COEN 6731 Winter 2026 Course Project

1 Project Overview

Each student or group (max two students) may choose to work on a designated topic from the provided suggestions. The course project requires the submission of the following:

- A proposal report (1 page) and proposal presentation slides (5 slides max).
- Midterm progress presentation slides (5 slides max).
- A final project report (6 pages max) and final presentation slides (8 slides max).

All reports must adhere to the USENIX paper format using the LaTeX template. Deadlines are stated on the course website.

The primary objective of the course project is to enable students to demonstrate their ability to research, design, and implement a well-scoped problem in the area of distributed systems. Emphasis is placed on the application of a **sound research methodology** throughout the entire project lifecycle, including problem formulation, system design, implementation, evaluation, and analysis.

Throughout the course, students will be exposed to research methodologies by reading and critically analyzing selected research papers, complemented by in-class discussions. These skills are expected to be reflected in the project work and final report.

For groups of two students, the final report must include a **contribution table** clearly documenting the responsibilities and contributions of each group member. In cases where significant imbalances in contribution are identified, individual grades may be adjusted accordingly.

1.1 Suggested Topics

1. **Building a Distributed Key-Value Store with Cabinet**

   In this project, you will build a distributed key-value store using **Cabinet**, a weighted consensus algorithm developed for fast convergence under heterogeneous environments. Unlike Raft, Cabinet assigns different voting weights to servers, allowing high-capacity nodes to contribute more strongly to consensus decisions.

   The implementation of the Cabinet consensus protocol is provided at: `https://github.com/gengruizhang/cabinet`

   Your task is to integrate Cabinet into a fully functional distributed key-value store that supports basic operations such as `PUT`, `GET`, and `DELETE`. The system should replicate data across multiple nodes and ensure consistency using Cabinet.

   Distributed systems concepts involved:

   - Consensus and replication
   - Leader-based coordination

- Fault tolerance under node failures
- Performance trade-offs with weighted voting

Programming language: Go (recommended), or re-implement Cabinet in C/C++/Java/Rust.

2. **Using Apache Spark for Distributed Computing**

   This project focuses on large-scale data processing using **Apache Spark**. You will design and implement a distributed data processing pipeline that runs on multiple worker nodes and performs non-trivial computations (e.g., joins, shuffles, iterative processing, and window analytics). Your pipeline must be reproducible and must report runtime/scale results across different cluster sizes or data sizes.

   Example Project A: E-commerce Analytics with Sessions, Funnels, and Attribution

   You find or generate the following datasets:

   - `events(user_id, timestamp, event_type, product_id, session_id?, device, referrer)`
   - `orders(order_id, user_id, timestamp, total_amount)`
   - `catalog(product_id, category, brand, price)`

   Implement a Spark pipeline that performs:

   (a) **Sessionization:** Build sessions per user using an inactivity threshold (e.g., 30 minutes). If `session_id` is missing, derive it using time gaps. (Requires window functions and stateful grouping.)

   (b) **Funnel conversion analysis:** Compute conversion rates for funnels such as

   $$\texttt{view} \rightarrow \texttt{add\_to\_cart} \rightarrow \texttt{purchase}$$

   at the granularity of `(category, device, referrer)` and by day/week.

   (c) **Attribution:** Attribute each order to the most recent non-direct referrer within the last k hours (e.g., last-touch attribution). (Requires joins between session events and orders with time constraints.)

   (d) **Anomaly detection (system-level):** Detect sudden drops/spikes in conversions using rolling baselines (e.g., comparing to trailing 7-day moving averages). Output top anomalies with explanations.

   Example Project B: Graph Analytics on Large Networks (PageRank + Community Signals)

   You create a directed graph dataset such as:

   - `edges(src, dst, timestamp?)` (optionally temporal)
   - `node_meta(node_id, type, region, attributes...)` (optional)

   Implement a Spark-based graph analytics workflow that includes:

   (a) **Iterative PageRank (distributed iteration):** Implement PageRank for T iterations using Spark DataFrames/RDDs (or GraphFrames if you choose, but you must still explain the execution and performance). Report convergence trends and runtime vs. T and graph size.

(b) **Community signal / triangle analysis:** Compute triangle counts or clustering coefficients for high-degree nodes (or approximate versions if exact is too expensive). Discuss trade-offs between exact and approximate approaches.

