

# CSE 490h, Autumn 2008

## Scalable Systems: Design, Implementation and Use of Large Scale Clusters



Instructors: Ed Lazowska & Aaron Kimball

TA: Slava Chernyak



# Content



- The fun stuff from
  - Programming Languages
  - Architecture
  - Operating Systems
  - Networking
  - Databases
  - Parallel Programming
- It doesn't obviate those courses!
  - There is a huge amount of important material that we won't cover

# Format



- Like a graduate course
  - Readings from the literature for each class session, *which you really must do*
  - Many superb guest speakers
    - | Jeff Dean
    - | Werner Vogels
    - | Mike Cafarella
    - | Steve Gribble
    - | Phil Bernstein
    - | Barry Brumitt
    - | James Hamilton
    - | Atul Adya
    - | perhaps others
  - No quiz section

# Consequence of content and format



- Things will be a wee bit chaotic
  - There will be pieces of background that are missing
    - | You need to stop us and make us fill it in!
  - Topics won't always be presented in the ideal order
    - | We sought the ideal guest speaker for each topic, rather than the person who could be there on the ideal day
  - There will be some "glue" missing
    - | We'll spend portions of our own class sessions filling in these gaps
  - We'll get you programming right away
    - | The rest of the course will be about how it works - what's under the covers


# Requirements



- Active participation
- Several programming assignments on our Hadoop cluster
- One programming assignment using Amazon Web Services
- A final exam focused on the readings and class sessions
- No killer project
  - We'll offer a project course next quarter, which we hope you'll take

# What's a large-scale cluster?

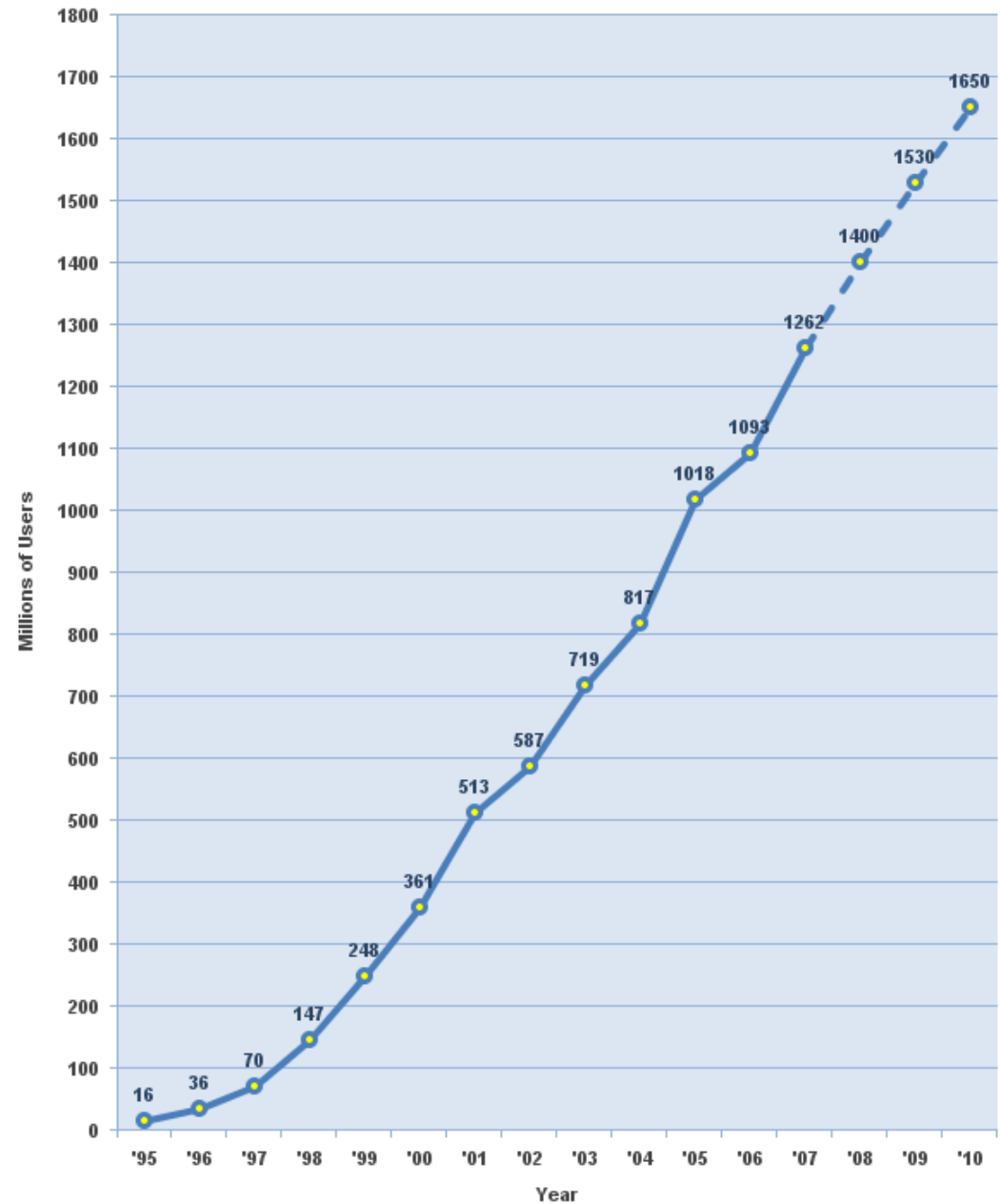
- In September 2007, Google used 11,081 "machine-years" (roughly, CPU-years) on MapReduce jobs alone
  - If the machines were busy 100% of the time (they're not), and if Google did nothing but run MapReduce jobs (this is in fact a modest proportion of Google's workload - MapReduce is used to build the index, but not to process the searches), that would be 132,972 machines!
  - It's hard to believe they don't have **1,000,000 machines**
  - A rack holds 176 CPUs (88 1U dual-processor boards), so that's about **6,000 racks**
  - A rack requires about 50 square feet (given datacenter cooling capabilities), so that's about **300,000 square feet** of machine room space (more than 6 football fields of real estate - although of course Google divides its machines among dozens of datacenters all over the world)

- 
- A rack requires about 10kw to power, and about the same to cool, so that's about 120,000 kw of power, or nearly **100,000,000 kwh per month** (\$10 million at \$0.10/kwh)
    - For comparison, annual power consumption - around 600M kwh - is equivalent to about 20% of Seattle City Light's generating capacity
  - Microsoft and Yahoo! are in the same business
  - Amazon.com is in a similar business

