18-640 Foundations of Computer Architecture

Lecture 20: “Cluster Computing Systems”

John Paul Shen
November 13, 2014

- Recommended Reading Assignments:
  - Luiz Andre Barroso, Jeffrey Dean, Urs Holzle, “Web Search For a Planet: The Google Cluster Architecture,” IEEE Micro, March-April 2003.
  - Jeffrey Dean and Sanjay Ghemawat, “MapReduce: Simplified Data Processing on Large Clusters,” OSDI, 2004.

Electrical & Computer ENGINEERING

18-640 Foundations of Computer Architecture

Lecture 20: “Cluster Computing Systems”

A. Computer Cluster
B. DEC VAXcluster
C. Google Cluster
D. MapReduce & Hadoop

Electrical & Computer ENGINEERING

A. Computer Cluster

GRID COMPUTING

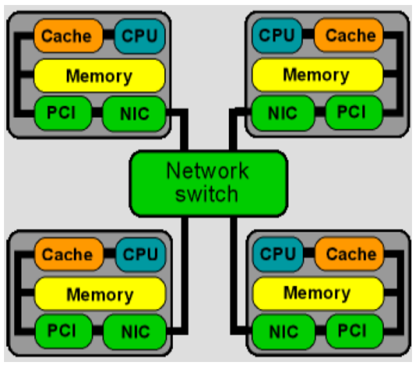
Distributed and Dynamically Networked Computers

CLUSTER COMPUTING

Shared File System and LAN Connected Multi-computers

MULTIPROCESSING

Shared Memory Multicores or Multiprocessors



Electrical & Computer ENGINEERING

What is a Computer Cluster?

- Cluster = a set of computers connected with a network
  - *Cluster can be viewed by the user as a single system (or a "multi-computer" system)*
  - Typically commodity computers (PCs) connected by commodity LAN (Ethernet)
  - Each computer (cluster node) runs its own OS and has its own address space
  - Supports execution of parallel programs via message passing with a master node
  - Typically supported by a shared file system and some "clustering middleware"
- Advantages
  - Easy to build: early systems were customer built using commodity PCs and networks
  - Relatively inexpensive: can get significant throughput performance at very low cost
  - Wide range of systems: can vary from small personal clusters to supercomputers
  - *Leverage concurrent advancements in personal computers and local area networks*

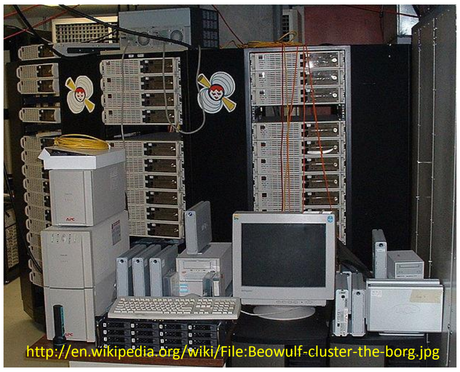
Beowulf Cluster

[Thomas Sterling & Donald Becker, NASA, 1994]

"Beowulf is a multi-computer which can be used for parallel computations. It is a system with one server node, and one or more client nodes connected via Ethernet. Beowulf also uses commodity software like FreeBSD, Linux, PVM (Parallel Virtual Machine) and MPI (Message Passing Interface). Server node controls the cluster and serves files to the client nodes."

"If you have two networked computers which share at least the `/home` *file system via NFS, and trust each other to execute remote shell (rsh), then it could be argued that you have a simple, two node Beowulf machine."*

http://en.wikipedia.org/wiki/File:Beowulf.jpg


http://en.wikipedia.org/wiki/File:Beowulf-cluster-the-borg.jpg

Example Clusters

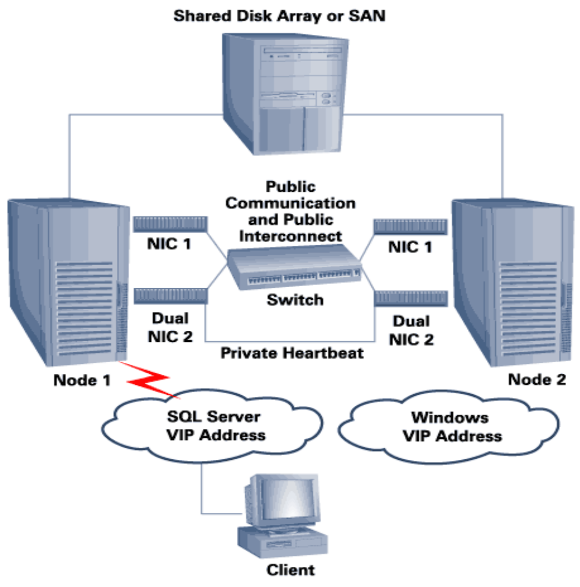
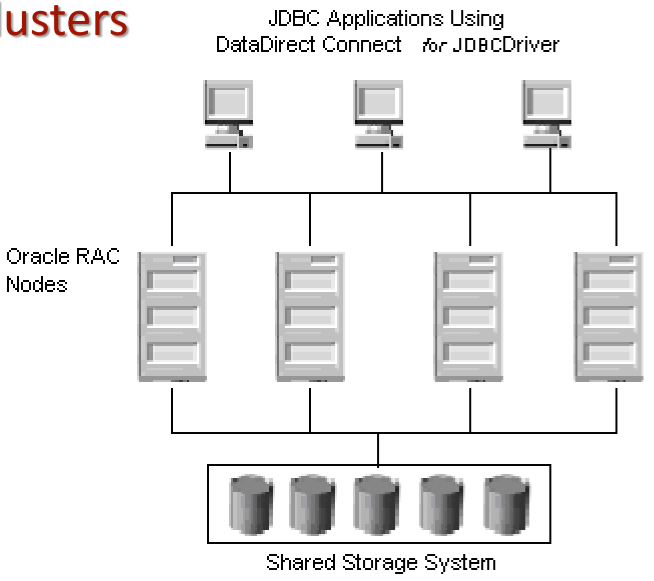
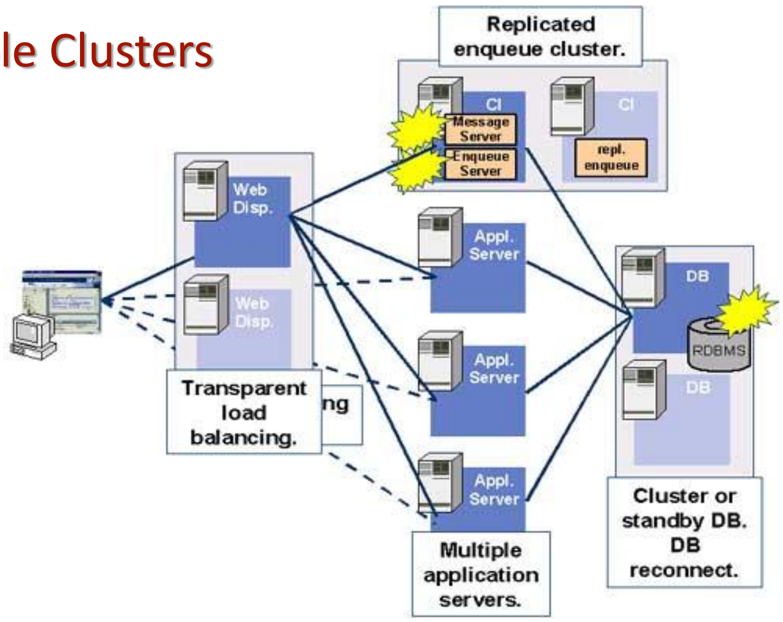


