Clusters

ECE/CS 757 Spring 2007
J. E. Smith

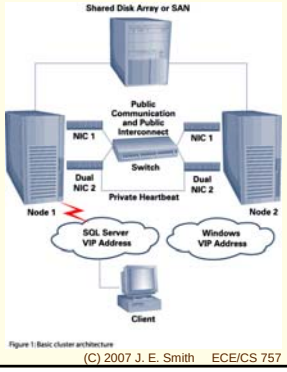
Lecture Outline

- **Introduction & Examples**
- **Case studies**
  - VAX Cluster: N. Kronenberg et al., VAXclusters: A Closely-Coupled Distributed System, ACM Transactions on Computer Systems, May 1986, pp. 130-146.
  - Google Cluster: **Luiz Andre Barroso, Jeffrey Dean, Urs Holzle, Web Search For a Planet: The Google Cluster Architecture, IEEE Micro, 23(2):22-28, March-April 2003.**
  - IBM Blade Center: **D. Desai, et al., BladeCenter System Overview, IBM Journal of R and D, Nov. 2005, pp. 809- 821.**
  - IBM 390 Parallel Sysplex: G. King, Cluster Architectures and S/390 Parallel Sysplex Scalability, IBM Systems Journal, 1197.

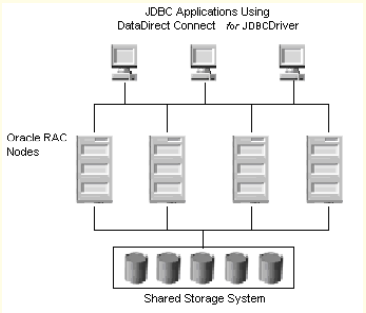
Definition etc.

- **A bunch of computers connected with a network**
  that can be viewed by the user as a single system
  - Each computer has its own address space
  - Programming commonly via message passing
- **Advantages**
  - Easy to design
  - Relatively inexpensive
    Commodity off-the-shelf parts
  - Can serve dual duty
    Desktop system + Network of Workstations (NOW)

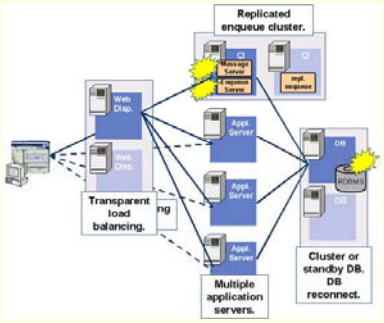
Example Clusters (via Google)



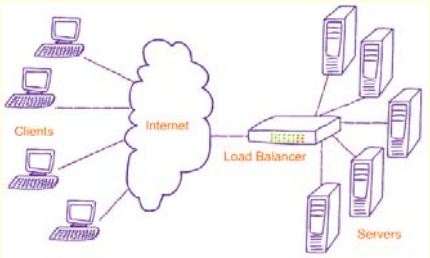
Example Clusters



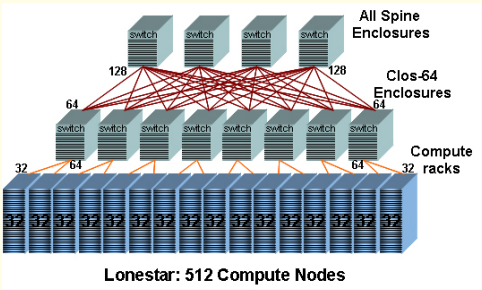
Example Clusters



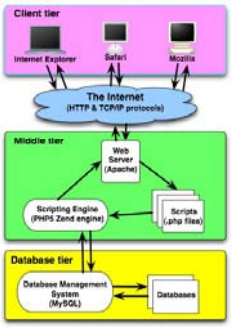
Example Clusters



Example Clusters



Example Clusters

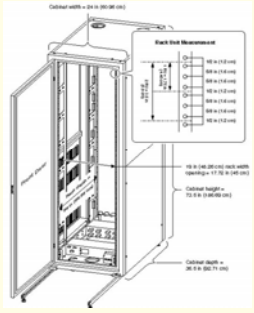


Packaging/Physical Design

- Workstations (PCs) on a LAN

Packaging/Physical Design

- Rack Mounted PC Boards



Packaging/Physical Design

- Blades
  - Shared power supply, cooling, etc.