We're drowning  
in users



Internet Users in the World  
Growth 1995 - 2010



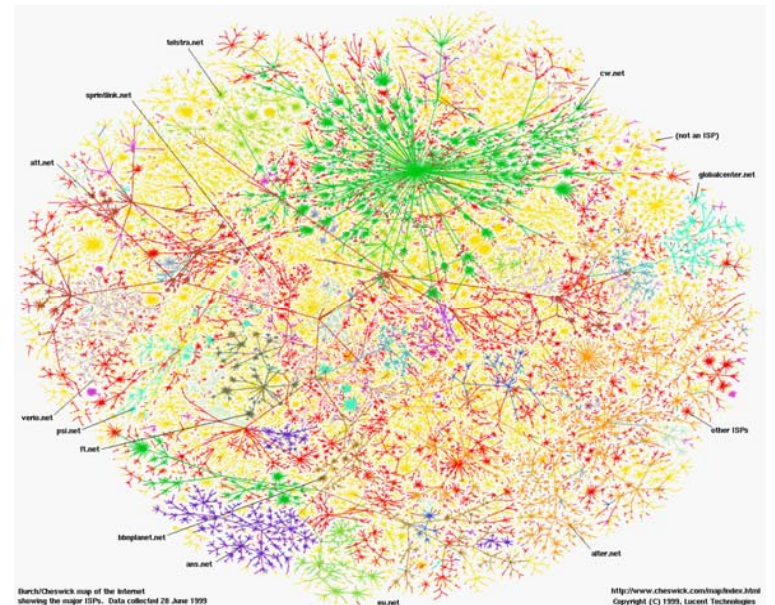
Source: [www.internetworldstats.com](http://www.internetworldstats.com) - January, 2008  
Copyright © 2008, Miniwatts Marketing Group



# We're drowning in data

## ■ The web

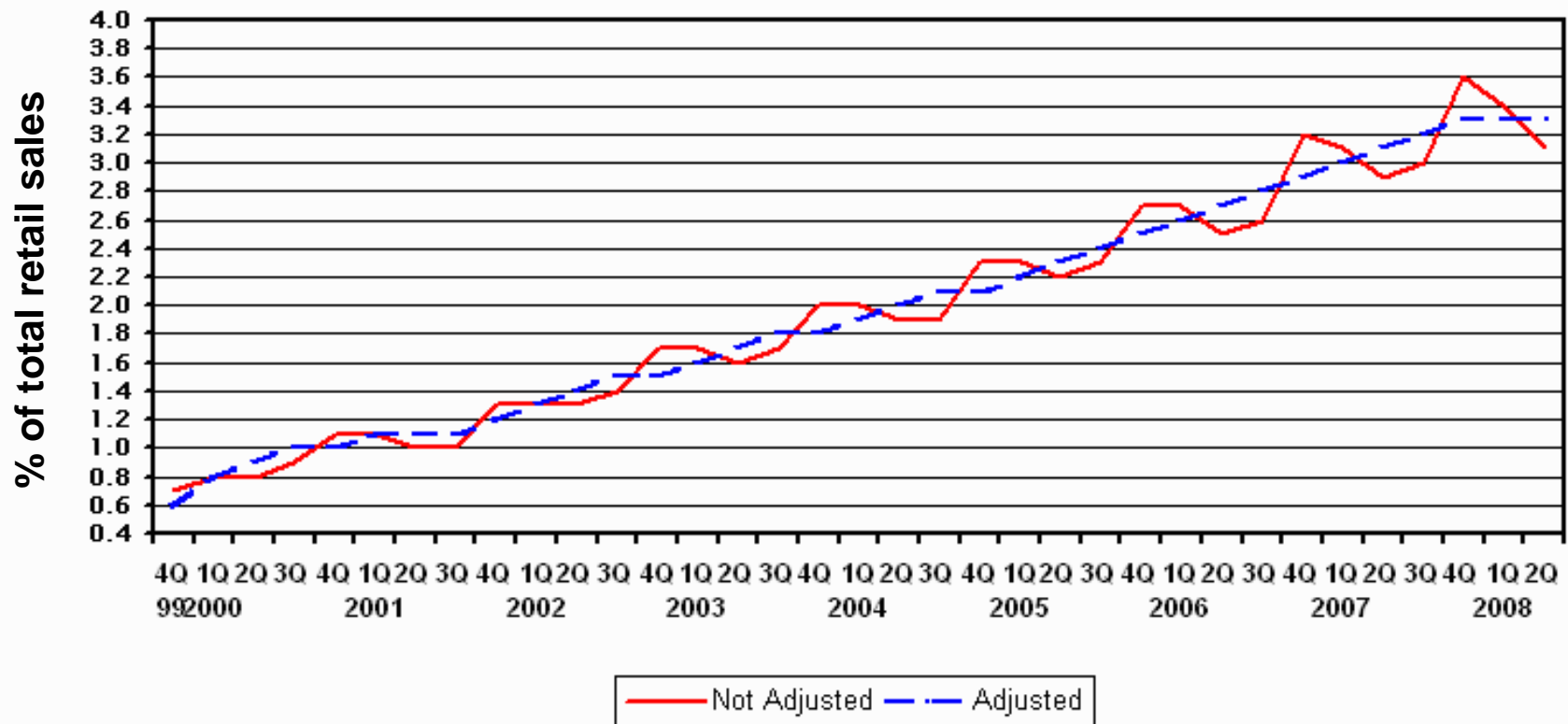
- 20+ billion web pages  $\times$  20KB = 400+ terabytes
  - | 400,000,000,000,000 (that's a lot of zeroes!)
  - | One computer can read 30-35 MB/sec from disk
  - | ~four months to read the web
  - | ~1,000 hard drives to store the web





## E-Commerce

~\$35B / quarter in the US



## ■ Sensors

- Point-of-sale terminals
- Gene sequencing machines
- Modern telescopes
- Large Hadron Collider



# We need to service those users and analyze that data

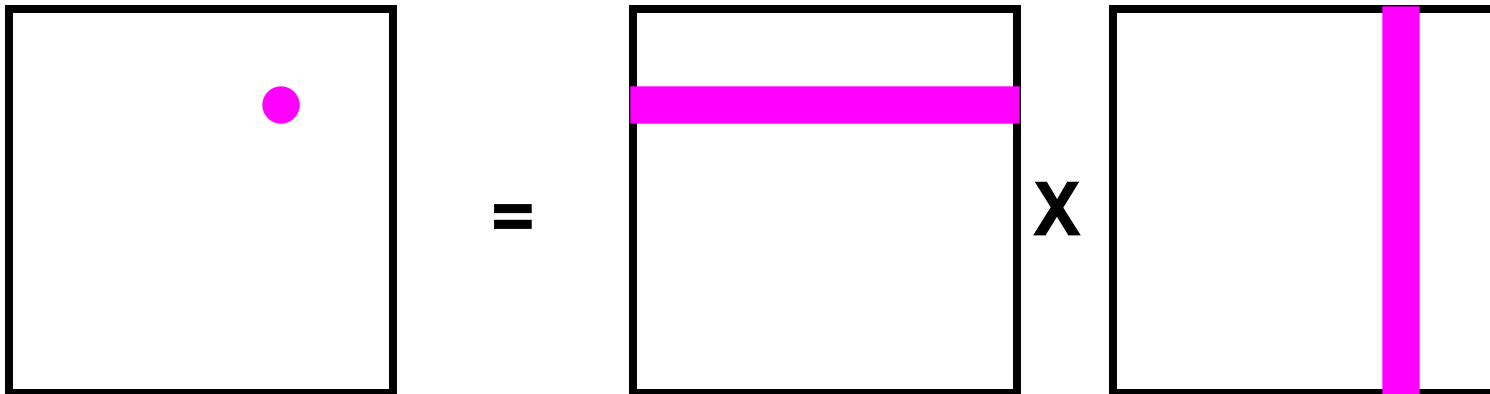
---

- Google not only stores (multiple copies of) the web, it handles an average of 3000 searches per second (7 billion searches per month)!
- The LHC will produce 700 MB of data per second - 60 terabytes per day - 20 petabytes per year
  - Hopefully they're going to analyze this data, because it cost \$6 billion to build the sucker
- **The only hope:** concurrent processing / parallel computing / distributed computing, at enormous scale