Figure 1: Basic cluster architecture

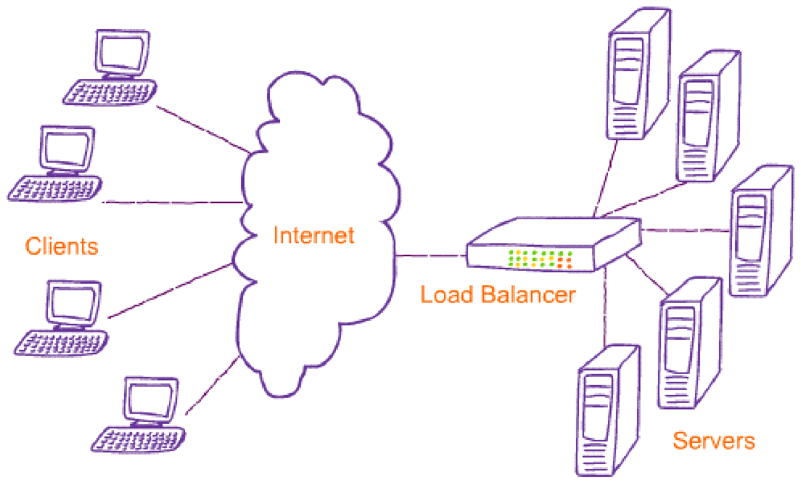
Example Clusters



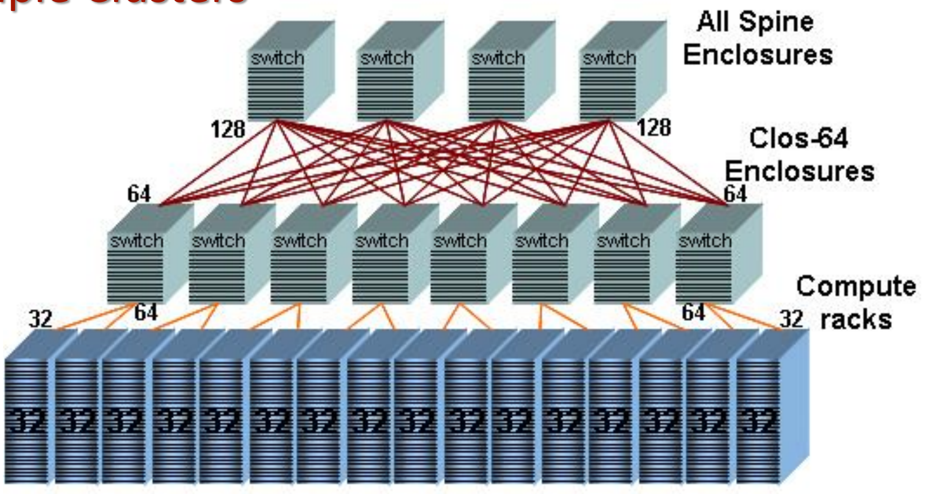
Example Clusters



Example Clusters

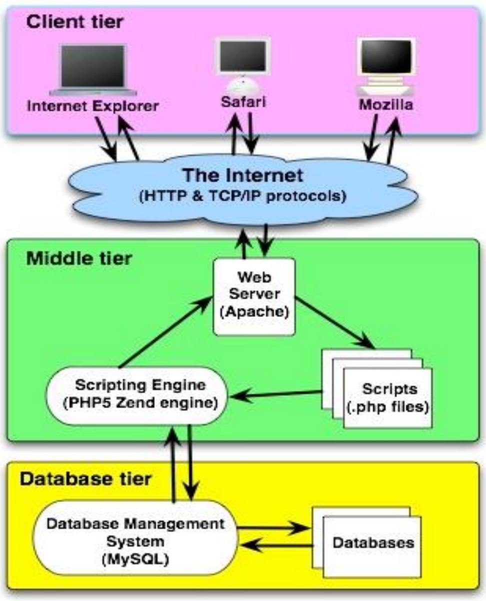


Example Clusters



Lonestar: 512 Compute Nodes

Example Clusters



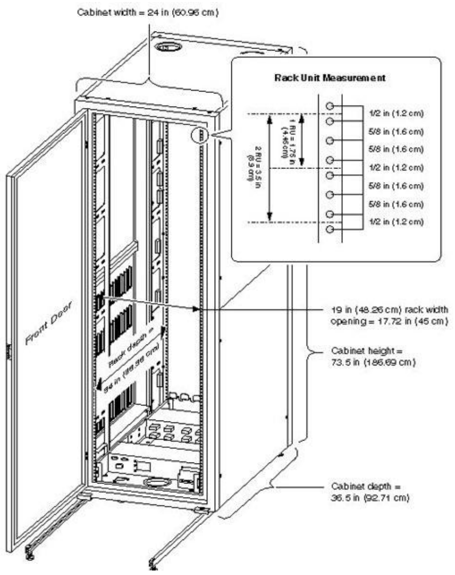
Physical Design

Workstations (PCs) on a LAN

Physical Design

Rack Mounted PC Boards

R-Cluster


Cabinet width = 24 in (60.96 cm)
Rack Unit Measurement
1 RU = 1.75 in (4.45cm)
2 RU = 3.5 in (8.9 cm)
1/2 in (1.2 cm)
5/8 in (1.6 cm)
5/8 in (1.6 cm)
1/2 in (1.2 cm)
5/8 in (1.6 cm)
5/8 in (1.6 cm)
1/2 in (1.2 cm)
Front Door
Rack depth = 34 in (86.36 cm)
19 in (48.26 cm) rack width opening = 17.72 in (45 cm)
Cabinet height = 73.5 in (186.69 cm)
Cabinet depth = 36.5 in (92.71 cm)

Physical Design

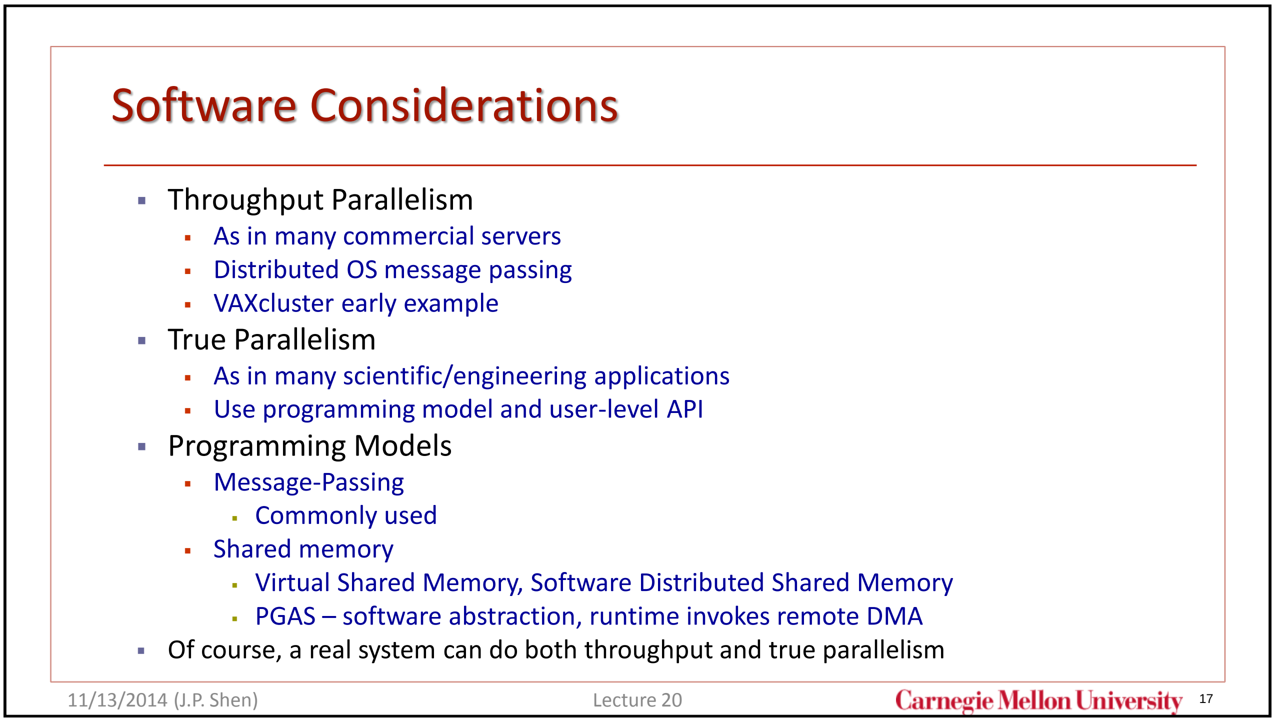
- Blades
  - Shared power supply, cooling, etc.

Physical Design

- Sun’s Project Black Box (Modular Datacenter)
  - “Portable computing” – up to 17 tons, 280 RU
  - BYOPS, BYOIU
  - No field-replaceable parts – why?