Packaging/Physical Design

- **Sun's Project Black Box (Modular Datacenter)**
  - "Portable computing" – up to 17 tons, 280 RU
  - BYOPS, BYOIU

Applications

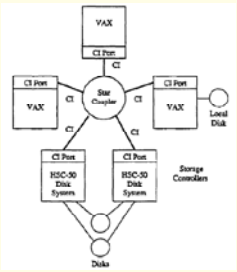
- **Commercial**
  - Large shared databases
  - Largely independent threads
  - "Architecture" often means software architecture
  - May use higher performance storage area network (SAN)
- **Scientific**
  - Inexpensive high performance computing
  - Based on message passing
  - Historical (research): Virtual Shared Memory
  - May use higher performance node-to-node network
  - Where HPC clusters end and MPPs begin isn't always clear

Software Considerations

- **Throughput Parallelism**
  - As in many commercial servers
  - Distributed OS message passing
  - VAXcluster early example
- **True Parallelism**
  - As in many scientific/engineering applications
  - Use programming model and user-level API
- **Programming Models**
  - Message-Passing
    Commonly used
  - Shared memory
    Virtual Shared Memory
    Subject of research; not commonly used, but interesting
- **Of course, a real system can do both throughput and true parallelism**

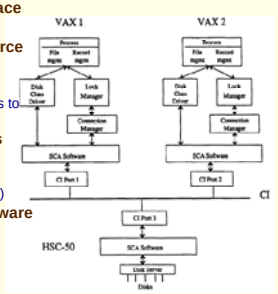
An Early System: VAXcluster

- **The first(?) cluster system, circa 1983**
- **Off-the-shelf computers (VAXes)**
  - Plus proprietary network
  - Plus communications interfaces (CI)
- **From the user perspective –**
  - User sees same system regardless of processor node being used
  - Single file system
  - Shared devices, print services, etc.
  - Throughput parallelism
- **Network**
  - Logically a bus, physically a star
  - Uses CSMA (ethernet-like) but different arbitration scheme



VAXcluster Software Architecture

- **Processes have own address space**
  - File mgmt service part of process
- **Lock Manager coordinates resource sharing**
- **Disk Class Driver**
  - Generic driver that sends messages to real disks through CI
- **Connection Manager coordinates nodes belonging to a cluster**
  - Cluster membership is dynamic (although changes are infrequent)
- **SCA (Systems Comm. Arch) software routes messages via CIs**

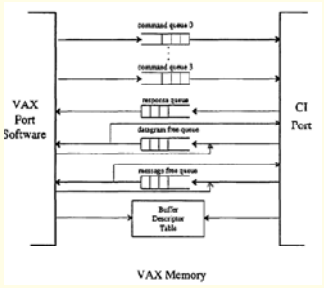


SCA (System Comm. Arch.)

- **Messages**
  - Cannot be lost; order of arrival maintained
    E.g. used for disk read/write commands
  - CIs assure reliability
  - Up to 112 bytes
- **Datagrams**
  - Can be dropped, lost duplicated, arrive out of order
  - Higher level software deals with unreliability, ordering, etc.
  - Up to 576 bytes (can hold a VAX page)
- **Block data transfers**
  - Contiguous in virtual address space
  - No size limit
  - Direct memory-to-memory (no buffering in CI)

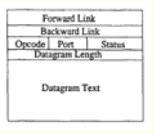
CI Architecture

- **Seven queues**
  - Reside in VAX memory
  - 4 command queues (allows priority levels)
  - Response queue for incoming messages/datagrams
  - Free queues
    Source of empty packets



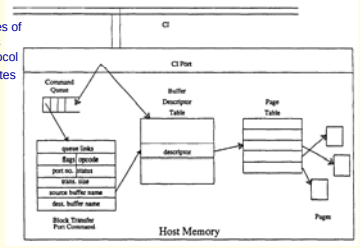
Example

- **Queue SEND DATAGRAM into a command queue**
  - Set response bit in packet if ack is needed
  - CI places packet into response queue after it is sent
  - Else CI places packet into free queue
- **When CI port receives a datagram**
  - Take a packet from datagram free queue
    else drops datagram if no free packet
  - Place DATAGRAM RECEIVED packet in response queue
- **Message similar – interrupts if no free packet on receive**



