Vertex University

Department of Computer Science & Engineering

CS-350
Fall Semester 2026
Location: Meridian Hall 102

CS-350: Distributed Systems & Cloud Infrastructure

Welcome Packet, Syllabus Highlights & Course FAQ

Professor: Dr. Evelyn Vance
Email: e.vance@vertex.edu
Office: Meridian Hall, Suite 405
Office Hours: Tue/Thu 2:00 PM – 3:30 PM
Lead TA: Jordan Lee
Email: j.lee@vertex.edu
Office Hours (TA): Wed 10:00 AM – 12:00 PM (Nexa Lab 202)

Welcome to CS-350! Distributed computing is the default mode of operation for modern technology platforms. Whether you are using a cloud-based storage service, a decentralized ledger, or a global search platform, you are interacting with distributed systems.
This course introduces the concepts, architectures, and algorithms that enable multiple independent computing nodes to work together as a single, cohesive, and resilient system. Over this semester, we will examine remote procedure calls, logical clocks, consensus protocols (specifically Raft), replication, partition tolerance, and cloud-native architectures. This packet contains critical logistics, tips for academic success, and answers to common student questions. Please review it thoroughly.

1 Course Overview & Objectives

A distributed system consists of multiple autonomous computers that communicate over a network and coordinate their actions by passing messages [Tanenbaum and Van Steen, 2017]. Designing such systems is challenging because developers must account for independent node failures, messages arriving out of order or being lost entirely, and the absence of a global shared clock.

This course explores both the theoretical foundations and practical realities of distributed computing. We will study the logical ordering of events in distributed environments [Lamport, 1978], explore fault tolerance techniques, and implement concrete consensus models like the Raft protocol [Ongaro and Ousterhout, 2014].

1.1 Key Learning Outcomes

Upon successful completion of CS-350, students will be able to:

- **Build Distributed Services:** Develop remote procedure call (RPC) communication frameworks and asynchronous socket networks.
- **Formulate Fault-Tolerant Designs:** Apply state-machine replication models that recover gracefully from machine crashes and network partitions.
- **Synthesize Consensus Mechanics:** Evaluate the trade-offs between Paxos and Raft, and build a replicated consensus log from scratch.
- **Navigate Trade-offs (CAP Theorem):** Explain the limits of consistency, availability, and partition tolerance in globally distributed databases.

- **Containerize & Deploy:** Pack distributed services into containers (Docker) and execute multi-node orchestration scenarios.

1.2 Course Resources & Software Environment

To ensure hands-on experience, the course focuses heavily on system implementation. The primary tools and references you will use include:

- **Programming Language:** We use the Go programming language (Golang) for all programming labs due to its first-class support for concurrency (goroutines and channels).
- **Development Platforms:** Git for version control and Docker/Docker-Compose for simulated cluster environments on your local machines.
- **Textbook:** Recommended readings are taken from *Distributed Systems* by Tanenbaum & Van Steen [Tanenbaum and Van Steen, 2017], supplemented by contemporary systems research papers.

2 Course Policies & Grading

To ensure transparency and fairness, CS-350 operates under a defined set of academic expectations. Your final grade is calculated based on five major assessment components, as detailed in Table 1.

Table 1: CS-350 Grading Components and Weights

Assessment Component	Weight	Description and Execution
Programming Labs (4 total)	35%	Hands-on coding assignments implementing RPC protocol engines, logical clocks, replication, and Raft consensus.
Midterm Examination	20%	Concepts-based in-class assessment covering distributed time, snapshotting, and synchronization.
Capstone Group Project	25%	Group project (3–4 students) designing, deploying, and testing a custom cloud database on Vertex Cloud.
Final Examination	15%	Comprehensive evaluation covering theoretical paper readings and advanced protocols.
Active Engagement	5%	Contributions to in-class discussions, feedback forms, and active peer assistance on course forums.

2.1 Late Submission Framework

Distributed systems development requires careful time management. To encourage progress, the late submission framework is defined as follows:

- **Grace Days Pool:** Each student has three (3) grace days to use across the semester for individual programming labs. A grace day extends a deadline by 24 hours.
- **Standard Deductions:** Once your grace days pool is depleted, late submissions lose 15% of the maximum points per 24-hour period. Submissions are capped at 72 hours late, after which a grade of 0 is recorded.
- **Exclusions:** Grace days cannot be applied to exams, quiz windows, or group project milestones.

2.2 Academic Integrity and AI Guidelines

Collaborative discussion of high-level concepts is highly encouraged [Felder and Brent, 2001], but the code you write must represent your own individual effort.

⚠ **Fictional Generative AI Policy:** Large Language Models (LLMs) and code assistants (e.g., Synapse Copilot, Vertex Assistant) are valuable for syntax lookup, compiler error debugging, and clarifying abstract algorithm specifications.
However, using AI to generate code templates, draft structural components, or solve core coding challenges is strictly prohibited. Submissions are screened with automated structural similarities checkers. Violations are automatically forwarded to the Vertex Academic Standards Committee.

