 3Ghz, try it at 2.5Ghz and sell it there...
- AMD now doing this with # cores
  - tri-core, though main reason is marketing
  - Process variations and yield errors create enough room for this to help
  - if 3 cores run @ 3Ghz but one runs @ 2.4 -> tri core 3.0 vs quad-core 2.4... (or 2-core...)

This paper

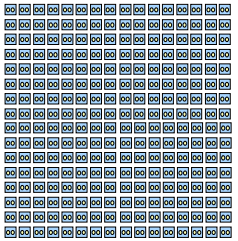
- Fault containment in shared-mem multiprocessors
- But, you say, we're moving to more clustered systems with high-speed interconnects!
- Well, yes, kind of. But what do we interconnect?
  - Cray XT5 blades are dual proc quad-core opterons
  - Cray XMT is a shared-mem MPP
    - 8000 CPUs, 64TB shared memory (CPUs are non-standard)
    - Still blades - but provides NUMA

Modern Server (IBM x3755)

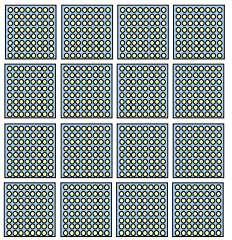
- Everything is multi-chip, multi-core, multi-mem
  - when core SW fails, it is likely to "wild write"



LANL systems and data



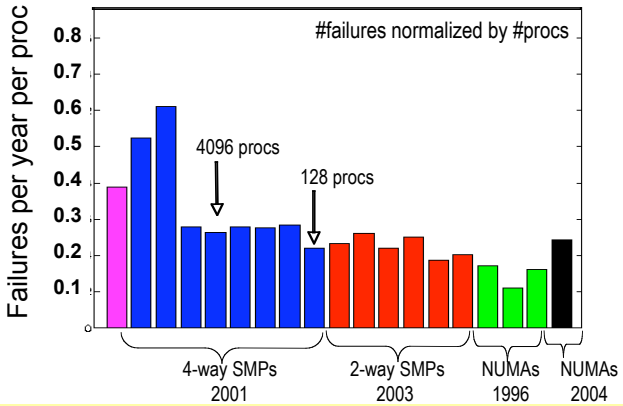
Clusters of 2/4-way **SMPs**
- **commodity components**
- 100s to 1000s of nodes.



Clusters of **NUMAs**
- **128-256 procs per node**
- 10s of nodes.

The failure data:
- Records of all failures leading to **node outage.**
- **23,000 records** over 9 years and 22 systems.

How does failure rate vary across systems?



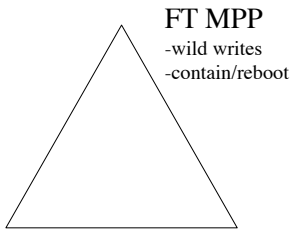
- Normalized failure rates similar, despite size differences => Failure rate grows ~linearly with system size.
- Similar even across systems across different type & age.

Fault Containment

- Resource management requirement:
  - A system provides fault containment if the probability that an application fails is proportional to the amount of resources used by that application, not to the total amount of resources in the system
- Whole-system procs see no benefit
  - But many (most?) uses of supercomputers are small
    - testing, development, small jobs
  - Rare but important ones are huge

FT@High Perf: Multicell Kernel

- MPP OS as distributed system
  - MPP (massively parallel processors) is more code/procs/complexity
  - Transactions & replication assumed to be too much overhead for perf goals
  - Increase node "independence"
    - Separate OS per node, minimal sharing
  - Model shared memory faults explicitly
    - True parallel codes in fact have higher failure rates, use checkpoint/restart
- Effect of failure
  - Wild writes (prevent)
    - 15-25% software faults: wild writes (IBM)
  - Access to memory in cell



Transform to Distributed System

- Each node runs "independent cell"
  - Contain failure to one cell, "mostly" not sharing resources with other cell
  - But shared memory allows cell to "wild write" (limited Byzantine)?
  - Kernel uses RPC not shared memory
- Hive OS for FLASH CC-NUMA
  - SW & HW (cache coherency) specialization
  - HW firewall for wild writes (node specific ACL per page)
  - HW messaging for fast RPC
  - Discard writeable pages on fault, making fa detection faster

