CS 582: Distributed Systems

Course Overview and Introduction

Dr. Zafar Ayyub Qazi

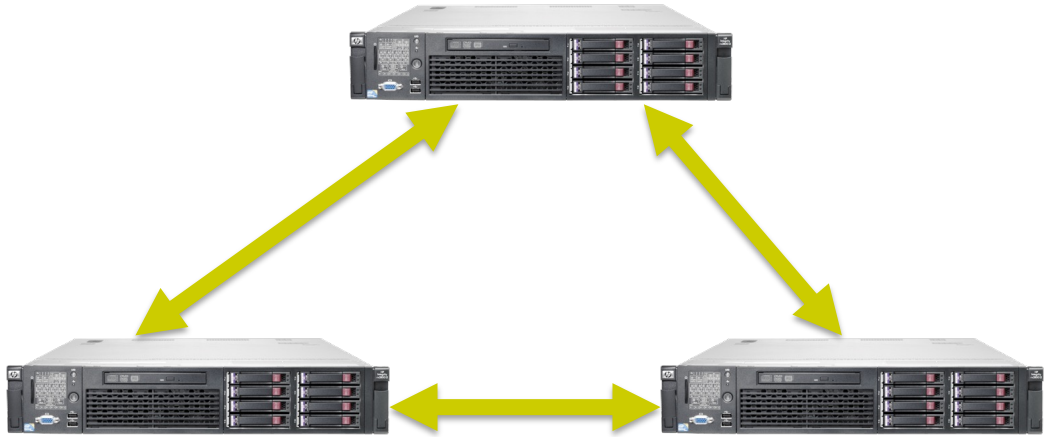
Fall 2024

Agenda for today's class

1. What is a distributed system, and why do we need them?
2. Where are distributed systems used?
3. Why should you care, and what is this course about?
4. What are the teaching methodology, workload, grading, & policies?
5. Who are the course staff?
6. Why is the Go language used for this course?

What is a distributed system?

What is a distributed system?



1. Multiple computers
2. Connected by a network
3. Doing something together

What is a distributed system?

- **"A collection of autonomous computing elements, connected by a network, which appears to its users as a single coherent system"**

— *Distributed Systems: Principles and Paradigms Book*
(Steen and Tanenbaum)

What is a distributed system?

"A distributed system is one in which the failure of a computer you didn't even know existed can render your own computer unusable"

— *Leslie Lamport*
(Turing Award Winner)

Why distributed systems?

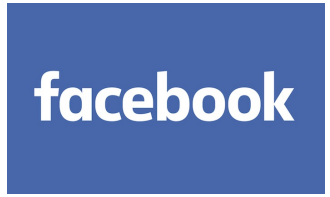
- Or, why not 1 computer?

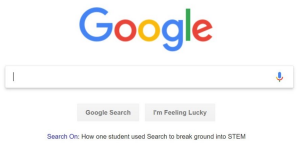
Why distributed systems?

- Failures
  - A service shouldn't fail when one computer does
- Limited computation/storage/…
  - As your service grows you want to add more CPU/GPU cycles, more memory, more storage, etc.
- Geographic separation
  - A web request in Pakistan is faster served by a server in Pakistan than US

Where are distributed systems used?

Where are distributed systems used?



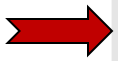


Where are distributed systems used?

- Web Search (e.g., Google, Bing)
- Online Shopping Services (e.g., Amazon, Walmart)
- File Synchronization (e.g., Dropbox, iCloud)
- Social Networks (e.g., Facebook, Twitter)
- Music (e.g., Spotify, Apple Music)
- Ride Sharing (e.g., Uber, Lyft)
- Video (e.g., YouTube, Netflix)
- Online gaming (e.g., Fortnite, DOTA2)
- Cellular Networks
- Online Banking Systems
- LLM applications (e.g., Perplexity, ChatGPT)
- …

Example: Scaling up Facebook

- 2004: Facebook startec
  - Web server frontend to a
  - Database store posts, fri
- 2008: 100M users
- 2010: 500M users



Scaling Facebook to 500 Million Users and Beyond

21 July 2010 at 11:06

Today we hit an important milestone for Facebook - half a billion users. It's particularly exciting to those of us in engineering and operations who build the systems to handle this massive growth. I started at Facebook four years ago when we had seven million users (which seemed like a really big number at the time) and the technical challenges along the way have been just as crazy as you might imagine.

