Introduction

DS 5110: Big Data Systems (Spring 2023)

Lecture 1

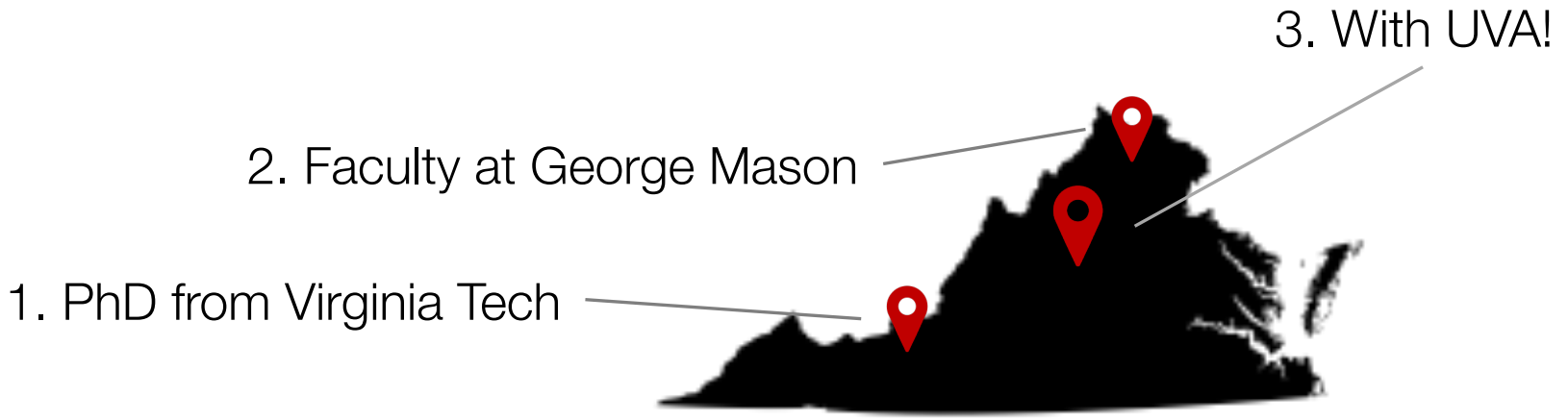
Yue Cheng

Some material taken/derived from:

- Wisconsin CS 744 by Shivaram Venkataraman.

Who am I?

- On the faculty of Data Science & Computer Science
  - Web: https://tddg.github.io
  - Email: mrz7dp@virginia.edu



- My current research: Designing scalable, high-performance, and easy-to-use computer **systems** that manage and process huge volume of **data**

Course staff and getting help

- Instructor: Yue Cheng
  - Office hours: Thursday, 11am-12pm on Zoom

- GTA: Jingyi Gao
  - Email: etc6bd@virginia.edu
  - Office hours: Friday, 6pm-10pm on Zoom

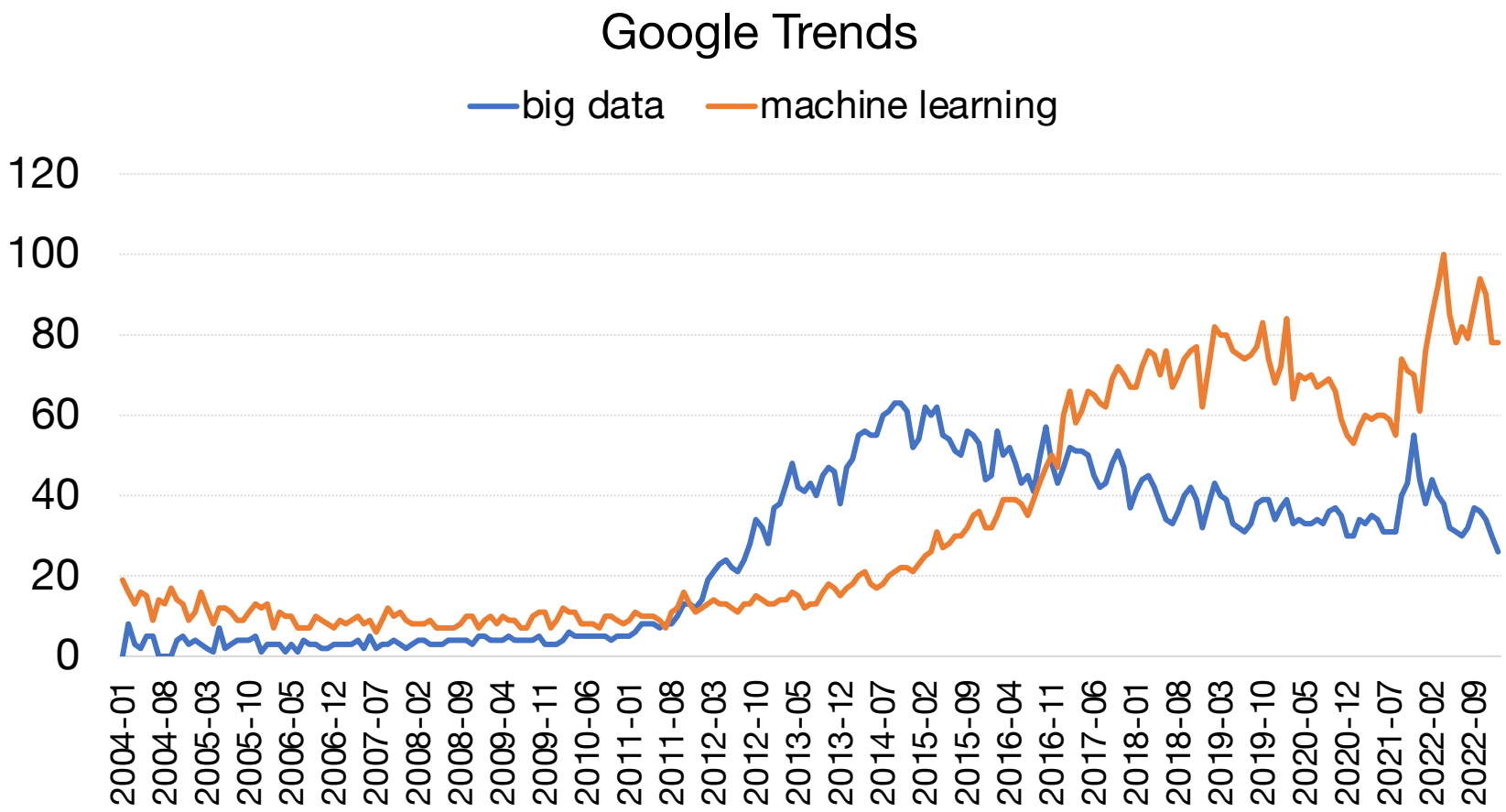
Course staff and getting help

- Instructor: Yue Cheng
  - Office hours: Thursday: 11am-12pm on Zoom
- GTA: Jingyi Gao
  - Email: etc6bd@virginia.edu
  - Office hours: Friday, 6pm-10pm on Zoom
- Discussion, questions: Ed
  - https://edstem.org/us/dashboard
  - Alternative place to ask questions about assignments, materials, and projects
  - No anonymous posts or questions
  - Can use private posts to instructor/GTA

Today's agenda

- What is this course about?
- Why are we studying Big Data Systems?
- What will you do in this course?