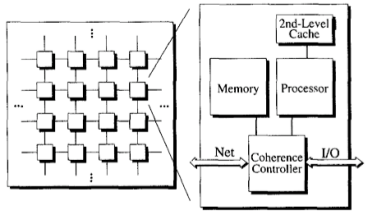
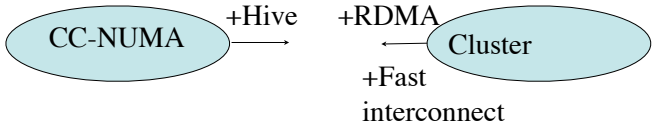
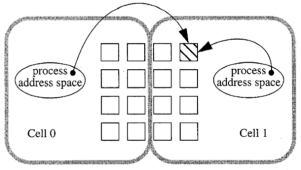


FIGURE 2.1. FLASH architecture.
The machine is structured as a set of nodes in a mesh network. Each node contains a portion of the main memory and a coherence controller which communicates with other nodes to maintain cache coherence. When a hardware fault occurs, the node is a likely unit of failure, so portions of main memory can be lost.

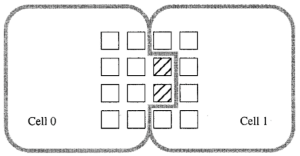


Fault Containment

- Fault containment in Hive:
  - Contained if prob of app failure proportional to resources used, not total resources
  - Large apps, using all machine, get no benefit
  - Traditional message/RPC verification in OS, but sped up with HW messaging
  - Focus on shared memory reading & writing
    - Debergalis like -- remote read allowed, and verified like RPC, and remote write not done in kernel (is done at user level)
  - Problem is failstop/fast -- use heuristics like probing, timeouts, bus errors
  - User level resource allocation is global shared memory app w/ microreboot recovery
  - File cache sharing & memory pressure balancing expands containment below app



(a) Logical-level sharing of data pages



(b) Physical-level sharing of page frames

Remote memory access control

- Cells protect all private pages from all others & blocks most cells from shared pages
  - 4 KB granulatity, 64 bit vector for 64 nodes (groups of nodes)
    - Page granularity b/c OS limitations; +4-6% remote write $ miss latency
  - Only needed if TLB mapping exists, but TLB misses are common, so share protection for all nodes of cell if any process needs mapping
- Declaring a cell failed requires synch w/ other cells
  - Non-Byzantine (Paxos/PTP) agreement to reboot/exclude cell
  - Double barrier: 1st complete remote ops, 2nd clean VM then resume
- Clean VM means pages writable by failed cell locked out
  - Flush all TLBs before 1st barrier so each reference tests appropriate use (allow stale file read after dirty block discard)
  - If failstop, this would be complete, but … changes might propogate
  - Aggressive testing of cells w/ heuristics is substitute for transactions
    - Performance vs containment tradeoffs

Eval

- Serious investment
  - Building HW (Flash)
  - Built cycle accurate simulator (SimOS)
  - Use real OS (IRIX)
  - Run "real" apps
- But hard to test scalability
  - 4 nodes is not a big system
- But hard to test reliability
  - Limited faults injected
  - Fault detection oracle leaves failfast assertion at risk

Workload	IRIX 5.2 time (sec)	Slowdowns on Hive		
		1 cell 4 CPUs/cell	2 cells 2 CPUs/cell	4 cells 1 CPU/cell
ocean	6.07	1 %	1 %	–1 %
raytrace	4.35	0 %	0 %	1 %
pmake	5.77	1 %	10 %	11 %

TABLE 7.2. Workload timings on a four-processor machine. As expected, the partition into cells has little effect on the performance of parallel scientific applications. It has a larger effect on compilation, which uses operating system services intensively.

