Imperial College London

LSDS
Large-Scale Data & Systems Group

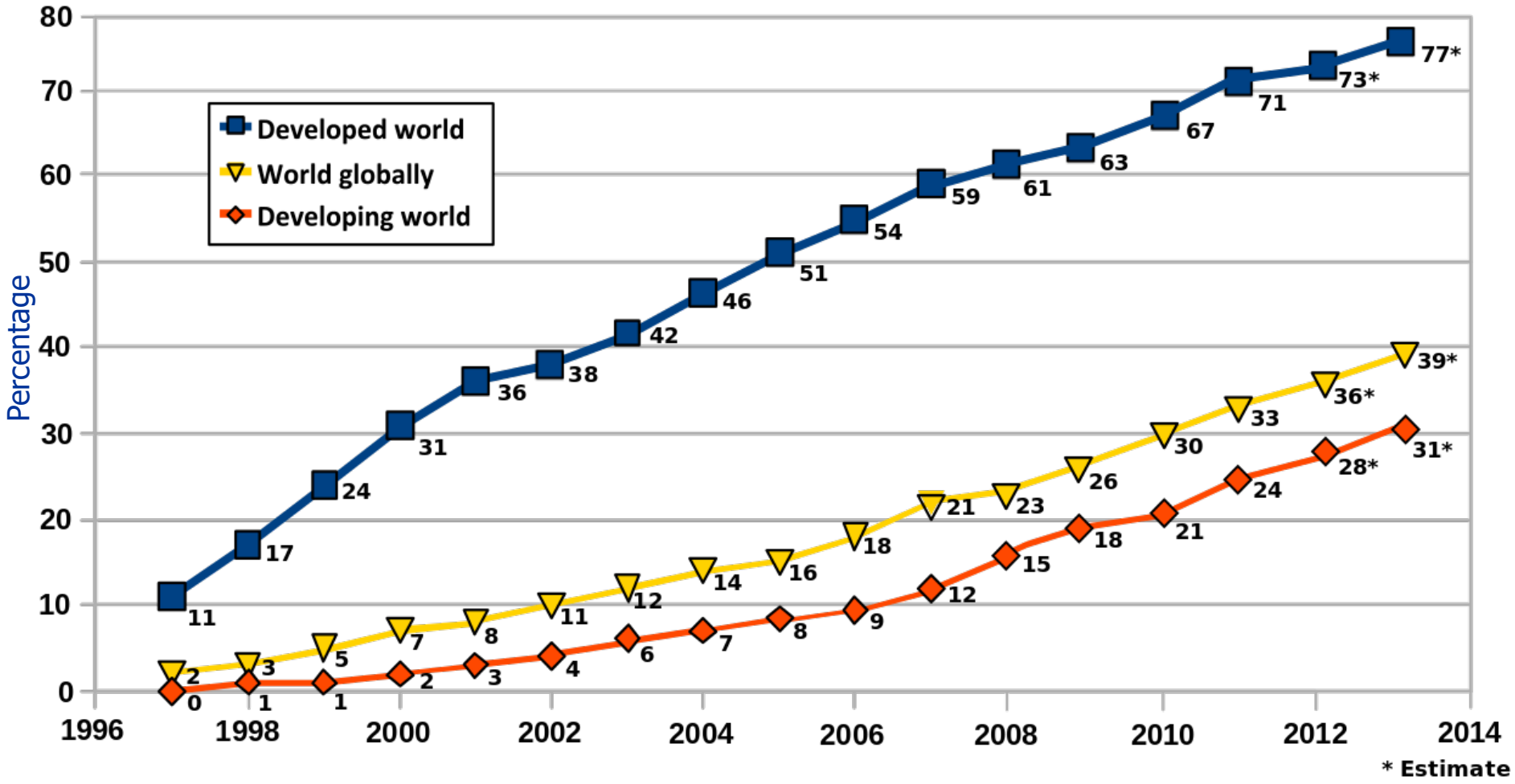
Scalable Cloud Systems: Introduction

Peter Pietzuch
prp@doc.ic.ac.uk

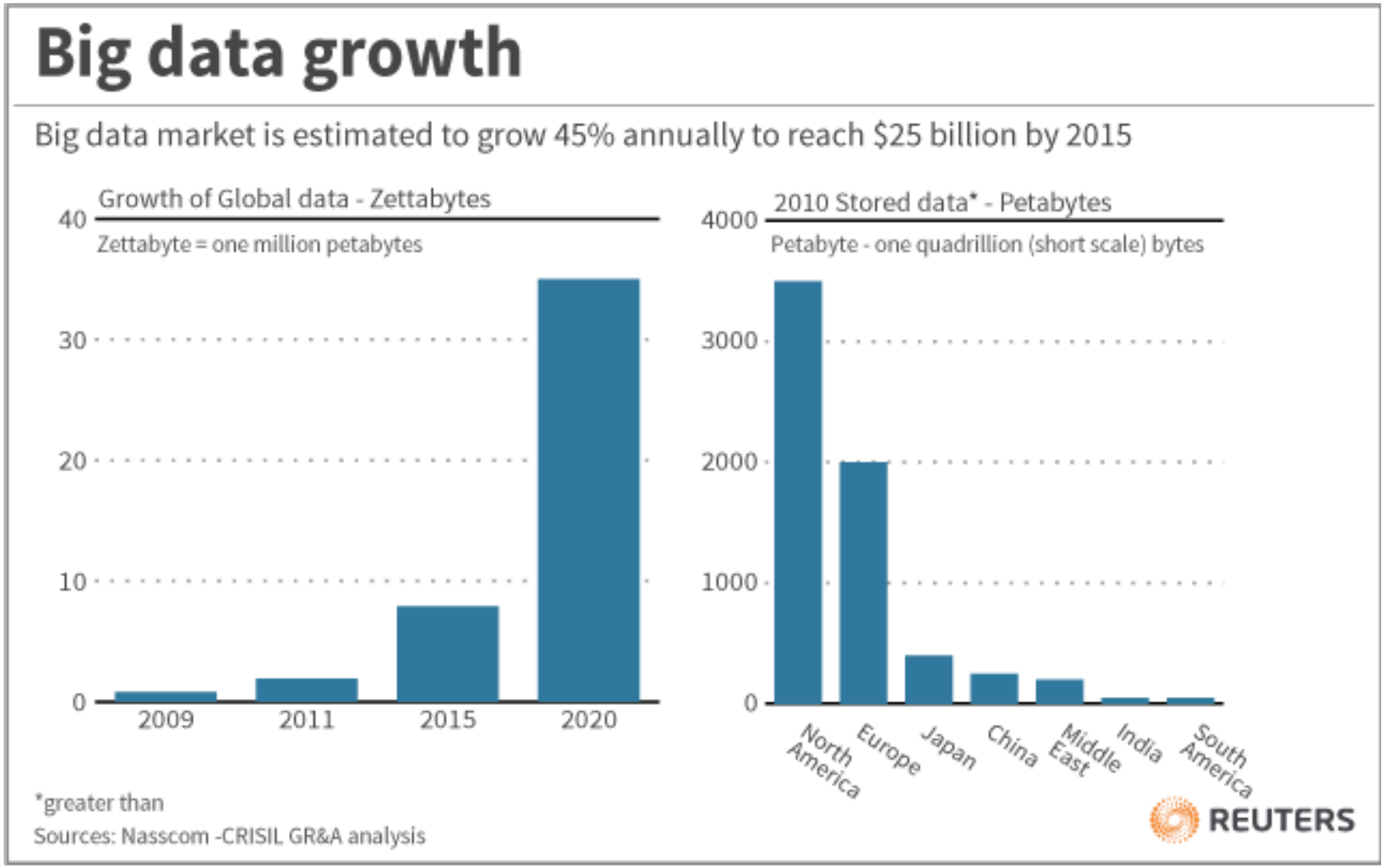
Department of Computing
Imperial College London
http://lsds.doc.ic.ac.uk

Autumn 2020

Facing Internet User Growth…



Facing Data Growth…



Data Storage Technologies

How much data is this?

