SMP & Tightly-Coupled Systems

- 15A Distributed Computing
- 15B Multi-Processor Systems
- 13E Distributed Systems - Consensus
- 13F Distributed Systems - Membership
- 14F Distributed Systems - Scalability
- 15C Tightly-Coupled Systems

Goals of Distributed Computing

- better services
  - scalability
    - apps too big to run on a single computer
    - grow system capacity to meet growing demand
  - improved reliability and availability
  - improved ease of use, reduced CapEx/OpEx
- new services
  - applications that span multiple system boundaries
  - global resource domains, services (vs. systems)
  - complete location transparency

Major Classes of Distributed Systems

- Symmetric Multi-Processors (SMP)
  - multiple CPUs, sharing memory and I/O devices
- Single-System Image (SSI), Cluster Computing
  - many computers, acting like a single computer
- loosely coupled, horizontally scalable systems
  - coordinated, but relatively independent systems
- application level distributed computing
  - peer-to-peer, application level protocols
  - distributed middle-ware platforms

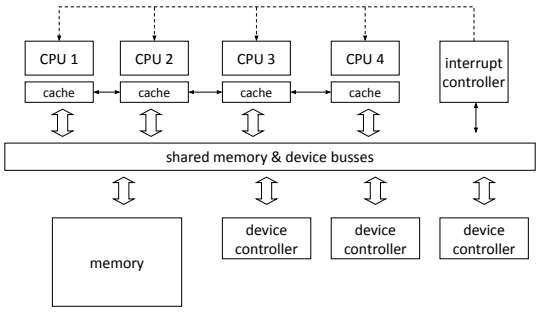
Evaluating Distributed Systems

- Performance
  - overhead, scalability, availability
- Functionality
  - adequacy and abstraction for target applications
- Transparency
  - compatibility with previous platforms
  - scope and degree of location independence
- Degree of Coupling
  - on how many things do distinct systems agree
  - how is that agreement achieved

SMP systems and goals

- Characterization:
  - multiple CPUs sharing memory and devices
- Motivations:
  - price performance (lower price per MIP)
  - incremental scalability (up to huge systems)
  - perfect application transparency
- Example:
  - multi-core Intel CPUs
  - multi-socket mother-boards

Symmetric Multi-Processors



SMP Price/Performance

- a computer is much more than a CPU
  - mother-board, disks, controllers, power supplies, case
  - CPU might cost 10-15% of the cost of the computer
- adding CPUs to a computer is very cost-effective
  - a second CPU yields cost of 1.1x, performance 1.9x
  - a third CPU yields cost of 1.2x, performance 2.7x
- same argument also applies at the chip level
  - making a machine twice as fast is ever more difficult
  - adding more cores to the chip gets ever easier
- massive multi-processors are obvious direction

SMP Operating System Design

- one processor boots with power on
  - it controls the starting of all other processors
- same OS code runs in all processors
  - one physical copy in memory, shared by all CPUs
- Each CPU has its own registers, cache, MMU
  - must cooperatively share memory and devices
- ALL kernel operations must be MT-Safe
  - protected by appropriate locks/semaphores
  - very fine grained locking to avoid contention

SMP Parallelism

- scheduling and load sharing
  - each CPU can be running a different process
  - just take the next ready process off the run-queue
  - processes run in parallel
  - most processes don't interact (other than in kernel)
- serialization
  - mutual exclusion achieved by locks in shared memory
  - locks can be maintained with atomic instructions
  - spin locks acceptable for VERY short critical sections
  - if a process blocks, that CPU finds next ready process

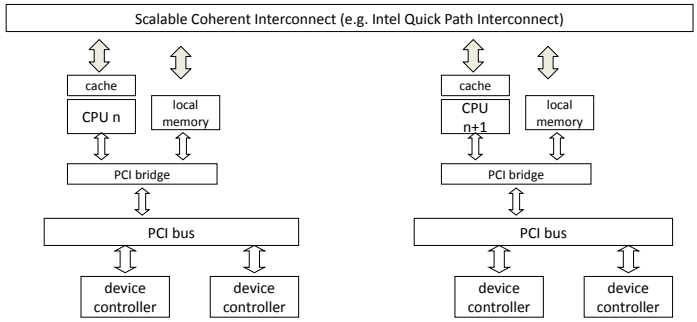
The Challenge of SMP Performance

- scalability depends on memory contention
  - memory bandwidth is limited, can't handle all CPUs
  - most references satisfied from per-core cache
  - if too many requests go to memory, CPUs slow down
- scalability depends on lock contention
  - waiting for spin-locks wastes time
  - context switches waiting for kernel locks waste time
- contention wastes cycles, reduces throughput
  - 2 CPUs might deliver only 1.9x performance
  - 3 CPUs might deliver only 2.7x performance

