CS 377 Group Project Proposal

Group Members: Shao Qin Tan and Simran Lekhwani

Part 1.1: Ideation

Response from Shao:

• What are the top three topics related to OS you find most interesting?

Distributed systems, concurrency, cpu scheduling

• If you have to pick one to explore in more depth, which one and why?

I would pick distributed systems. It seems like a hard but useful topic for getting a job in the industry

• Look online for some advanced algorithm(s) on the topic of your interest. They should be something you have not implemented in the projects. What algorithm did you pick, and why?

I want to implement an algorithm related to any heartbeat failure detection algorithm. It is related to distributed systems, and I think it is a very important part of it as detecting node failures are very important

Response from Simran:

I think the top 3 topics I find interesting are concurrency, file systems and distributed systems – mainly due to my curiosity to learn more in depth about the systems I use daily. While there are many components that define these systems (esp an operating system), we often overlook or underappreciated simple things like the file system and our ability to share resources in a multi-core system, which is why I am further intrigued by them to understand the functionality at the lower level.

I believe I would like to explore distributed systems in-depth for two main reasons: a) it mimics most of the systems we use daily, so I would like to understand how it works; b) the course doesn't venture much into this topic so exploring it through this project would be a great learning experience and a good challenge.

I came across a few algorithms like the Chandy-Lamport algorithm, bully algorithm and the gossip algorithm. I picked the bully algorithm because it's interesting to learn how a cluster of nodes would elect a leader, unlike K8s and Cassandra, where the master node is manually selected sometimes.

Part 1.2: Brainstorming

We both seemed to have a common interest in distributed systems, especially in understanding the implementation of a fault tolerance network/system like in K8s.

Therefore, we decided to focus on distributed systems, especially failure detection.

Part 1.3: Proposal

Our proposal is to implement failure detection in a simulated distributed system.

Algorithm we would like to implement:

The algorithm we would like to implement is a “gossip-style” failure detection, which is one of the epidemic protocols. This algorithm would allow us to detect node failures in a decentralized network, using peer-to-peer communication. In this algorithm, a node sends its state to some selected peers (ranging from 0 to n-1 in a n-node system) at a constant time interval, while also saving the states (of its peers) it receives. Through this constant communication, the nodes will converge their copy of nodes’ states and agree on which ones are alive or not.

Baseline algorithm:

The algorithm we will be comparing “gossip-style” failure detection with is known as the heartbeat failure detection. Unlike peer-to-peer communication in gossip-style, this algorithm utilizes a centralized server to communicate with all nodes. For the sake of simplicity, a master node will be manually selected and each remaining node in the centralized network is expected to periodically send a health status to the master node. If a message isn’t received, it is assumed dead.

While the above algorithm utilizes all the nodes to identify a failure, the heartbeat algorithm only has one node to do so, which can introduce potential problems if we were to scale and increase the number of nodes in the system.

Metrics used for comparison:

1. Latency/Response time: how long it takes for each algorithm to identify a node failure
2. Accuracy: the proportion of correctly identified failed nodes vs false positives vs missed failures (false negatives) – this might be something worth evaluating to understand the convergence algorithm in the gossip-style failure detection
3. Scalability: performance and overhead of the algorithm as the number of nodes increase
4. Network traffic/number of messages sent: The volume of heartbeat messages exchanged among nodes, indicating the overhead introduced by the protocol.

Test cases:

1. Single Node Failure:

Simulate the failure of a single node and verify if the failure is detected within the expected timeframe.

2. Multiple Concurrent Failures:

Simulate simultaneous failures of several nodes and assess whether the algorithm accurately detects all failed nodes.

3. Temporary Network Partition:

Introduce a temporary network partition and observe if the algorithm correctly handles partitions without erroneously marking active nodes as failed.

4. High Load Scenario:

Evaluate the algorithm under heavy network traffic conditions to ensure that it maintains accurate detection without performance degradation.

5. Recovery After Failure (potential stretch):

Test the scenario where nodes recover after a failure and validate if the system accurately detects their reintegration.