Injected fault type and workload (P = *pmake*, R = *raytrace*)		# tests	Latency until last cell enters recovery(mscc)	
			Avg	Max
Node failure:				
during process creation	P	20	16	21
during copy-on-write search	R	9	10	11
at random time	P	20	21	45
Corrupt pointer in:				
process address map	P	8	38	65
copy-on-write tree	R	12	401	760

TABLE 7.4. Fault injection test results. The tests used a four processor system booted with four cells. In all tests Hive successfully contained the effects of the fault to the cell in which it was injected.

Impact

- Long term impact: Vmware
  - SimOS: fast accurate HW sim w/ small changes in real OS
  - VMware: SW fault containment for commercial OSs

VMWare aside

- SimOS/Embra (Witchel/Rosenblum)
  - Dynamic binary translation
    - Orig code block -> Native code block that does same thing
      - A lot like JIT compiling in Java
      - Theirs: could translate to specific architectures preserving features
      - But really important part was...
- VMWare (Rosenblum et al.)
  - Dynamic binary translation...
  - x86 -> x86 with important things changed (protected access, etc.)
  - Virtual devices, all the bells & whistles

VMWare server consolidation

- Popular trend
  - Multiple VMs on single hardware
    - Can run multiple "independent" machines together
      - Faults, security, etc. isolated
      - But don't have to buy the H/W for it
    - Eases provisioning
      - Can move VMs around to meet demand
      - If needed, can move to dedicated server
    - Not a panacea (perf cost, still have complexity if run multiple OS, etc), but very useful & hot topic today

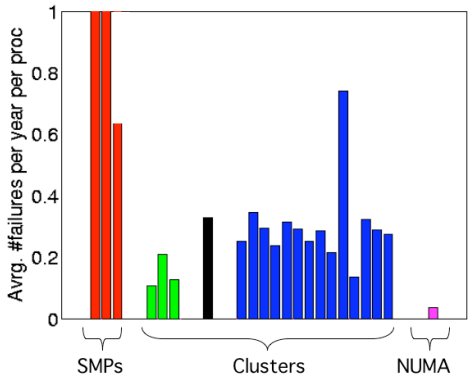
Mondriaan Memory Protection

- Mondrix (Witchel, Ree, Asanovic - SOSP'05)
- Fine-grained memory protection in one OS
  - Didn't target large SMP, but could
  - Provides word-level protection. What kind of prot?
    - no perm, r/o, r/w, execute-read
    - Simple? r/o program text, r/o data, r/w data, stack
    - Better: each malloc object mapped r/w, surrounding words inaccessable (malloc state - 4-8 bytes of header before each allocated block)

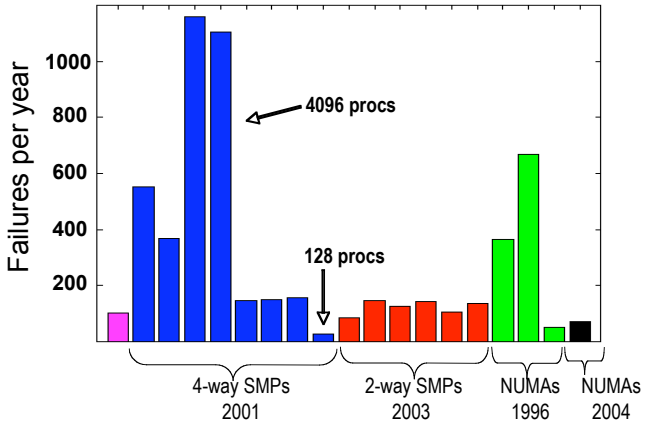
Why MMP?

- Prevent & detect accidental corruption (at hardware speeds, no efence/valgrind overhead)
- Buffer overrun protection (partial) -- protected region at end of buffer, yell if accessed
- Arbitrary data watchpoints (debugging - tell me if word X is modified)
- Generational GC - tell me if object is modified
- Flexible DSM

Failure rate (normalized by **size** of system)



What do failure rates look like?



- Large variability -- even within systems of same HW type.