# CS592 Advanced Distributed Systems and Reliability

## Course Description

Cloud services, such as Amazon AWS and Google Cloud, now have become the backbone of the internet. These cloud systems are composed of large distributed systems that run on thousands of machines. Therefore, how to design scalable, efficient, and reliable distributed systems becomes increasingly important. This class aims to bring students to the state of the art in research and practice on distributed systems as well as to identify a set of open research problems.

The first part of this course focuses on the design of distributed systems. Topics include advanced distributed consensus, practical byzantine fault tolerance, distributed storage & databases, peer-to-peer systems & blockchains.

The second part of this class focuses on reliability. Large distributed systems tend to be failure-prone and failures on them have severe consequences. For instance, a failure that brought down Amazon on its 2018 PrimeDay cost Amazon 100 million dollars in just one hour. Unfortunately, these problems have no simple solutions. Distributed systems need to be designed to detect, isolate, and recover from these problems. Topics in this part include bug detection, testing, debugging & diagnosis, fault isolation, failure recovery, formal verification.

This syllabus is subject to change and changes will be announced appropriately.

## Time & Location

TBD

## Prerequisites

Basic knowledge of software systems (CS354/CS307) and basic programming skills (CS381/CS252). Students without this background should have a discussion with the instructor prior to registering the course. Knowledge about distributed systems (CS505) is preferred but not required.

## Administrative Information

Instructor: Yongle Zhang; yonglezh@purdue.edu

Teaching Assistant: TBD

Course website: Notifications and slides will be via Brightspace (https://purdue.brightspace.com/).

## Textbooks

There is no textbook for this course due to the emphasis on current techniques. Lecture notes and pointers to papers and other references on the web will be provided. Some helpful background references for classical topics are:

- Guide to Reliable Distributed Systems by Kenneth P. Birman
- Distributed Systems: Principles and Paradigms by Andrew S. Tanenbaum and Maarten Van Steen

## Grading

There are no midterm and final exams.

- Class participation & in-class discussion: 8%
- Paper reading and review: 12%
- Paper presentation: 30%
- Final project: 50%
    - Project proposal: 5%
    - Midterm report: 10%
    - Final presentation & demo: 15%
    - Final report: 20%

**Paper Presentation**: Each class, one student prepares to present a paper and lead class discussion. Depending on the number of students enrolled in the course, each student will present one, two, or three times over the course of the semester.

**Paper Review**: Two times over the course of the semester each student must write a two-page summary of the previous week's paper, and this paper must not be a paper the student presented to the class.

**Project**: Students are encouraged to work as teams of two or three.

**Final report** should be submitted using Latex using ACM sigplan conference template.

## Project Topics

We will provide a few project directions for your reference. You are encouraged to choose a specific project within the overarching theme of one research direction in the list, although you are free to choose any project at your will as long as it relates to distributed systems or reliability for scientific discovery and design. The goal is to nurture projects that have potentials to be developed into innovative research papers in the future.

## Late Policy

- All assignments due at 11:59:00 PM
- 1 day late (1 second == 1 day): 25% off

## Paper List

The paper list is preliminary and subject to change.

Preparation

- Efficient Readings of Papers in Science and Technology
- How (and How Not) to Write a Good Systems Paper

Distributed System Basics

- Time, Clocks, and the Ordering of Events in a Distributed System
- Leases: An Efficient Fault-Tolerant Mechanism for Distributed File Cache Consistency
- Everything You Always Wanted to Know About Synchronization but Were Afraid to Ask

Recent Cloud Systems

- MapReduce: Simplified Data Processing on Large Clusters
- Bigtable: A Distributed Storage System for Structured Data
- Dynamo: Amazon's Highly Available Key-value Store

More Consensus

- There Is More Consensus in Egalitarian Parliaments
- Mencius: Building Efficient Replicated State Machines for WANs
- Just say NO to Paxos Overhead: Replacing Consensus with Network Ordering
- Designing Distributed Systems Using Approximate Synchrony in Data Center Networks

More Consistency

- Seeing is Believing: A Client-centric Formulation of Database Isolation
- Replicated Data Consistency Explained Through Baseball
- Customizable and Extensible Deployment for Mobile/Cloud Applications

Bazyntine Fault Tolerance

- Practical Byzantine Fault Tolerance
- Making Byzantine Fault Tolerant Systems Tolerate Byzantine Faults
- Farsite: Federated, Available, and Reliable Storage for an Incompletely Trusted Environment
- Secure Untrusted Data Repository (SUNDR)

Distributed Logging

- Virtual Consensus in Delos
- CORFU: A Shared Log Design for Flash Clusters
- Tango: Distributed Data Structures over a Shared Log
- TrInc: Small Trusted Hardware for Large Distributed Systems
- PeerReview: Practical Accountability for Distributed Systems

Peer-to-peer Systems

- Chord: A Scalable Peer-to-peer Lookup Protocol for Internet Applications
- Epidemic Algorithms for Replicated Database Maintenance
- Using Lightweight Modeling To Understand Chord
- How robust are gossip-based communication protocols?

Bug Detection

- Bugs as Deviant Behavior: A General Approach to Inferring Errors in Systems Code
- KLEE: Unassisted and Automatic Generation of High-Coverage Tests for Complex Systems Programs
- How to Build Static Checking Systems Using Orders of Magnitude Less Code

Debugging

- Lazy Diagnosis of In-Production Concurrency Bugs
- Finding and Reproducing Heisenbugs in Concurrent Programs
- Automated Concurrency-Bug Fixing
- Towards Practical Default-On Multi-Core Record/Replay
- ODR: Output-Deterministic Replay for Multicore Debugging

Failure Recovery

- Exploring Failure Transparency and the Limits of Generic Recovery.
- Rx: Treating Bugs As Allergies---A Safe Method to Survive Software Failures.
- ASSURE: Automatic Software Self-healing Using REscue points

Formal Verification

- IronFleet: Proving Practical Distributed Systems Correct
- Verdi: A Framework for Implementing and Formally Verifying Distributed Systems
- SibylFS: formal specification and oracle-based testing for POSIX and real-world file systems
- Push-Button Verification of File Systems via Crash Refinement
- Using Crash Hoare Logic for Certifying the FSCQ File System
- SAMC: Semantic-Aware Model Checking for Fast Discovery of Deep Bugs in Cloud Systems