3 Frequently Asked Questions (FAQ)

Below are answers to the most common questions students raise during the first few weeks of the semester.

Q: I have never written code in Go. Will I be at a disadvantage?
A: No. We assume strong programming fundamentals in structured languages (like Java, C++, or Rust) but do not require prior experience with Go. We host a "Lab 0" tutorial during the first week to familiarize you with Go's unique syntax, pointer arithmetic, interfaces, and concurrency model. Most students find Go intuitive to learn.

Q: How are the programming labs graded?
A: Grading is automated via the **Vertex Sandbox** autogradiator. The autograder runs your code against intensive test suites that simulate real-world conditions, including network delays, package loss, and random machine failures. You may submit your code multiple times before the deadline; your highest score will be kept.

Q: My code passes all tests on my local computer, but fails the autograder. Why?
A: Local development environments are typically fast and lack network jitter, which often conceals concurrency bugs. The autograder runs tests under constrained execution threads, which frequently exposes race conditions, deadlock scenarios, and resource leaks. Always compile and test your code locally using the race detector command: `go test -race ./...`.

Q: What if I cannot find a group for the Capstone Project?
A: We will run a matching session during Week 4 to help form project teams. We also host a dedicated thread on our student forum. We will ensure every student is placed in a functional team by the end of Week 5.

Q: Can I get extra credit if I struggle with a lab?
A: Yes, through structured extension tasks. Labs 2, 3, and 4 include optional "challenge exercises" (such as building dynamic group membership changes or log optimization). These challenges can boost your grade by up to 5% on that specific lab. We do not offer manual extra credit assignments at the end of the term.

4 Course Schedule & Milestones

CS-350 is structured as a fast-paced journey through theory and implementation. The weekly topics, associated readings, and primary course milestones are detailed in Table 2.

Table 2: Weekly Course Syllabus and Lab Milestones

Weeks	Core Subject Area	Key Readings	Deliverable
Week 1–2	Distributed Time & Clocks	Lamport Clocks [Lamport, 1978]	Lab 0 (Ungraded Go Lab)
Week 3–4	RPC & Socket Architectures	Tanenbaum Ch. 4 [Tanenbaum and Van Steen, 2017]	Lab 1 (Multi-threaded RPC)
Week 5–7	Log Replication & Consensus	Raft Protocol [Ongaro and Ousterhout, 2014]	Lab 2 (Raft Leader Election)
Week 8	Midpoint conceptual Sync	Practice exercises	In-class Midterm Exam
Week 9–11	Replicated Key-Value Storage	Tanenbaum Ch. 7	Lab 3 (Fault-tolerant DB)
Week 12–14	Scaling & Partitioning	CAP Theorem publications	Capstone Project Beta
Week 15	System Showcases & Demos	Peer team surveys	Capstone Final Presentation

4.1 Milestone Progression Map

Figure 1 depicts how the lab assignments build upon one another, progressing from foundational RPC concepts to a fully scale-out consensus engine.

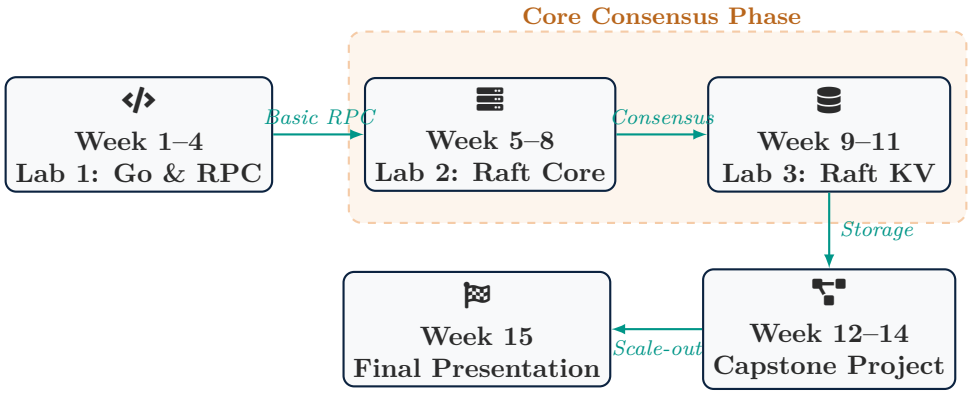

Figure 1: Progression of programming milestones and project development in CS-350.

References

Richard M. Felder and Rebecca Brent. Effective strategies for cooperative learning. *Journal of Cooperation & Collaboration in College Teaching*, 10(2):69–75, 2001.

Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565, 1978.

Diego Ongaro and John Ousterhout. In search of an understandable consensus algorithm. In *2014 USENIX Annual Technical Conference (USENIX ATC 14)*, pages 305–320, 2014.

Andrew S. Tanenbaum and Maarten Van Steen. *Distributed Systems: Principles and Paradigms*. Pearson, 3rd edition, 2017.