# Traditional parallel computing

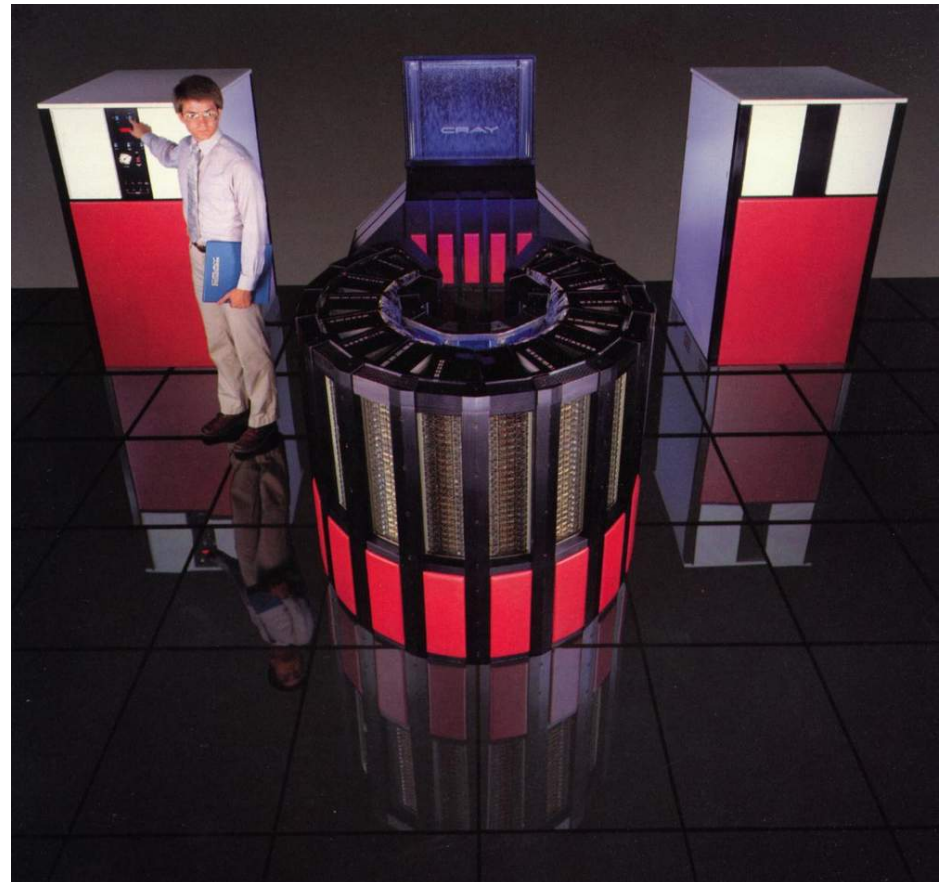
- Consider a really stupid example: multiplying two  $N \times N$  matrices for huge  $N$

- $C[i,j] = \sum_{k=1..N} A[i,k] * B[k,j]$



## ■ SIMD / vector processors

- Vector programming model
- Performance and reliability through hugely expensive components



## ■ MIMD / shared-memory multiprocessors

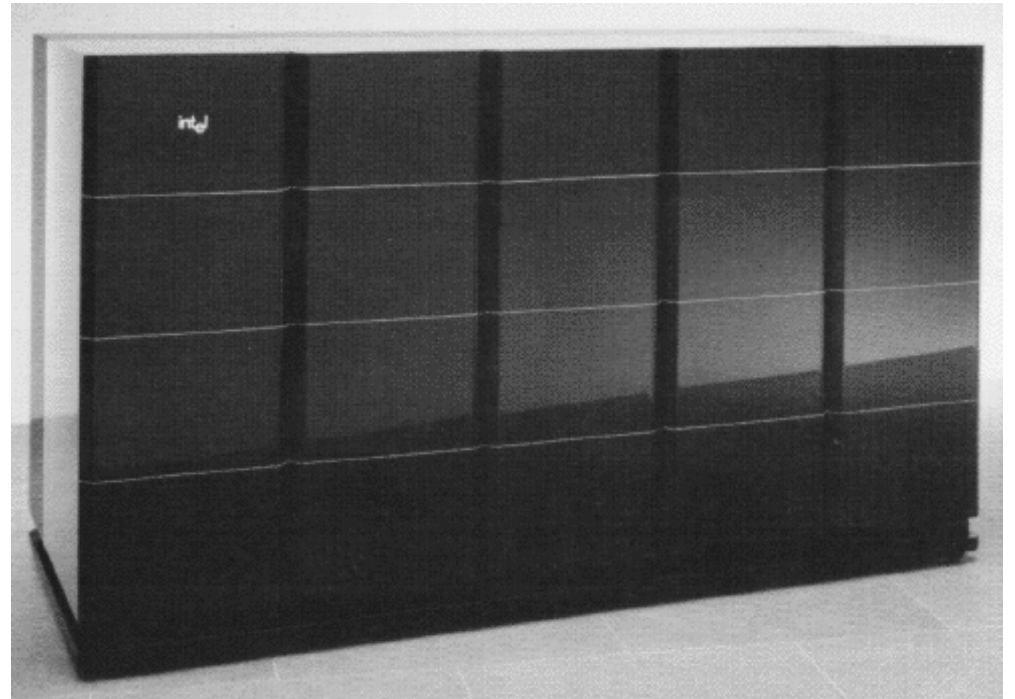
- Shared memory programming model
- High-performance interconnect providing cache coherence
- Single copy of the OS
- All CPUs are equally "close" to all memory and disks





## ■ MIMD / message-passing multiprocessors

- Message-passing programming model
- Multiple copies of the OS
- High-performance interconnect
- Typically, processors do not have local disk storage