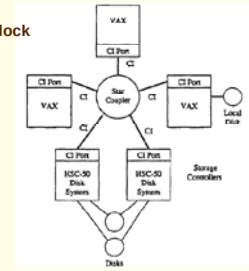
Block Transfers

- **Block transfers are memory-to-memory**
  - Don't use packet queues
- **Buffer descriptors provide CI w/ needed information**
  - Initiating software gives names of sending and receiving buffers and size via higher level protocol
  - Single command packet initiates transfer



Example: Disk Read

- **Disk class driver sends read message to CI port on HSC disk controller**
  - Request contains device type, unit number, logical disk address, requestor's buffer name, size of transfer
- **Controller reads data and sends it via block transfer**
- **When complete, controller sends completion/status message back to requester**



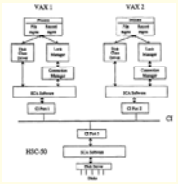
Locking

- **Locks are distributed**
  - Each shared resource has a master node responsible for granting locks
  - Resource directory maps name of resource to name of master node
    -- Distributed among nodes; given resource name, any node can determine directory location
- **To Lock a resource:**
  - Lock manager sends lock request message via SCA to directory for resource

    1. If directory on master node for resource, then performs lock request and responds

    2. If directory not on master node, directory responds with location of master node
       -- Then lock message is sent to master node

    3. If resource undefined; respond telling requester to master the resource itself

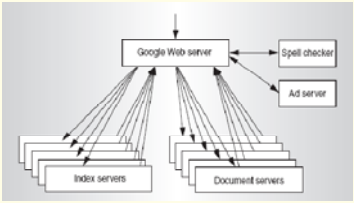


Coordinated Disk Caching

- **Multiple sharers**
  - Use *Value block* associated w/ resource lock
    Value passed with resource ownership
  - Usage
    Hold version number of block of disk data
    Sharers of data cache copies;
    Increment version number when data is modified
    If version number matches cached data, then data valid
    else, re-load copy from disk
- **Deferred writeback**
  - Use interrupt when resource lock request is blocked
  - Usage
    Get lock and perform repeated reads/write w/ cached version
    When interrupted, release lock

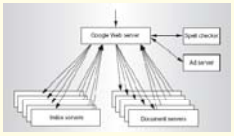
Case Study: Google Server

- **Architecture Overview**
  - Provide reliability in software
    - Use cluster of PCs
    - Not high-end servers
  - Design system to maximize aggregate throughput
    - Not server response time
    - (Can parallelize individual requests)



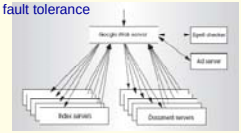
Application Summary

- **Geographically distributed clusters**
  - Each with a few thousand machines
- **First perform Domain Name System (DNS) lookup**
  - Maps request to nearby cluster
- **Send HTTP request to selected cluster**
  - Request serviced locally w/in that cluster
- **Clusters consist of Google Web Servers (GWSes)**
  - Hardware-based load balancer distributes load among GWSes



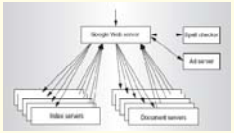
Query Execution

- **Phase 1:**
  - Index servers consult inverted index that maps query word to list of documents
  - Intersect hit lists and compute relevance index
  - Results in ordered list of document ids (*docids*)
- **Both documents and inverted index consume terabytes of data**
- **Index partitioned into "shards", shards are replicated**
  - Index search becomes highly parallel; multiple servers per shard load balanced requests
  - Replicated shards add to parallelism and fault tolerance



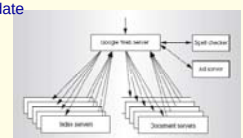
Query Execution

- **Phase 2:**
  - Start w/ docids and determine title, resource locator, document summary
- **Done by document servers**
  - Partition documents into shards
  - Replicate on multiple servers
  - Route requests through load balancers



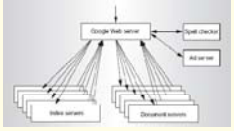
Parallelism/Fault Tolerance

- **High levels of "natural" parallelism**
  - Both inter-query and intra-query
  - Near-linear speedups
- **Replication helps both parallelism and fault tolerance**
  - Permits software-based fault tolerance
  - (The requirements for fault tolerance aren't very high, though)
- **Most accesses are read-only**
  - Combined w/ replication, simplifies updates
  - Queries are diverted during update