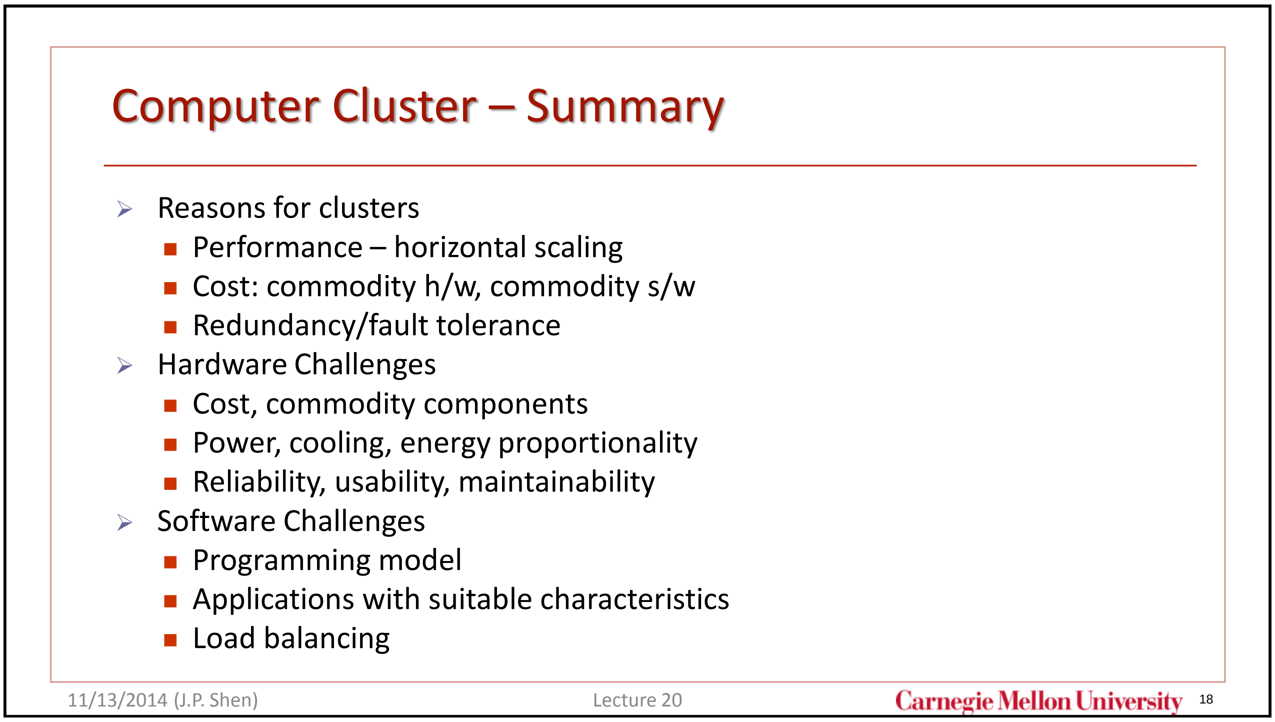
Applications

- 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
  - Also PGAS (partitioned global address space); software 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, Software Distributed Shared Memory
    - PGAS – software abstraction, runtime invokes remote DMA
- Of course, a real system can do both throughput and true parallelism

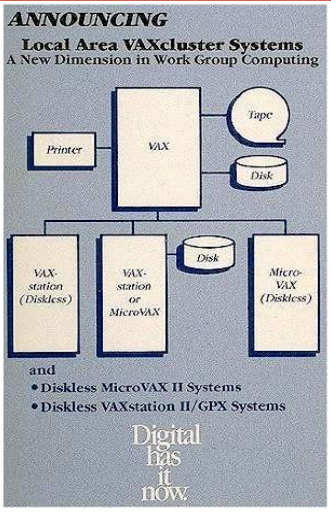


Computer Cluster – Summary

- 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

B. DEC VAXcluster

[Digital Equipment Corporation, 1984]



ANNOUNCING

Local Area VAXcluster Systems

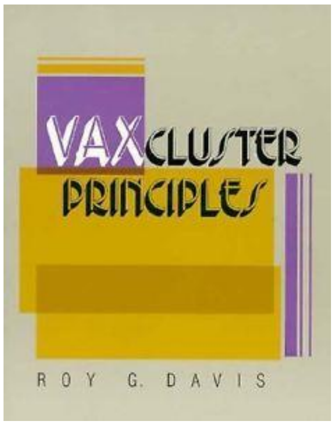
A New Dimension in Work Group Computing

Printer
VAX
Tape
Disk
VAX-station (Diskless)
VAX-station or MicroVAX
Disk
Micro-VAX (Diskless)

and

- Diskless MicroVAX II Systems
- Diskless VAXstation II/GPX Systems

Digital has it now.

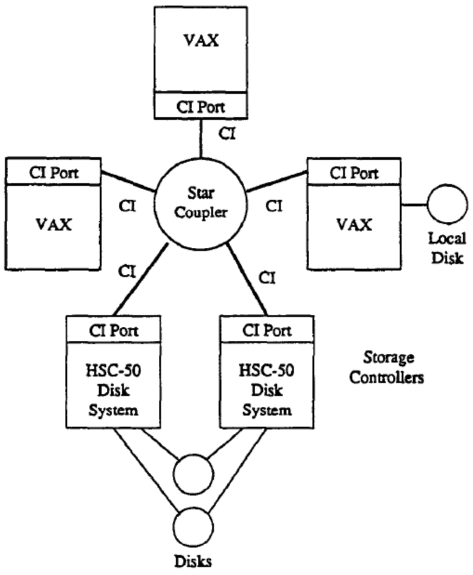


https://www.montagar.com/~patj/vhobcluster1.jpg

Electrical & Computer ENGINEERING

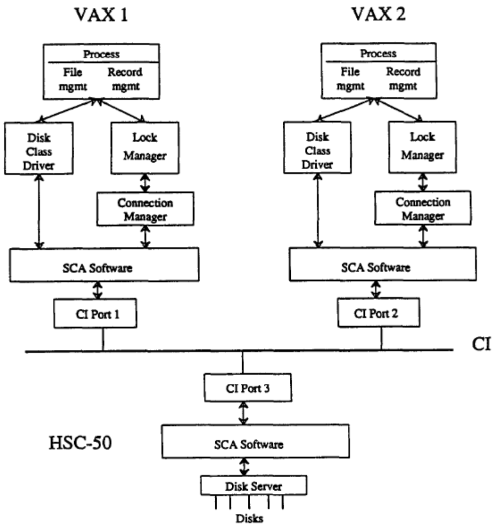
DEC VAXcluster (An Early System)

- The first commercial 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

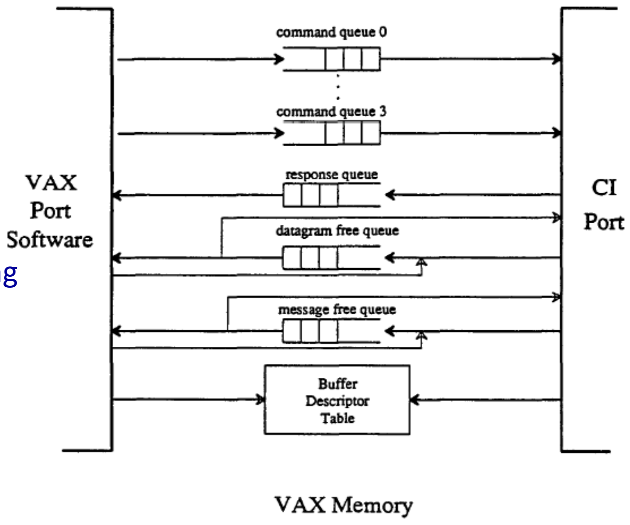


SCA (System Comm. Arch.)

- 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)

