COMP 755 Advanced Operating Systems
Fall 2025
Philip S. Bell

Mid-Semester Project Progress Report

What to include:

- Group information
  - Solo: Philip S. Bell
- Project title and topic:

Decentralized Load Balancer Simulation:

Implement a distributed, routerless peer simulation in C++ where each node (thread or process) exchanges load information with peers and dynamically routes work requests to less-busy neighbors — modeling the decentralized coordination logic of my future Mixture of Experts (MoE) system for my PhD research project.

Timeline:

Week 1: Project Setup & Design Specification

Goals

- Finalize project scope and architecture.
- Prepare build environment and development workflow.
- Write pseudocode and class design.

Milestones

1. **Literature & Design Research**
   - Review related work on distributed load balancing:
     - Chord / Consistent Hashing
     - Gossip protocols (e.g., SWIM, Serf)
     - Linux kernel scheduling as analogy
   - Define how simulation models “load” (e.g., integer queue length).
2. **System Design Document**
   - Diagram the architecture:
     - PeerNode class (thread or process)
     - Message class for exchanging load and task info
     - NetworkSimulator or socket layer abstraction
   - Define protocols (message types, update frequency).
3. **Environment Setup**
   - Initialize C++ project with CLion.
   - Set up GitHub repo and build scripts.

- Verify multithreading, networking, and logging libraries (e.g., <thread>, <mutex>, <asio>).

4. **Deliverable:**
   - design_spec.pdf including architecture diagram, pseudocode, and workflow.
   - Confirm project compiles (“Hello World” test).

Week 2: Core Implementation (Single-Node Simulation)

Goals

- Implement node logic and basic local routing.
- Simulate load change and task queueing.

Milestones

1. **Implement Node Core**
   - PeerNode manages:
     - A local load queue (tasks)
     - Thread-safe metrics for current load
     - Periodic load adjustment
   - Add worker threads to “process” tasks (sleep-based simulation).
2. **Add Basic Task Routing**
   - Local node receives tasks.
   - If load exceeds threshold, offload to random peer (placeholder routing).
3. **Logging and Metrics**
   - Log per-node load, task transfers, and timing.
   - Implement local performance stats collection.
4. **Deliverable:**
   - Single-process simulation with 3–5 nodes (threads).
   - Output showing dynamic task balancing locally.

Week 3: Distributed Communication Layer

Goals

- Implement inter-node communication.
- Replace placeholder routing with peer-to-peer message passing.

Milestones

1. **Networking Layer**
   - Implement communication via:
     - **Option A:** TCP sockets (real networking)
     - **Option B:** Message queues or shared memory (simulated networking)
   - Define message types: LOAD_UPDATE, TASK_REQUEST, TASK_TRANSFER.
2. **Peer Discovery**
   - Implement static peer list or basic gossip broadcast.
   - Peers periodically exchange load info.
3. **Decentralized Routing**
   - Each node uses peer load info to pick the least-loaded node.
   - Route new or excess tasks dynamically.

4. **Deliverable:**
    - Multi-process distributed load balancer demo.
    - Logs showing load distribution and routing stability.

Week 4: Testing, Analysis, and Report Preparation

Goals

- Evaluate performance, complete the short paper, and polish code.
- Connect findings to MoE research framework.

Milestones

1. **Performance Evaluation**
    - Run tests varying number of peers and load intensity.
    - Collect metrics: mean load variance, message rate, convergence time.
2. **Analysis**
    - Compare results to ideal load distribution.
    - Identify bottlenecks or instability causes.
3. **Documentation**
    - Write **5 page report**:
        1. Introduction & Goal
        2. Related Work (Distributed scheduling, gossip routing)
        3. Interface (Sockets, threads, system calls used)
        4. Software Implementation (class structure, flow diagrams)
        5. Conclusion (difficulties, future work)
    - Include architecture diagram and simulation graphs.
4. **Code Finalization**
    - Add README with compilation/run instructions.
    - Comment all public classes and functions.
5. **Deliverables:**
    - Final C++ source code in GitHub repo.
    - Completed 5 page project paper.
    - Performance plots (optional).

Stretch Goals

- Add a **Trust Score** to each node (randomly fluctuating or based on past reliability).
- Modify routing to avoid low-trust peers (basic trust-aware scheduling).
- Implement lightweight encryption or signature in messages (OpenSSL or libsodium).
- Visualize peer load over time using matplotlib or gnuplot.