Carnegie Mellon University in Qatar

Distributed Systems

15-440 - Fall 2023

Problem Set 4

Out: October 31, 2023

Due: November 12, 2023

Problem I: Quorum-Based Protocol (15 Points)

Now that you are taking 15-440 at CMUQ, you might start getting offers for developing real-world distributed systems. Assume an offer came from *MyBlogPost.com*, a popular website for posting and following blogs, asking you to build a highly-available storage system for them. You accepted the offer and, after much thought, decided to use Gifford's quorum consensus replication as a technique to achieve high availability. You analyzed the user accessibility patterns at *MyBlogPost.com* over the past 12 months and realized that they can be classified into two major categories, say, **CAT1** and **CAT2**. For efficiency reasons, you planned to treat the two categories differently. In addition, to keep your technique simple, you decided to give a weight of 1 to each voting node in the read and write quorums of each category.

5pts a. Based on your analysis, **CAT1** is read-intensive, with 98% read and 2% write accesses. You are planning to set up a 5-replica system for this category and seeking to optimize for the common case (i.e., make reads fast). What read and write quorum sizes would you select so as to achieve fast reads, while ensuring correctness? Explain your answer.

5pts b. Based on your analysis, **CAT2** is write-intensive, with 99% write and 1% read accesses. You are planning to set up a 4-replica system and seeking to optimize for the common case (i.e., make writes fast). What read and write quorum sizes would you select so as to achieve fast writes, while ensuring correctness? Explain your answer.

5pts c. Suppose on each access, a node may fail with a probability, $p = 0.1$. You may also assume that failures are independent. For the read and write quorum values you computed in part (b), what is the probability that an access will fail?

Problem II: Decentralized Permission-based Synchronization (25 Points)

In the permission-based decentralized mutual-exclusion algorithm proposed by Lin *et al.*, every resource replica has its own coordinator for controlling access. Whenever a process wants to access the resource, it will have to get a majority vote from $m > n/2$ coordinators (assuming there are n coordinators). If a coordinator does not want to vote for a process (e.g., because it has already voted for another process), it will send a permission-denied message to the process.

10pts a. Explain how a deadlock can happen when multiple processes try to get permission to access a resource at the same time.

15pts b. Propose a protocol that allows for avoiding such deadlocks in the system (*Hint: consider adding a super-coordinator to manage concurrent requests for permissions*).

Problem III: Programming Models (60 Points)

30pts

a. Every graph problem can be solved using matrix-matrix or matrix-vector multiplication. Figure 1 illustrates how a graph, $G(E, V)$, where E is the set of edges and V is the set of vertices, can be represented as an adjacency matrix. A cell a_{ij} in the matrix is equal to 1 when edge $(i, j) \in E$. A simple graph problem example of calculating in-degree is shown in Figure 1. Multiplying the transpose of the graph adjacency matrix (unweighted graph) with a vector of all ones produces a vector of vertex in-degrees. To get the out-degrees, one can multiply the adjacency matrix with a vector of all ones.

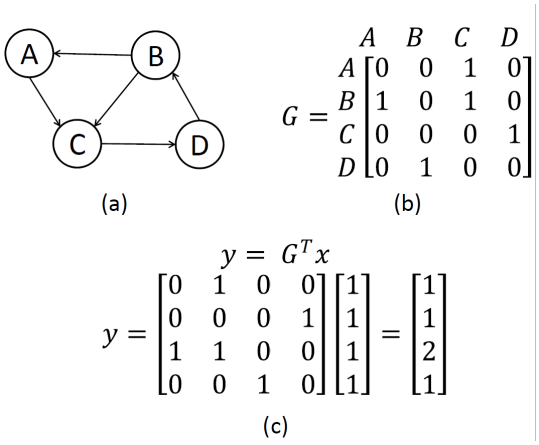

Figure 1: Graph (a) Logical representation (b) Adjacency matrix (c) In-degree calculation as SPMV GT x = y. Vector x is all ones. The output vector y indicates the number of incoming edges for each vertex.

In this part, you will implement Breadth First Search (BFS) using matrix-vector multiplication. In particular, starting at the source (level 0), at every iteration, BFS visits the set of unvisited neighbor nodes at the next level in the graph; until all nodes are visited. The latest set of visited nodes at a given iteration is called the *frontier*. A frontier can be represented as a vector f where $f[i] = 1$ if node i is in the current frontier. A node (k) is a candidate to be in the new frontier, if there is a node (j) in the current frontier $(f[j] = 1)$ such that $a_{jk} = 1$. Hence, at each iteration, you will multiply the transpose of the graph adjacency matrix with the frontier vector, to get a vector of all the candidates. However, only the ones that haven't been visited so far will be in the new frontier.

Use MPI for Python (mpi4py) to implement BFS as an iterative matrix-vector multiplication process.

30pts

b. Read the following 2 papers and provide a summary, alongside at least 3 "strengths" and at least 3 "weaknesses" per each paper. Your summary for each paper should not exceed 4 paragraphs.

LA3 Paper
Ray Paper