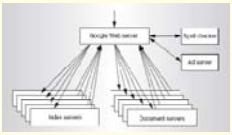
Hardware

- **Performance/Price beats pure Performance**
  - Use dual CPU servers rather than quad
  - Use IDE rather than SCSI
- **Use commodity PCs**
  - Essentially mid-range PCs except for disks
  - No redundancy as in many high-end servers
- **Use rack mounted clusters**
  - Connect via ethernet
- **Result is order of magnitude better performance/price than using high-end server components**



Power

- **Power is a very big issue**
  - Google servers 400 watts/ft^2
  - High end servers 700 watts/ft^2
  - Typical commercial data center – 70-150 watts/ft^2
    ⇒ special cooling or additional space, anyway
    using high-end servers would make matters worse



Energy-Proportional Computing

Luiz André Barroso and Urs Hölzle, "The Case for Energy-Proportional Computing," IEEE Computer, Dec 2007.

- **Cluster nodes not easy to power up/down**
  - Response time due to load variation
  - Persistent state stored at nodes
- **Usually run under partial/low load**
  - Energy use should be proportional to load
  - CPUs pretty good at this (power management)
  - System level is worse (DRAM)
  - I/O not great (disk spin management)
  - Networking gear is terrible (all static power)

Application Characterstics

- **For 1GHz Pentium III**
  - Index server
- **Data is percent per instruction retired**
  - Br mispredict – 50 per 1K insts.
  - L1 I miss – 4 per KI
  - L1 D miss 7 per KI
  - etc
- **CPI actually isn't bad**
  - Branch mispredict rate is not very good
  - Data organized for optimal searches will likely have this property
- **Argues for CMP or SMT or both**
  - ILP is limited (1 mispredict per 20 insts)
  - Pentium 4 has twice the CPI and similar branch prediction perf.
  - Observe 30% speedup w/ hyperthreading (at high end of Intel projections)

Table 1. Instruction-level measurements on the index server.

Characteristic	Value
Cycles per instruction	1.1
Ratios (percentage)	
Branch mispredict	5.0
Level 1 instruction miss*	0.4
Level 1 data miss*	0.7
Level 2 miss*	0.3
Instruction TLB miss*	0.04
Data TLB miss*	0.7

* Cache and TLB ratios are per instructions retired.

Memory System

- **For 1GHz Pentium III**
- **Good temporal locality for instructions**
  - Loops doing searches
- **Data blocks**
  - Very low temporal locality
  - Good spatial locality within index data block
    So data cache performance is relatively good.
- **Memory bandwidth not a bottleneck**
  - Memory bus about 20% utilized
  - Relatively high computation per data item
    (or maybe just high mispredicts per data item)
  - Modest L2, longer (128 byte) cache lines seem to be a good design point

Table 1. Instruction-level measurements on the index server.

Characteristic	Value
Cycles per instruction	1.1
Ratios (percentage)	
Branch mispredict	5.0
Level 1 instruction miss*	0.4
Level 1 data miss*	0.7
Level 2 miss*	0.3
Instruction TLB miss*	0.04
Data TLB miss*	0.7

* Cache and TLB ratios are per instructions retired.

Architecture Bottom Line

- **Large clusters of small-SMP nodes**
  - Not large scale shared memory MPs
- **Commodity PC components**
- **Characteristics of Google application**
  - Focus on Performance/Price
  - No private state
- **Many web servers (although smaller than Google) have similar characteristics**

IBM BladeCenter

- **Motivation –**
  - Circa 1999 -- Rack mounted servers becoming difficult
    - 42 1u servers require a lot of cables in a very small space
    - No redundant power supplies/cooling make servicing difficult
- **Solution : Use server blades**
  - Similar to technology used in telecommunications/network switches and hubs
  - Chassis w/ backplane for interconnect (network and power)
  - Shared (and redundant) power supplies/cooling

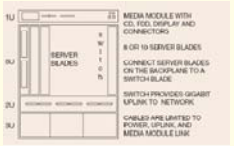
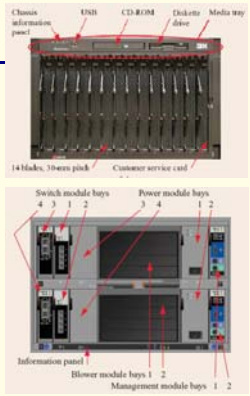


Figure 1
The earliest concept drawing of the BladeCenter architecture, created on October 20, 1999.