A few of the big numbers we deal with:
* 500 million active users
* 100 billion hits per day
* 50 billion photos
* 2 trillion objects cached, with hundreds of millions of requests per second
* 130TB of logs every day

Example: Scaling up Facebook

Scaling Memcache at Facebook

Rajesh Nishtala, Hans Fugal, Steven Grimm, Marc Kwiatkowski, Herman Lee, Harry C. Li,
Ryan McElroy, Mike Paleczny, Daniel Peek, Paul Saab, David Stafford, Tony Tung,
Venkateshwaran Venkataramani

{rajeshn,hans}@fb.com, {sgrimm, marc}@facebook.com, {herman, hcli, rm, mpal, dpeek, ps, dstaff, ttung, veeve}@fb.com

Facebook Inc.

Abstract: `Memcached` is a well known, simple, in-memory caching solution. This paper describes how Facebook leverages `memcached` as a building block to construct and scale a distributed key-value store that supports the world's largest social network. Our system handles billions of requests per second and holds trillions of items to deliver a rich experience for over a billion users around the world.

1 Introduction

Popular and engaging social networking sites present significant infrastructure challenges. Hundreds of millions of people use these networks every day and impose computational, network, and I/O demands that traditional web architectures struggle to satisfy. A social network's infrastructure needs to (1) allow near real-time communication, (2) aggregate content on-the-fly from multiple sources, (3) be able to access and update very popular shared content, and (4) scale to process millions of user requests per second.

however, web pages routinely fetch thousands of key-value pairs from `memcached` servers.

One of our goals is to present the important themes that emerge at different scales of our deployment. While qualities like performance, efficiency, fault-tolerance, and consistency are important at all scales, our experience indicates that at specific sizes some qualities require more effort to achieve than others. For example, maintaining data consistency can be easier at small scales if replication is minimal compared to larger ones where replication is often necessary. Additionally, the importance of finding an optimal communication schedule increases as the number of servers increase and networking becomes the bottleneck.

This paper includes four main contributions: (1) We describe the evolution of Facebook's `memcached`-based architecture. (2) We identify enhancements to `memcached` that improve performance and increase memory efficiency. (3) We highlight mechanisms that improve our ability to operate our system at scale. (4) We characterize the production workloads imposed on

Example: High availability at Amazon

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.

An excerpt from a paper by Amazon's engineering team describing the design of their data store, Dynamo.
Paper: "Dynamo: Amazon's Highly Available Key-value Store", SOSP'07

Failure are common in large systems

Failure are common in large systems

Failure statistics from clusters in Google data centers*

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.

Design, Lessons, and Advice for Building Large Scale Distributed Systems by Jeff Dean, Google Senior Fellow and lead of Google AI, 2009

Example: Strong consistency at scale in Google

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, Dale Woodford

Google, Inc.

Abstract

Spanner is Google's scalable, multi-version, globally-distributed, and synchronously-replicated database. It is the first system to distribute data at global scale and support externally-consistent distributed transactions. This paper describes how Spanner is structured, its feature set, the rationale underlying various design decisions, and a novel time API that exposes clock uncertainty. This API and its implementation are critical to supporting external consistency and a variety of powerful features: non-blocking reads in the past, lock-free read-only transactions, and atomic schema changes, across all of Spanner.

tency over higher availability, as long as they can survive 1 or 2 datacenter failures.

Spanner's main focus is managing cross-datacenter replicated data, but we have also spent a great deal of time in designing and implementing important database features on top of our distributed-systems infrastructure. Even though many projects happily use Bigtable [9], we have also consistently received complaints from users that Bigtable can be difficult to use for some kinds of applications: those that have complex, evolving schemas, or those that want strong consistency in the presence of wide-area replication. (Similar claims have been made by other authors [37].) Many applications at Google

However, according to the CAP theorem, no system can simultaneously achieve strong consistency, availability, and partition tolerance

Example: Scaling Distributed ML

Scaling Distributed Machine Learning with the Parameter Server

Mu Li*‡, David G. Andersen*, Jun Woo Park*, Alexander J. Smola*†, Amr Ahmed†,
Vanja Josifovski†, James Long†, Eugene J. Shekita†, Bor-Yiing Su†

*Carnegie Mellon University ‡Baidu †Google

{muli, dga, junwoop}@cs.cmu.edu, alex@smola.org, {amra, vanjaj, jamlong, shekita, boryiingsu}@google.com

Abstract

We propose a parameter server framework for distributed machine learning problems. Both data and workloads are distributed over worker nodes, while the server nodes maintain globally shared parameters, represented as dense or sparse vectors and matrices. The framework manages asynchronous data communication between nodes, and supports flexible consistency models, elastic scalability, and continuous fault tolerance.