A brief history about Big Data



Google circa 1997

Everything is about data

"… Storage space must be used efficiently to store indices and, optionally, the documents themselves. The indexing system must process hundreds of gigabytes of data efficiently…"

"The system… downloading the last 11 million pages in just 63 hours… The sorter can be run completely in parallel; using four machines, the whole process of sorting takes about 24 hours…"

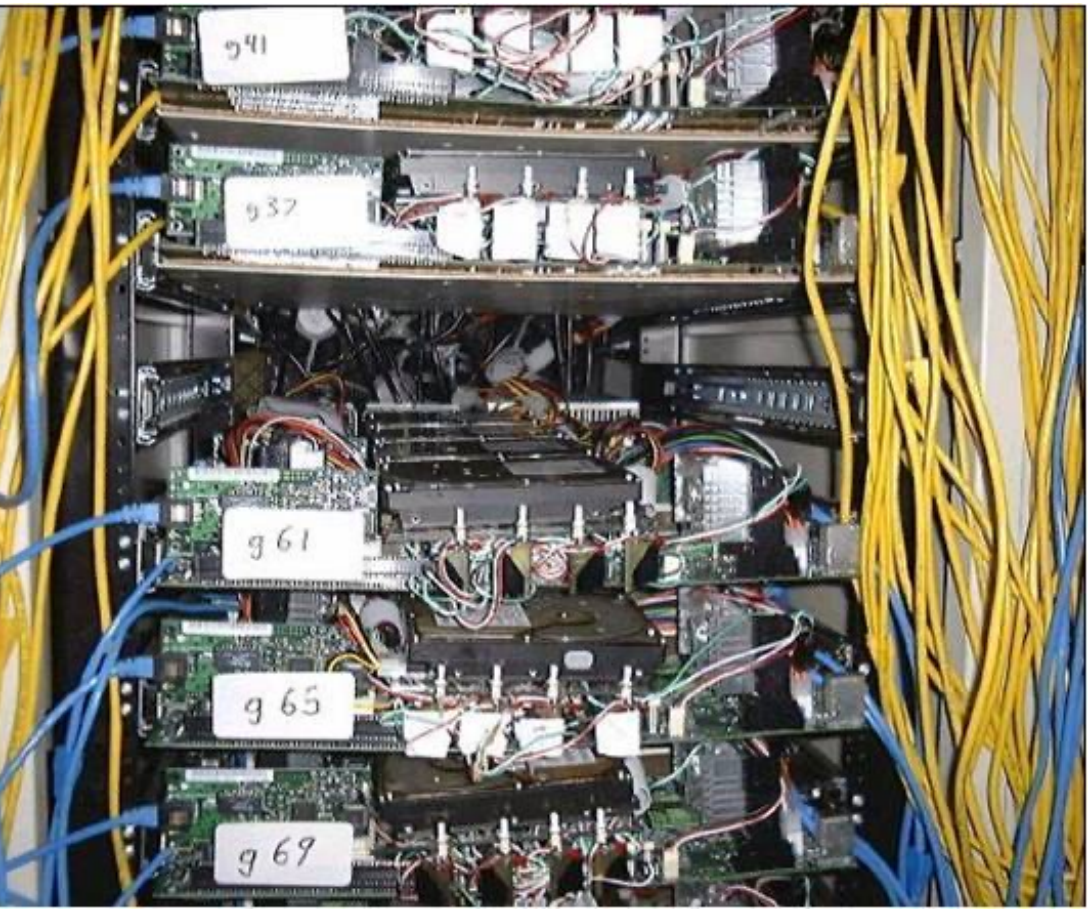
The anatomy of a large-scale hypertextual Web search engine [1]

Sergey Brin [2], Lawrence Page [*,2]

Computer Science Department, Stanford University, Stanford, CA 94305, USA

Abstract

Google circa 2000



Commodity CPUs

Lots of disks

Low bandwidth network

Cheap!


