Case Studies of Cache Coherence

<announcements>

Ivy, AFS/Web, Coda, Bayou

Why is Ivy cool?
All the advantages of *very* expensive parallel hardware.
On cheap network of workstations.
No h/w modifications required! Software runs as is!

Do we want a single address space? Or have programmer understand remote references? Shall we make a fixed partition of address space?
I.e. first megabyte on host 0, second megabyte on host 1, &c?
And send all reads/writes to the correct host?

Ivy: We can detect reads and writes using VM hardware.
I.e. read- or write-protect remote pages.

What if we didn't do a good job of placing the pages on hosts?
Maybe we cannot predict which hosts will use which pages?

Could move the page each time it is used.
When I read or write, I find current owner, and I take the page.
So need a more dynamic way to find current location of the page.
What if lots of people read a page?

Move page for writing, but allow read-only copies.
When I write a page, I invalidate r/o cached copies.
When I read a non-cached page, I find most recent writer.
Works if pages are r/o and shared, or r/w by one host.

Invariants:
1. every page has exactly one current owner
2. current owner has a copy of the page
3. if mapped r/w by owner, no other copies
4. if mapped r/o by owner, maybe identical other r/o copies
5. manager knows about all copies

How well does this work?

Instruction latency

1st level cache hit: 1 cycle
DRAM: 100-200 cycles

Remote DRAM: 100K-1M cycles
(1Gb Ethernet => 40 usec for a 4KB page)

False sharing

What happens if two shared variables are both on the same page, concurrently written by different processors?

Will ping page back and forth multiple times.

Partial solution: release consistency

Needless communication:

Example: Parallel mesh computation, where each row of the array is one page
What will be sent? Result is that the computation runs at the speed of the network

Solution: perhaps send only the differences?

Common theme in systems: how much do you sacrifice in performance for using a particular abstraction? If not much, great! Exponentially better hardware will fix the problem soon. If quite a lot, abstraction is probably not going to work. (What is "quite a lot"?)

Another theme: ideas from one branch of CS get used in other branches. Ideas in Ivy and successor systems, are now common in multicore designs.

AFS/Web:

AFS: local and wide area distributed file system, with cache coherence.

Client gets copy on file open; modifies file; write through on close.

In early 90's, commonly used for distributed development projects.

Web provided distributed file system, but without cache coherence or security.

So: why did the web win?

<speculate?> Browser!

Easy system administration: no centralized user accounts

Extension model. URL could trigger a program at the remote end, and not just a file read. Meant you could integrate databases and other software as back end to a web site.

Coda: Followup to AFS, to deal with disconnected operation. At time paper was written, laptops were in infancy, no wifi. We are quickly moving now to a wireless, but connected world.

Coda solution:

a) prefetch things you will need in future
b) allow others to change files while you are disconnected (no exclusive access)
c) log all changes to local disk – replay log preserves order of writes
d) transactional updates to metadata, to simplify local recovery
e) apply log at server when connected
f) conflict resolution

Coda conflict resolution policy: changes appear in processor order, applied at time when notebook reconnects.

Is that sequentially consistent? Linearizable? Does it matter?

Could we make Coda sequentially consistent? Suppose we kept the replay log at the server, and integrated the various replay logs together, to decide on the order? (This is how Bayou does it.)

A few questions:

What happens on directory updates to different files by different users? Can that be resolved automatically? What about different updates to different parts of the same file by different users?

How would you implement google docs or windows live today?

Assume perfect connectivity? What if we wanted to allow offline work?

Probably: you would keep a local copy of the entire directory, to ensure all the files are available. Would an explicit check in/check out model work better?

Example: email sync: over a slow link, this can take minutes.

Bayou: p2p disconnected operation, application-level conflict resolution

Idea is that you should be able to sync a set of portables, without access to a server.

Exchange updates with neighbor; not committed until everyone sees it (and you know that no other earlier updates can occur).

Uses replay logs, but replay logs are preserved.

Conflict resolution is programmed – each write is provided some code to run when a conflict occurs, e.g., for a calendar program, what to do if someone else booked the room while you were disconnected.