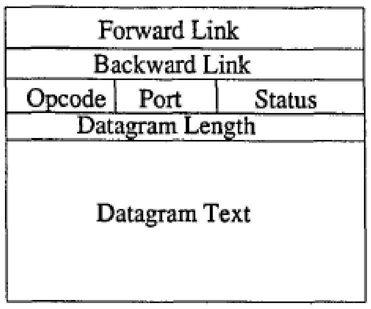
CI Architecture

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



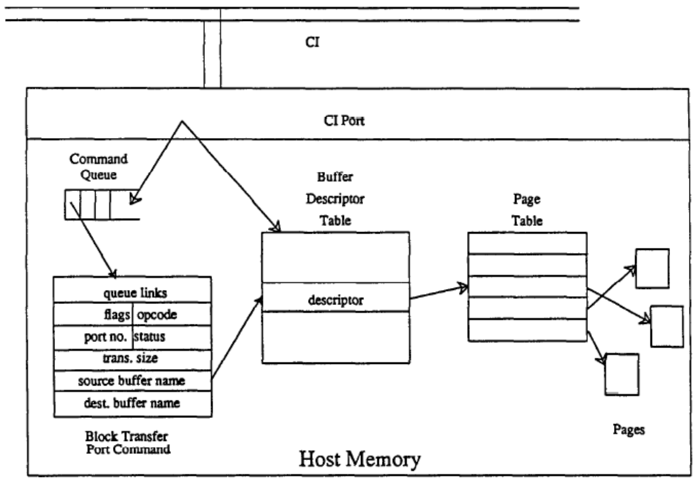
Example

- 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



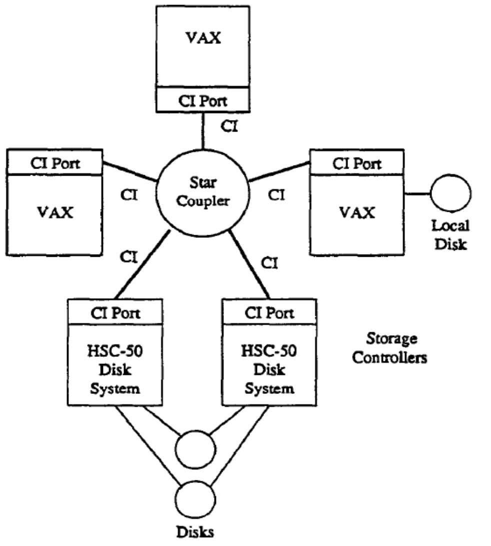
Block Transfers

- 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

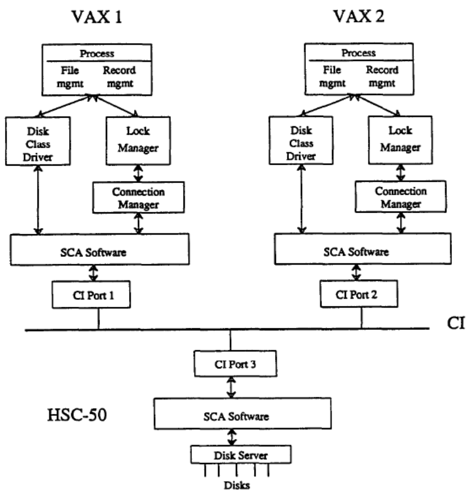
- Disk class driver sends read message to CI port on HSC disk controller
  - Request contains device type, unit number, logical disk address, requestor's buffer name, size of transfer
- Controller reads data and sends it via block transfer
- When complete, controller sends completion/status message back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

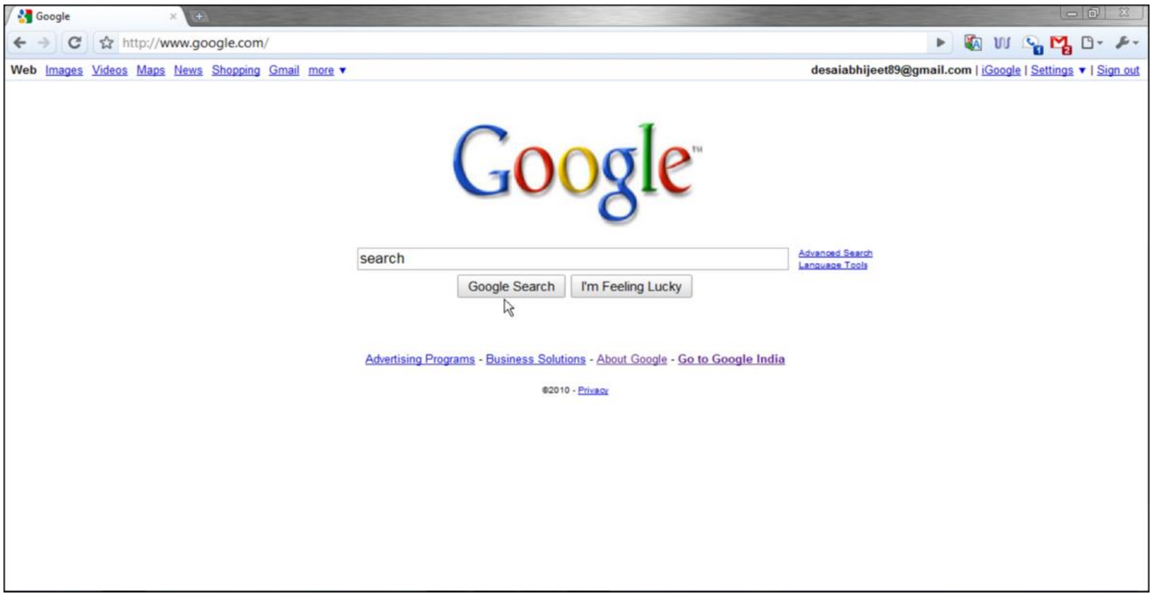
C. Google Cluster [Luiz André Barroso , Jeffrey Dean , Urs Hölzle, 2003]

Web Search For a Planet:
The Google Cluster Architecture
Luiz Andre Barroso, Jeffrey Dean, Urs Holzle
Google

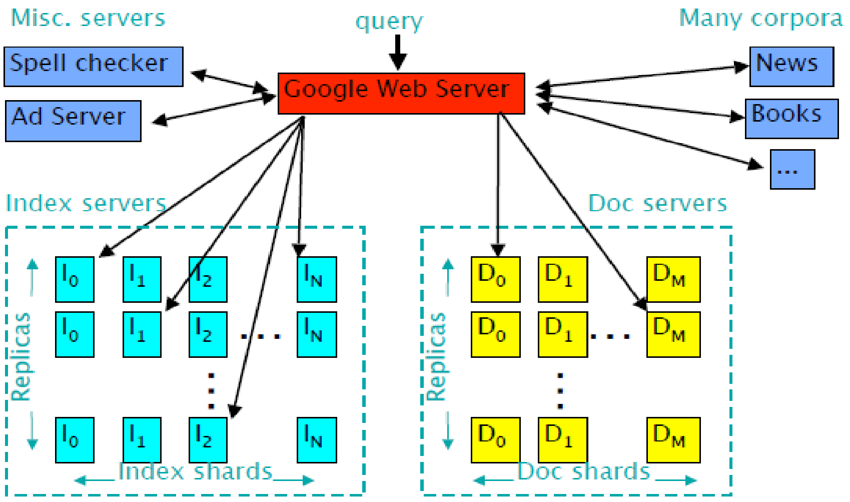
Google Cluster Architecture
Web Search for a Planet and much more….
Abhijeet Desai
desaiabhijeet89@gmail.com
Electrical & Computer ENGINEERING

Design Principles of Google Clusters

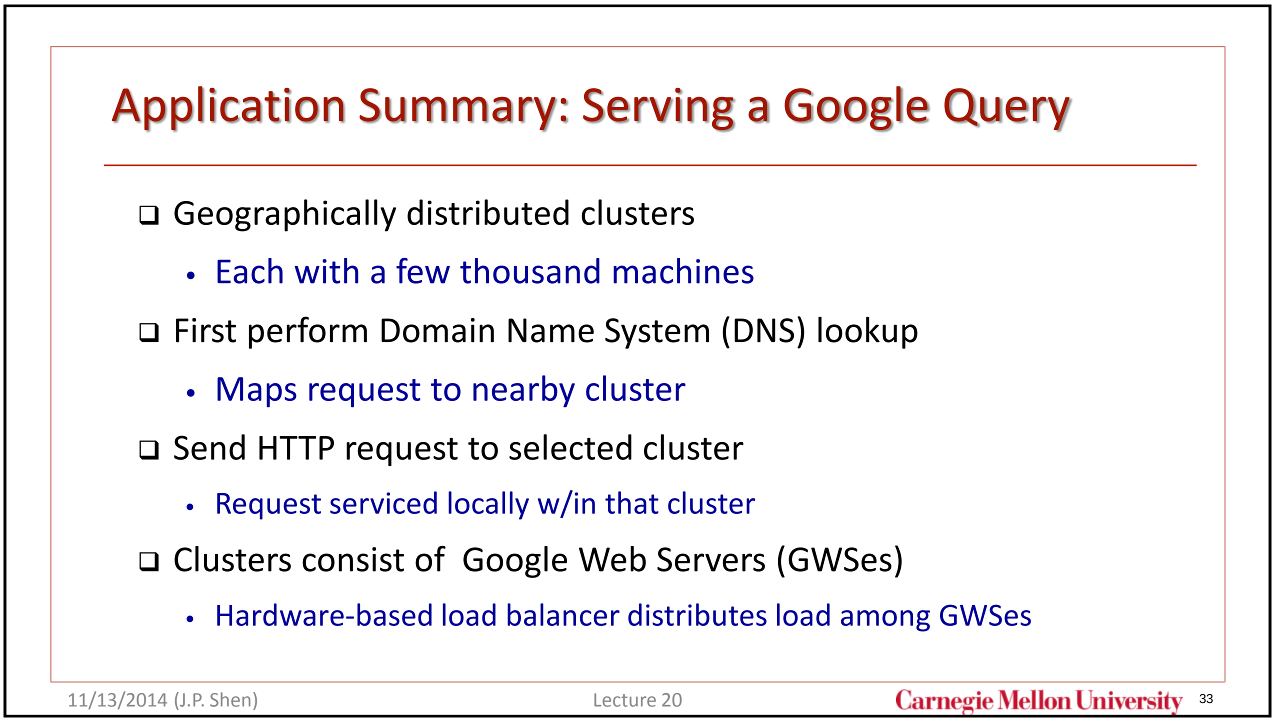
- ❑ Software level reliability
  - No fault-tolerant hardware features; e.g.
    - redundant power supplies
    - A redundant array of inexpensive disks (RAID)
- ❑ Use replication
  - for better request throughput and availability
- ❑ Price/performance beats peak performance
  - CPUs giving the best performance per unit price
  - Not the CPUs with best absolute performance
- ❑ Using commodity PCs
  - reduces the cost of computation


