Distributed System Final Project Instructions

For your final project, you will **design, implement, and deploy** a real-world **distributed system** or experiment with **distributed algorithms** in a suitable simulation. Your project should demonstrate key distributed systems scalability, fault tolerance, or consistency.

The project should have a working prototype by early Nov, with improvements in late Nov, leading to a mature system by the end of the course. Choose a feasible, ambitious idea that aligns with the course timeline. You may select from provided examples or propose your own.

Projects must be completed in **teams of 2-3 students**; **solo projects are not allowed**.

Proposal – Literature Review

You can use google scholar or IEEE to find "recent" research papers related to distributed systems. In general, you should use research articles such as those from conferences or journals published by ACM or IEEE, since they have more depth and are peer-reviewed.

NOTE: You must evaluate information from at least 4 articles and papers for 2 person teams

Each student should deeply understand the material presented in the papers s/he has undertaken. Most specifically, each student should:

- Know the algorithms and the techniques presented in the paper;
- Be able to answer to questions of the style «Why is each line of the code useful in the algorithms s/he will present and what could go wrong if any line was removed»;
- Evaluate the algorithm presented and devise own good/bad scenarios of execution and understand how the algorithms cope with these scenarios
- Study/devise a big number of examples to deeply understand how the algorithms work;
- Understand the high-level idea of the analysis of the algorithms included in the paper;

Each team should submit a project proposal of 2-3 pages providing more details on the project you are going to implement. your proposal should explicitly state the following:

- A description of the problem
- The motivation for the problem (e.g., why is the problem interesting, why is it challenging, who will benefit from a solution to the problem, etc.)
- A brief discussion of previous work related to this problem.
- Your initial ideas on how to attack the problem. This includes a (rough) methodology and plan for your project including high level architecture, implementation, algorithms and deployment
- Be sure to structure your plan into a set of incremental, implementable milestones and include a milestone schedule
- The resources needed to carry out your project, software, hardware, cloud, etc
- Citation and link to resources used or will be used.

ONE submission per team.

Project Timeline

Component	Due Date
Partner Selection	9/25/2025 @11:59pm
Proposal	10/09/2025 @11:59pm
Presentations	12/04/2025 @11:59pm
Final Report	12/05/2025 @11:59pm
Code Submission	12/04/2025 @11:59pm

Project Ideas

You can choose any of the ideas presented below or you should suggest your own project based on the research papers selected.

1. Distributed Social Network
Implement a **distributed social network application** where each node represents an individual or organization, connected through various interdependencies like friendship, common interests, or beliefs. Nodes can join and leave the network at any time, with the ability to specify or modify their parameters, such as personal interests, beliefs, or relationships. The system should be capable of correlating nodes with similar parameters and dynamically forming "groups of relevance" to facilitate proper networking. These groups will ensure that users can easily connect with others who share similar traits or goals, promoting interaction within the network.

However, exiting the network at any time introduces significant challenges, especially when nodes have active connections or belong to various groups. These abrupt exits can affect the network's stability and disrupt ongoing interactions or group formations. The project must identify these challenges and propose solutions that ensure a smooth transition when nodes join or leave.

The selection of services and algorithms will be crucial in managing these situations efficiently, ensuring consistency, scalability, and minimal disruption in the network. By addressing these challenges, the system will provide a seamless user experience even with fluctuating participation.

2. Library of Collaborating Tools
Implementation of a library of collaborating tools, such as:

- Distributed Calendar (like the google calendar but probably with enhanced functionality)
- Distributed Meeting Arrangement (like doodle but with enhanced functionality)
- Distributed Meeting and Sharing (like facebook) where nodes can chat, form chat-rooms or groups, and send messages to multiple recipients. Ensure that messages are displayed in the same order for all participants, both in one-on-one chats and within groups.
- Distributed Editor (like recent version of emacs). Users can create and edit documents, which can be concurrently edited by several other users.

  Choose **some** of the above items so that the result should be a comprehensive project.

3. Middleware
Implementation of middleware that provides **some** of the following functionality:

- Naming service. Obviously, each node has to define a "name" when it enters the network. This service requires implementing a mechanism that permits to some node to find the information of any other node by searching its "name" and this must be implemented in a distributed way.
- Group creation, node joining and exiting, and communication. Maintaining the groups of relevance in a distributed way, when nodes can join and exit the network arbitrarily, is challenging.
- Security service: The security service may include the following: (1) authentication of principals, (2) access control on the reception of remote method invocations, (3) security of communication between clients and servers, (4) facilities for non-repudiation, etc.
- Trading service: Allows the location of objects by their attributes, i.e., it is kind of a directory service. Its database contains a mapping from service types and their associated attributes onto remote memory references of objects. Clients make queries specifying the type of service required, together with other arguments specifying constraints on the values of attributes, and preferences for the order in which to receive matching offers.
- Transaction and concurrency control services.
- Persistent state service.
- Any other functionality you would like to suggest.