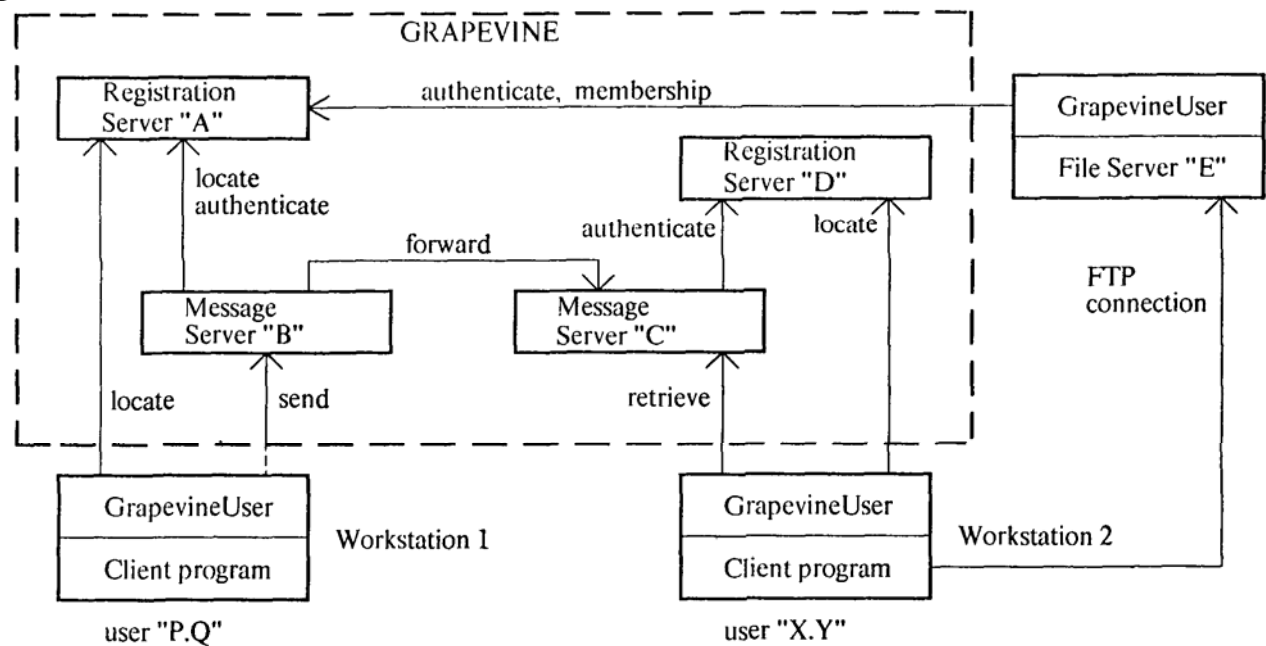




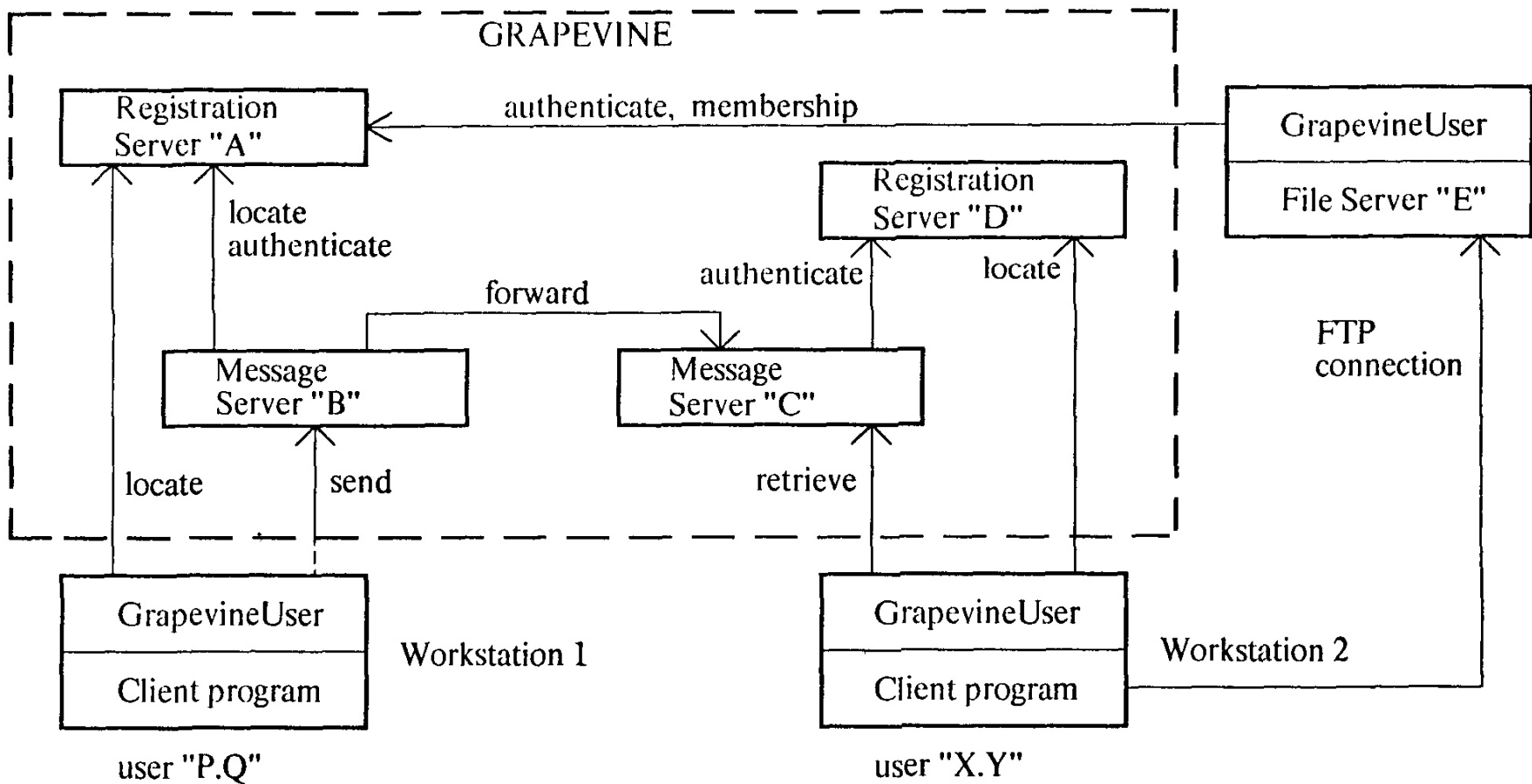
# Traditional distributed computing

## Client/server

- Lots of concurrency, but no attempt to speed up a single app
- Cheap hardware; modest inter-machine bandwidth
- Reliability via software
- Highly scalable



## Let's talk about reliability



# Is there a middle ground - best of both worlds?



- Speeds up the solution of extremely large problems/applications
- Uses a distributed system structure
  - Highly scalable
  - Cheap components
  - Reliability achieved via software
  - Modest interconnect bandwidth
- Simple programming model

# We need problems that are "embarrassingly parallel"



- We're dealing with thousands of independent computers with local memory and local disk (cheesy disks - not even SCSI)
- Interconnection bandwidth is significant, but not monstrous - commodity fast Ethernet switches
- Thus, communication is relatively expensive - the problem must be decomposable into pieces that are pretty much independent of one another
- Can you think of some problems that are like this?



SETI  HOME

# Rosetta@home

Protein Folding, Design, and Docking



## What is Rosetta@home?



**Rosetta@home** needs your help to determine the 3-dimensional shapes of proteins in research that may ultimately lead to finding cures for some major human diseases. By running the Rosetta program on your computer while you don't need it you will help us speed up and extend our research in ways we couldn't possibly attempt without your help. You will also be helping our efforts at designing new proteins to fight diseases such as HIV, Malaria, Cancer, and Alzheimer's (See our [Disease Related Research](#) for more information). Please [join us](#) in our efforts! **Rosetta@home is not for profit.**

 Site search

### Join Rosetta@home

1. [Rules and policies](#)
2. [System requirements](#)
3. [Download, install, and run BOINC](#)  
(enter the project URL: <http://boinc.bakerlab.org/rosetta/>)
4. [A welcome from David Baker](#)

### About

- [10 reasons why users crunch Rosetta@home](#)
- [Quick Guide to Rosetta@home and Its Graphics](#)
- [Play the interactive rosetta game, FoldIt!](#)
- [Rosetta@home FAQ](#)
- [Rosetta@home Science FAQ](#)
- [Disease Related Research](#)
- [Research Overview](#)
- [News & Articles about Rosetta](#)
- [David Baker's Rosetta@home Journal](#)
- [Rosetta@home promo video](#)
- [Technical news](#)

### Returning participants

### User of the day



[Jamison Kingfield](#)

### Server Status as of 21 Sep 2008 22:55:54 UTC

[ Scheduler running ] Queued: 18,682  
 In progress: 402,920  
 Successes last 24h: 231,583  
 Users (last day ) : 215,789 (+109)  
 Hosts (last day ) : 618,568 (+329)  
 Credits last 24h : 7,873,217  
 Total credits : 4,864,673,040  
 TeraFLOPS estimate: 78.732

Sep 21, 2008

**Predictor of the day:** Congratulations to [C G Edwards](#) (Team [EDS](#)) for predicting the lowest energy structure for workunit `1opd__JUMPRELAX_PREFIX_PREFRAG_SAVE_ALL_OUT-1opd_-_4237_0!`

[...more](#)

Available as an [RSS feed](#)..