Google

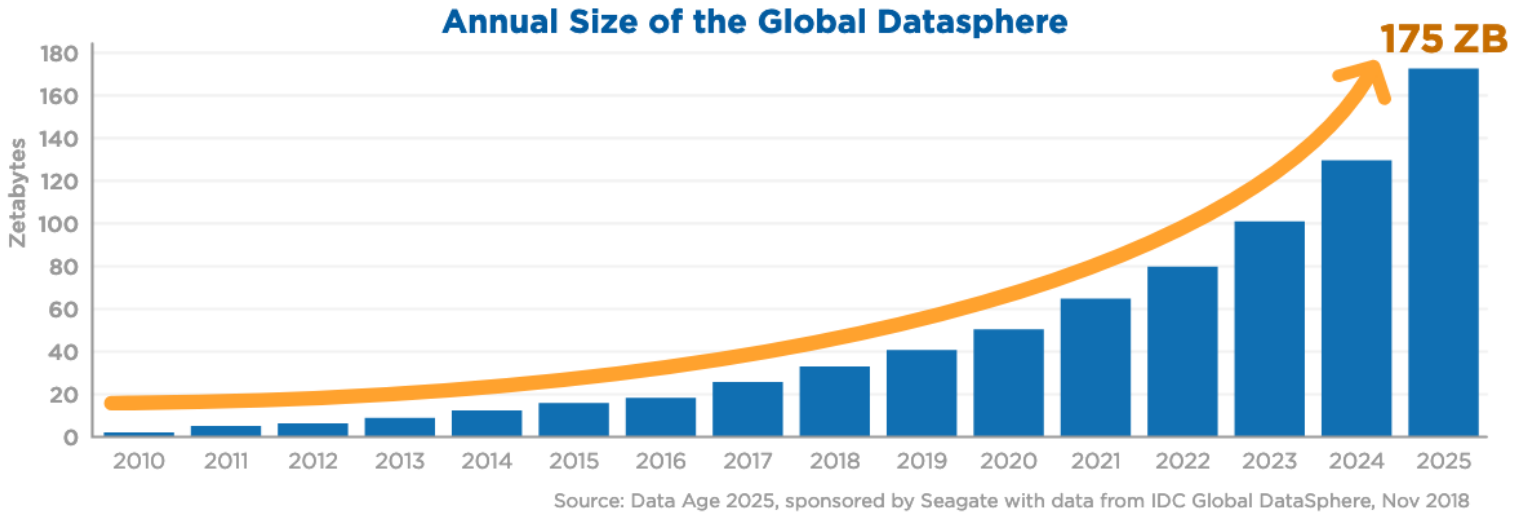

A Google Datacenter in Hamina, Finland


Facebook

Microsoft's Datacenter to be deployed on the seafloor

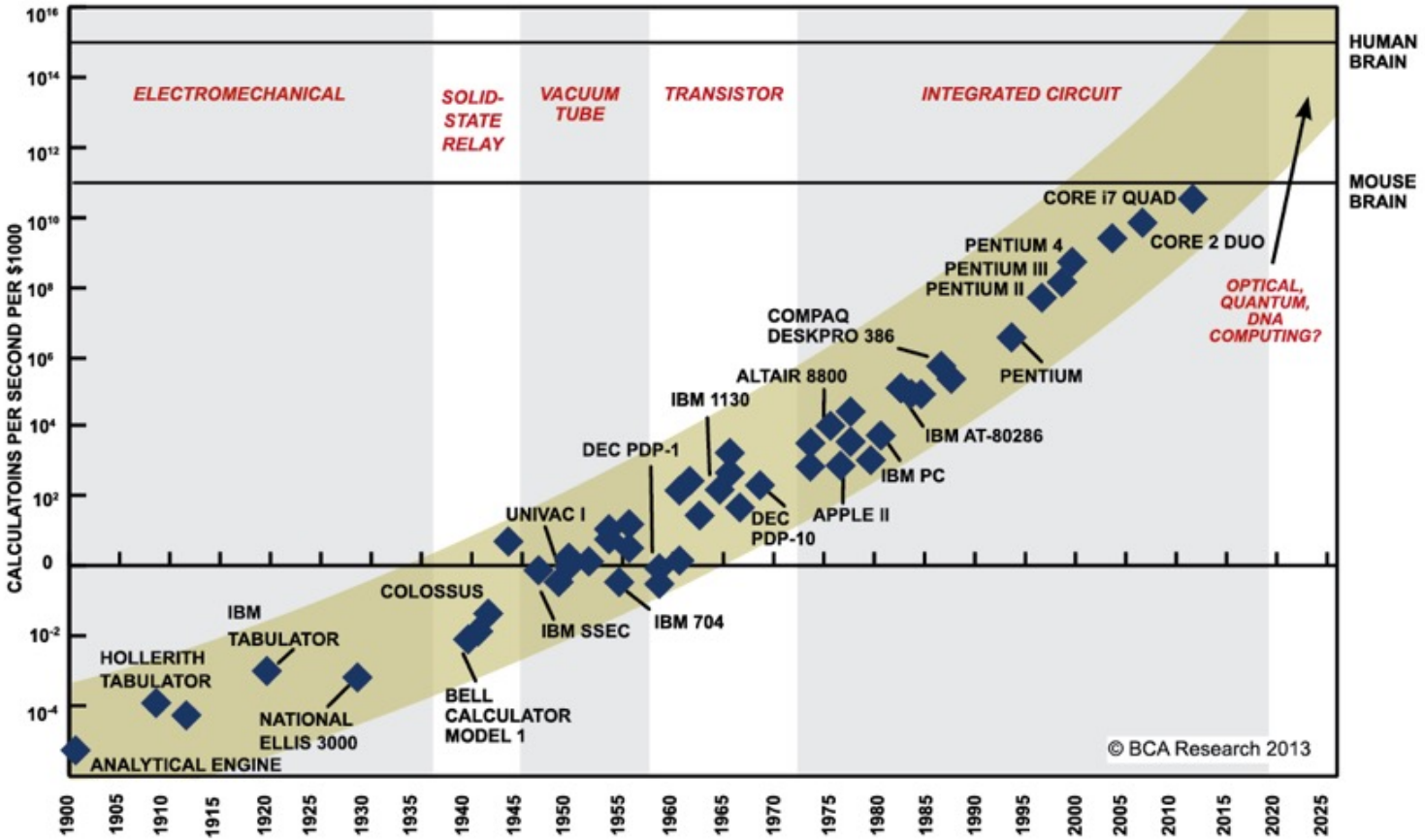
Data explosion

- Facebook's (now Meta) daily logs: 60 TB
- Google web index: 10+ PB



Exciting time in big data systems

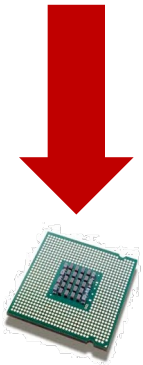
Moore's law ending → many challenges



SOURCE: RAY KURZWEIL, "THE SINGULARITY IS NEAR: WHEN HUMANS TRANSCEND BIOLOGY", P.67, *THE VIKING PRESS*, 2006. DATAPOINTS BETWEEN 2000 AND 2012 REPRESENT BCA ESTIMATES.