4. Publish-Subscribe System
Develop a **Publish-Subscribe (Pub-Sub)** system for a distributed network where nodes can **publish** information and **subscribe** to topics of interest. Publishers broadcast updates, while subscribers receive information based on their selected topics. The system should include **efficient information diffusion** using techniques like **gossip protocols** and **mobile agents**, ensuring that updates reach relevant subscribers with minimal redundancy and network congestion.

Key considerations include **fault tolerance** to handle node failures and network partitions, **scalability** to support growing networks, and **performance evaluation** to assess metrics like latency and message reliability. Additionally, the system must support **dynamic subscription management**, **security** features like authentication and encryption, and ensure that information is consistently delivered to subscribers while optimizing resource usage.

5. Distributed Hash Tables
Implement a **Peer-to-Peer (P2P)** system using a **Distributed Hash Table (DHT)** for efficient data storage and retrieval. The system can either use a **structured DHT** with a deterministic topology (e.g., Chord, Kademlia) for efficient lookups or an **unstructured DHT** where nodes use heuristics for data discovery. Key features include **data storage and retrieval** through hashing keys, **node join/leave** functionality, **routing mechanisms** for efficient queries, and **replication and fault tolerance** to prevent data loss.

The system must address challenges such as **scalability**, ensuring performance with increasing nodes and requests, and handling **network partitioning** to maintain operations during failures. Additionally, consistency models like **Eventual Consistency** should be considered, balancing **availability** and **partition tolerance** (CAP Theorem). Security measures like **encryption** and **authentication** are also essential to ensure privacy and protect the system from malicious nodes.

6. Distributed Data Structures
Develop a library of **distributed data structures** designed for **distributed memory machines**, ensuring efficiency in parallel and distributed operations while supporting **consistency**, **fault tolerance**, and **scalability**. Key structures to implement include a **distributed hash map** for key-value pairs, a **distributed list/queue** for managing elements across nodes, a **distributed set** supporting operations like union and intersection, a **distributed graph** for parallel node and edge traversal, and **distributed trees** (e.g., B-trees) for efficient insertions, deletions, and searches across nodes.

The system must address challenges such as **consistency**, ensuring data integrity across nodes despite failures, and **fault tolerance**, with mechanisms for replication and recovery. It should also ensure **scalability** as the system grows and support **concurrency** for concurrent read and write operations across nodes without conflicts or inconsistencies.

7. Consistency and Replication
Develop a **quorum-based consistency system** to ensure strong consistency in distributed environments, where a read or write operation succeeds only when a quorum of nodes agrees. This system guarantees consistency by forming read and write quorums and using majority voting techniques to prevent conflicts and divergent replicas, ensuring a consistent state across all nodes.

In addition, implement a **Revision Control System** to manage concurrent updates in distributed file systems. Each change is tracked with a unique identifier (revision code), timestamp, and author, and the system includes features for comparing, merging, and resolving conflicts between revisions. Key features include **concurrency control**, **versioning and history management**, and **fault tolerance** to ensure data integrity, consistency, and availability even in the event of node failures.

8. Distributed Memory Allocator or Garbage Collection
Develop a **distributed memory allocator** to efficiently manage memory across multiple nodes, optimizing usage and minimizing latency. Implement a **distributed garbage collection** system that reclaims memory from unused objects, using techniques like leases to track object lifetimes and determine when they can be safely freed.

Key concepts include **leases** for tracking object references, **reference counting** to ensure proper memory management, and **concurrency** to allow smooth memory allocation and garbage collection without conflicts. **Fault tolerance** mechanisms are also essential to handle node failures during memory operations, ensuring consistent memory management across the system.

9. Distributed Database from Scratch
Develop a **basic distributed database** that addresses consistency, scalability, concurrency, and fault tolerance. Use a lightweight database like SQLite, focusing on distributed aspects such as quorum-based reads/writes, leader-follower replication, and conflict resolution for data consistency across nodes.

Address **scalability** by employing techniques like sharding or partitioning for horizontal scaling. Implement **concurrency control** using Multi-Version Concurrency Control (MVCC) to handle efficient transactions and concurrent access without blocking. Ensure **fault tolerance** through strategies like data replication, leader election, and failure recovery, making the system reliable even in the event of node failures.