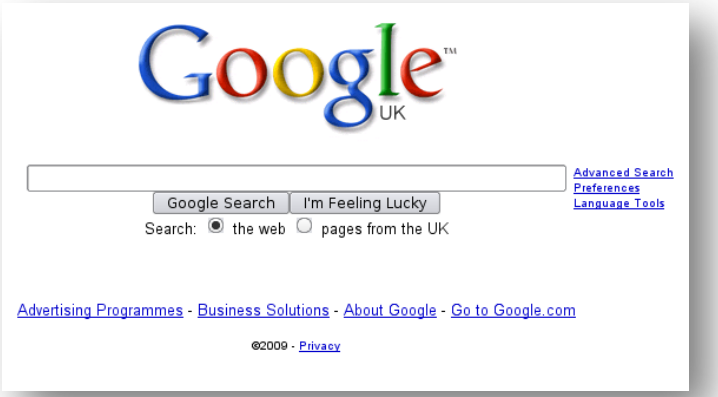
Scalable Systems for Web Companies

Connecting users to data

Scalable Distributed Systems

Search engines (e.g. Google, Yahoo, ...)

- Global crawling, indexing and search



Social networking sites (e.g. Facebook, Twitter, LinkedIn, ...)

- Facebook: serves 200 million users and stores 40 billion photos

Cloud computing applications (e.g. Amazon, Microsoft, Google, ...)

- Pay-as-you-use storage and computation for applications
  - Amazon: bought servers worth $86 million in 2008 alone

Content delivery networks (CDNs) (e.g. Akamai, Limelight, ...)

- Scalable web hosting, file distribution, media streaming, ...
  - Akamai: hosting for Microsoft.com, CNN.com, BBC iPlayer, ...

About this course

SSD Course Format – Second Half

Live (video) lectures

- Introduce core concepts and technologies

Paper discussion live sessions

- Required reading for lecture (1-2 papers)
- Discussion questions per paper
- Be prepared to answer questions
- Slides to provide summary/background on paper

Webpage with course schedule:

http://materials.doc.ic.ac.uk

Acknowledgements: Paolo Costa, UW Scalable Systems class, Columbia DS class

Scalable Systems (Peter Pietzuch)

1. Scalable distributed systems design

2. Cloud computing & Data centres

3. Scalable storage and querying

- "**Bigtable**: A Distributed Storage System for Structured Data", Seventh Symposium on Operating System Design and Implementation (OSDI), Seattle, WA, November 2006
- "**Dynamo**: Amazon's Highly Available Key-Value Store", ACM Symposium on Operating Systems Principles (SOSP), Stevenson, WA, October 2007
- "**Spanner**: Google's Globally-Distributed Database", Tenth Symposium on Operating System Design and Implementation (OSDI), Hollywood, CA, October 2012

4. Scalable compute

- "**MapReduce**: Simplified Data Processing on Large Clusters", Sixth Symposium on Operating System Design and Implementation (OSDI), San Francisco, CA, December 2004
- "**Resilient Distributed Datasets**", 9th USENIX conference on Networked Systems Design and Implementation (NSDI), San Jose, CA, April 2012

Scalable Systems - Timetable

		Topic	
Week 4	Nov 6	Introduction	
Week 5	Nov 10	Cloud Computing & Data Centres	
	Nov 13	BigTable	Paper link
Week 6	Nov 17	Dynamo	Paper link
	Nov 20	Spanner	Paper link
Week 7	Nov 24	MapReduce	Paper link
	Nov 27	RDD	Paper link

BigTable Questions

Week 5:

"Bigtable: A Distributed Storage System for Structured Data", Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E. Gruber, Seventh Symposium on Operating System Design and Implementation (OSDI), Seattle, WA, November 2006

- What is the problem that this paper tries to solve? How would summarise its main idea in a few sentences? How does it work in more detail?
- What is good about the paper? What is not good about the paper?
- How does the design of BigTable compare to that of a parallel relational database management system (RDBMS)?
- What limits the scalability of the BigTable design?

Dynamo Questions

Week 6:

"**Dynamo: Amazon's Highly Available Key-Value Store**", Guiseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swami Sivasubramanian, Peter Vosshall, and Werner Vogels, ACM Symposium on Operating Systems Principles (SOSP), Stevenson, WA, October 2007

- What is the problem that this paper tries to solve? How would summarise its main idea in a few sentences? How does it work in more detail?
- What is good about the paper? What is not good about the paper?
- To what extent is the design of Dynamo inspired by Distributed Hash Tables (DHTs)? What are the advantages and disadvantages of such a design?
- How does the design of Dynamo compare to that of BigTable?

Spanner Questions

Week 6:

"**Spanner: Google's Globally-Distributed Database**", James C. Corbett, Jeffrey Dean, Michael Epstein, Andrew Fikes, Christopher Frost, JJ Furman, Sanjay Ghemawat, Andrey Gubarev, Christopher Heiser, Peter Hochschild, Wilson Hsieh, Sebastian Kanthak, Eugene Kogan, Hongyi Li, Alexander Lloyd, Sergey Melnik, David Mwaura, David Nagle, Sean Quinlan, Rajesh Rao, Lindsay Rolig, Yasushi Saito, Michal Szymaniak, Christopher Taylor, Ruth Wang, and Dale Woodford, Tenth Symposium on Operating System Design and Implementation (OSDI), Hollywood, CA, October 2012

- What is the problem that this paper tries to solve? How would summarise its main idea in a few sentences? How does it work in more detail?
- What is good about the paper? What is not good about the paper?
- How does the performance of Spanner depend on the workload?
- What other applications could TrueTime have?

MapReduce Questions

Week 7:

"MapReduce: Simplified Data Processing on Large Clusters", Jeffrey Dean and Sanjay Ghemawat, Sixth Symposium on Operating System Design and Implementation (OSDI), San Francisco, CA, December 2004

- What is the problem that this paper tries to solve? How would summarise its main idea in a few sentences? How does it work in more detail?
- What is good about the paper? What is not good about the paper?
- What algorithms cannot be easily expressed in the MapReduce model?
- Can you think of other techniques for handling stragglers?

RDD Questions

Week 7:

"**Resilient Distributed Datasets**", Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael J. Franklin, Scott Shenker, Ion Stoica, 9th USENIX conference on Networked Systems Design and Implementation (NSDI), San Jose, CA, April 2012

- What is the problem that this paper tries to solve? How would summarise its main idea in a few sentences? How does it work in more detail?
- What is good about the paper? What is not good about the paper?
- Is the comparison with Hadoop fair?
- How well can Spark be used to process graph data?

