Multiprocessors and Miscellaneous

<u>Memory Coherence</u>

- create shared virtual memory on loosely-coupled multiprocessors
- why?
    - Easier to program than msg passing
    - Commodity parts
    - More scalable than bus-based SMP
    - Migration
- VM managers handle coherence in this system:

  Centralized
    - Single manager
    - Manager keeps who owner is
    - Owner node keeps the copy set and lock, and basically synchronizes all accesses to that page
    - Still a bottleneck because all requests must go to centralized manager to find out who owner is

  Fixed Distributed Mgr
    - addrspace split up among processors
    - each node manages its set of pages
    - hard to divide up addrspace correctly to properly load balance for diff. apps.

  Broadcast Distributed Mgr
    - owner manages the pages it has
    - all requests/inv./etc. broadcast
    - broadcast means it doesn't scale well

  Dynamic Distributed Mgr
    - everybody keeps track of all pages' probOwner
    - no fixed owner or manager
    - keep track of ownership of all pages in each processor's local page table
    - probOwner field checked/updated on every fault
    - on request, if you forward then you set your probOwner to the requester. This collapses the chain.
    - Algorithm does not degrade based on the number of processors in the system but on number of processors contending for a page.
    - It's possible to broadcast every M page faults to collapse chain. (this isn't that important.)
    - Sends fewer forwarding messages than other algorithms.
    - Can also make copy set into a tree:
        - Writes propagate in parallel down tree
        - For reads, you only have to find one node that has data. (you get added to copy set of that node instead of owner.)
- not good for fine-grained write sharing

<u>Hive</u>

- shared memory multiprocessor (FLASH)
- fault containment
- resource sharing (memory)

Facts:

- each cell has independent copy of kernel running on it
  - less sharing of kernel resources (scalability)
  - kernel only has control of local HW (reliability)

Defn:
Fault containment – system provides if probability that application fails is proportional to amount of resources used by that application not to the total amount of resources in system

HW: FLASH network remains fully connected with high reliability
Each memory page has HW permission bit vector
Cache coherence controller checks permission attached to each memory block before modifying memory (not too much overhead).

Ways in which a fault on one cell can damage another cell:
1.) send bad msg
2.) provide bad data or errors to remote reads
3.) cause erroneous writes

How they handle:
1.) sanity checks all RPC info from other cells
timeout while waiting for replies
2.) careful references (you can read any page on another cell and write any user-level page on another cell)
3.) user determines which page to protect with firewall
once a cell failure is detected, all pages writable by that cell are discarded

Failure Detection:

- heuristic checks every once in a while
- timeouts
- attempt to write to another node's memory causes bus error
- if you read data from another and fails consistency check

All of these are perceived as failure and cause consensus algorithm to run.

On failure detection:

- 2 phase barrier
- all TLB entries and remote mappings flushed

Resource Sharing (pages)

- logical sharing – pfdat points to page on another cell. That other cell manages the page.

- Physical sharing – pfdat points to physical location on other cell, but you manage the page.
    - You only get your page back (to manage) when borrower frees or failure detected.
    - Permission bits managed by data home but only writable by memory home.
- User-level process called Wax handles global resource allocation
    - Runs on multiple cells (multi-threaded) and synchronizes
    - Whenever failure, dies and is reincarnated.

<u>Munin</u>

- distributed memory coherence managed on page level
- manage objects based on consistency model they actually need
- in particular, they talk about release consistency
    - distinguish between acquires and releases
    - only need to make updates seen by all processors at release or next acquire on that synch variable
        - ⇨ delay updates. Therefore, less communication and fewer runs of coherence protocols
- make synchronization obvious
- specify how data is used/sharing. This determines coherence protocol used.
- Adds a layer to system in order to do all of this
- They say this is easier than msg passing for programmer and performance is similar.

<u>Trace Driven</u>

- good scheduling method must be preemptive and must prevent any job from capturing CPU for too long a period of time
- trace driven modeling becomes less costly than "live" experiments. Also allows you to disregard a lot of other variables and examine the portion of system you're interested in.

<u>General Hint Papers</u>

- keep it simple
- make the normal case fast and special case everything else
- keep it small and understandable; complexity undermines reliability
- use a good idea again. Don't generalize
- caching is good
- every syntactically correct program with error should give predictable error message. (assertion programming is a good thing.)
- provide fast mechanisms not powerful constructs
    - people can use them as needed
- keep interface stable and hide implementation
- make it correct
- end to end