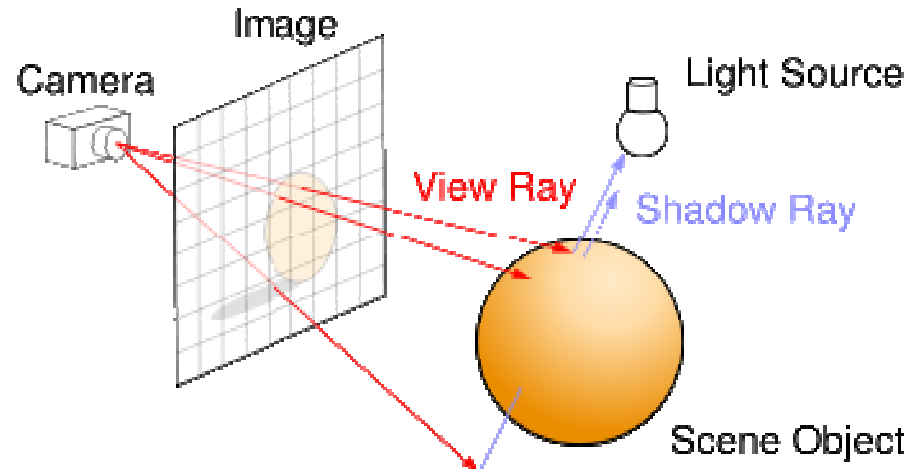
### News

Sept 14, 2008  
We are currently participating to the first competition for modeling the structure of an important biological receptor. This protein is much larger than all the targets submitted



## ■ Ray tracing in computer graphics

- Render a scene by tracing rays from the eye to objects in the scene
  - Each ray is independent

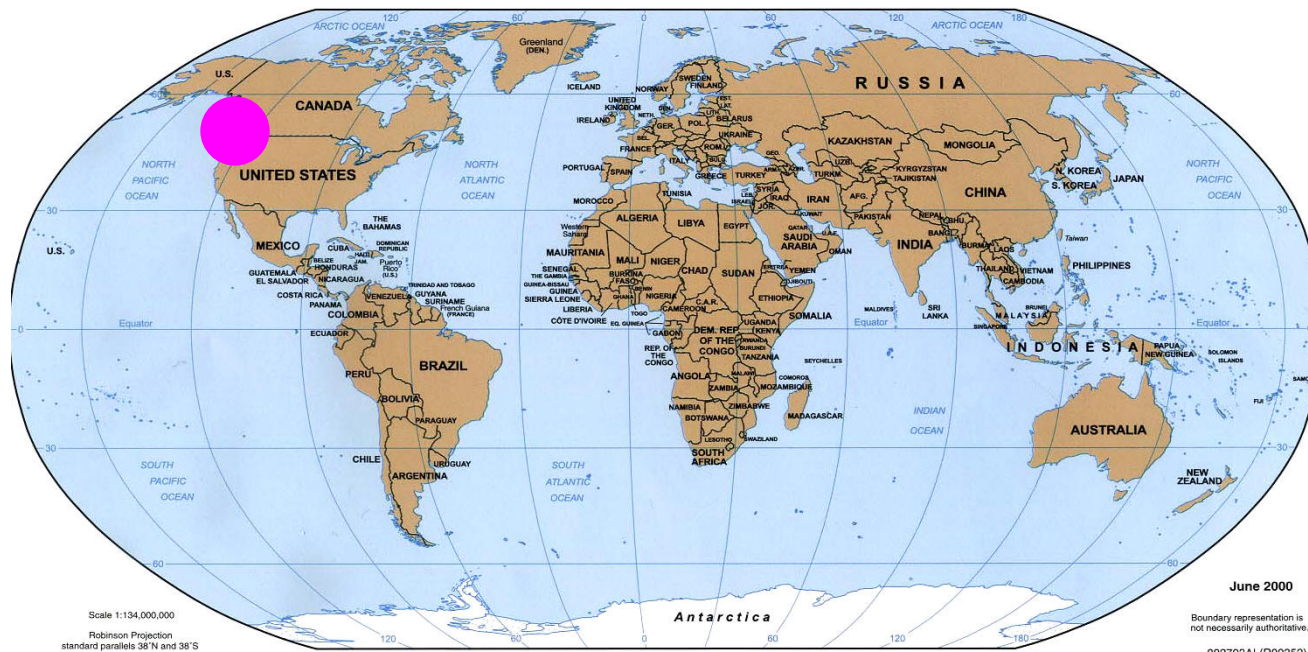


## ■ Compute a web index

- For each of 20+ billion web pages, make a list of the words it contains
- Invert this index - for each of however many words there are, make a list of the web pages on which it appears

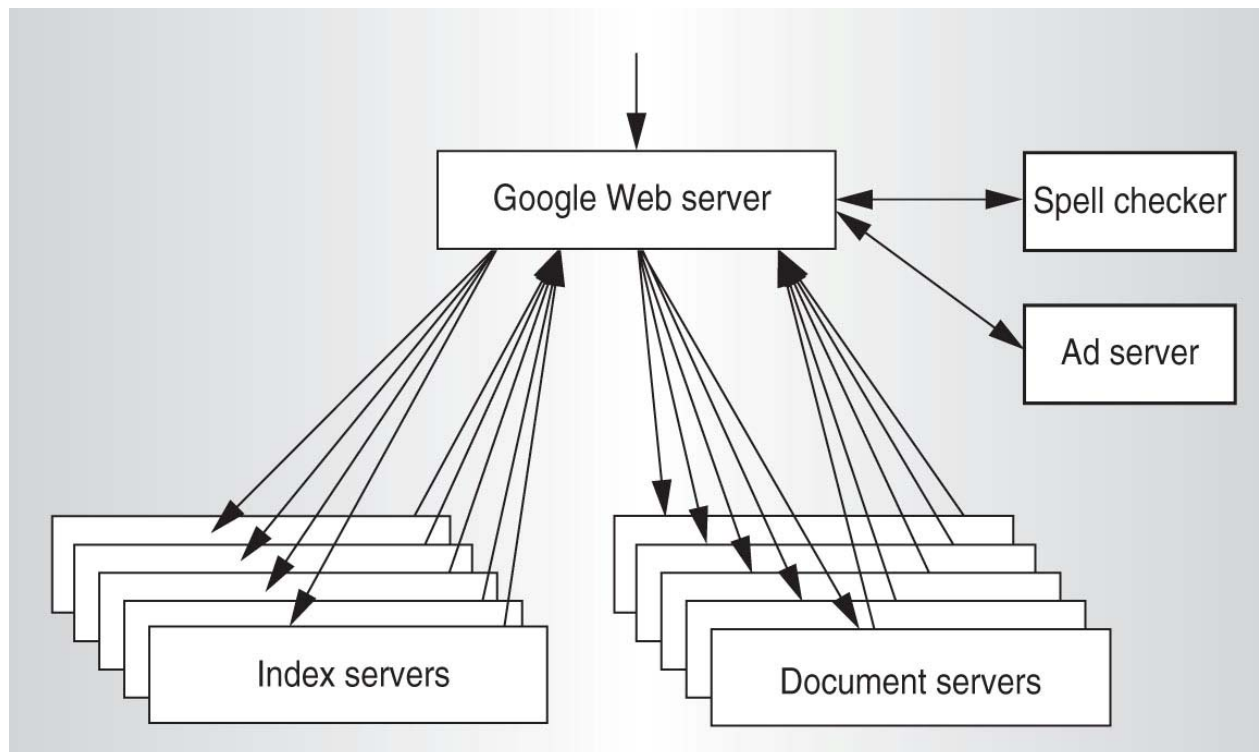
## ■ Answering a Google search request - a *single* search request