Managing Memory Contention

- Fast n-way memory is <u>very</u> expensive
  - without it, memory contention taxes performance
  - cost/complexity limits how many CPUs we can add
- Non-Uniform Memory Architectures (NUMA)
  - each CPU has its own memory
    - each CPU has fast path to its own memory
  - connected by a <u>Scalable Coherent</u> Interconnect
    - a <u>very fast</u>, <u>very local</u> network between memories
    - accessing memory over the SCI may be 3-20x slower
  - these interconnects can be highly scalable

CC-Non-Uniform Memory Architecture Symmetric Multi-Processors



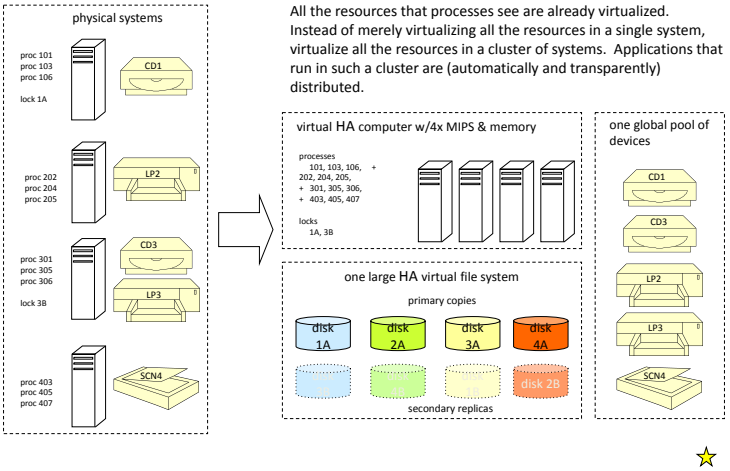
S/W design for NUMA systems

- it is all about local memory hit rates
  - every outside reference costs us 3-20x performance
  - we need 75-95% hit rate just to break even
- How can the OS ensure high hit-rates?
  - replicate shared code pages in each CPU's memory
  - assign processes to CPUs, allocate all memory there
  - migrate processes to achieve load balancing
  - spread kernel resources among all the CPUs
  - attempt to preferentially allocate local resources
  - migrate resource ownership to CPU that is using it

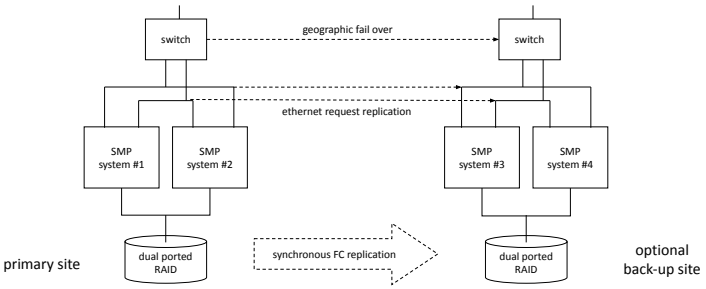
Single System Image (SSI) Clusters

- Characterization:
  - a group of seemingly independent computers collaborating to provide SMP-like transparency
- Motivation:
  - higher reliability, availability than SMP/NUMA
  - more scalable than SMP/NUMA
  - excellent application transparency
- Examples:
  - Locus, MicroSoft Wolf-Pack, OpenSSI
  - Oracle Parallel Server

The Dream

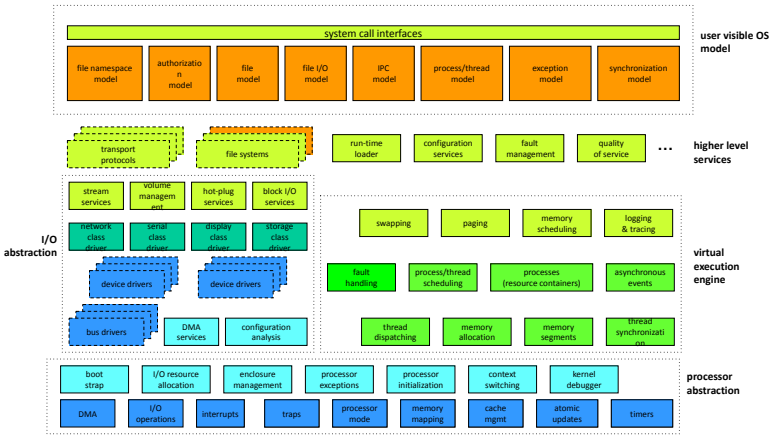


Modern Clustered Architecture



Active systems service independent requests in parallel. They cooperate to maintain shared global locks, and are prepared to take over partner's work in case of failure. State replication to a back-up site is handled by external mechanisms.