(c) **Temporal extension (optional but encouraged):** If timestamps exist, compute how PageRank changes over time windows (e.g., daily snapshots) and identify nodes with the largest rank volatility.

(d) **Skew handling and scaling study:** Demonstrate at least one real performance issue (e.g., high-degree node skew causing stragglers), then mitigate it (e.g., salting keys, custom partitioning, broadcast joins, or graph preprocessing). Provide before/after measurements.

Programming language: PySpark or Scala.

3. **Distributed Training System**

   In this project, you will design and implement a **distributed machine learning training system** in which multiple worker nodes collaboratively train a shared model. Training may be coordinated using either a **centralized parameter-server architecture** or a **decentralized (peer-to-peer) synchronization mechanism**.

   The machine learning model itself may be simple (e.g., linear or logistic regression); however, the **dataset must be sufficiently large** to require distributed computation. Students are expected to focus on **system-level challenges**, including efficient synchronization, consistency of model updates, communication overhead, and tolerance to worker or network failures.

   The system should be able to **synchronize model updates efficiently** and **handle failures gracefully** (e.g., straggler workers, dropped messages, or node crashes).

   The primary focus of this project is on **distributed system design and implementation**, not on achieving high model accuracy.

   Distributed systems concepts involved:

   - Distributed coordination and synchronization
   - Consistency vs. performance trade-offs
   - Fault tolerance and failure recovery during training

   Programming language: Python or any suitable programming language.

4. **Cloud Database System**

   This project involves building a simplified cloud database service composed of multiple backend nodes and a client-facing interface. The system should support concurrent clients and store data across at least three server processes.

   You should explicitly address:

   - Load balancing across database nodes
   - Replication and fault tolerance
   - Client request routing

The database does not need to support SQL but must demonstrate distributed storage and coordination.

Distributed systems concepts involved:

- Replication strategies
- Failure recovery
- Scalability and concurrency

Programming language: C/C++/Go/Java/Rust.

5. **Distributed Chat and File Transfer System**

   In this project, you will implement a distributed chat and file transfer application. The system should support multiple clients, non-blocking communication, and reliable message delivery.

   Requirements include:

   - Use of gRPC for communication
   - Non-blocking or asynchronous I/O
   - Support for multiple concurrent clients

   You may design either a centralized or partially decentralized architecture, but the system must involve multiple backend processes.

   Distributed systems concepts involved:

   - Client-server communication
   - Concurrency and scalability
   - Fault handling and reconnection

   Programming language: Frontend: Any. Backend: C/C++/Go/Java/Rust

2 Project Proposal and Presentation

You will need to submit a **one-page proposal** using the provided LaTeX template and your **presentation slides** (maximum of 5 slides). The proposal and slides should clearly articulate your project idea and intended approach.

2.1 Project Proposal

The proposal must adhere to the following requirements and must not exceed **ONE page**. Use the provided one-pager LaTeX template for submission. The proposal should include:

- **Problem Statement**: Clearly and concisely identify the problem you aim to address. If in doubt, start with: *"The problem is..."* (Just a few sentences)
- **Relevance**: Explain why the problem is important. Highlight its potential impact—what would solving this problem change or improve?
- **Interest**: Convince the audience that the problem is challenging and non-trivial. Demonstrate that it requires thought and effort to solve.

- **Related Approaches**: Briefly review what others have done to address this problem. Provide context for your project by referencing related work.
- **Approach Overview and Methodology**: Sketch your intended approach to solving the problem. Provide a high-level summary of your plan. In addition, briefly describe how you plan to tackle the problem. Specify whether you will use implementation and evaluation, modeling and simulation, theorem definition and proof, or another method.
- **Anticipated Difficulties**: Identify potential challenges or obstacles you expect to face during your project.

2.2 Project Proposal Presentation

You will be given **5 minutes** to present your proposal. Your slides should be structured as follows:

- Use **1–2 slides** to present the problem you are trying to solve, including the background and motivation.
- Use **1–2 slides** to describe your proposed approach and methodology. Provide a concise overview of your plan. Include at least **one figure** illustrating your proposed system architecture. The figure should help explain aspects such as:
  - What does your proposed system look like (what are the major components/modules)?
  - What are the inputs and outputs of your system?
  - What is the overall workflow?
  - How do the different components interact with each other?
- Use **1 slide** to outline the anticipated difficulties. Focus on difficulties that are **objective**. For example, *"We may need more machines to simulate distributed scenarios"* or *"We may need to identify appropriate workloads for evaluation"* are valid. Avoid subjective challenges such as *"I am not good at programming"*.