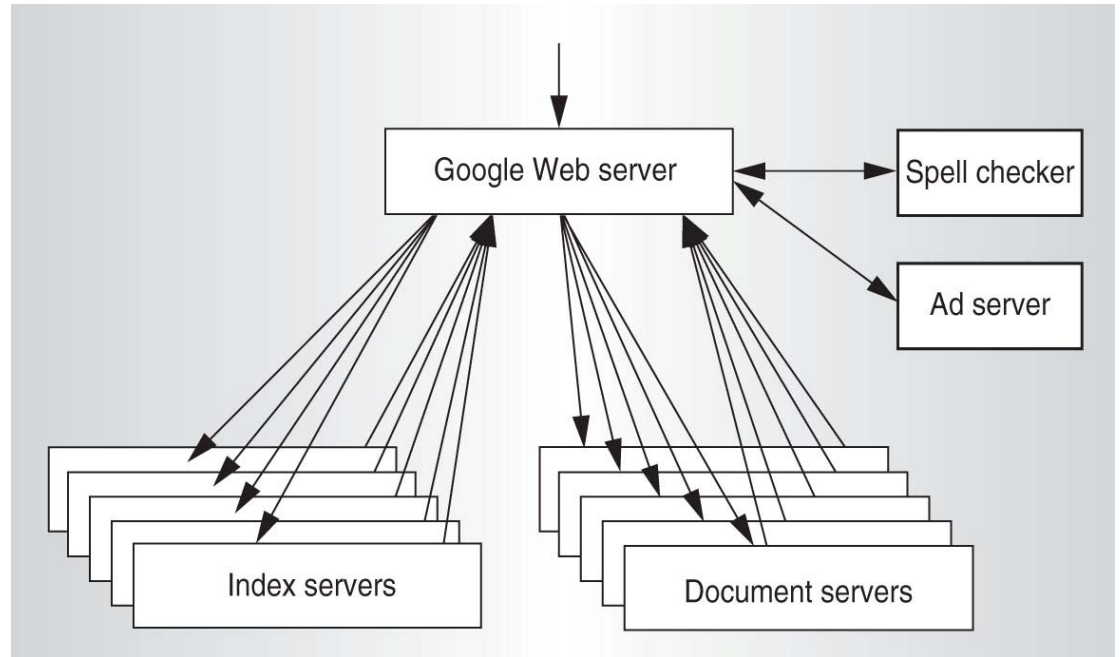
- There are multiple clusters (of thousands of computers each) all over the world
- DNS routes your search to a nearby cluster



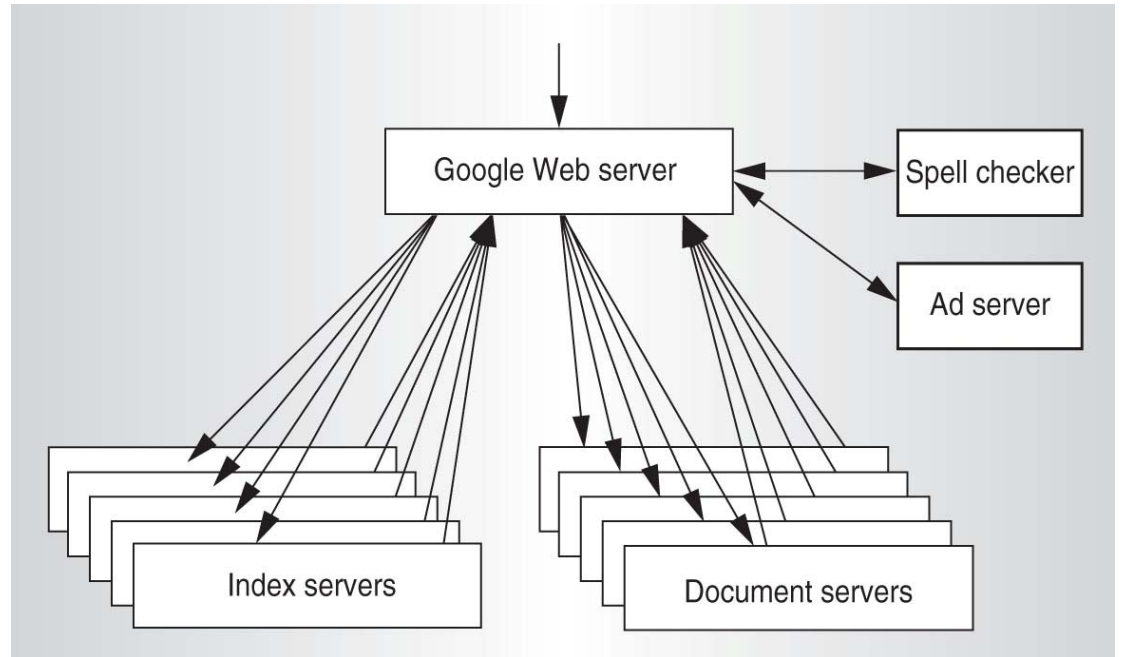


- A cluster consists of Google Web Servers, Index Servers, Doc Servers, and various other servers (ads, spell checking, etc.)
- These are cheap standalone computers, rack-mounted, connected by commodity networking gear