Structure of a Modern OS



OS design for SSI clustering

- all nodes agree on the state of all OS resources
  - file systems, processes, devices, locks, IPC ports
  - any process can operate on any object, transparently
- they achieve this by exchanging messages
  - advising one-another of all changes to resources
    - each OS's internal state mirrors the global state
  - request execution of node-specific requests
    - node-specific requests are forwarded to owning node
- implementation is large, complex, difficult
- the exchange of messages can be very expensive

SSI Clustered Performance

- clever implementation can minimize overhead
  - 10-20% overall is not uncommon, can be much worse
- complete transparency
  - even very complex applications "just work"
  - they do not have to be made "network aware"
- good robustness
  - when one node fails, others notice and take-over
  - often, applications won't even notice the failure
- nice for application developers and customers
  - but they are complex, and not particularly scalable

Commitment Protocols

- used to implement distributed commitment
  - provide for atomic all-or-none transactions
  - simultaneous commitment on multiple hosts
- challenges
  - asynchronous conflicts from other hosts
  - nodes fail in the middle of the commitment process
- multi-phase commitment protocol:
  - Confirm no conflicts from any participating host.
  - All participating hosts told to prepare for commit.
  - All participating hosts told to “make it so”.

Distributed Consensus

- achieving simultaneous, unanimous agreement
  - even in the presence of node & network failures
  - required: agreement, termination, validity, integrity
  - desired: bounded time
- consensus algorithms tend to be complex
  - and may take a long time to converge
- they tend to be used sparingly
  - e.g. use consensus to elect a leader
  - who makes all subsequent decisions by fiat

Typical Consensus Algorithm

1. Each interested member broadcasts his nomination.
2. All parties evaluate the received proposals according to a fixed and well known rule.
3. After allowing a reasonable time for proposals, each voter acknowledges the best proposal it has seen.
4. If a proposal has a majority of the votes, the proposing member broadcasts a claim that the question has been resolved.
5. Each party that agrees with the winner’s claim acknowledges the announced resolution.
6. Election is over when a quorum acknowledges the result.

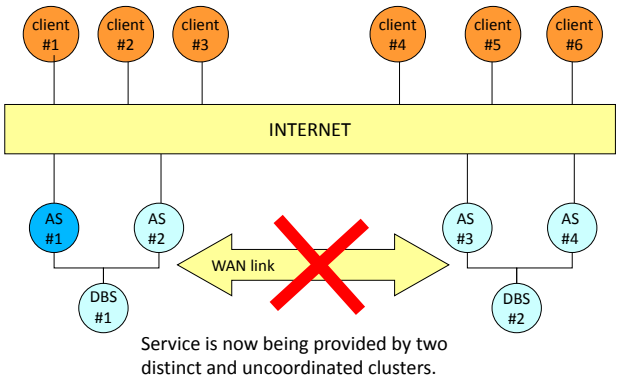
Cluster Membership

- a Cluster is a group of nodes …
  - all of whom are in communication with one-another
  - all of whom agree on an elected cluster master
  - all of whom abide by the cluster master’s decisions
    - he may (centrally) arbitrate all issues directly
    - he may designate other nodes to make some decisions
- Highly Available service clusters
  - cluster master assigns work to all of the other nodes
  - if a node falls out of the cluster, its work is reassigned

Heart-beats

- “I’m still alive” messages, exchanged in cluster
- cluster master monitors the other nodes
  - regularly confirm each node is working properly
  - promptly detect any node falling out of the cluster
  - promptly reassign work to surviving nodes
- some nodes must monitor the cluster master
  - to detect the failure of the cluster master
  - to trigger the election of a new cluster master

Split-Brain



Service is now being provided by two distinct and uncoordinated clusters.