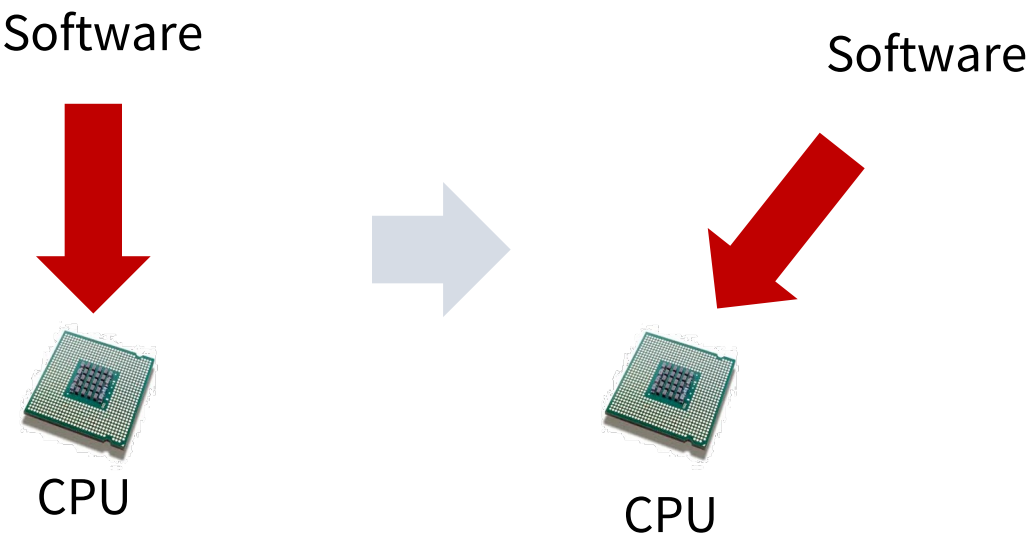
Increased complexity – Computation

Software

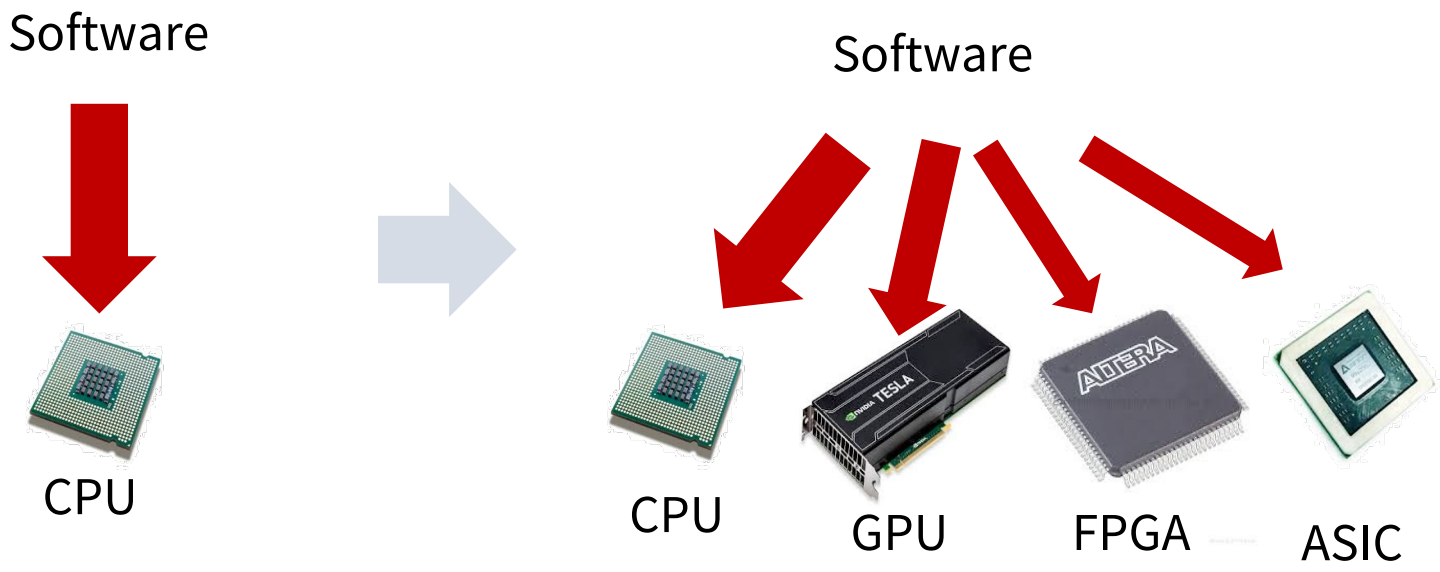


CPU

Increased complexity – Computation

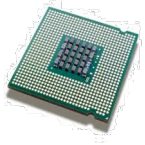


Increased complexity – Computation



Increased complexity – Memory

2015



L1/L2 cache	~1 ns
L3 cache	~10 ns
Main memory	~100 ns / ~80 GB/s / ~100GB
NAND SSD	~100 usec / ~10 GB/s / ~1 TB
Fast HDD	~10 msec / ~100 MB/s / ~10 TB

Increased complexity – Memory

2015

L1/L2 cache	~1 ns
L3 cache	~10 ns
Main memory	~100 ns / ~80 GB/s / ~100GB
NAND SSD	~100 usec / ~10 GB/s / ~1 TB
Fast HDD	~10 msec / ~100 MB/s / ~10 TB

2020

L1/L2 cache	~1 ns
L3 cache	~10 ns
HBM	~10 ns / ~1TB/s / ~10GB
Main memory	~100 ns / ~80 GB/s / ~100GB
NVM	~1 usec / ~10GB/s / ~1TB
NAND SSD	~100 usec / ~10 GB/s / ~10 TB
Fast HDD	~10 msec / ~100 MB/s / ~100 TB

Increased complexity – more and more choices

Basic tier: A0, A1, A2, A3, A4
Optimized Compute : D1, D2, D3, D4, D11, D12, D13
D1v2, D2v2, D3v2, D11v2,…
Latest CPUs: G1, G2, G3, …
Network Optimized: A8, A9
Compute Intensive: A10, A11,…

Microsoft Azure

t2.nano, t2.micro, t2.small
m4.large, m4.xlarge, m4.2xlarge, m4.4xlarge, m3.medium, c4.large, c4.xlarge, c4.2xlarge, c3.large, c3.xlarge, c3.4xlarge, r3.large, r3.xlarge, r3.4xlarge, i2.2xlarge, i2.4xlarge, d2.xlarge d2.2xlarge, d2.4xlarge,…

Amazon EC2

n1-standard-1, ns1-standard-2, ns1-standard-4, ns1-standard-8, ns1-standard-16, ns1highmem-2, ns1-highmem-4, ns1-highmem-8, n1-highcpu-2, n1-highcpu-4, n1-highcpu-8, n1-highcpu-16, n1-highcpu-32, f1-micro, g1-small…

Google Cloud

CLOUD

Extended AWS outage disrupts services across the globe

By **Diana Goovaerts** • Dec 7, 2021 05:48pm

COMPANIES > AMAZON

Breaking: AWS Experienced an Outage in Its US-East 2 Availability Zone

Yet another outage at AWS, this time at its US-East 2 availability region, leads
effects of cloud outages

Lisa D Sparks | Dec 05, 2022

MAY 21, 2020

Cows responsible for short outages to Google fiber network

Abner Li - May. 21st 2020 2:20 pm PT @technacity

Tech / Big Tech

Alibaba cloud services unit's review finds system breakdown week ago 'longest major-scale failure' in Hong Kong and Macau

- The Alibaba subsidiary's incident review found that the system failure was caused by 'a malfunction the data centre's chillers'
- The incident resulted in a lengthy service outage that stretched for more than 24 hours at some customer sites

Tracy Qu in Shanghai + myNEWS

Published: 9:00pm, 26 Dec, 2022

Why you can trust

Y. Cheng

The Joys of Real Hardware

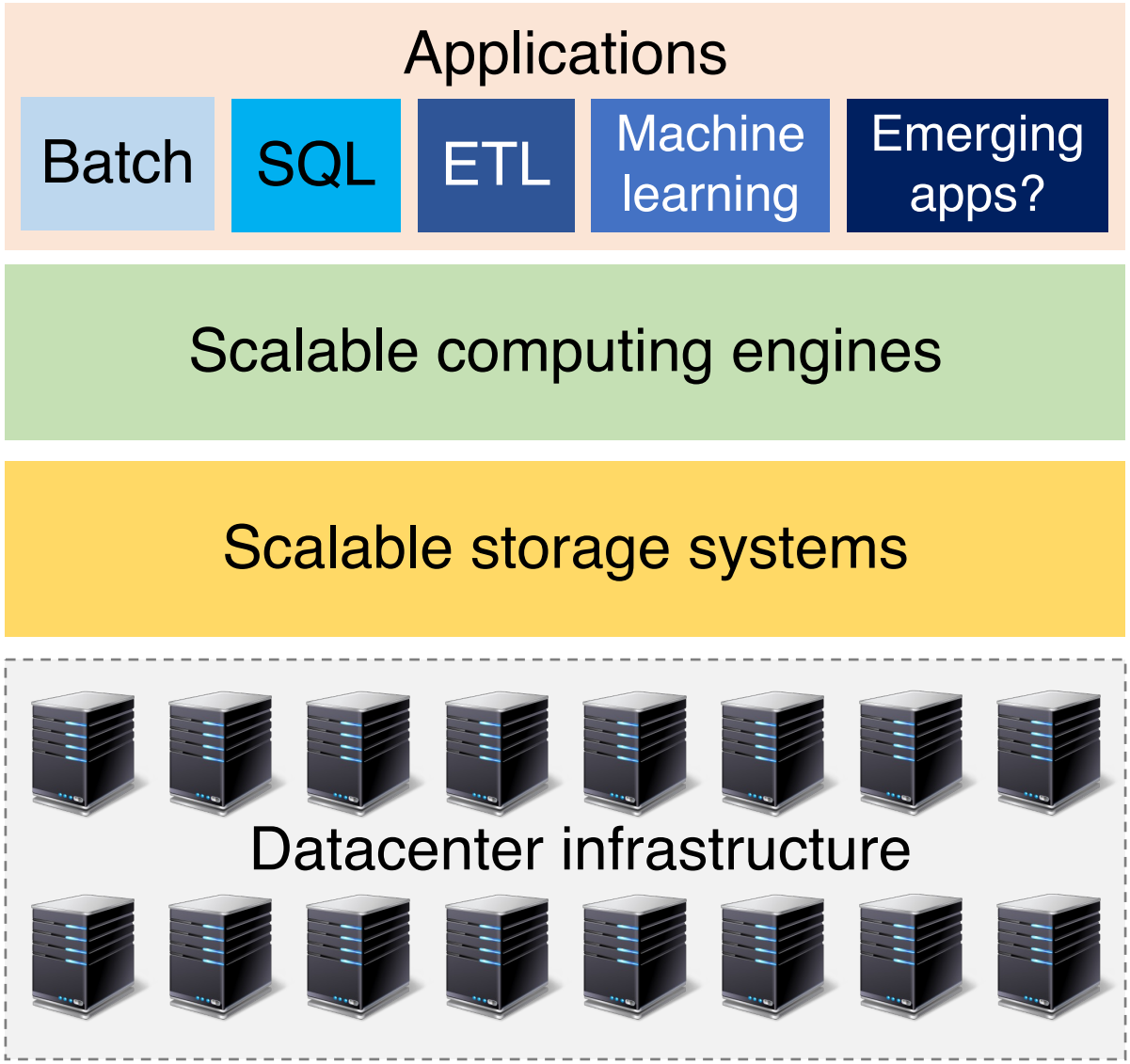
Typical first year for a new cluster:

~0.5 overheating (power down most machines in <5 mins, ~1-2 days to recover)
~1 PDU failure (~500-1000 machines suddenly disappear, ~6 hours to come back)
~1 rack-move (plenty of warning, ~500-1000 machines powered down, ~6 hours)
~1 network rewiring (rolling ~5% of machines down over 2-day span)
~20 rack failures (40-80 machines instantly disappear, 1-6 hours to get back)
~5 racks go wonky (40-80 machines see 50% packetloss)
~8 network maintenances (4 might cause ~30-minute random connectivity losses)
~12 router reloads (takes out DNS and external vips for a couple minutes)
~3 router failures (have to immediately pull traffic for an hour)
~dozens of minor 30-second blips for dns
~1000 individual machine failures
~thousands of hard drive failures
slow disks, bad memory, misconfigured machines, flaky machines, etc.

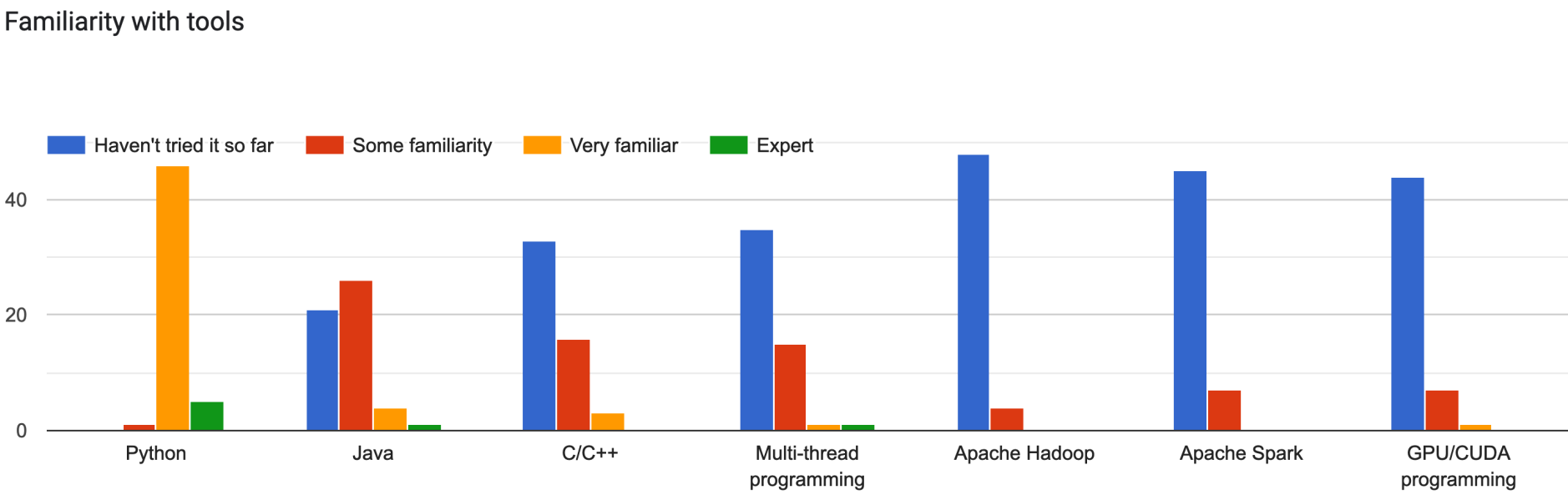
Long distance links: wild dogs, sharks, dead horses, drunken hunters, etc.

* Jeff Dean, LADIS'09

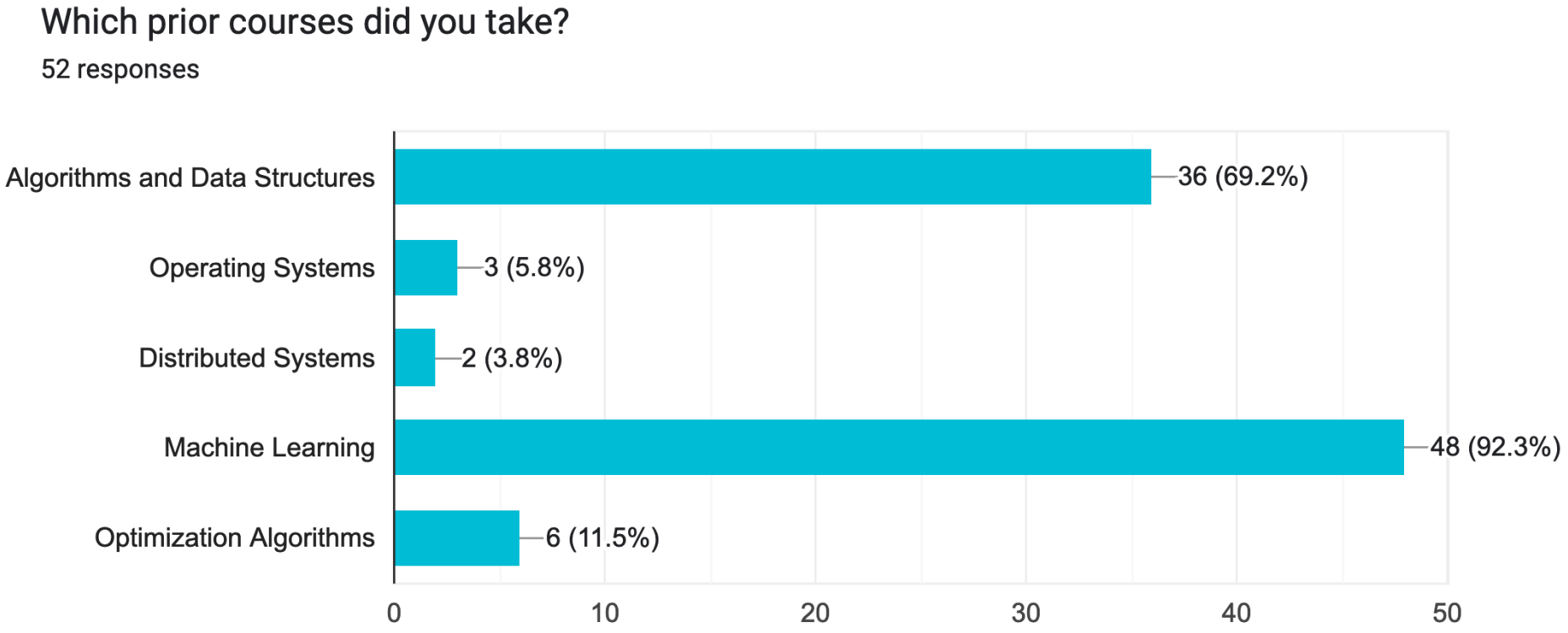
But how do we program this for processing big data?


