Advanced Topics in Computer Systems, CS262B

Prof. Eric A. Brewer

The Case for NOW

January 29, 2004

I. Background

Huge effort in parallel computing throughout early 90's. It was the "P2P" of its time, with new conferences, lots of top researchers, and lots of funding...

Key goals:

- scalability, raw FLOPS (how many CPUs)
- network bandwidth
- programming languages (parallel fortran)
- shared memory models

But some big problems:

- workstations had much better single CPU performance -- mostly due to 18 month lag in usage of a particular processor
- very high engineering cost spread over relatively few machines (sold) => high cost per CPU for these machines to amortize the R&D
- awkward development environments
- custom OS was also behind in features/reliability
- hard to upgrade
- somewhat less reliable than workstations
- hard to program even with some language help
- various PhDs on all of these topics...

Only real advantage of the large parallel machine: backplane bandwidth

- solution: create a cluster network that has similar bandwidth (if not latency)
- this was eventually fixed by AM work at Berkeley and the U-Net work at Cornell (and also partially in the VI interface for Windows).

Clusters:

- key idea: have to reuse the whole box, not just the CPU
- implies: better performance, much better cost, latest OS and tools
- challenges: even harder to program, network still not as good
- extra challenge: can you make use of idle workstations? (more on this later)
- huge amount of aggregate disk I/O (and seeks)
- software RAID (rather than hardware) -- this took many years to really work

Big Issues:

- how do you program a cluster?
- how do you deal with partial failure? (you how potentially have all of the problems of distributed systems!)
- how to get a global system view: scheduling, file systems (easy), shared caching, namespaces?

Killer App?

- turned out to be web servers, led by Inktomi work in particular
- Advantages for web servers: incremental scalability, fault tolerance, cost
- Eventually worked well for traditional science applications as well, but mostly for those without the need for fine-grain cluster-wide sharing. E.g. rendering works great, but sparse matrix apps are much harder

Things that didn't work out as planned:

- no real use of idle workstations -- there are some counterexamples, mostly in graphics and simulation (and SETI !). Machine cost is minor compared to other costs and complexities.
- Winner wasn't workstation vendors: really PC vendors plus Linux (see Beowulf project for example)
- network RAM has never really made it big. Possibly due to security?
- not much use of software fault isolation for the GLUnix layer. More a traditional layer under processes.
- security? in practice it is provided by physical security for big clusters (not by software on users' desktops)