Allocating Workload & Consistency

Allocating Workload **models:**

- 1. processor pool
  - processor -> process p for p's lifetime
  - sharing at *process granularity*
  - e.g. C compilation, multiple modules
  - user workstation maybe just a user interface (eg X)

Allocating Workload **models:**

- 2. NOW (Shoja's Martlet; Paterson's NOW)
  - steal cycles from idle workstations
  - aided by *process migration*

    (when the owner of the ws logs in!)

Allocating Workload **models:**

- **3. Shared Mp multiprocessors . . .**
  - each cpu has private cache and (possibly) private Mp
  - all share a single *shared Mp* in which programs and data are resident
  - shared Mp can be used to implement (emulate) message passing
  - popular for servers, O(10-100 cpus)

Allocating Workload **models:**

and
Maintaining them. . .

Kinds of Consistency and Maintaining them. . .

Update consistency

Update consistency

- means that a series of transactions on a single data item should not interact
- the effect of each should be independent

  of the others

- sufficient condition:
  - each should be *atomic* :

Update consistency

- each should be *atomic* :
  - 1] all of it is done or none of it is done
  - 2] the state change should be as though the transaction was *instantaneous*
  -

Replication Consistency:

- databases are often not monolithic or *partitioned* but *replicated*
- changes to one copy of the data must be "quickly " reflected in all copies
- a sequence of changes (updates) must be passed against all copies in the same time sequence (Lamport)

Cache Consistency

- Cache: when a client receives data from a server

  it may keep its copy around

  in case it needs it again soon.

- such data is *cached* and the store is a *cache*.

- origin: hardware cache for instructions,
  - interposed between Mp and cpu.

Cache Consistency problem:

- when the original data is changed in the server
  how to ensure the cache copy changes too?

- in a cpu with one Mp, *writethrough* techniques

- in a distributed system with n clients of the data server, where n varies continually and unpredictably
  NOT CLEAR!

Cache Consistency problem:

- why bother?
- 1000:1 speedups are common

Failure consistency:

- consistent recovery of all processes
  from failure of one process or processor

- requires checkpoint/restart techniques

Clock consistency

- Consistent view of time, or at least of temporal sequences (A happened before B)
- there is no common hardware clock
- Lamport, Fidge, . . .

and see . . .

- functionality (emulate unix)

- QOS
  - performance
  - availability/reliability
  - security

- reconfigurability (short & long term)