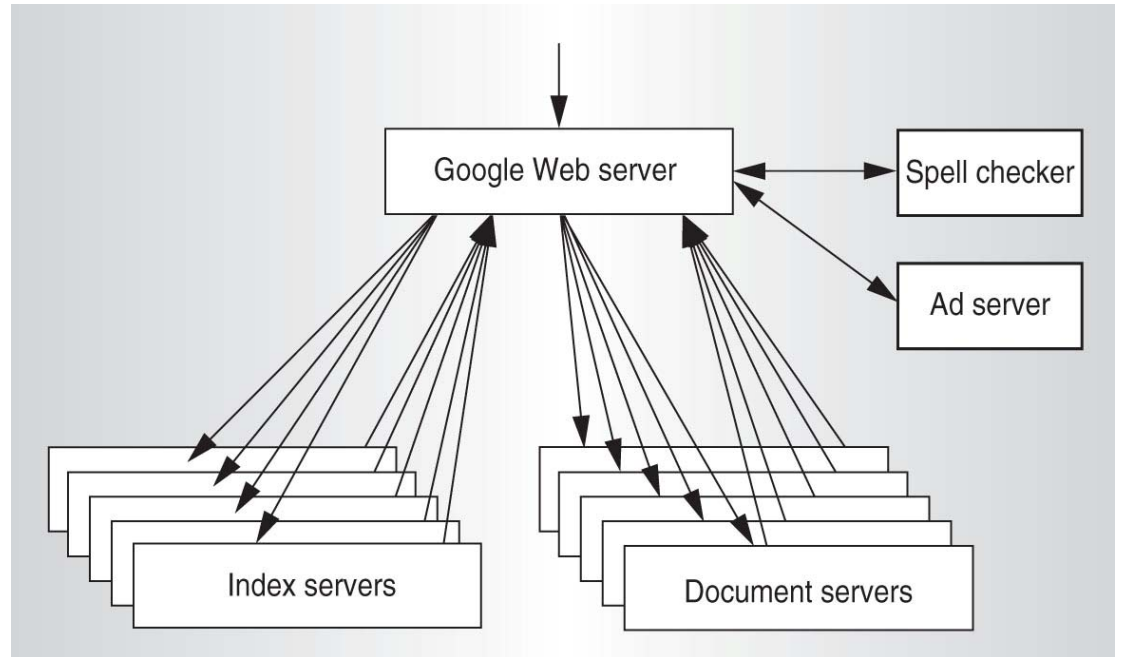




- Within the cluster, load-balancing routes your search to a lightly-loaded Google Web Server (GWS), which will coordinate the search and response
- The index is partitioned into "shards." Each shard indexes a subset of the docs (web pages). Each shard can be searched by multiple computers - "index servers"
- The GWS routes your search to one index server associated with each shard, through another load-balancer
- When the dust has settled, the result is an ID for every doc satisfying your search, rank-ordered by relevance



- The docs, too, are partitioned into "shards" - the partitioning is a hash on the doc ID. Each shard contains the full text of a subset of the docs. Each shard can be searched by multiple computers - "doc servers"
- The *GWS* sends appropriate doc IDs to one doc server associated with each relevant shard
- When the dust has settled, the result is a URL, a title, and a summary for every relevant doc



- Meanwhile, the ad server has done its thing, the spell checker has done its thing, etc.
- The *GWS* builds an HTTP response to your search and ships it off

# Many hundreds of computers are involved in responding to a single search request



- The system must have the following characteristics:
  - Fault-Tolerant
    - | It can recover from component failures without performing incorrect actions
  - Highly Available
    - | It can restore operations, permitting it to resume providing services even when some components have failed
  - Recoverable
    - | Failed components can restart themselves and rejoin the system, after the cause of failure has been repaired
  - Consistent
    - | The system can coordinate actions by multiple components, often in the presence of concurrency and failure



- Scalable

- It can operate correctly even as some aspect of the system is scaled to a larger size

- Predictable Performance

- The ability to provide desired responsiveness in a timely manner

- Secure

- The system authenticates access to data and services



- The system also must support a straightforward programming model

- Mere mortals must be able to write apps

- And it must be *cheap*

- A Google rack (176 2-GHz Xeon CPUs, 176 Gbytes of RAM, 7 Tbytes of disk) costs about \$300K; 6,000 racks ~ \$2B
- You could easily pay 2x this or more for “more robust” hardware (e.g., high-quality SCSI disks, bleeding-edge CPUs)
- And if you wanted a “traditional” multiprocessor with very high bisection bandwidth, the cost would be astronomical (and you couldn't achieve anything like this scale)



- You cannot make any of the following assumptions

- Hardware

- | Components are reliable
- | Components are homogeneous

- Software

- | It's correct

- Network

- | Latency is zero
- | Bandwidth is infinite
- | It's secure

- Overall system

- | Configuration is stable
- | There is one administrator



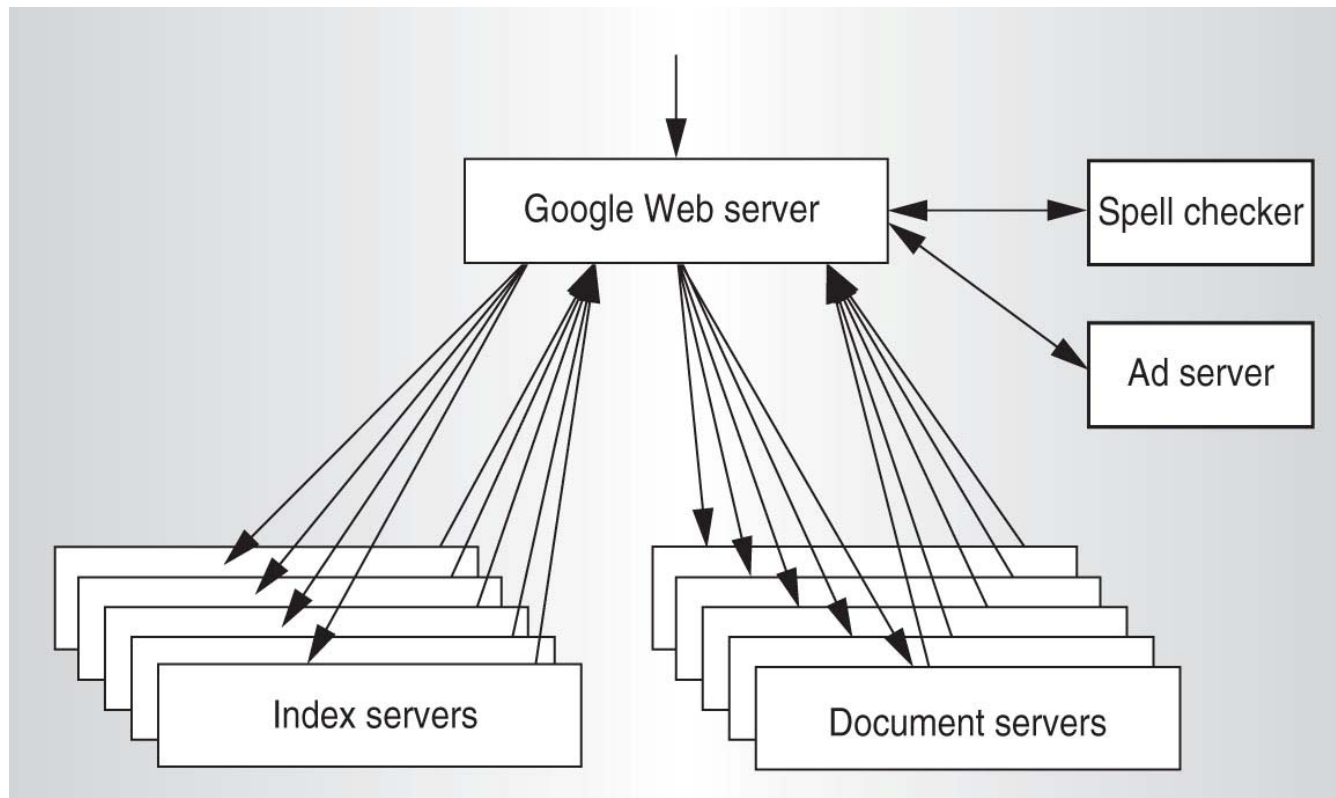
How to pull this off is the subject of this course