Quorum

- the simplest solution to split-brain is 'quorum'
  - in a cluster that has been provisioned for N nodes, becoming cluster master requires (N/2)+1 votes
  - this completely prevents split-brain
    - it also prevents recovering from the loss of N/2 nodes
- some systems use a 'quorum device'
  - e.g. a shared (multi-ported) disk
    - cluster master must be able to reserve/lock this device
- some systems use special 'election' hardware

Scalability - Bottlenecks

- avoid a single control points
  - partition responsibility over many nodes
- separated data- and control-planes
  - control nodes choreograph the flow of data
    - where data should be stored or obtained from
    - ensuring coherency and correct serialization
  - data flows directly from producer to consumer
    - data paths are optimized for throughput/efficiency
- dynamic re-partitioning of responsibilities
  - in response to failures and/or load changes

Scalability: Cluster Protocols

- Consensus protocols do not scale well
  - they only work for small numbers of nodes
- Minimize number of consensus operations
  - elect a single master who makes decisions
  - partitioned and delegated responsibility
- Avoid large-consensus/transaction groups
  - partition work among numerous small groups
- Avoid high communications fan-in/fan-out
  - hierarchical information gathering/distribution

Lessons Learned

- consensus protocols are expensive
  - they converge slowly and scale poorly
- systems have a great many resources
  - resource change notifications are expensive
- location transparency encouraged non-locality
  - remote resource use is much more expensive
- a greatly complicated operating system
  - distributed objects are more complex to manage
  - complex optimizations to reduce the added overheads
  - new modes of failure w/complex recovery procedures
- Bottom Line: Deutsch was right!

Reading and Assignments

Reading:

- Eventual Consistency
- Kampe: Horizontally Scaled Systems

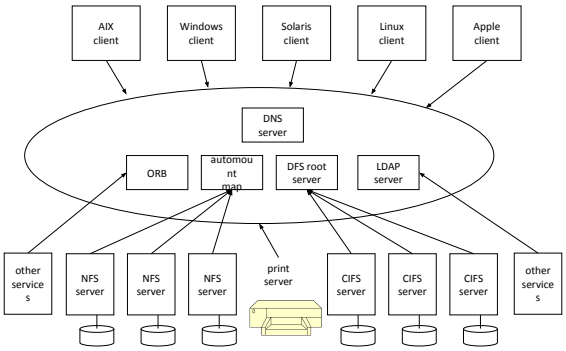
Prep for Final Exam

Supplementary Slides

SMP Device I/O

- all processors can access all memory/devices
  - any processor can initiate an I/O operation
    - initiating processor need not be one that requested the I/O
  - any processor can service an I/O interrupt
    - servicing processor need not be one that initiated I/O
- interrupt controller picks which CPU to interrupt
  - dynamic priorities, always interrupt lowest priority CPU
  - fixed binding of some or all interrupts to one CPU
  - automatic round-robin delivery

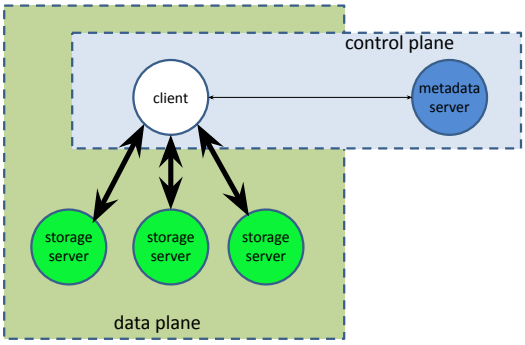
Global Resource View (heterogeneous systems &resources)



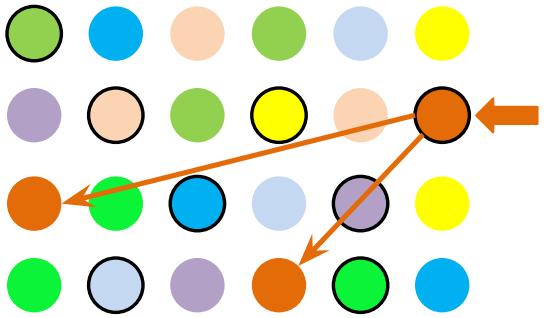
Scalability – Traffic

- network messages are expensive
  - NIC and network capacity to carry them
  - server CPU cycles to process them
  - client delays awaiting responses
- minimize messages/client/second
  - cache results to eliminate requests entirely
  - enable complex operations w/single request
  - buffer up large writes in write-back cache
  - pre-fetch large reads into local cache

Control and Data Planes



Data Plane: small transaction clusters



Control Plane: hierarchical reporting