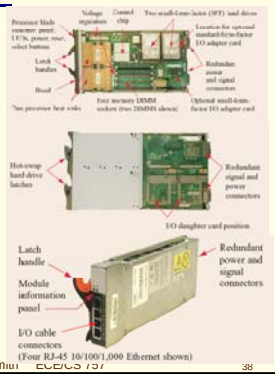
Hardware

- Midplane
- 14 server blades in front
  - Plus shared media stuff
  - Double density of 1U server
- Switch, Power supply modules, etc. in back
- Airflow front to rear
- Redundant power supplies



Hardware, contd.

- Processor Blade
- Storage Expansion Blade
- Switch Module

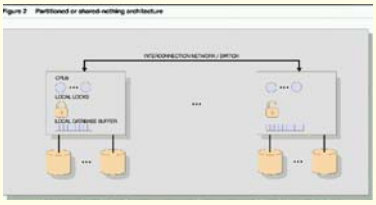


Commercial Cluster Architectures

- For scalable, commercial applications
  - OLTP, Web servers, Parallel database servers
- Disk sharing creates major division
  - Partitioned disks
  - Shared disks
- Tradeoffs involve:
  1. update concurrency control method
  2. database buffer cache coherency control
  3. provision for shared memory among the nodes

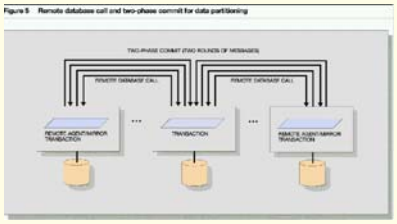
Partitioned Architecture

- Disks are not shared
- Function shipping model
  - Perform remote function call to node holding data
  - Local locks and buffer cache (simplify implementation)
- I/O shipping model
  - Remote I/O access – similar to virtual shared disks



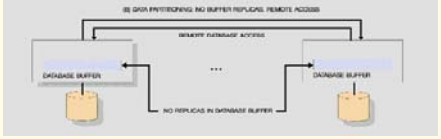
Partitioned Architecture Concurrency Ctl

- Remote database call – communication delay
- Requesting node context switches (because of long remote call delay)
- On remote node – allocate an agent



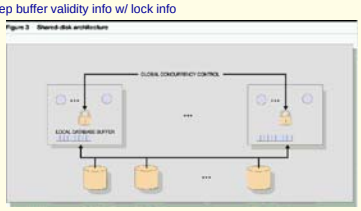
Partitioned Architecture Data Buffering

- Data buffering more efficient than with shared disk scheme



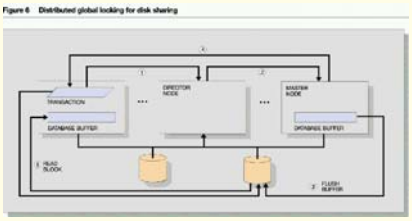
Shared Disk Architecture

- All nodes have direct access to disks
- Requires global buffer coherency and concurrency control (locks)
- Centralized or Distributed concurrency control
  - VAXcluster is example of distributed
- Buffer coherency
  - Broadcast invalidate – simple, but high overhead in large systems
  - Integrated concurrency/coherency
    - Keep buffer validity info w/ lock info



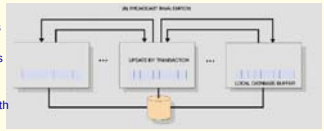
Shared Disk Architecture Concurrency Ctl.

- Must have global concurrency control (locks)
- Distributed locking *typically* done
  - As in VAXcluster
  - Overhead in finding master, and then communicating with master
- 390 Sysplex uses global w/ assist



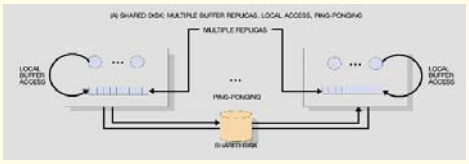
Shared Disk Architecture Buffering

- Buffer cache coherency
- Invalidate protocol
  - Overhead of invalidation messages
    - Linear in number of nodes
  - Overhead due to coherence misses
- Check on access
  - Lazy method
  - Track the validity (in conjunction with locks)
  - Explicitly check the validity before every access
  - Invalid data stays in buffers (and may decrease hit rate)
- If needed block dirty on remote node
  - Write back to disk by remote node
  - Read from disk by local node
  - Adds to latency and consumes disk bandwidth
  - Really bad when there is "ping-ponging"