Coursework + Exams

Coursework

- 1 x coursework answering questions about another research paper
- Coursework acts as exam preparation

Exams

- Traditional exam questions, see past exam papers
- All taught material is examinable
- Questions about discussed research papers
- Slight adaptation due to open book format

Resources – Scalable Systems

Textbooks

- "Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems", Martin Kleppmann, O'Reilly Media, September 2014
- "The Art of Scalability: Scalable Web Architecture, Processes and Organizations for the Modern Enterprise", Martin L. Abbott, Michael T. Fisher, Addison Wesley, 1st Edition, December 2009

Blogs

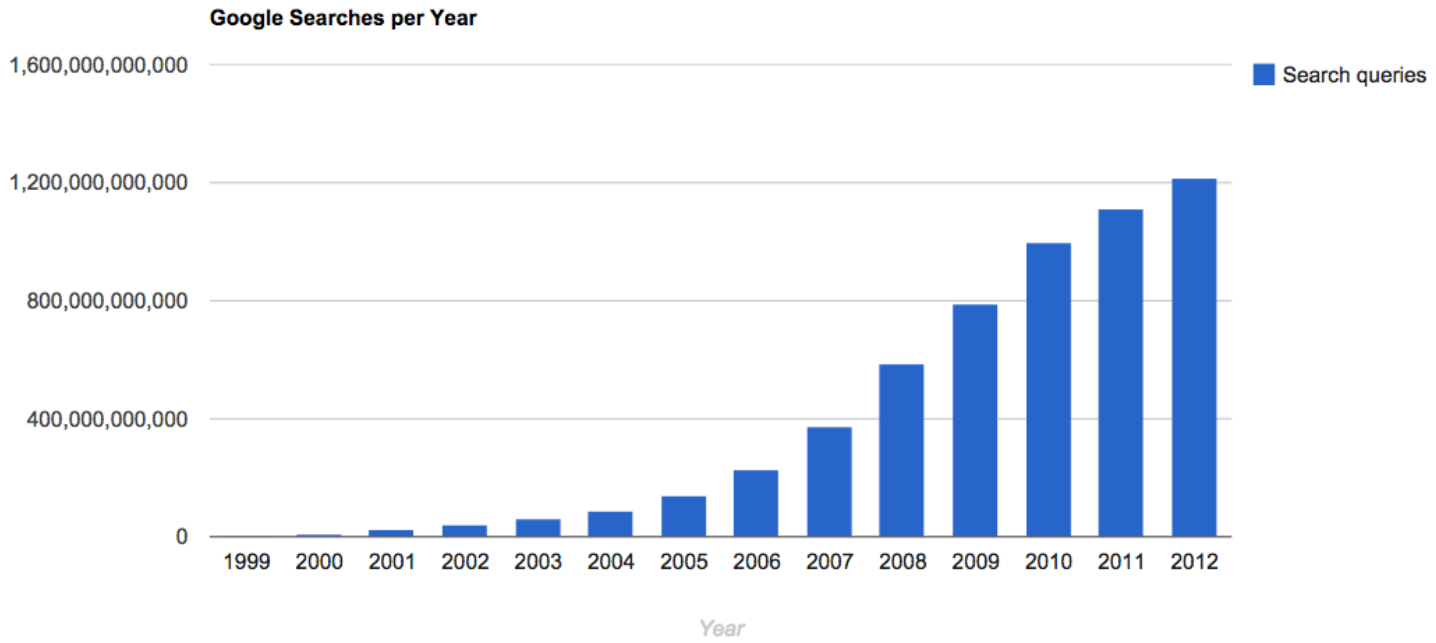
- http://highscalability.com/
- http://www.allthingsdistributed.com/ (Werner Vogel's blog)
- http://perspectives.mvdirona.com/ (James Hamilton's blog)

Conferences

- ACM/USENIX SOSP/OSDI, ACM SOCC, USENIX ATC, USENIX NSDI, ACM SIGCOMM, ACM CoNEXT, ACM EuroSys, ACM SIGMOD, VLDB, IEEE ICDE

Scalable Distributed Systems

Example: Google Search



Processes ~ 40,000 queries per second (estimate)

Crawling times:

- 1 month for 50 million pages in 1999; <1 min in 2012

Typical search query uses 1000 machines in 200 ms

source: Internetlivestats.com

Example: Facebook

What 20 minutes on Facebook looks like

Shared links: 1,000,000 every 20 minutes

Tagged photos: 1,323,000

Event invites sent out: 1,484,000

Wall Posts: 1,587,000

Status updates: 1,851,000

Friend requests accepted: 1,972,000

Photos uploaded: 2,716,000

Comments: 10,208,000

Message: 4,632,000

facebook

400,000,000 requests/s
200 TB of data in RAM

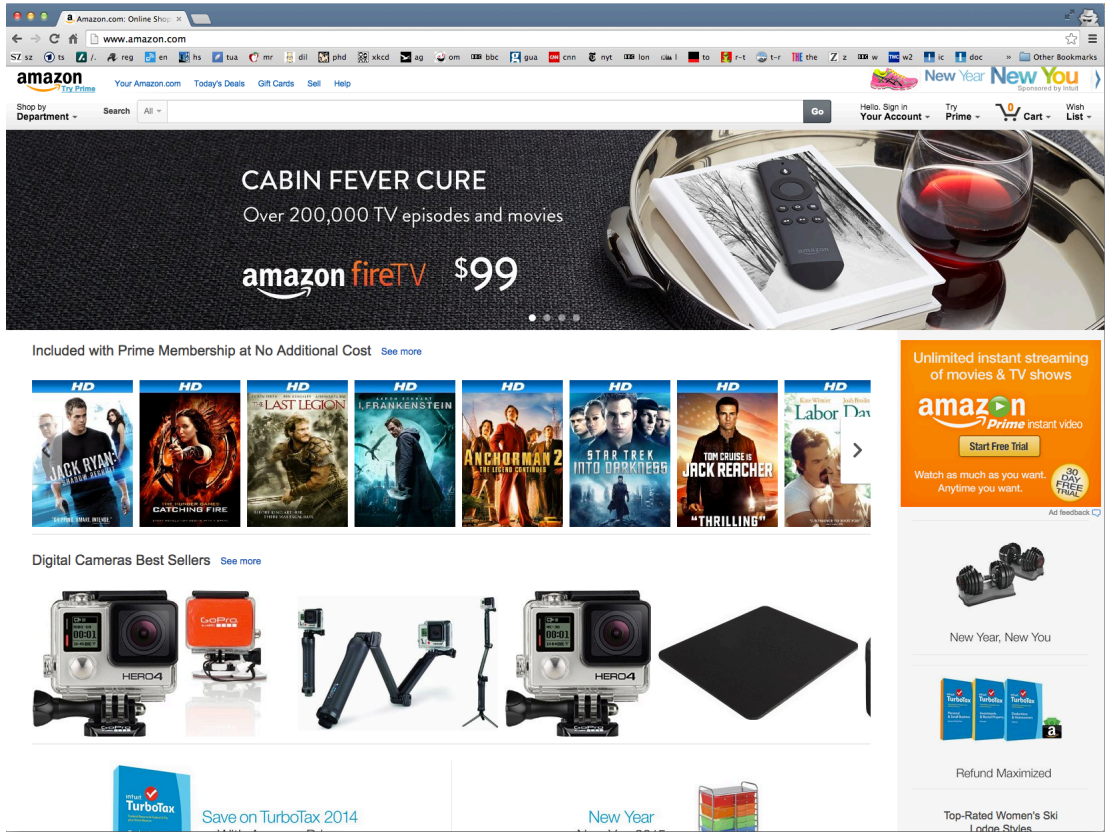
Performance Matters

Online services (eg Facebook, Google Search, Bing):

- Expected response time <100 ms

Performance affects revenue

- Amazon: every 100 ms of latency cost them 1% in sales
- Google found an extra 0.5 secs drops traffic by 20%