Google
http://www.google.com/
Web Images Videos Maps News Shopping Gmail more ▾
desaiabhijeet89@gmail.com | iGoogle | Settings ▾ | Sign out
Google
search
Advanced Search
Language Tools
Google Search
I'm Feeling Lucky
Advertising Programs - Business Solutions - About Google - Go to Google India
©2010 - Privacy

Google Query Serving Infrastructure

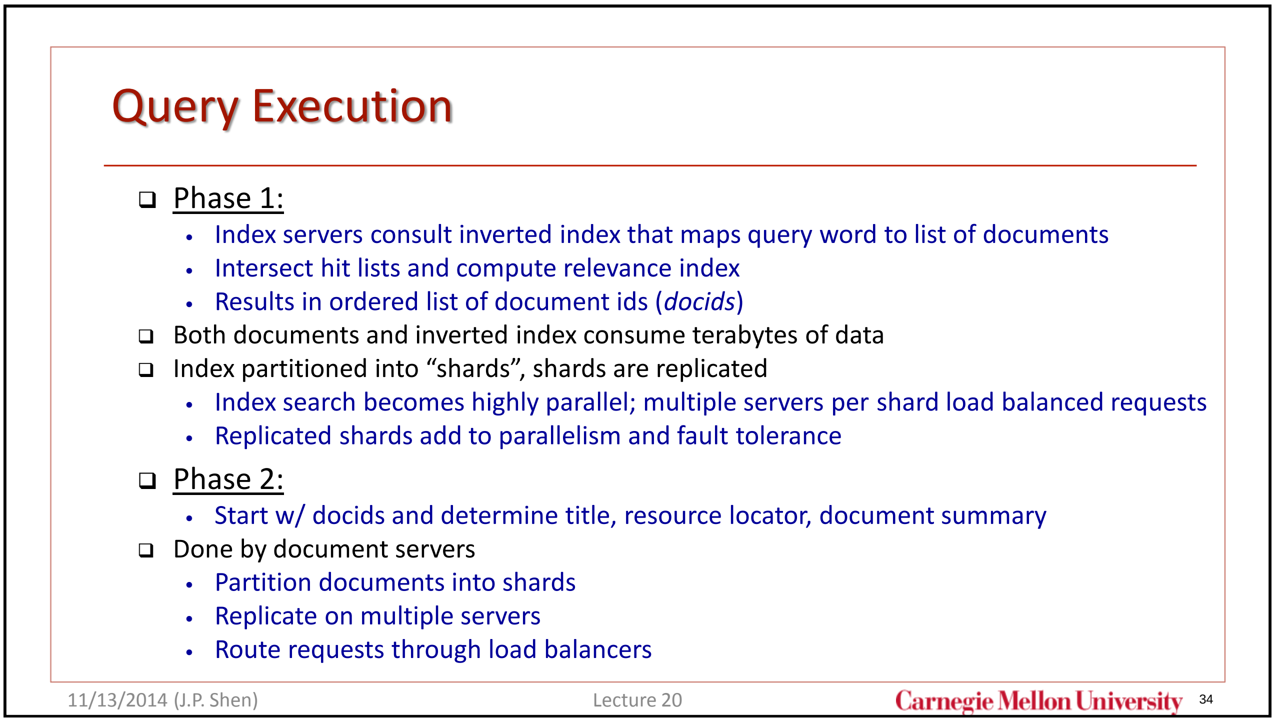


Elapsed time: 0.25s, machines involved: 1000s+



Application Summary: Serving a Google Query

- 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

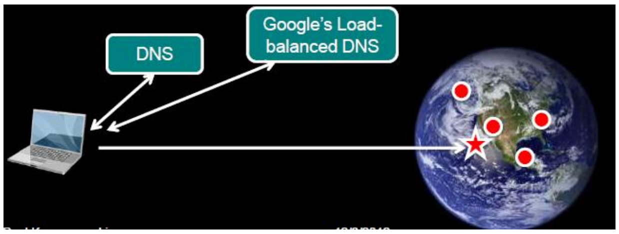


Query Execution

- 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
- 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

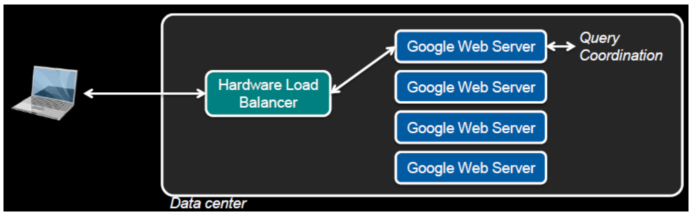
Step 1 – DNS

- User's browser must map google.com to an IP address
- "google.com" comprises
  - Multiple clusters distributed worldwide
  - Each cluster contains thousands of machines
- DNS-based load balancing
  - Select cluster by taking user's geographic proximity into account
  - Load balance across clusters
  - [similar to Akamai's approach]



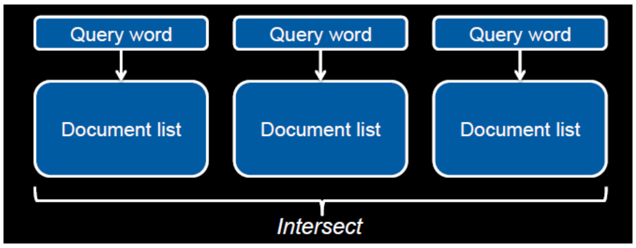
Step 2 – Send HTTP Requesst

- IP address corresponds to a load balancer within a cluster
- Load balancer
  - Monitors the set of Google Web Servers (GWS)
  - Performs local load balancing of requests among available servers
- GWS machine receives the query
  - Coordinates the execution of the query
  - Formats results into an HTML response to the user



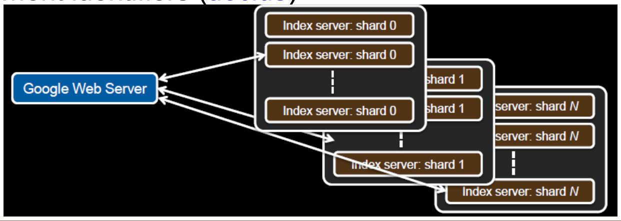
Step 3 – Find Document via Inverted Index

- Index Servers
  - Map each query word → {list of documents} (hit list)
    - Inverted index generated from web crawlers → MapReduce
  - Intersect the hit lists of each per-word query
    - Compute relevance score for each document
    - Determine set of documents
    - Sort by relevance score



Parallelizing the Inverted Index

- Inverted index is 10s of terabytes
- Search is parallelized
  - Index is divided into *index shards*
    - Each index shard is built from a randomly chosen subset of documents
    - Pool of machines serves requests for each shard
    - Pools are load balanced
  - Query goes to one machine per pool responsible for a shard
- Final result is ordered list of document identifiers (*docids*)

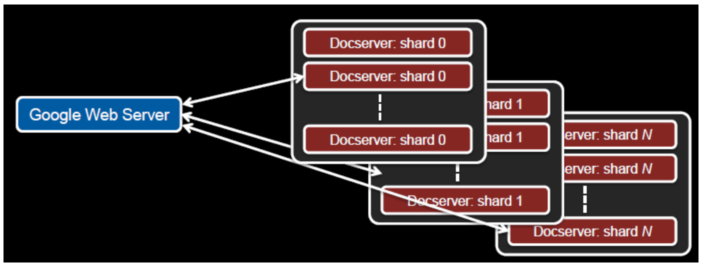


Step 4 – Get Title and URL for Each docid

- For each docid, the GWS looks up
  - Page title
  - URL
  - Relevant text: document summary specific to the query
- Handled by document servers (docservers)

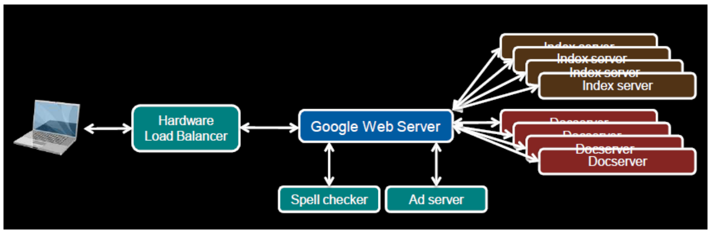
Parallelizing Document Lookup

- Like index lookup, document lookup is partitioned & parallelized
- Documents distributed into smaller shards
  - Each shard = subset of documents
- Pool of load-balanced servers responsible for processing each shard
- Together, document servers access a cached copy of the entire web!