Applications
Batch
SQL
ETL
Machine learning
Emerging apps?
Scalable computing engines
Scalable storage systems
Datacenter infrastructure

Background survey: Familiarity w/ tools



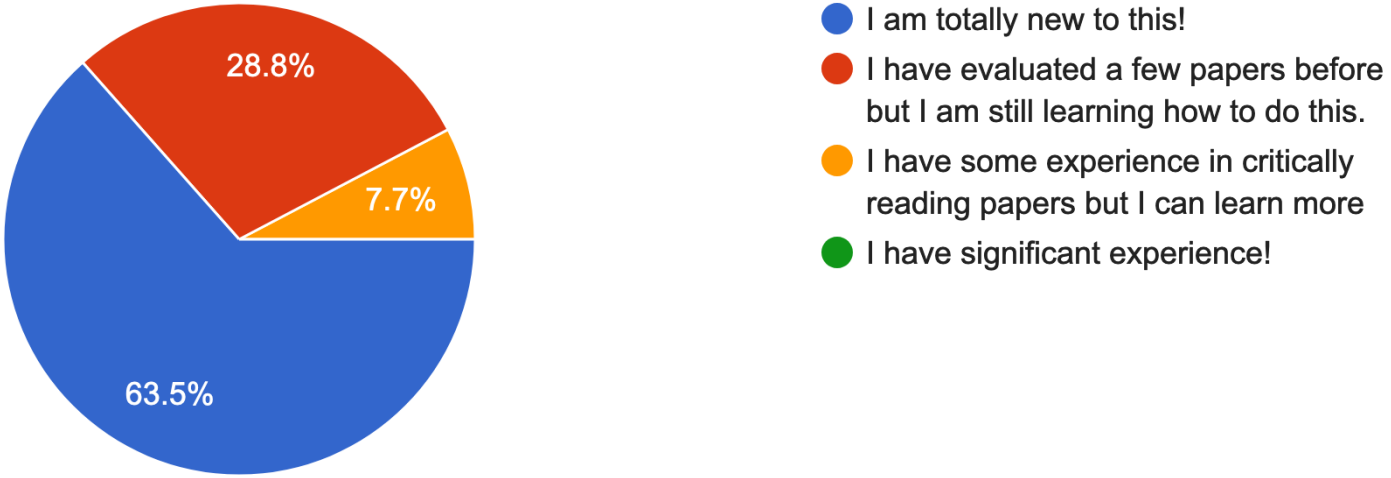
Prior courses



Paper reading

How much experience do you have in critically reading and evaluating systems research papers?

52 responses



What do you hope to learn from this course?

“Through this course, I hope to gain a deeper understanding of how data systems can be designed and constructed in such a way as to achieve superb scalability and parallelization”

“Through hands-on programming experiences, I hope to become adept at engaging with existing big data and cloud computing resources.”

“Skills associated with cloud computing and a greater knowledge of the behind-the-scenes systems that support big data.”

“Fundamentals of common cloud storage hosting services, particularly AWS”

“Building scalable machine learning algorithms that can serve millions of users”

Course syllabus

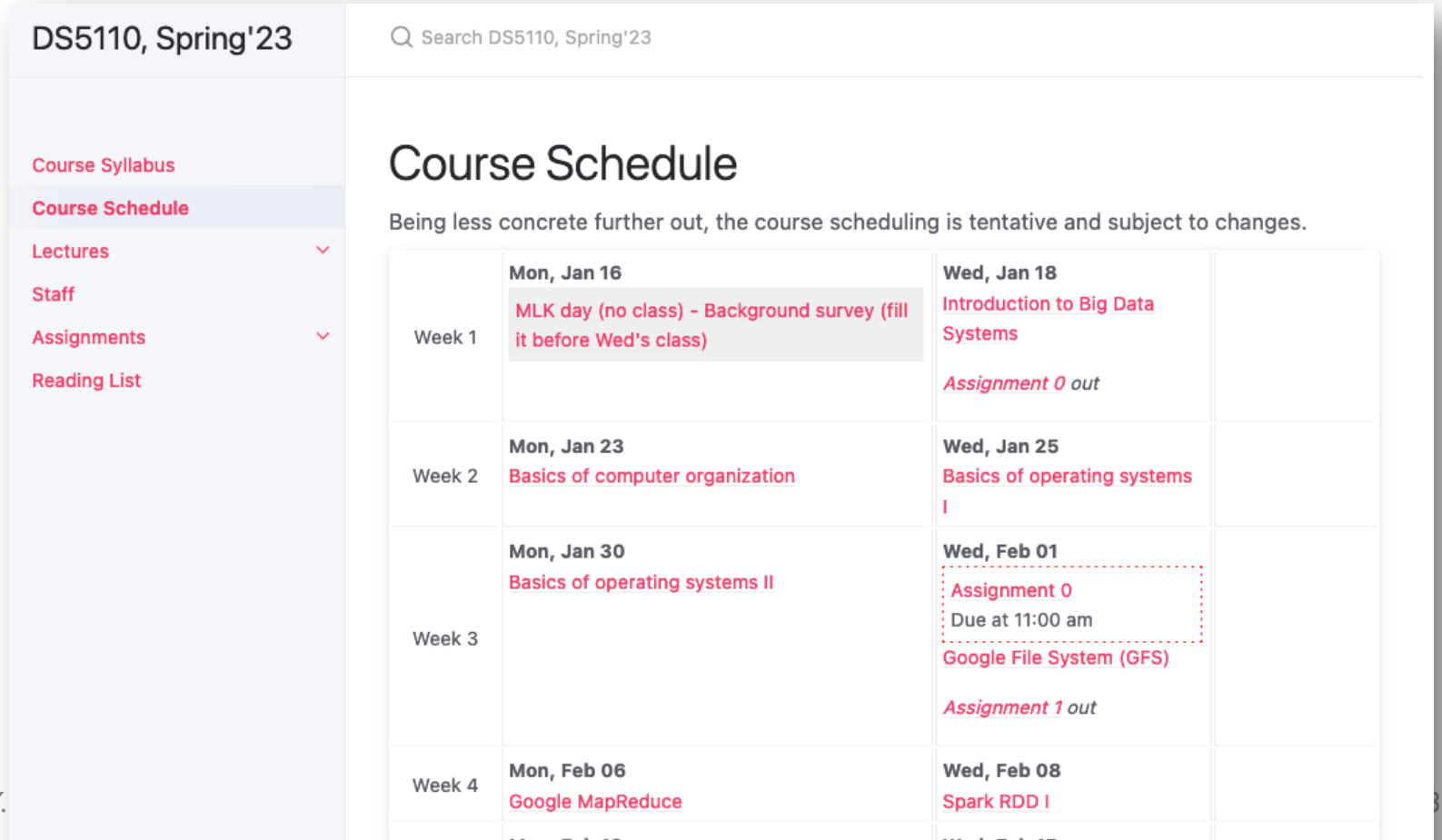
Big picture course goals

- Learn about some of the most influential works in big data systems
- Explain the design and architecture of big data systems
- Read and evaluate research papers
- Develop and deploy applications on open-source big data frameworks
- Design and report some research ideas