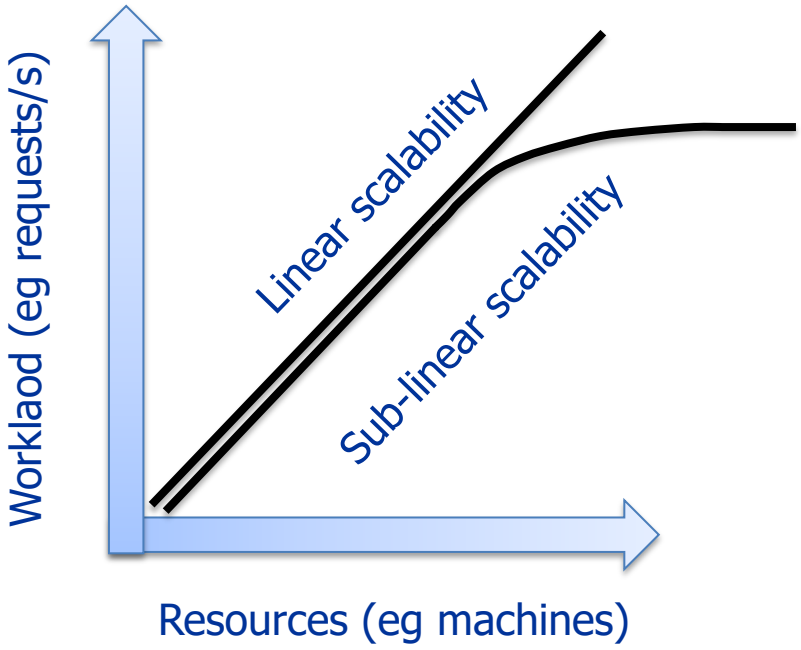
Scalability Matters

Ideally: adding 2x more servers supports 2x more users

Linear scalability hard to achieve

- Overheads + synchronisation
- Load-imbalances create hotspots (eg popular content, poor hash function)

Need to partition compute + data



HPC and Parallel Computing

Supercomputers, mainframes give high performance...

- Eg IBM Systems z13 (z stands for "zero downtime")
  - Up to 141 CPUs, 5.5 Ghz core, 10 TB of RAM, 2.5 billions tx/sec
- Price tag >$1 million

IBM System Z mainframe

Advantages

- Single solution
- Powerful hardware
- Reliable design

Disadvantages

- ???

Distributed Computing

Client/server model

- Request typically too large for single server
- Carefully partition problem across many servers

Cheap commodity hardware

- Multi-core x86 servers
- Standard local disks for storage
- Best performance per $$$

Limited inter-machine bandwidth

- Think regular 10Gbps Ethernet

Unrealiable hardware

- Reliability achieved via software mechanisms

Communication expensive → need embarssingly parallelisable algorithms

Properties of Distributed Systems

Scalability

- Aggregation of many resources
  - Compute: cluster of shared nothing machines
  - Storage: distributed file systems (eg GFS, HDFS, NFS, ...)
  - Memory: cluster memory (in-memory key/value stores, caches, ...)
  - Bandwidth: data centres networks, CDNs, ...

Location transparency

- Client doesn't care which server handles request

High availability

- Mask hardware failures: power outages, disk failures, memory corruption, network switch failures
- Mask software failures: bugs, misconfiguration, updates failures

Composable services

- Rely on composition of independent services
- Good software engineering practice

Answering a Google Search Request

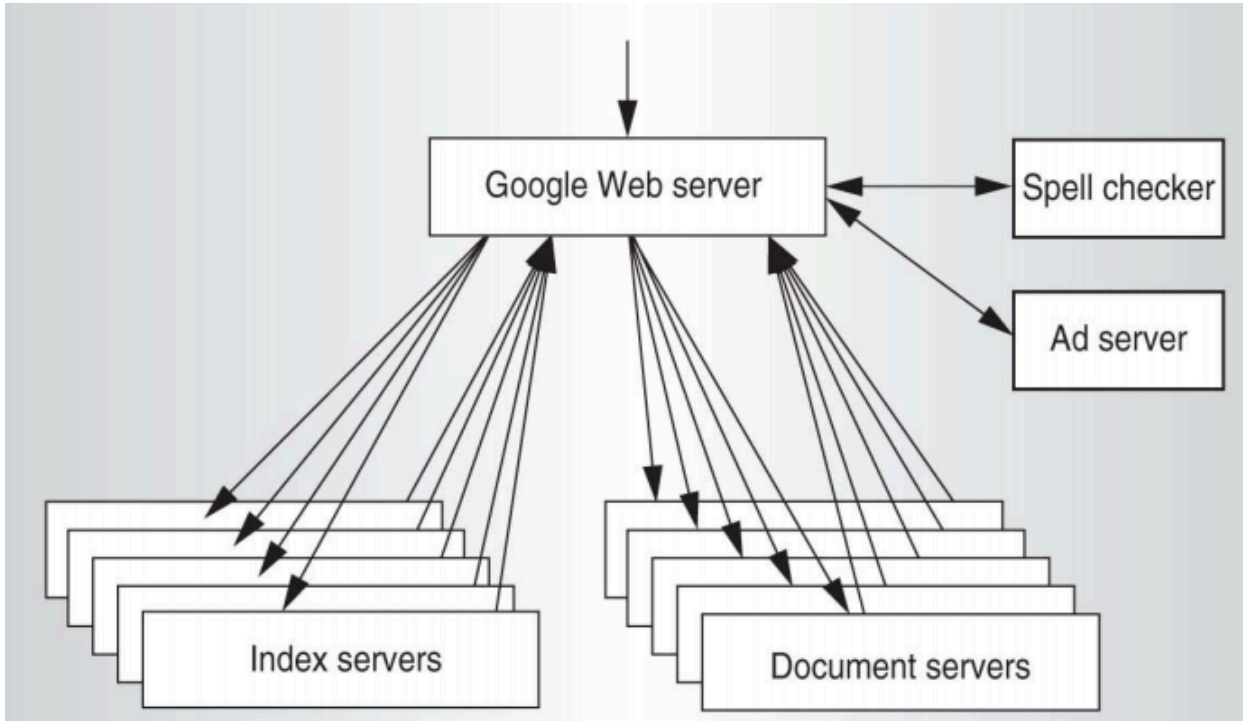
Multi data centres (DC) distributed throughout the world

