COMP 3010 – Distributed Computing

Course Description

Calendar entry

An introduction to the development of client server and peer-to-peer systems through web applications, distributed programming models, and distributed algorithms. Prerequisite: [[COMP 2150 and COMP 2080] or [ECE 3740 and ECE 3790]] and [one of STAT 1150, STAT 1000, STAT 1001, STAT 2220, or PHYS 2496].

General Course Description

Applications don't run in isolation on a single computer. In this course we go from the isolated applications written in second year (and most of third year) and focus on how an application can interact with other applications to solve interesting problems. We'll discuss how to design applications such that they *can* interact reliably. We'll also discuss how to implement the common client-server applications we use every day.

Distributed computing doesn't get interesting until we have a group of peers working to solve a common problem. To do that we need to understand and be able to implement several really interesting/complex algorithms. Doing that well is a challenge. A challenge involving concurrency and failures — where a failure in code you didn't write *has* to be something that your code handles *cleanly*. It may be a challenge but solving those challenges is at the foundation of some interesting Computer Science.

Detailed Prerequisites

Before entering this course, a student should be able to:

- Implement and manipulate data structures such as lists, queues, and trees.
- Design, implement, and build complex applications requiring multiple modules with well-defined interfaces.
- Perform well-defined and structured tests on code.
- Use a variety of standard development tools such as those that automate builds or allow for the inspection of active program state.
- Use standard tools to remotely connect to a computer and then edit/run code.

Course Goals

By the end of this course students will:

- See how real-world distributed applications are designed and implemented.
- Build their own client-server and peer-to-peer applications.

- Experience messaging protocols such as HTTP.
- Design and implement a messaging protocol that must correctly interact with code written by other students.
- See how real-world peer-to-peer applications that share and store information are built using fundamental algorithms that ensure the scalability and reliability needed by applications that involve an unknown number of peers from around the world.

Learning Outcomes

Design of Distributed Applications

Students should be able to:

1. Explain how a distributed resource is located using a distributed naming service.
2. Identify potential failure modes in a distributed design and propose solutions to mitigate each failure mode.
3. Analyze the scalability of a distributed design and propose improvements that enhance the scalability of the design.
4. Propose and justify the design of a distributed application.

Web-based Computing

Students should be able to:

1. Write server-side code that processes HTTP messages *without* the use of an existing library or framework.
2. Implement server-side session management *without* the use of an existing library, framework, or database management system.
3. Write client-side code that provides sufficient testing of their own server-side code.

Distributed Programming

Students should be able to:

1. Write code that uses stream sockets to implement a client-server application.
2. Write code that uses datagram sockets to implement some peer-to-peer application.
3. Explain how the message passing model provides synchronization primitives (e.g., blocking versus non-blocking) needed by concurrently running applications that interact.
4. Identify synchronization issues (e.g., deadlock, live-lock, and race conditions) that arise in a piece of code that interacts with other instances of the same code.
5. Write error handling and recovery code that mitigates failure modes identified through timeouts and corrupt messages.

Distributed Algorithms

Students should be able to:

1. Show that a given algorithm correctly or incorrectly provides mutually exclusive access/update of a shared resource by a group of peers.
2. Show that a given algorithm correctly or incorrectly reaches consensus among a group of peers.
3. Show that a given algorithm correctly or incorrectly elects a new tracker/server from a group of peers.
4. Show that a given algorithm correctly or incorrectly manages a ring of peers that join and leave the ring non-deterministically.
5. Explain how a group of peers can manage a shared data store.
6. Discuss the happens-before relation and its use to define a partial ordering of events between a group of interacting peers.
7. Use the happens-before relation to identify whether a given algorithm will result in message passing race conditions, live-lock, and/or deadlock and propose a solution that prevents/avoids the situation.