Schedule (tentative)

https://tddg.github.io/ds5110-spring23/

- Readings, assignments, due dates
- Less concrete further out; don't get too far ahead



Course format: Lectures

- Some lecture + some discussion
  - Slides available on course website (night before)
- First 3 weeks: Basics of computer organization and operating systems
  - Mostly from textbook
- Week 3-9: GFS/MapReduce, Spark, Python analytics
- Week 7: Midterm exam
- After midterm: Cloud data systems, ML systems, and data warehousing / datacenters
- Some lectures have required readings
  - Review forms to help drive lecture discussion

Course format: Review forms

- Goal: read and be prepared for discussion
- Review form will be posted on Ed few days before lecture
  - You need to fill out review form **by 11am** same day of lecture
  - Review form will cover required reading with a strong focus on stimulating a fruitful discussion

- No late submission will be accepted (with two wildcards)
- Contact instructor for exceptions in severe circumstances only

How to read a paper: GFS

The Google File System

Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung

Google*

ABSTRACT

We have designed and implemented the Google File System, a scalable distributed file system for large distributed data-intensive applications. It provides fault tolerance while running on inexpensive commodity hardware, and it delivers high aggregate performance to a large number of clients.

While sharing many of the same goals as previous distributed file systems, our design has been driven by observations of our application workloads and technological environment, both current and anticipated, that reflect a marked departure from some earlier file system assumptions. This has led us to reexamine traditional choices and explore radically different design points.

The file system has successfully met our storage needs. It is widely deployed within Google as the storage platform for the generation and processing of data used by our service as well as research and development efforts that require large data sets. The largest cluster to date provides hundreds of terabytes of storage across thousands of disks on over a thousand machines, and it is concurrently accessed

1. INTRODUCTION

We have designed and implemented the Google File System (GFS) to meet the rapidly growing demands of Google's data processing needs. GFS shares many of the same goals as previous distributed file systems such as performance, scalability, reliability, and availability. However, its design has been driven by key observations of our application workloads and technological environment, both current and anticipated, that reflect a marked departure from some earlier file system design assumptions. We have reexamined traditional choices and explored radically different points in the design space.

First, component failures are the norm rather than the exception. The file system consists of hundreds or even thousands of storage machines built from inexpensive commodity parts and is accessed by a comparable number of client machines. The quantity and quality of the components virtually guarantee that some are not functional at any given time and some will not recover from their current failures. We have seen problems caused by application

How to read a paper: Summary

- Start your reading early
- Repeat, give time between iterations
- 1st pass: Read abstract, introduction, section headings, conclusion
- 2nd pass: Read all sections, make notes
- Iterate… (if your schedule fits)

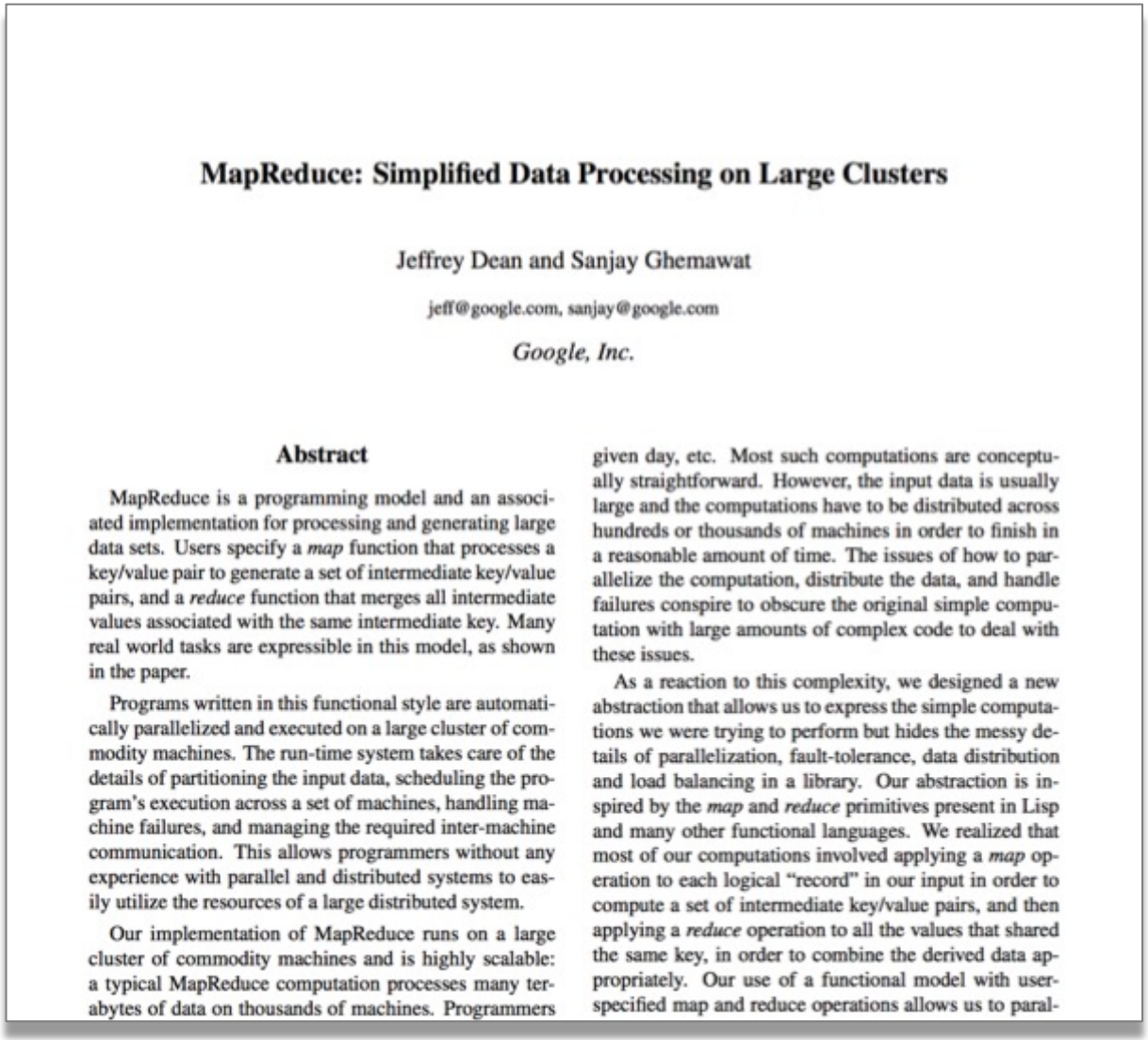
- Some key points (examples):
  - What is the problem being solved?
  - What are the main contributions? What is the design?
  - What workloads, setups were considered in evaluation?
  - How do they compare to prior work?
  - What parts of the claims are adequately backed up?