Additional Operations

- In parallel with search:
  - Send query to a spell-checking system
  - Send query to an ad-serving system to generate ads
- When all the results are in, GWS generate HTML output:
  - Sorted query results
    - With page titles, summaries, and URLs
    - Ads
    - Spelling correction suggestions



Parallelism & Fault Tolerance

- High levels of "natural" parallelism
  - Both inter-query and intra-query; Near-linear speedups
- Parallelism:
  - Lookup of matching docs in a large index
  - Many lookups in a set of smaller indexes followed by a merge step
  - A query stream is of multiple streams each handled by a cluster
  - Adding machines to a pool increases serving capacity
- 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

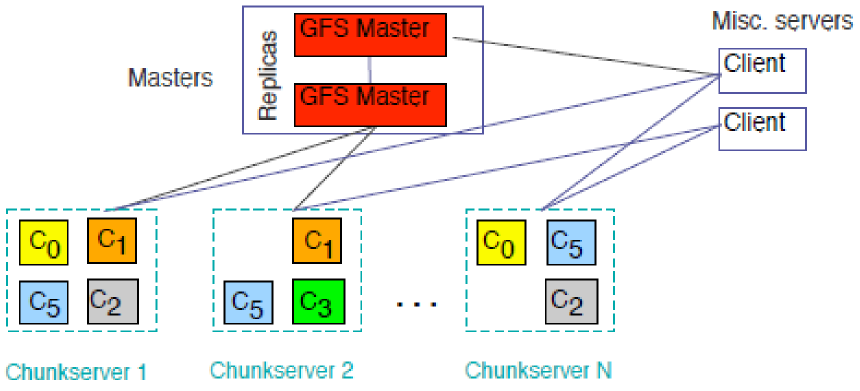
Hardware Considerations

- ❑ 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

The Power Problem

- ❑ Power is a very big issue
  - Google server consumes 400 watts/ft^2
  - High end server consumes 700 watts/ft^2
  - Typical commercial data center consumes 70-150 watts/ft^2
- ❑ Power Solutions:
  - Can reduce power but will also reduce performance
  - Lower power servers must not cost more
  - Must provide adequate cooling to the clusters
  - A very active area of research

GFS (Google File System) Design



- Master manages metadata
- Data transfers happen directly between clients/chunk servers
- Files broken into chunks (typically 64 MB)

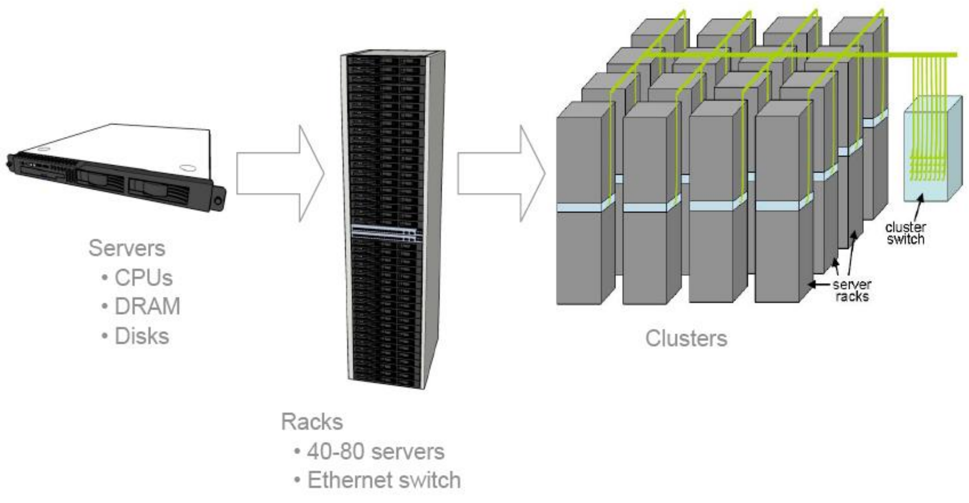
GFS Usage @ Google

- 200+ clusters
- Many clusters of 1000s of machines
- Pools of 1000s of clients
- 4+ PB Filesystems
- 40 GB/s read/write load
  – (in the presence of frequent HW failures)

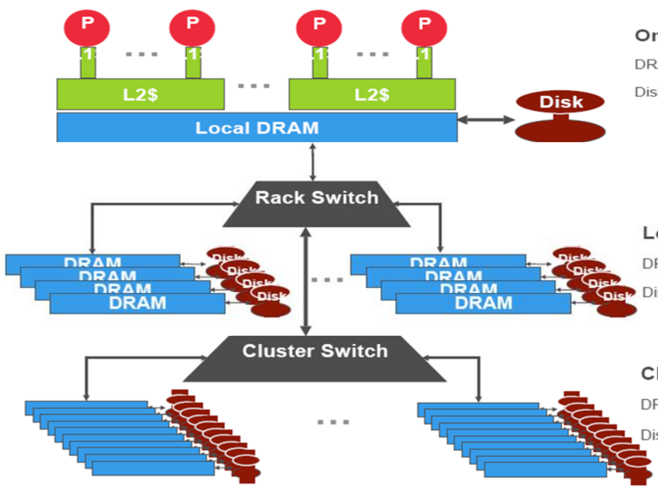
Change to Caffeine

- In 2010, Google remodeled its search infrastructure
- Old system
  - Based on MapReduce (on GFS) to generate index files
  - Batch process: next phase of MapReduce cannot start until first is complete
    - Web crawling → MapReduce → propagation
  - Initially, Google updated its index every 4 months. Around 2000, it reindexed and propagated changes every month
    - Process took about 10 days
    - Users hitting different servers might get different results
- New system, named *Caffeine*
  - Fully incremental system: Based on BigTable running on GFS2
  - Support indexing many more documents: ~100 petabytes
  - High degree of interactivity: web crawlers can update tables dynamically
  - Analyze web continuously in small chunks
    - Identify pages that are likely to change frequently
  - BTW, MapReduce is not dead. Caffeine uses it in some places, as do lots of other services.

The Machinery



Architectural View of the Storage Hierarchy



One server
DRAM: 16GB, 100ns, 20GB/s
Disk: 2TB, 10ms, 200MB/s

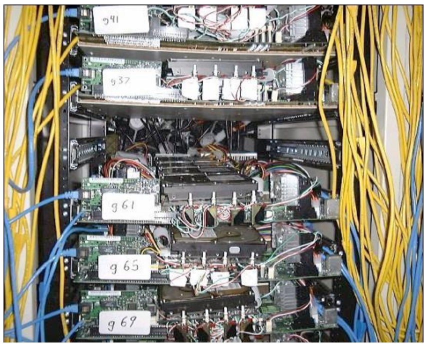
Local rack (80 servers)
DRAM: 1TB, 300us, 100MB/s
Disk: 160TB, 11ms, 100MB/s

Cluster (30+ racks)
DRAM: 30TB, 500us, 10MB/s
Disk: 4.80PB, 12ms, 10MB/s

Clusters Through the Years

“Google” Circa 1997 (google.stanford.edu)

Google (circa 1999)



Clusters Through the Years

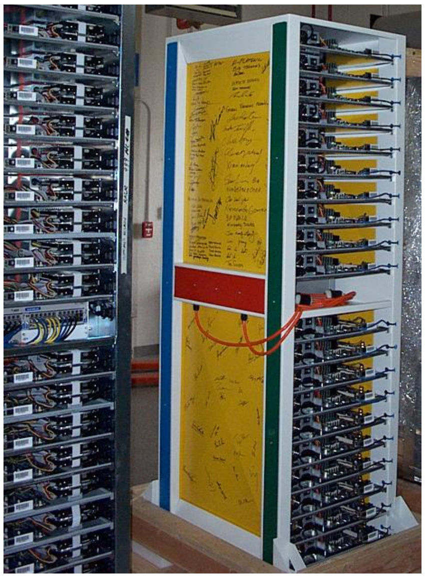
Google Data Center (Circa 2000)

Google (new data center 2001)

3 days later

Current Design

- In-house rack design
- PC-class motherboards
- Low-end storage and networking hardware
- Linux
- + in-house software