To demonstrate the scalability of the proposed framework, we show experimental results on petabytes of real data with billions of examples and parameters on problems ranging from Sparse Logistic Regression to Latent Dirichlet Allocation and Distributed Sketching.

≈ #machine × time	# of jobs	failure rate
100 hours	13,187	7.8%
1,000 hours	1,366	13.7%
10,000 hours	77	24.7%

Table 1: Statistics of machine learning jobs for a three month period in a data center.

cost of synchronization and machine latency is high.

- At scale, fault tolerance is critical. Learning tasks are often performed in a cloud environment where machines can be unreliable and jobs can be preempted.

To illustrate the last point, we collected all job logs for a three month period from one cluster at a large internet company. We show statistics of batch machine learning

Large models can have billions/trillions of parameters. GPT-4 has more than a trillion parameters

Why should you care?

Why should you care?

- If you plan to work in industry

Wagner Vogels,
Amazon CTO

Job openings in my group

What kind of things I am looking for in you?

You know your distributed systems theory: You know about **logical time, snapshots, stability, message ordering, but also acid and multi-level transactions** …You have at least once tried to understand **Paxos** by reading the original paper."

You have a good sense for distributed systems practice

You have actually built some real systems yourself

*AllThingsDistributed by Wagner Vogels

Why should you care?

- If you plan to go for graduate school
  - Necessary for pursing research in distributed systems and computer networks
  - Important for ML, big data processing besides other areas

What will you learn in the course?

First Part: Fundamentals

- Distributed systems principles, concepts, and algorithms
  - How do detect failures?
  - How to reason about timing and event ordering?
  - How to get multiple computer to reach consensus?
  - How design a system to achieve a specific consistency requirement?
  - How to handle distributed concurrent transactions?
  - …

Fundamentals

- System Models
- Failure Detectors
- Remote Procedure Calls (RPCs)
- Physical time synchronization
- Logical Clocks: Lamport Clocks and Vector Clocks
- Theoretical results: CAP and FLP
- Types of consistency models:
  - E.g., linearizability, sequential consistency, causal consistency
- Consensus algorithms: Paxos, Raft, PBFT

Second Part: Real-World Case Studies

- Distributed data stores
  - Apache ZooKeeper: A Widely used wait-free coordination service
  - Dynamo at Amazon: Amazon's highly available key-value storage system.
  - Memcache at Facebook: Facebook's scalable caching system
  - Google's Spanner database: Globally distributed & consistent databases

- Distributed big data processing and ML applications
  - MapReduce: Big Data Processing Framework
  - Spark: A Fault-Tolerant In-Memory Big Data Processing Framework
  - Distributed Parameter Server: Scaling Machine Learning with Parameter Server
  - Ray: Framework for Emerging AI Applications

Key Learning Outcomes

- Develop a solid understanding of fundamentals of distributed sys.
- Design, implement and debug distributed systems
- Identify suitable distributed systems given an application

What is the teaching methodology, workload, grading, and policies?

Make sure you carefully go over the course outline

You are responsible for reading everything in the course outline

Teaching Philosophy (1/2)

- Freely and actively engage with the content
  - Ask questions and participate in class

- Building strong intuition for complex problems
  - **"Why"** does this work
  - "**What if**" if we design it in some other way

Teaching Philosophy (2/2)

- Focus on fundaments and first principles
  - Develop a clear and deep understanding of the foundations of the discipline
- Case study driven approach
  - How fundamental concepts are synthesized to design practical distributed sys.?
- To facilitate this process, you must engage

"As a Junior, I was really intimidated when I first enrolled in the course. However, soon afterward, Distributed Systems became my favourite course this semester. … We started from the foundational concepts and gradually built upon them. We would break down complex, sophisticated concepts into bite size pieces that all of us could understand. At the start of every lecture, we would reflect on what we were previously doing and more importantly why. This ensured that all of us got the bigger picture of why certain distribution algorithms/systems were designed the way they were."