DNS redirection sends HTTP search request to nearby DC

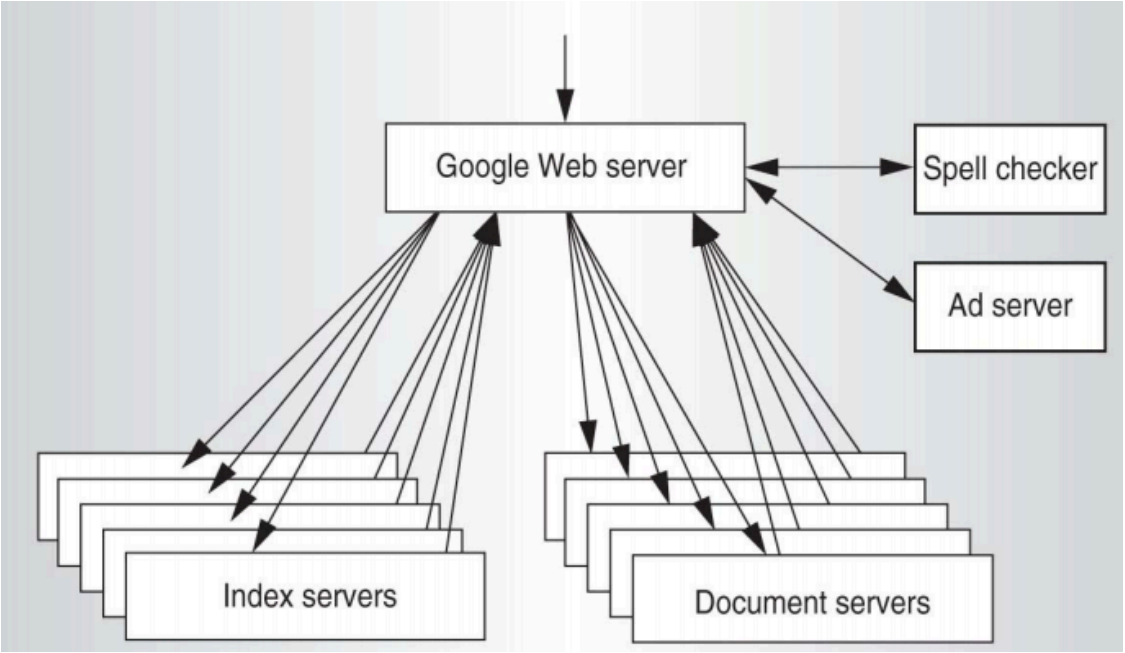
Answering a Google Search Request II

DC contains Load Balancers, Google Web Servers, Index Servers, Document Servers, and various others (ads, spell-checking etc)

- Web index partitioned into shards for subset of web pages; Index Servers search shards
- Web pages (documents) also partitioned into shards; Document Servers store shards

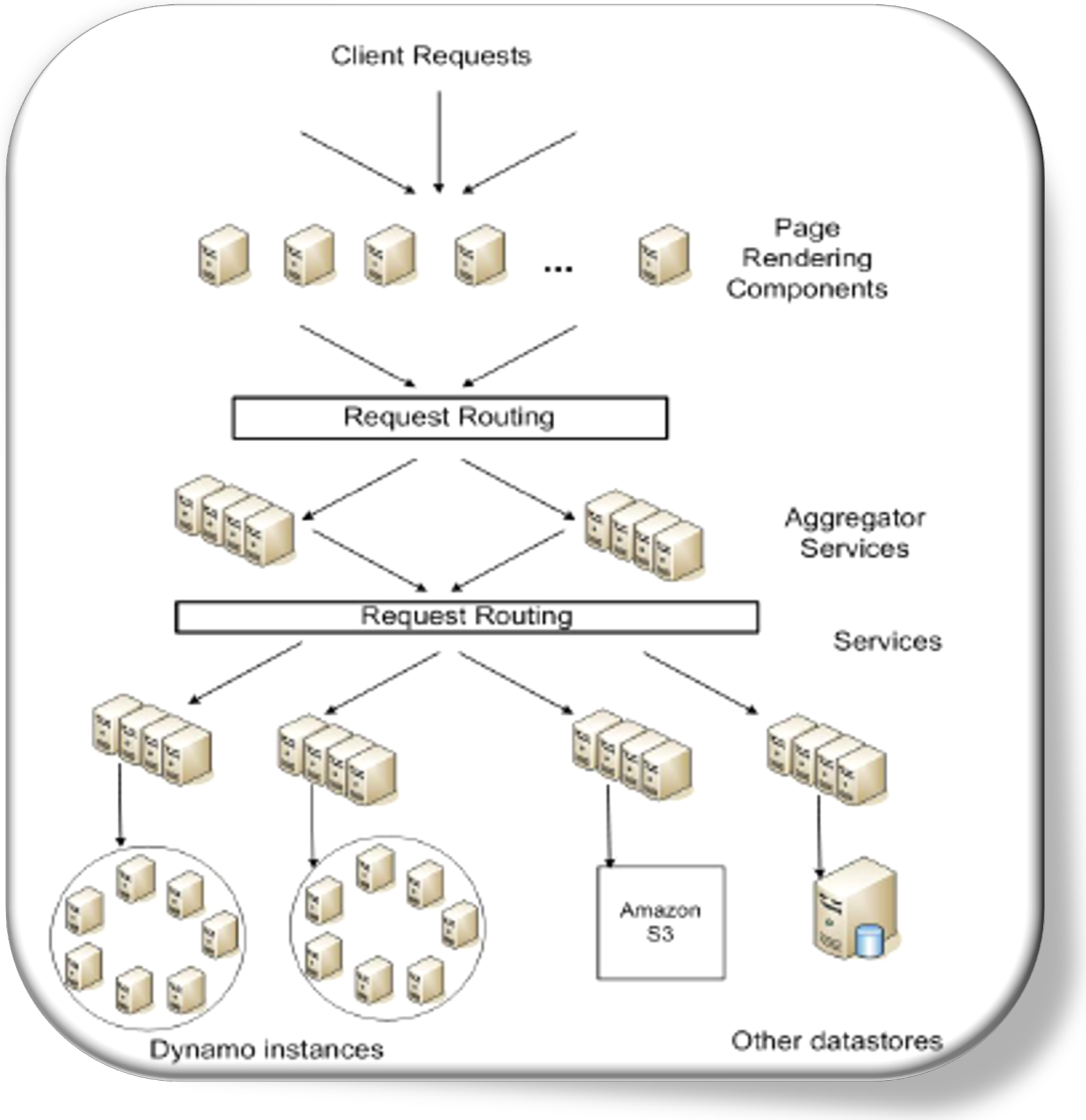


Answering a Google Search Request III



1. Load-balancer routes request to lightly-loaded Google Web Server (GWS)
2. GWS routes search to one Index Server for each shard through load-balancer
3. Results are aggregated (IDs for matching documents), ie ordered by relevance
4. GWS sends appropriate IDs to Doc Servers to retrieve URL, title, summary
5. Results are aggregated (+ad, spell), producing search result page

Generating the Amazon Front Page



Data Centres

Data Centres for Scalable Systems

Data Centers

- Large group of networked servers
- Typically built from commodity hardware
- Network components often non-commodity

Assumptions:

- **Scale out** and not scale up
  - Commodity servers with local disks
  - Parallelism is king
- Software designed for **failure**

Data Centres from Provider's Point of View

Scalable service = data centre hardware + software

Typically built from commodity hardware

- Although network components often non-commodity

Economies of scale

- Cheaper electricity, hardware, network, operations
- Custom software and hardware solution for efficiency

Technology	Cost in Medium-Sized DC	Cost in Very Large DC	Ratio
Network	$95 per Mbit/sec/month	$13 per Mbit/sec/Month	7.1
Storage	$2.20 per GByte/month	$0.40 per Gbyte/month	5.7
Administration	≈ 140 Servers / Administrator	> 1000 Servers / Administrator	7.1