Container Datacenter

Container Datacenter

Comparison: Custom Built vs. High-end Servers

	Typical x86 - based server	Custom built x86 - based server	
PROCESSORS	8 2-GHz Xeon CPUs	176 2-GHz Xeon CPUs	22x
RAM	64 Gbytes of RAM	176 Gbytes of RAM	3x
DISK SPACE	8 Tbytes of disk space	7 Tbytes of disk space	-1 TB
PRICE	$758,000	$278,000	$480,000

The Google "Cloud"

CANADA
WASHINGTON
IDAHO
Seattle
Wenatchee
Quincy
90
Columbia R.
82
Pacific Ocean
Portland
The Dalles
84
Salem
Snake R.
5
OREGON
Miles
0
100
[NYT: June 8, 2006]

Google Cluster – Conclusions

- For a large scale web service system like Google
  - Design the algorithm which can be easily parallelized
  - Design the architecture using replication to achieve distributed computing/storage and fault tolerance
  - Be aware of the power problem which significantly restricts the use of parallelism
- Several key pieces of infrastructure for search systems:
  - GFS
  - MapReduce
  - BigTable

Google Cluster – References

1. Luiz André Barroso , Jeffrey Dean , Urs Hölzle, Web Search for a Planet: The Google Cluster Architecture, IEEE Micro, v.23 n.2, p.22-28, March 2003 [doi>10.1109/MM.2003.1196112]
2. S. Brin and L. Page, "The Anatomy of a Large-Scale Hypertextual Web Search Engine," Proc. Seventh World Wide Web Conf. (WWW7), International World Wide Web Conference Committee (IW3C2), 1998, pp. 107-117.
3. "TPC Benchmark C Full Disclosure Report for IBM eserver xSeries 440 using Microsoft SQL Server 2000 Enterprise Edition and Microsoft Windows .NET Datacenter Server 2003, TPC-C Version 5.0," http://www.tpc.org/results/FDR/TPCC/ibm.x4408way.c5.fdr.02110801.pdf.
4. D. Marr et al., "Hyper-Threading Technology Architecture and Microarchitecture: A Hypertext History," Intel Technology J., vol. 6, issue 1, Feb. 2002.
5. L. Hammond, B. Nayfeh, and K. Olukotun, "A Single-Chip Multiprocessor," Computer, vol. 30, no. 9, Sept. 1997, pp. 79-85.
6. L.A. Barroso et al., "Piranha: A Scalable Architecture Based on Single-Chip Multiprocessing," Proc. 27th ACM Int'l Symp. Computer Architecture, ACM Press, 2000, pp. 282-293.
7. L.A. Barroso, K. Gharachorloo, and E. Bugnion, "Memory System Characterization of Commercial Workloads," Proc. 25th ACM Int'l Symp. Computer Architecture, ACM Press, 1998, pp. 3-14.

D. MapReduce and Hadoop

An Introduction to MapReduce

Presented by Frane Bandov
at the Operating Complex IT-Systems seminar
Berlin, 1/26/2010

Hadoop MapReduce Fundamentals

@LynnLangit

Electrical & Computer ENGINEERING

MapReduce

- *"A simple and powerful interface that enables automatic parallelization and distribution of large-scale computations, combined with an implementation of this interface that achieves high performance on large clusters of commodity PCs."*
- More simply, MapReduce is:
  - A parallel programming model and associated implementation.

Dean and Ghermawat, "MapReduce: Simplified Data Processing on Large Clusters," Google Inc.

Programming Models

- Parallel Programming Models
  - Message Passing
    - Independent tasks encapsulating local data
    - Tasks interact by exchanging messages
  - Shared Memory
    - Tasks share a common address space
    - Tasks interact by reading and writing this space asynchronously
  - Data Parallelization
    - Tasks execute a sequence of independent operations
    - Data usually evenly partitioned across tasks
    - Often referred to as "Embarrassingly parallel"

MapReduce

- A simple programming model that applies to many large-scale computing problems
- Hide messy details in MapReduce runtime library:
  - automatic parallelization
  - load balancing
  - network and disk transfer optimizations
  - handling of machine failures
  - robustness
  - **improvements to core library benefit all users of library!**

MapReduce: Programming Model

- Process data using special map() and reduce() functions
  - The map() function is called on every item in the input and emits a series of intermediate key/value pairs
  - All values associated with a given key are grouped together
  - The reduce() function is called on every unique key, and its value list, and emits a value that is added to the output

Google MapReduce - Idea

Map:
Apply function to all elements of a list

```
square x = x * x;
map square [1, 2, 3, 4, 5];
```

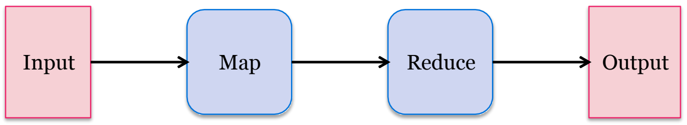
→ [1, 4, 9, 16, 25]

Reduce:
Combine all elements of a list

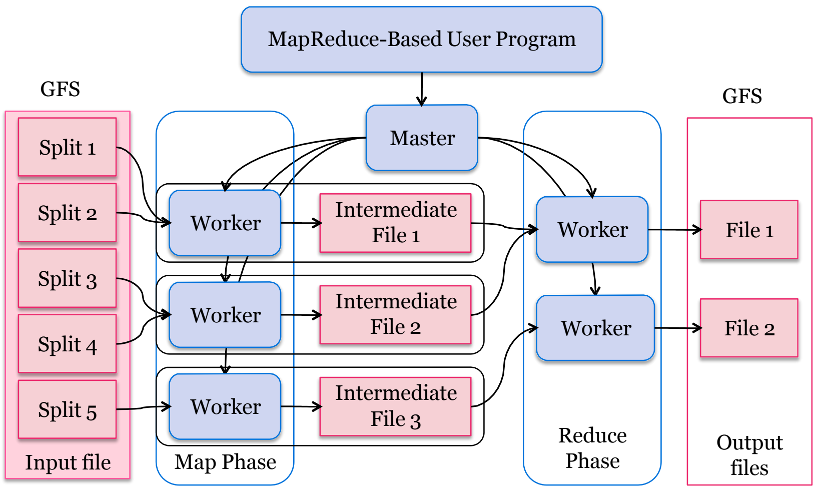
```
reduce (+)[1, 2, 3, 4, 5];
```

→ 15

Basic functioning



Google MapReduce - Overview



MapReduce Job – Logical View

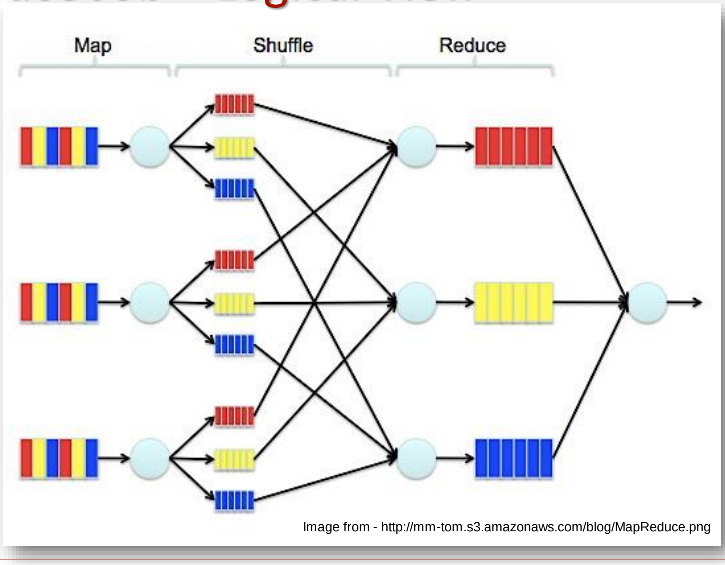
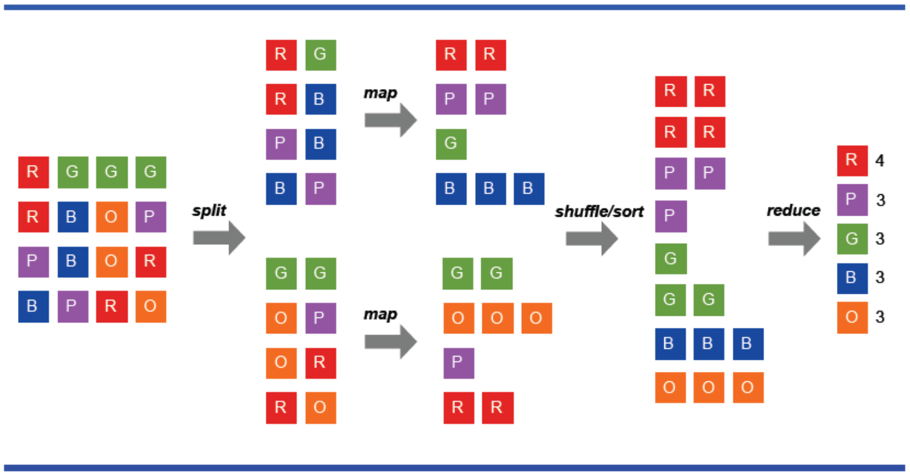


