Workstation Clusters

- replace big mainframe machines with a group of small cheap machines
- get performance of big machines on the cost-curve of small machines
- technical challenges
  - meeting the performance goal
  - providing single system image

Supporting Trends

- economics
  - consumer market in PCs leads to economies of scale and fierce competition among suppliers
    - » result: lower cost
  - Gordon Bell's rule of thumb: double manufacturing volume, cut cost by 10%
- technology
  - PCs are big enough to do interesting things
  - networks have gotten really fast

Models

- machines on desks
  - pool resources among everybody's desktop machine
- virtual mainframe
  - build a "cluster system" that sits in a machine room
  - use dedicated PCs, dedicated network
  - special-purpose software

Model Comparison

- advantage of machines on desks
  - no hardware to buy
- advantages of virtual mainframe
  - no change to client OS
  - more reliable and secure
  - resource allocation easier
  - better network performance

Resource Pooling

- CPU
  - run each process on the best machine
  - stay close to user
  - balance load
- memory
  - use idle memory to store VM pages, cached disks blocks
- storage
  - distributed file system (already covered)

CPU Pooling

- How should we decide where to run a computation?
- How can we move computations between machines?
- How should shared resources be allocated?

Efficiency of Distributed Scheduling

- queueing theory predicts performance
- assume
  - 10 users
  - each user creates jobs randomly at rate C
  - machine finishes jobs randomly at rate F
- compare three configurations
  - separate machine for each user
  - 10 machines, distributed scheduling
  - a single super-machine (10x faster)

Predicted Response Time

separate machines	$\frac{1}{F-C}$
super-machine	$\frac{1}{10(F-C)}$
pooled machines	between the other two like separate under light load like super under heavy load

Independent Processes

- simplest method (on vanilla Unix)
  - monitor load-average of all machines
  - when a new process is created, put it on the least-loaded machine
  - processes don't move
- pro: simple
- con: doesn't balance load unless new processes are created; Unix isn't location-transparent

Location Transparency

- principle: a process should see itself as running on the machine where it was created
- location-dependencies: process-Ids, parts of file system, sockets, etc.
- usual solution
  - run "proxy" process on machine where process was created
  - "system calls" cause RPC to proxy

Process Migration

- idea: move running processes around to balance load
- problems:
  - how to move a running process
  - when to migrate
  - how to gather load information

Moving a Process

- steps
  - stop process, saving all state into memory
  - move memory image to another machine
  - reactivate the memory image
- problems
  - can't move to machine with different architecture or OS
  - image is big, so expensive to move
  - need to set up proxy process

Migration Policy

- migration can be expensive, so do rarely
- migration balances load, so do often
- many policies exist
- typical design: let imbalance persist for a while before migrating
  - "patience time" is several times the cost of a migration

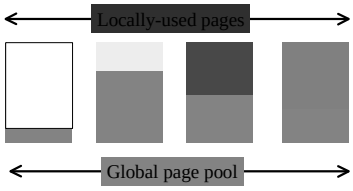
Pooling Memory

- some machines need more memory than they have; some need less
- let machines use each other's memory
  - virtual memory backing store
  - disk block cache
- assume (for now) all nodes use distinct pages and disk blocks

Failure and Memory Pooling

- might lose remotely-stored pages in a crash
- solution: make remote memory servers stateless
- only store pages you can afford to lose
  - for virtual memory: write to local disk, then store copy in remote memory
  - for disk blocks, only store "clean" blocks in remote memory
- drawback: no reduction in writes

Local Memory Management



within each block, use LRU replacement

Issues

- how to divide space between local and global pools
  - goal: throw away the least recently used stuff
    - » keep (approximate) timestamp of last access for each page
    - » throw away the oldest page
- what to do with thrown-away pages
  - really throw away, or migrate to another machine
  - where to migrate

Random Migration

- when evicting page
  - throw away with probability P
  - otherwise, migrate to random machine
    - » may immediately re-do at new machine
- good: simple local decisions; generally does OK when load is reasonably balanced
- bad: does 1/P as much work as necessary; makes bad decisions when load is imbalanced

N-chance Forwarding

- forward page N times before discarding it
- forward to random places
- improvement
  - gather hints about oldest page on other machines
  - use hints to bias decision about where to forward pages to
- does a little better than random

Global Memory Management

- idea: always throw away a page that is one of the very oldest
- periodically, gather state
  - mark the oldest 2% of pages as “old”
  - count number of old pages on each machine
  - distribute counts to all machines
- each machine now has an idea of where the old pages are

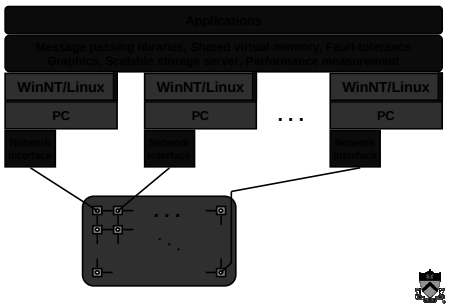
Global Memory Management

- when evicting a page
  - throw it away if it’s old
  - otherwise, pick a machine to forward to
    - » prob. of sending to M proportional to number of old pages on M
- when a node that had old pages runs out of old pages, stop and regather state
- good: old throws away old pages; fewer multi-migrations
- bad: cost of gathering state

Virtual Mainframe

- challenges are performance and single system image
- lots of work in commercial and research worlds on this
- case study: SHRIMP project
  - two generations built here at Princeton
    - » focus on last generation
  - dual goals: parallel scientific computing and virtual mainframe apps

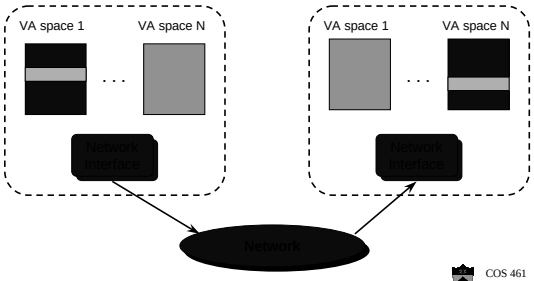
SHRIMP-3



Performance Approach

- single user-level process on each machine
  - cooperate to provide single system image
  - client connects to any machine
- optimized user-level to user-level communication
  - low latency for control messages
  - high bandwidth for block transfers

Virtual Memory Mapped Comm.



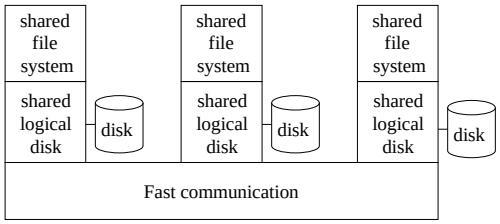
Communication Strategy

- separate permission checking from communication
  - establish "mapping" once
  - move data many times
- communication looks like local-to-remote memory copy
  - supported directly by hardware

Higher-Level Communication

- support sockets and RPC via specialized libraries
- calls do extra sender-to-receiver communication to coordinate data transfer
- bottom line for sockets
  - 15 microsecond latency
  - 90 Mbyte/sec bandwidth
  - much faster than alternatives

Pulsar Storage Service



Single Network-Interface Image

- want to tell clients there is just one server, even when there are many
  - balance load automatically
- methods
  - DNS round-robin
  - IP-level routing
    - » based on IP address of peer
    - » dynamic, based on load

Summary

- clusters of cheap machines can replace mainframes
  - keys: fast flexible communication, carefully implemented single system image
  - experience with databases too
- this method is becoming mainstream
- more work needed to make machines-on-desks model work