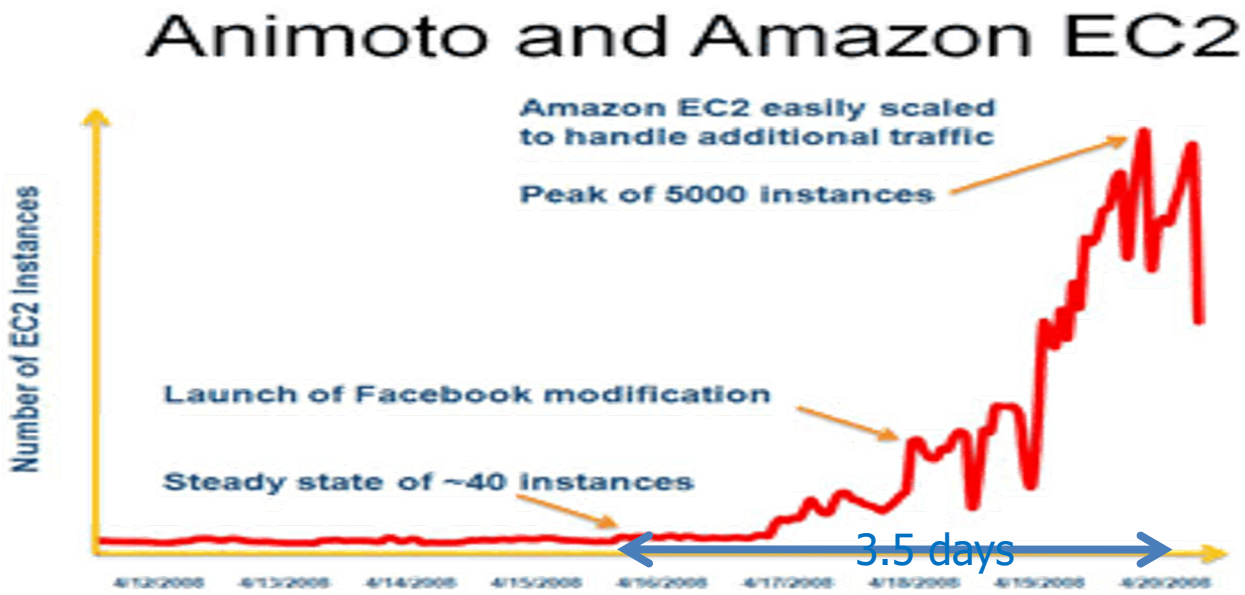
Huge DCs 5-7x as Cost Effective as Medium-Scale DCs

Elasticity: Resources on-demand

Scalability and elasticity: scale-out vs. scale-up

- Many low-cost PCs rather than few expensive high-end servers

Illusion of infinite resources from perspective of tenants



Google Data Centre (circa 1997)

Google Data Centre (circa 2000)

Google Data Centre (today)

Google: 30+ data centre locations

- over 1 million servers
- 260 MW (0.01% of global energy)
- Exabytes (10^{18}) of storage

Cooling infrastructure

Microsoft’s Data Center Footprint

Design Principles

Types of Scalable Systems

Online systems (< 100 ms)

- Web content serving, interactive features, transaction processing (OLTP), ...

Batch processing systems (> 1 hours)

- Offline data processing
- Data analytics (Hadoop, Spark), data warehousing (OLAP), ...

Nearline systems (< 1 secs or mins)

- Dynamic content presented to users where freshness matters
- Recommender systems, ad prediction, dashboards, performance monitoring, interactive debugging, ...

Search for people, jobs, companies, and more... Advanced

Home Profile Connections Jobs Interests Business Services Try Premium for free

Avg Bid for Devs: 55k - Want to move out of your industry? Work with a new stack? Try Hired today!

Share an update...

All Updates

Pulse recommends this news for you

Study: The Most Important Characteristic In A CEO Is...

Paul Petrone on LinkedIn · What's the most important characteristic in a CEO?Korn Ferry, the world's largest executive search firm, conducted a...

2h

My First 90 Days: Slow Down and Take the Time to Learn

By Josh Bersin · 19h

Can Big Data Make Medicine Better?

By Dr. Chris Stout · 1h

How to Get People to Do What You Want

By Colin Shaw · 4h

See your news

Grow your network. Connect with people you may know

Andrea Michi
Undergraduate Teaching Assistant
+ Connect

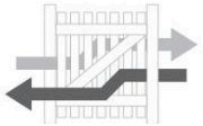
Ioan Raicu
Assistant Professor at Illinois
+ Connect

Anmol Rajpurohit
Graduate Student, Data Science at
+ Connect

Rocco De Nicola
Full Professor at IMT Lucca
+ Connect

Keep growing your network. Find more connections

Colin Robbins



An introduction to Information Exchange Gateways via Martin Cooper

cybermatters.info · Today marks the beginning of an exciting week for me. I am on site at one of our major clients installing an Information Exchange Gateway demonstrator that I've been working on for the last few months. Over that time I've...

Like (1) · Comment · Share · 1d ago

Simon Khan

Add a comment...

Say happy work anniversary!

Giuliano Casale has been at Imperial College London 5 years this January.

People You May Know

Zengbin Zhang, Software Engineer at Google
Connect

Ajit Jaokar, founder futuretext
Connect

Manuel Roman, Technical Yahoo, Architect, Sr. Principal at
Connect

See more »

Ads You May Be Interested In

London's Top Devs
Want to move out of your industry? Work with a new stack? Try Hired today!

5 Big Data Requirements
[Report] Learn How 600 Companies are Implementing Big Data & Future Trends

Pharmaceuticals Event
Network with pharmaceutical experts at the Quality by Design Symposium

You Recently Visited

Andrew Warfield's post:
Letting it all scale out
See all of Andrew's posts

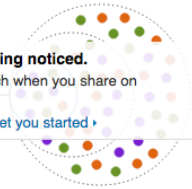
Who's Viewed Your Profile

10 Your profile has been viewed by 10 people in the past 15 days.

14 Your rank for profile views moved down by 14% in the past 15 days.

Unlock the full list with LinkedIn Premium

Who's Viewed Your Updates



Peter, start getting noticed.
See who you reach when you share on LinkedIn.
Great articles to get you started

Your LinkedIn Network

492 Connections

Distributed Computing Challenges

Scalability

- Independent parallel processing of sub-requests/tasks
- Eg adding more servers permits serving more concurrent requests

Fault tolerance

- Must mask failures and recover from hardware/software failure
- Must replicate data and service for redundancy

High availability

- Service must operate 24/7

Consistency

- Data stored/processed by multiple services must lead to consistent results

Performance

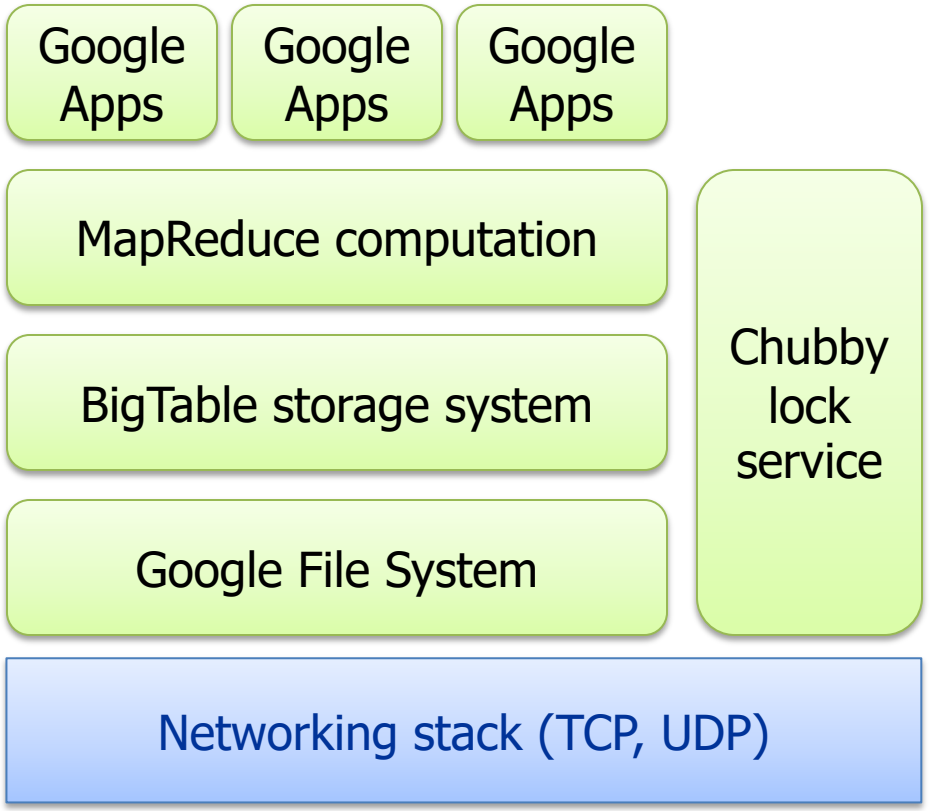
- Predictable low-latency processing with high throughput