Image from - http://mm-tom.s3.amazonaws.com/blog/MapReduce.png

MapReduce: Colored Square Counting Example



MapReduce: WordCount Example

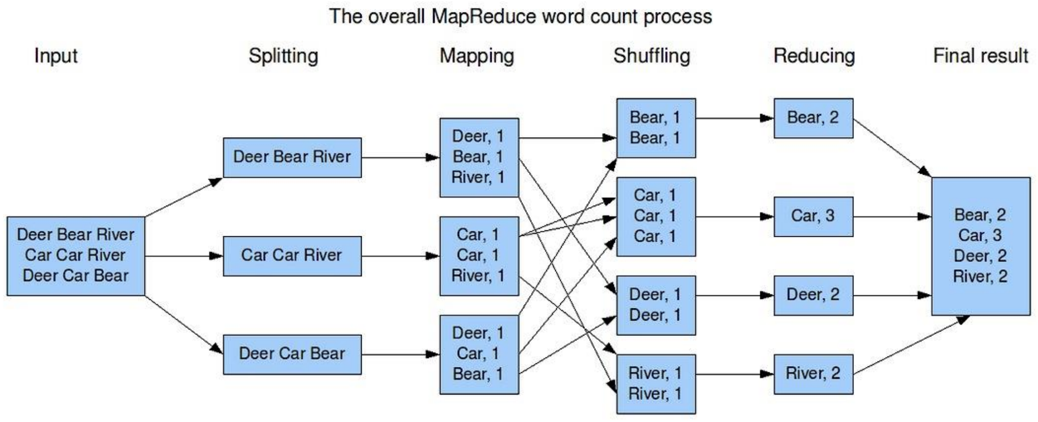


Image from: http://blog.jteam.nl/wp-content/uploads/2009/08/MapReduceWordCountOverview1.png

MapReduce – Fault Tolerance

- Workers are periodically pinged by master
- No answer over certain time → worker failed
- ➢ Mapper fails:
  - – Reset map job as *idle*
  - – Even if job was completed → intermediate files are inaccessible
  - – Notify reducers where to get the new intermediate file
- ➢ Reducer fails:
  - – Reset its job as *idle*
- ➢ Master fails:
  - – Periodically sets checkpoints
  - – In case of failure MapReduce-Operation is aborted
  - – Operation can be restarted from last checkpoint

Google MapReduce - GFS

Google File System

- In-house distributed file system at Google
- Stores all input an output files
- Stores files…
  - – divided into 64 MB blocks
  - – on at least 3 different machines
- Machines running GFS also run MapReduce

Alternative Implementation

Apache Hadoop

- Open-Source-Implementation in Java
- Jobs can be written in C++, Java, Python, etc.
- Used by Yahoo!, Facebook, Amazon and others
- Most commonly used implementation
- HDFS as open-source-implementation of GFS
- Can also use Amazon S3, HTTP(S) or FTP
- Extensions: Hive, Pig, HBase

MapReduce with Hadoop

Now, let's take a closer look at how Hadoop implements MapReduce from [White, 2011]. . .

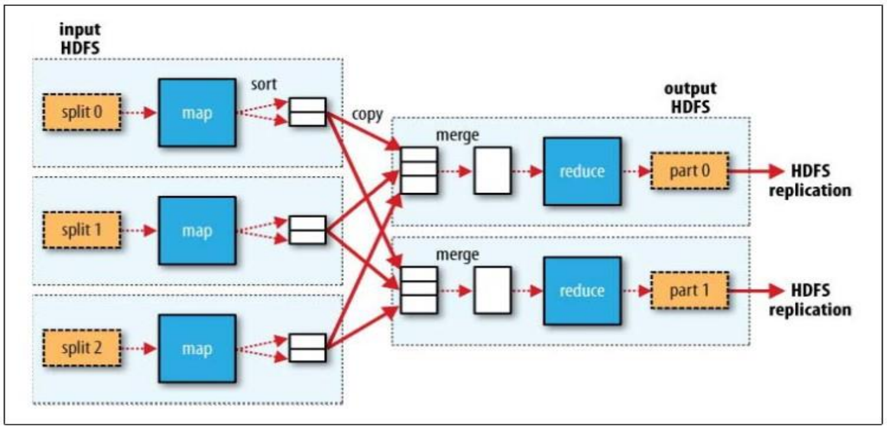
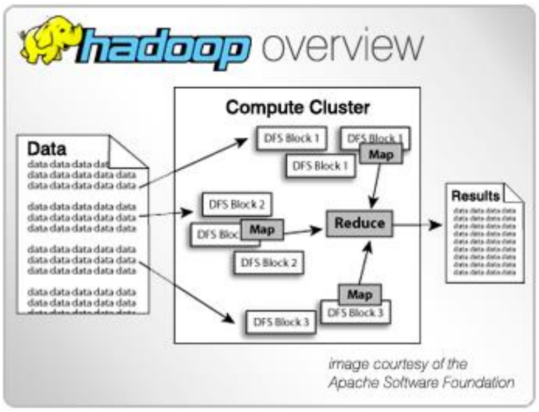


Figure 2-3. MapReduce data flow with multiple reduce tasks

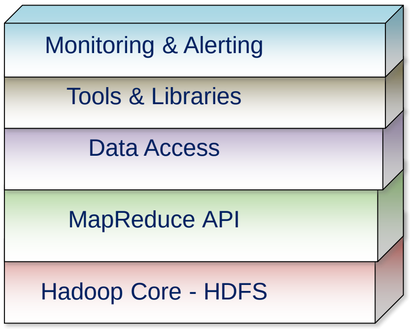
What is Hadoop?

- Open-source data storage and processing API
- Massively scalable, automatically parallelizable
  - Based on work from Google
    - GFS + MapReduce + BigTable
  - Current Distributions based on Open Source and Vendor Work
    - Apache Hadoop
    - Cloudera – CH4 w/ Impala
    - Hortonworks
    - MapR
    - AWS
    - Windows Azure HDInsight



Hadoop is a Set of Apache Frameworks and more…

- Data storage (HDFS)
  - Runs on commodity hardware (usually Linux)
  - Horizontally scalable
- Processing (MapReduce)
  - Parallelized (scalable) processing
  - Fault Tolerant
- Other Tools / Frameworks
  - Data Access: HBase, Hive, Pig, Mahout
  - Tools: Hue, Sqoop
  - Monitoring: Greenplum, Cloudera



More Alternative Implementations

Mars

- MapReduce-Implementation for nVidia GPU using the CUDA framework

MapReduce-Cell

- Implementation for the Cell multi-core processor

Qizmt

- MySpace's implementation of MapReduce in C#

Reception and Criticism

- Yahoo!: Hadoop on a 10,000 server cluster
- Facebook analyses the daily log (25TB) on a 1,000 server cluster
- Amazon Elastic MapReduce: Hadoop clusters for rent on EC2 and S3
- IBM and Google: Support university courses in distributed programming
- UC Berkley announced to teach freashmen programming MapReduce

Reception and Criticism

Criticism mainly by RDBMS experts DeWitt and Stonebraker

- MapReduce
  - is a step backwards in database access
  - is a poor implementation
  - is not novel
  - is missing features that are routinely provided by modern DBMSs
  - is incompatible with the DBMS tools

Response to criticism

- MapReduce is no RDBMS
- It suits well for processing and structuring huge amounts of unstructured data
- MapReduce's big innovation is that it enables distributing data processing across a network of cheap and possibly unreliable computers

More MapReduce Developer Resources

- Based on the distribution – on premises
  - Apache
    - MapReduce tutorial - http://hadoop.apache.org/docs/r1.0.4/mapred_tutorial.htmlCloudera
  - Cloudera
    - Cloudera University - http://university.cloudera.com/
    - Cloudera Developer Course (4 day) - *RECOMMENDED* - http://university.cloudera.com/training/apache_hadoop/developer.html
  - Hortonworks
  - MapR
- Based on the distribution – cloud
  - AWS MapReduce
    - Tutorial - http://aws.amazon.com/elasticmapreduce/training/#gs
  - Windows Azure HDInsight
    - Tutorial - http://www.windowsazure.com/en-us/manage/services/hdinsight/using-mapreduce-with-hdinsight/
    - More resources - http://www.windowsazure.com/en-us/develop/net/tutorials/intro-to-hadoop/