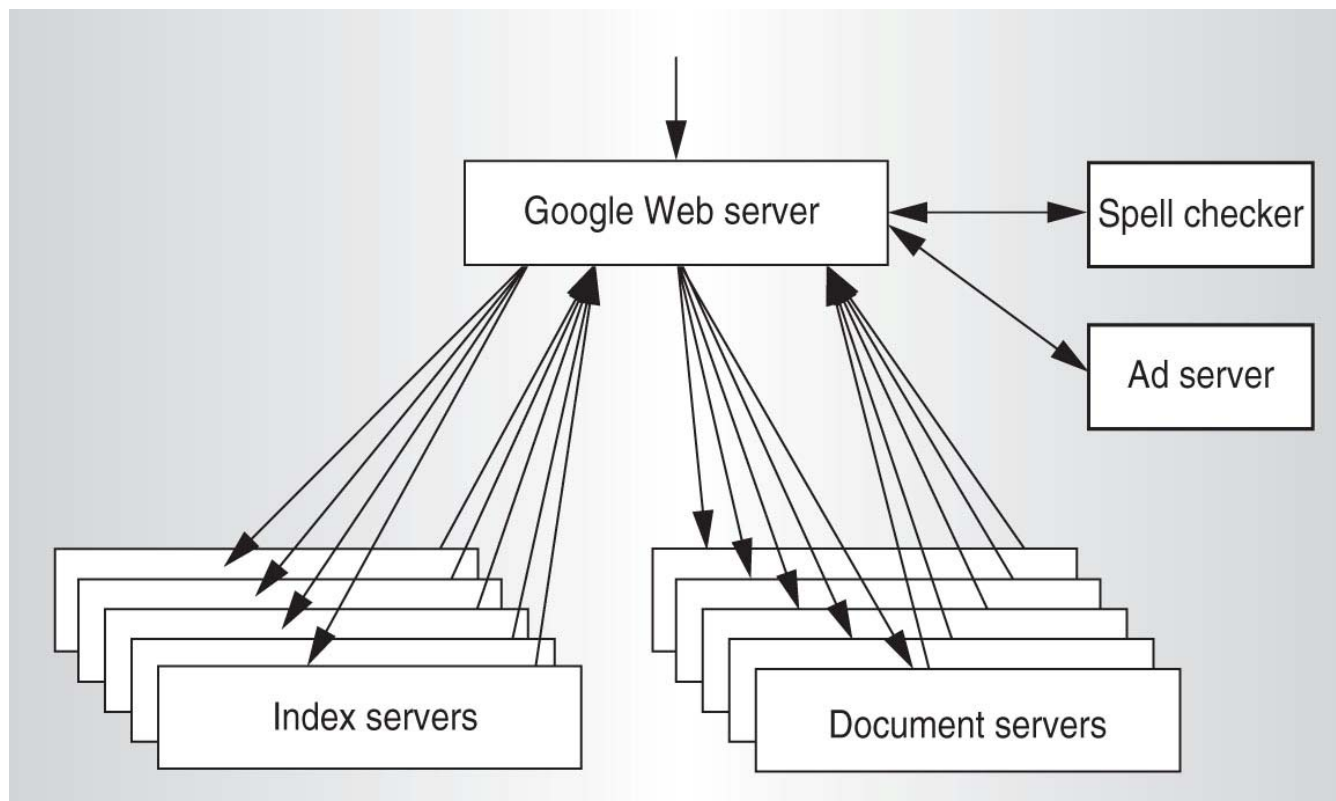
- You'll learn how to program these systems, and you'll learn how they're built

## But you tell me ...

- How does the Google search application achieve those characteristics (fault-tolerant, highly-available, recoverable, scalable, etc.)?




- Where does the Google search application use replication? Where does it use partitioning?



# How on earth would you enable mere mortals write hairy applications such as this?



- Recognize that many Google applications have the same structure
  - Apply a "map" operation to each logical record in order to compute a set of intermediate key/value pairs
  - Apply a "reduce" operation to all the values that share the same key in order to combine the derived data appropriately
- Example: Count the number of occurrences of each word in a large collection of documents
  - Map: Emit <word, 1> each time you encounter a word
  - Reduce: Sum the values for each word

- 
- Build a runtime library that handles all the details, accepting a couple of customization functions from the user - a Map function and a Reduce function
  - That's what MapReduce is
    - Supported by the Google File System and the Chubby lock manager
    - Augmented by the BigTable not-quite-a-database system

# Some terminology



## ■ MapReduce

- The LISP functional programming "Map / Reduce" way of thinking about problem solving
- Also, the name of Google's runtime library supporting this programming paradigm at enormous scale

## ■ Hadoop

- An open source implementation of the MapReduce functionality

## ■ Dryad

- Microsoft's version



## ■ Cloud computing

- Computing “out there somewhere” rather than on your desktop or in your datacenter
- Gmail, Google Docs, Salesforce.com, etc.
- Because cloud computing takes place at enormous scale, it requires many of the techniques we’ll discuss in this class

## ■ Amazon Web Services

- A commercial service that lets you deploy your own cloud computing applications
- Offers computing, storage, database services, etc.

## Instances

### Standard Instances

Instances of this family are well suited for most applications.

\$0.10 - Small Instance (Default)

1.7 GB of memory, 1 EC2 Compute Unit (1 virtual core with 1 EC2 Compute Unit), 160 GB of instance storage, 32-bit platform

\$0.40 - Large Instance

7.5 GB of memory, 4 EC2 Compute Units (2 virtual cores with 2 EC2 Compute Units each), 850 GB of instance storage, 64-bit platform

\$0.80 - Extra Large Instance

15 GB of memory, 8 EC2 Compute Units (4 virtual cores with 2 EC2 Compute Units each), 1690 GB of instance storage, 64-bit platform

### High-CPU Instances

Instances of this family have proportionally more CPU resources than memory (RAM) and are well suited for compute-intensive applications.

\$0.20 - High-CPU Medium Instance

1.7 GB of memory, 5 EC2 Compute Units (2 virtual cores with 2.5 EC2 Compute Units each), 350 GB of instance storage, 32-bit platform


\$0.80 - High-CPU Extra Large Instance

7 GB of memory, 20 EC2 Compute Units (8 virtual cores with 2.5 EC2 Compute Units each), 1690 GB of instance storage, 64-bit platform

Pricing is per instance-hour consumed for each instance type. Partial instance-hours consumed are billed as full hours.

*EC2 Compute Unit (ECU)* - One EC2 Compute Unit (ECU) provides the equivalent CPU capacity of a 1.0-1.2 GHz 2007 Opteron or 2007 Xeon processor. See [Amazon EC2 Instance Types](#) for details on available instance configurations and a complete description of an EC2 Compute Unit.



- 
- This includes
    - | Purchase + replacement
    - | Housing
    - | Power
    - | Operation
    - | Instantaneous expansion and contraction
  - Enables VC-less (or VC-lite) web service companies
  - Enables focus on your core business, not generic infrastructure
  - 1000 processors for 1 hour costs the same as 1 processor for 1000 hours
    - | Revolutionary!
  - Your application doesn't run decently on this environment?
    - | Start figuring out how to change that!

# Animoto