Implementation

You can choose to implement any of the projects described above and you are encouraged to propose and implement any other service you want (which is related to the topics of this course, i.e. it incorporates any class of distributed algorithms discussed during the lectures).

The implementation of your project must consider issues such as fault tolerance, performance, scalability, availability, consistency, concurrency and other challenges discussed in class.

Your choice should be so that **at least** three of the following list of algorithms should be implemented in the project:

1. Broadcast and/or multicast through spanning tree construction
2. Mobile Agents
3. Gossiping Protocols
4. Mutual Exclusion
5. Leader Election
6. Distributed Snapshots
7. Timestamps
8. Resource allocation algorithms
9. Concurrency control
10. Resource discovery
11. Agreement protocols
12. Replication and consistency protocols
13. Distributed shared memory protocols

Final Reports

Your team's report should be minimum of 10-15 pages (excluding images and diagrams) using the IEEE double column template, and include at least these sections:

1. Team member names, student IDs and email addresses on cover page
2. Project Goals and motivation
3. Description of Distributed System challenges addressed
4. Detailed discussion of previous work related to this problem (Literature review)
5. Project Design
    a. Describe key design goals in designing your distributed systems i.e how your design allows the system to handle the core distributed systems challenges: Heterogeneity, Openness, Security, Failure Handling, Concurrency, Quality of Service, Scalability, and Transparency.
    b. Describe the key components and algorithms
    c. Describe your architecture and how the components will interact. You must include several UML diagrams to illustrate your design (at least 3 diagrams from below list to illustrate proper overview of the system.
        i. Use Case style diagram to show the overview of your system functionality and different functions in your system.
        ii. Use the Deployment and Component diagrams to show the software and hardware components of your system.
        iii. Communication diagram to illustrate the connections between the components,

iv. Sequence diagram explains the sequence of actions, events, and processes between the components, objects, and actors in your system.
v. Activity and State diagrams to explain more detail about a process or object in your system.

6. Evaluation
    a. Evaluate the selected approach and analyze why the selected approach is good? Provide an intuitive description of the algorithms, their correctness and their complexity
    b. How are the following issues addressed in your project
        i. Fault Tolerant
        ii. Performance
        iii. Scalability
        iv. Consistency
        v. Concurrency
        vi. Security
7. Implementation Details
    a. Describe the choice of Architectural Style
    b. You must include several UML diagrams to illustrate your implementation such Class Diagrams, Object diagrams, Interaction diagrams, etc (At least 3 diagrams)
    c. Describe Software components and services implemented and how they interact (i.e RPC/RMI, Web Services, Socket APIs, etc)
8. Demonstration of example system run
    a. Define and explain a successful scenario and include snapshots
    b. Define and explain 3 failure scenarios and include snapshots for each and how your system tackle such failure scenarios
9. Analysis of expected performance (growth rate of major algorithms, etc).
    a. Describe the overall result of your project and reflect on your design decisions.
    b. Define measurements metrics to analyze and understand how it performs or works (ideally in comparison to another system).
    c. Simulate workloads and measure the performance of your system against the metrics you developed above (Example in terms of throughput and latency)
10. Testing results
    a. Develop and explain set of test cases (at least 5)
    b. Report the result of test cases and discuss how well your systems works.
11. Conclusions, lessons learned possible improvements, etc.

Final Project Presentation

A recorded presentation is required for the project. The presentation will approximately be 15 minutes for each team. Your presentation slides should be understandable to anyone who has taken our distributed systems course.

An ideal presentation will:

- Introduce your project and the problem that it solves.
- Briefly present related work/systems.
- Describe the key design choices of your system, with a discussion of how they relate to some of our distributed systems design challenges.
- Overview of components, architecture and features of the system
- Live demo of your running project. Be sure to demo and focus on core distributed systems concepts such as reliability, scalability , fault tolerance, concurrency, etc
  For example:

    1. Show what happens when you cause a crash.
    2. How your system scales as you increase concurrent access to the system
    3. How is performance affected as you increase the number of users
- Discuss the overall impact of your design and present some evaluation of how well the system works (use metrics).
- Describe what you plan on doing next
- Conclude your presentation with a summary of your project's key points.

What to submit

These deliverables are due at or before the last class of the quarter

1. Final report.
2. Recorded presentation/demo,
3. Presentation slides used to illustrate your presentation
4. Code submission
5. Readme file containing
    1. How to setup/compile your project and step by step instructions on setting up your project
    2. Division of work: you are responsible for making it clear which portions each student was responsible for

ONE submission per team.