--- *Anonymous student from an earlier offering of CS 582*

Teaching Methodology

- In-Person Lectures
  - Lectures will not be recorded

- Slack for course-related discussion
  - Announcements, questions, clarifications
  - You all **must** signup on Slack
  - You will receive a signup email after class

Learning the Material (1/3)

- Attend lectures
  - Participate in class discussions
  - You are responsible for everything covered in class

- Read the book
  - Distributed Systems: Principles and Paradigms, 3rd Edition, by Andrew S. Tanenbaum and Maarten Van Steen.
  - Other optional books recommended on the course outline
  - Books will be made available on LMS under the Books folder
  - Recommended readings for lectures given in the course outline
  - You will not be tested on topics we didn't cover in lectures

Learning the Material (2/3)

- Post questions on Slack
  - Also, help each other by answering queries on Slack
  - Don’t send course staff direct emails
  - Do not expect course staff to respond outside office hours and on weekends

- Drop by the instructor and TAs office hours

Learning the Material (3/3)

- Make sure you do all programming assignments
  - Provide invaluable experience of designing, implementing, debugging distributed systems

- Attend programming tutorials
  - About Go and individual assignments

Grading

- Four programming assignments: 35%
- Quizzes: 25%
  - In-class, Announced
  - N-1 policy (to accommodate for different issues – no retakes)
- Homeworks (on assigned paper readings): 15%
- Final Exam (Comprehensive): 25%

Programming Assignments

- Practice designing, implementing and debugging dist. systems
- Assignments will be in Go language

Tentative release and due dates are available on the course outline

- You will build a fault-tolerant key-value storage service using Raft consensus algorithm
- Assignment 1 is designed to help you get started with Go
- Tutorial on Go on this Friday (tentative)

Policies: Write your own code

- All assigned work must be done individually (unless specified otherwise)
- You cannot copy other people's code, online solutions, or from any other sources. If you are unsure, then ask the course staff
- Students are not allowed to look at anyone else's code or allow any else to look at their code
- Students must be prepared to explain any program code they submit
- All submissions are subject to plagiarism detection
- Students are strongly advised that any act of plagiarism will be **directly** reported to the Disciplinary Committee
- Students are responsible for protecting their LMS and Zambeel credentials and must not share them with any other student
- Students are not allowed to use AI-based applications to generate code …

…

Policies: Write your own code

- All assigned work must be done individually (unless specified otherwise)
- You cannot copy other people's code, online solutions, or from any other sources. If you are unsure, then ask the course staff
- Students are not allowed to look at anyone else's code or allow any else to look at their code
- Students must be prepared to explain any program code they submit
- All submissions are subject to plagiarism detection
- Students are strongly advised that any act of plagiarism will be **directly** reported to the Disciplinary Committee
- Students are responsible for protecting their LMS and Zambeel credentials and must not share them with any other student
- Students are not allowed to use AI-based applications to generate code …

Don't Plagiarize!

…

Integrity Violations

- MOSS: plagiarism detection tool for programming assignments
- We do MOSS + manual inspection
- Cases will be directly reported to the Disciplinary Committee

Late day policy for programming assignments

- 90% for work submitted up to 24 hours late
- 80% for work submitted up to 2 days late
- 70% for work submitted up to 3 days late
- 60% for work submitted up to 4 days late
- 50% for work submitted after 5 days late

- In addition, you have five “free” late days during the semester

Workload

- Assignments will take substantial time
  - Key is to start early and leave time for debugging
  - There is going to be gradual rampup in the difficulty of assignments

- We will have 11 paper readings in the second part of the course
  - Typically one reading in a week

Who are the course staff?

Instructor: Dr. Zafar Ayyub Qazi

- Academic Background
  - Postdoctoral scholar at UC Berkeley, 2016-2017
  - PhD in Computer Science from Stony Brook University, 2010-2015
- Area of Research
  - Cellular Networks, Distributed Storage, Internet Affordability, Online Safety & Privacy, and Generative AI
- Recipient of Google Research Award
- Recognized in the AI 2000 list, among the 100 most influential researchers in the area of computer networking from 2011-20*

*https://www.aminer.cn/ai2000/cn

Teaching Assistants

- Danish Athar
- Muhammad Ahmed
- Hamna Shafqat
- Muhammed Ayain
- Maryam Usman
- Chaudhary Hammad Javed

Why Go?

Go Background

- Go is a programming language designed by Google to help solve Google's problems

- Designed for writing, reading, debugging and maintaining large software systems

Go is a compiled, concurrent, garbage-collected, statically typed language developed at Google.

Who are the founders?

- Ken Thompson (B, C, Unix, UTF-8)
- Rob Pike (Unix, UTF-8)
- Robert Griesemer (Hotspot, JVM)

...and a few others engineers at Google

Why are we using it in this course?

- Garbage collected, type safe → eliminates many bugs
- Excellent abstractions for building distributed systems
  - E.g., Go channels and routines
- Useful RPC libary that we will be use in programming assignments

Who are using Go in industry?

Who are using Go?

- https://github.com/golang/go/wiki/GoUsers

Questions

Next Lecture

- System models and failure detectors