Course format: Discussion

- Your participation is very important
  - As an indicator of how well you've prepared

- Paper review form examples
  - One or two sentence summary of the paper
  - Description of the problem or assumptions made
  - Comparison to other papers that you read in class?
  - Experimental setup and what the results mean?
  - One flaw or thing that can be improved?
  - …

Research results matter: MapReduce



MapReduce: Simplified Data Processing on Large Clusters

Jeffrey Dean and Sanjay Ghemawat

jeff@google.com, sanjay@google.com

Google, Inc.

Abstract

MapReduce is a programming model and an associated implementation for processing and generating large data sets. Users specify a *map* function that processes a key/value pair to generate a set of intermediate key/value pairs, and a *reduce* function that merges all intermediate values associated with the same intermediate key. Many real world tasks are expressible in this model, as shown in the paper.

Programs written in this functional style are automatically parallelized and executed on a large cluster of commodity machines. The run-time system takes care of the details of partitioning the input data, scheduling the program's execution across a set of machines, handling machine failures, and managing the required inter-machine communication. This allows programmers without any experience with parallel and distributed systems to easily utilize the resources of a large distributed system.

Our implementation of MapReduce runs on a large cluster of commodity machines and is highly scalable: a typical MapReduce computation processes many terabytes of data on thousands of machines. Programmers

given day, etc. Most such computations are conceptually straightforward. However, the input data is usually large and the computations have to be distributed across hundreds or thousands of machines in order to finish in a reasonable amount of time. The issues of how to parallelize the computation, distribute the data, and handle failures conspire to obscure the original simple computation with large amounts of complex code to deal with these issues.

As a reaction to this complexity, we designed a new abstraction that allows us to express the simple computations we were trying to perform but hides the messy details of parallelization, fault-tolerance, data distribution and load balancing in a library. Our abstraction is inspired by the *map* and *reduce* primitives present in Lisp and many other functional languages. We realized that most of our computations involved applying a *map* operation to each logical "record" in our input in order to compute a set of intermediate key/value pairs, and then applying a *reduce* operation to all the values that shared the same key, in order to combine the derived data appropriately. Our use of a functional model with user-specified map and reduce operations allows us to paral-

Research results matter: NoSQL

Dynamo: Amazon's Highly Available Key-value Store

Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall and Werner Vogels

Amazon.com

ABSTRACT

Reliability at massive scale is one of the biggest challenges we face at Amazon.com, one of the largest e-commerce operations in the world; even the slightest outage has significant financial consequences and impacts customer trust. The Amazon.com platform, which provides services for many web sites worldwide, is implemented on top of an infrastructure of tens of thousands of servers and network components located in many datacenters around the world. At this scale, small and large components fail continuously and the way persistent state is managed in the face of these failures drives the reliability and scalability of the software systems.

This paper presents the design and implementation of Dynamo, a highly available key-value storage system that some of Amazon's core services use to provide an "always-on" experience. To achieve this level of availability, Dynamo sacrifices consistency under certain failure scenarios. It makes extensive use of object versioning and application-assisted conflict resolution in a manner that provides a novel interface for developers to use.

Categories and Subject Descriptors

D.4.2 [**Operating Systems**]: Storage Management; D.4.5 [**Operating Systems**]: Reliability; D.4.2 [**Operating Systems**]: Performance;

General Terms

Algorithms, Management, Measurement, Performance, Design, Reliability.

1. INTRODUCTION

Amazon runs a world-wide e-commerce platform that serves tens of millions customers at peak times using tens of thousands of servers located in many data centers around the world. There are

One of the lessons our organization has learned from operating Amazon's platform is that the reliability and scalability of a system is dependent on how its application state is managed. Amazon uses a highly decentralized, loosely coupled, service oriented architecture consisting of hundreds of services. In this environment there is a particular need for storage technologies that are always available. For example, customers should be able to view and add items to their shopping cart even if disks are failing, network routes are flapping, or data centers are being destroyed by tornados. Therefore, the service responsible for managing shopping carts requires that it can always write to and read from its data store, and that its data needs to be available across multiple data centers.

Dealing with failures in an infrastructure comprised of millions of components is our standard mode of operation; there are always a small but significant number of server and network components that are failing at any given time. As such Amazon's software systems need to be constructed in a manner that treats failure handling as the normal case without impacting availability or performance.

To meet the reliability and scaling needs, Amazon has developed a number of storage technologies, of which the Amazon Simple Storage Service (also available outside of Amazon and known as Amazon S3), is probably the best known. This paper presents the design and implementation of Dynamo, another highly available and scalable distributed data store built for Amazon's platform. Dynamo is used to manage the state of services that have very high reliability requirements and need tight control over the tradeoffs between availability, consistency, cost-effectiveness and performance. Amazon's platform has a very diverse set of applications with different storage requirements. A select set of applications requires a storage technology that is flexible enough to let application designers configure their data store appropriately

amazon DynamoDB

Textbooks?

- Papers and documentations (required or optional) serve as reference for many topics that aren't directly covered by a text

- Slides/lecture notes

- Two optional textbooks (free PDFs available online)
  - "**Operating Systems: Three Easy Pieces (OETEP)**" by Remzi H. Arpaci-Dusseau and Andrea C. Arpaci-Dusseau
  - "**Distributed Systems 3rd edition**" by van Steen and Tenenbaum will supply optional alternate explanations

Assignments

- Two programming assignments in Python on AWS
  - Assignment 0: Using AWS Academy and AWS EC2
  - **Assignment 1:** A tour of Apache HDFS and Spark
  - **Assignment 2:** Dask.distributed analytics

- All assignments are individual

- Short coding-based assignments
  - Preparation for the course project

Course project

- Goal: Explore new research ideas or implementation in the area of big data systems
  - Define the problem
  - Execute the research
  - Write up and present your research

- Two project styles (tentative)
  - **Data analysis:** Work towards analyzing large, real-world dataset of interest using big data tools
  - **System implementation:** Work towards open-source contribution to big data tools

Course project steps

- I will distribute a list of project ideas (Week 4)
  - You can either choose one or come up with your own
  - We will meet together to discuss

- (Not mandatory) Pick your teammates: a team of up to 3 students

- Milestones (tentative)
  - Project bid + team composition due Friday, Feb 24
  - Project checkpoint 1 due Friday, Mar 24
  - Project checkpoint 2 due Friday, Apr 14
  - Final project presentation on Wed, Apr 26
  - Final project everything due Wed, May 3

Grading

- Assignments (20% total)
  - Assignment 0 (0%)
  - Assignment 1 (10%)
  - Assignment 2 (10%)
- Reading review forms (5%)
- Quizzes (5%) and in-class participation (5%)
- Midterm exam (15%): open-book, open-note
- Project (50%)

Assignment 0

- Assignment 0 (0%): Using AWS Academy and AWS EC2

FAQ: Why take this course?

- **Interesting** – hard problems, powerful solutions
- **Used by real systems** – driven by the rise of large businesses (e.g., Google, Amazon)
- **Active research area** – lots of progress + big unsolved problems
- **Hands-on** – you'll build something by the end of the semester

FAQ: What this course is not about

- **Not** a course on databases, relational models, or SQL
- **Not** a course on internals of RDBMSs
- **Not** a training module for how to use Spark
- **Not** a course on ML theories or data mining algorithms