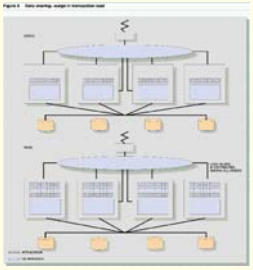
Shared Disk Buffer Inefficiency

- Redundant cached copies
  - Especially if transaction affinity is not maintained



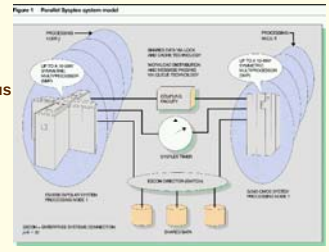
Load Balancing

- Multiple transaction classes each w/ affinity
  - E.g. TPC-C affinity w/ a warehouse
- Data is also partitioned (or buffered on nodes)
- Affinity classes should match partitioning of data
- On a "load surge", front-end can balance load, but then affinity class/data partition correspondence is lost ⇒ performance loss
- The stronger the affinity for data, the worse the performance effect



IBM 390 Parallel Sysplex

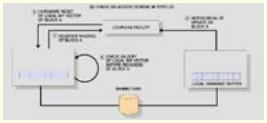
- Cluster architecture developed by a vertically integrated company
  - With lots of large system experience
- Clusters w/ shared disks plus connection via coupling facilities
- Coupling facility
  - Hardware plus microcode
  - 100 MB per sec links
  - Commands complete in microseconds
  - Three functions –
    - Lock, cache, list



Coupling Facility

- **Locks**
  - Lock table mapped to data blocks
  - Initial request to lock managed directly by coupling facility (microcode?) – works 99%
  - If contention, handled by software lock manager
  - Lock release can be eager or lazy
- **Lists**
  - General purpose queueing structure (in microcode)
  - Remove/add elements FIFO or LIFO or by key
  - Programs register "interest" in a list
    - Notified when transition from empty to non-empty
  - Can be used for load balancing, for example

Coupling Facility



- **Check on Access**
- **Database Caching**
  - Local or Global data blocks in systems
    - Bit vector indicating validity
  - Cache directory in coupling facility
    - Indexed by block name; points to validity bits
  - If database manager performs an update on a cached block
    - Lookup directory entry, send invalidates (clears bit in validity vector)
  - All done in microcode
- **Coupling facility may also cache copies**
  - Like L2 cache for local copies
  - Avoids high overheads due to ping-ponging

Coupling Facility, contd.

- **Load Balancing**
  - Data partitioning/processing affinity mismatches can be efficiently handled w/ coupling facility
  - I.e. ping-ponging might occur if other nodes have to take over some of the affinity load when there is a load surge; coupling facility permits robustness.
- **Fault Tolerance**
  - Assumes processing node fails, but shared data is still available
  - Effect similar to load balancing

Performance

- **Systems**
  - Two and Eight node 390 systems w/ one coupling facility
  - Sixteen node 390 system w/ two coupling facilities
- **Software Systems**
  - Customer Information Control System (CICS) Transaction Manager (TM)
  - IMS Database Manager
- **Benchmarks**
  - OLTP – Warehousing, Reservations, Banking

Performance -- Scaling

- **Very good scalability**
- **Most of the cost comes in scaling from one system to two**

System Size	2	8	16
CICS-TM/IMS-DB	1.00	3.89	7.40
IMS-TM/DB2	1.00	3.87	

Performance – I/O per Transaction

- **Normalize to one node system**
- **Inversely proportional to buffer hit ratio**
- **IMS-DB does not use buffering in coupling facility; DB2 system does**
  - More total system buffers
  - Reduced ping-ponging

System Size	1	2	8	16
CICS-TM/IMS-DB	1.00	1.03	1.08	1.13
IMS-TM/DB2	1.00	.96	.94	

Clusters - Summary

- **Reasons for clusters**
  - Performance – horizontal scaling
  - Cost: commodity h/w, commodity s/w
  - Redundancy/fault tolerance
- **Hardware Challenges**
  - Cost, commodity components
  - Power, cooling, energy proportionality
  - Reliability, usability, maintainability
- **Software Challenges**
  - Programming model
  - Applications with suitable characteristics
  - Load balancing