3 Progress Presentation

For the progress presentation, you will only need to **present your progress** and submit your **slides**. A written report is **NOT required**. You will have **5 minutes** to present your progress. Your presentation should address the following points:

- **Changes Since the Proposal**: Highlight any modifications or updates to your original plan since the proposal. Discuss why these changes were necessary.
- **Current Progress**: Provide an overview of the progress you have made so far. Clearly state the current status of your project.
- **Achievements**: Summarize the key accomplishments you have achieved to date.

4 Final Project Presentation and Report

You are required to present your final project in the last lecture and submit your final project report by the deadline.

4.1 Final presentation

During the final lecture, all students will present their project work. Each presentation will be allotted **10 minutes**. Your presentation should cover the following components:

1. **Problem Statement:** Use a **single slide** to succinctly present the problem. Avoid covering background or motivation, as these were already addressed in your proposal and progress presentations.

2. **Proposed Solutions:** Allocate **2-4 slides** to provide a detailed explanation of your solution. This should include how the problem is addressed, a presentation of your system architecture, an analysis of its strengths and limitations, and any other pertinent details. This part should include at least the two following diagrams:

   - An overview of the system architecture, highlighting major components, workflow, and interactions among components.
   - A summary of the implementation, including how the system was built and the major modules in your codebase.

3. **Evaluation Results:** Use **2-3 slides** to showcase quantifiable evaluation results. Highlight how your system or algorithm performs under relevant workloads and representative scenarios.

4.2 Final report

The final project should not exceed **six pages**, excluding references. If additional space is needed for supplementary material, you may include an appendix. However, ensure the appendix is well-organized and separate from the main content.

The report should be of "publishable quality," meaning it must be free of typos, grammatical errors, and other issues. However, this does not imply that the report needs to be ready for publication in a major venue. Achieving a perfect experiment result is encouraged but not required.

Your final report should include the following sections:

1. **Problem Statement:** Clearly articulate the motivation for and background of the project.

2. **Solutions:** Provide a detailed description of the solution, including high-quality figures to clearly illustrate how the problem is addressed. The following figures are mandatory:

   - **System architecture figure:** A detailed diagram showing all major components, their interactions, and end-to-end data flow. The figure must clearly indicate inputs, outputs, and how data moves through the system.
   - **Implementation structure figure:** A code-level diagram showing the structure of your implementation, including modules/classes, key functions, and their calling relationships (e.g., control flow or interaction between components).

   The quality of the figures matters. All figures must be original and created by the students. **The use of AI-generated figures is strictly prohibited.** Figures should be carefully designed to accurately reflect your system and implementation. Low-effort or auto-generated diagrams that do not precisely match your work may result in a penalty.

3. **Results:** Present any quantifiable results, comparisons, and relevant findings. This section must include:

- **Evaluation setup:** Clearly describe the experimental setup, including the testbed (e.g., machines, cluster, environment) and workloads/benchmarks used.
- **Quantitative results and comparisons:** Report measurable results (e.g., latency, throughput, accuracy, resource usage) and provide comparisons under different settings (e.g., varying input sizes, number of nodes, configurations, or baselines).
- **Analysis and explanation:** Go beyond reporting numbers: explain why the results occur. Provide insights into system behavior, trade-offs, and any observed trends.

4. **Lessons learned:** Report lessons learned that are **specific to your project** (minimum 2, maximum 4). These lessons must be derived from your own design, implementation, and evaluation experience. General or textbook-level observations (e.g., "distributed systems are hard") will not receive credit. Each lesson should be concrete, technically grounded, and supported by evidence from your project (e.g., design decisions, debugging experience, or experimental results).

5. **Availability:** Include a link to your GitHub repository where the project code is accessible. The code must be runnable and reproducible, with:

   - A clear README describing setup and usage instructions
   - All necessary dependencies specified. Either:
     - A Docker container for environment setup (recommended), or
     - An automated installation script to configure dependencies