Can’t solve all challenges in a custom manner for each application

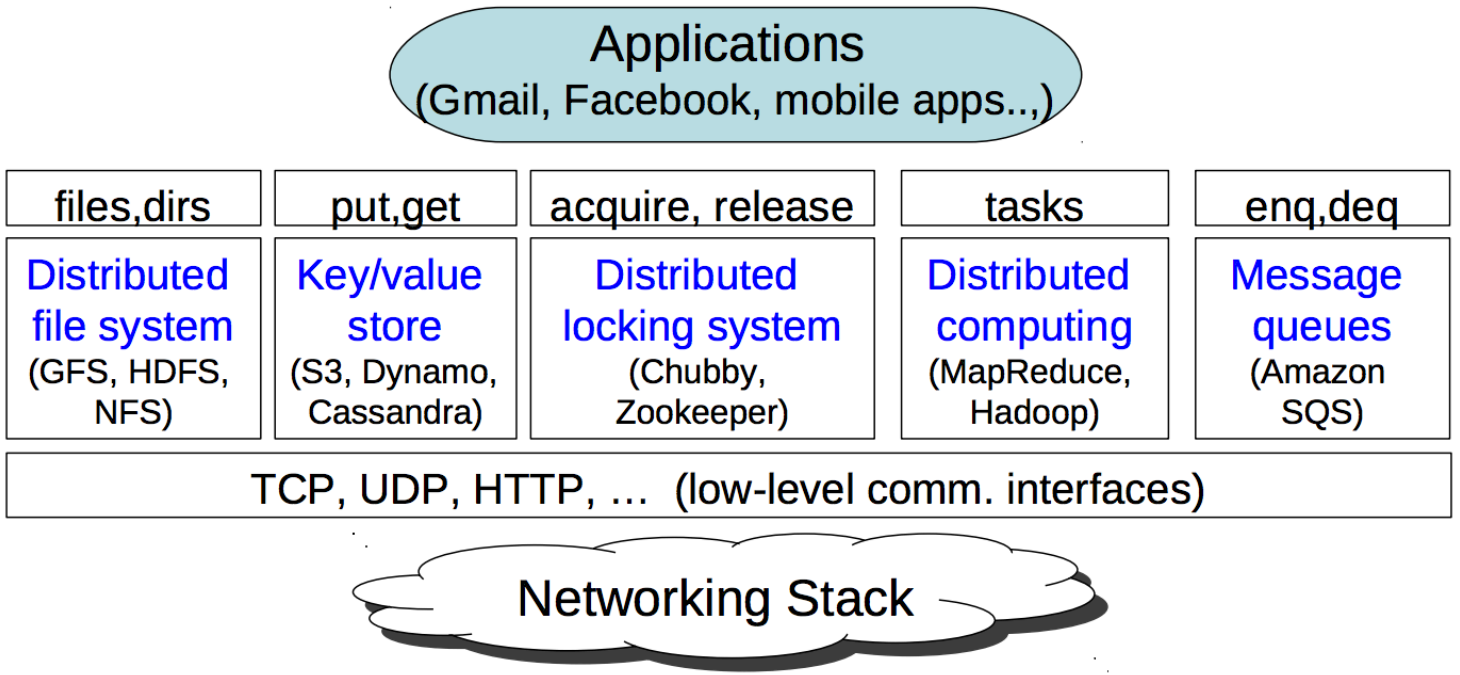
Abstractions for Scalable Systems

Eg Google uses several layers of abstraction

- Runs applications (search, mail, ...) on top of highest layer
- Each layer is scalable, network-aware and fault-tolerant



Modern Scalable Distributed Systems Stacks



Many internal services

- Microsoft: Autopilot, Dryad
- Google: GFS, BigTable, MapReduce
- Yahoo: Hadoop, HDFS
- Amazon: Dynamo, Astrolabe
- Facebook: Cassandra, Scribe, HBase

Design Principles for Scalable Systems

1. Stateless services
2. Caching
3. Partition/aggregration pattern
4. Weaker consistency
5. Efficient failure recovery

Others?

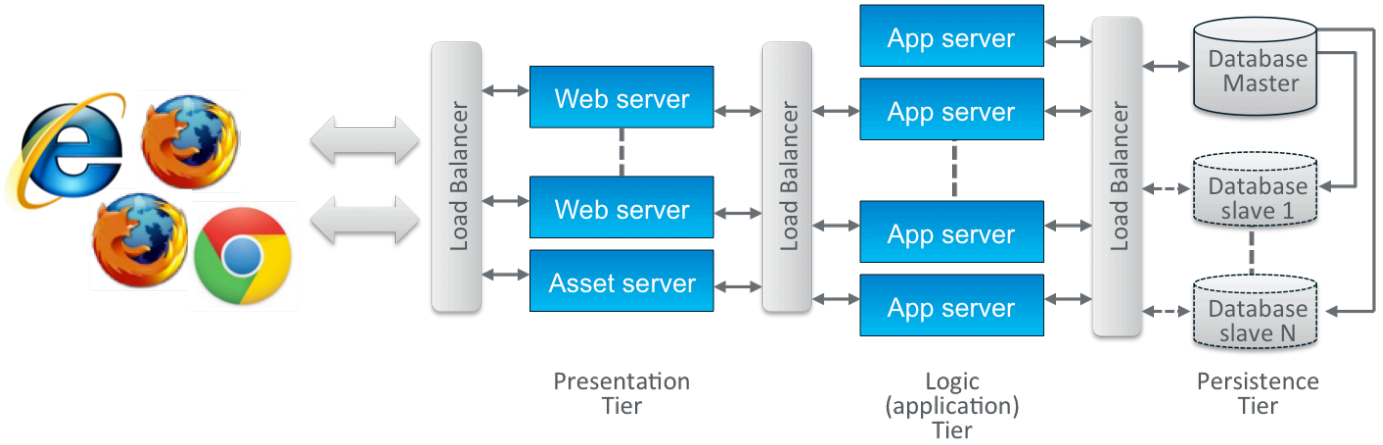
1. Stateless Services

Data consistency is major headache

- Avoid consistency issues by not maintaining data!

Example: HTTP as a stateless protocol

- RESTful designs of web applications
- Simplicity of of scaling three-tiered web architectures



Where is the state?

Example: Separation of data + meta-data in dist. file systems

- Permits data access to be stateless

If state is needed, use leases to expire state gracefully

2. Caching

Latency is king

- Need to design systems for low latency

Think about the sources of latency and add caches...

Numbers Everone Should Know

L1 cache reference	0.5 ns
Branch mispredict	5 ns
L2 cache reference	7 ns
Mutex lock/unlock	100 ns
Main memory reference	100 ns
Compress 1K bytes with Zippy	10,000 ns
Send 2K bytes over 1 Gbps network	20,000 ns
Read 1 MB sequentially from memory	250,000 ns
Round trip within same datacenter	500,000 ns
Disk seek	10,000,000 ns
Read 1 MB sequentially from network	10,000,000 ns
Read 1 MB sequentially from disk	30,000,000 ns
Send packet CA->Netherlands->CA	150,000,000 ns

source: Jeff Dean (Google)

3. Partition/Aggregation Pattern

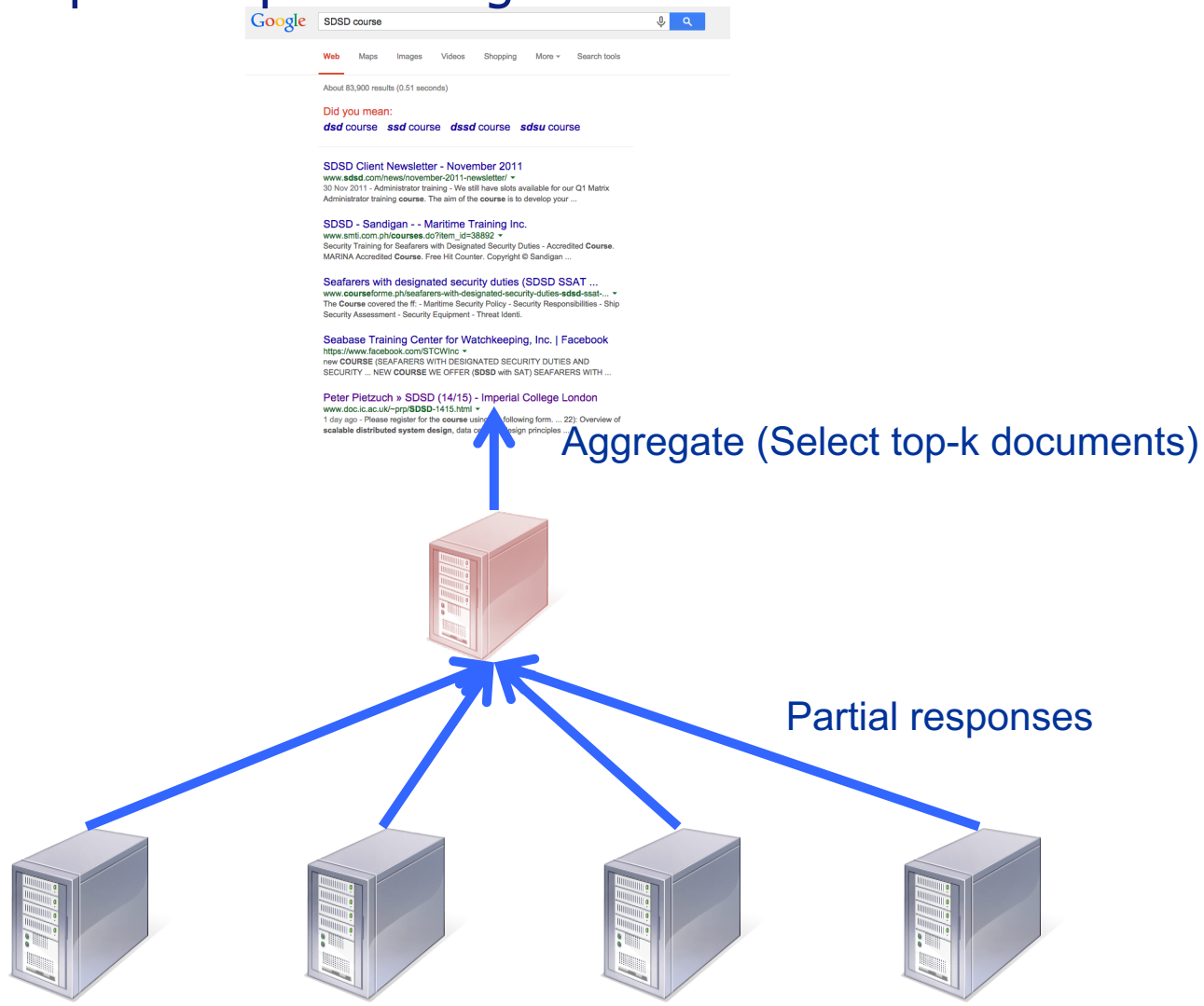
How do data-intensive application scale?

Frontend

Backends

Partition/Aggregation Pattern

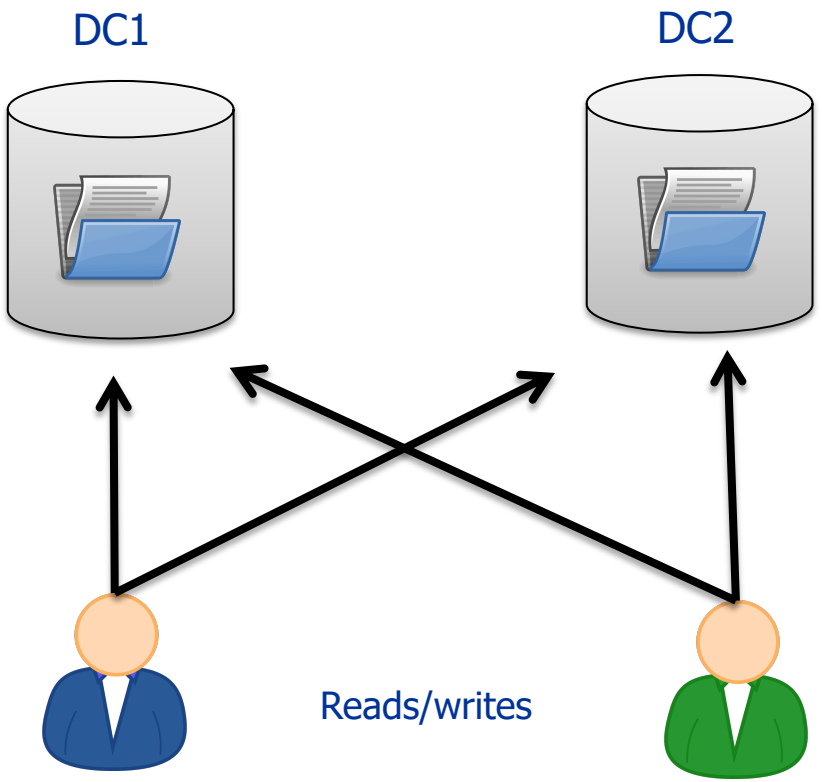
Enables task-parallel or data-parallel processing



4. Weaker Consistency

Many applications need state replicated across a wide area

- For reliability, availability, and low latency access



Consistency Models

Two main choices:

Strongly consistent operations (eg use Paxos)

- often imposes additional latency for common case

Inconsistent operations

- better performance/availability, but apps harder to write and reason about in this model

Many apps need to use a mix of both of these:

- eg Gmail: marking a message as read is asynchronous, sending a message is a heavier-weight consistent operation
- Order of posts in LinkedIn news feed? Access from multiple devices?
- Count of song popularity in Spotify?

5. Efficient Failure Recovery

Think of failure as the common case

Full redundancy too expense → use failure recovery

- Impossible to build redundant systems at scale
- Rather reduce the cost of failure recovery

Failure recovery: replication vs recomputation

- Depends on respective costs

Replication

- Need to replicate data and service
- Introduces consistency issues

Recomputation

- Easy for stateless protocols (eg HTTP requests)
- Remember data lineage for compute jobs