INDIAN INSTITUTE OF INFORMATION TECHNOLOGY, ALLAHABAD
End-semester Examination, May 2018
Distributed Systems (IDSS630E)

Max. Time: 3 Hours **Max. Marks: 75**

There are seven questions in the paper. Answer all.

Q.1. Short answer type questions : **[5*2 =10]**

a. Which techniques are used in DNS to improve scalability?
i. *Replication* ***ii.*** *Recursive queries* ***iii.*** Caching / Soft-State ***iv.*** Partitioning of Namespace

b. For each of the following distributed mutual exclusion solutions, which properties does that solution have? Circle all that apply.

(i) Ring-based algorithm:	fair	reliable	long synchronization delay
(ii) Majority rules:	fair	reliable	long synchronization delay
(iii) Maekawa voting:	fair	reliable	long synchronization delay

c. For each critical section (CS) execution, Maekawa's algorithm requires _____ messages per CS execution and the Synchronization delay in the algorithm is _____.

d. The global state recording part of a single instance of the Chandy-Lamport algorithm requires _____ messages and _____ time, where e is the number of edges in the network and d is the diameter of the network

e. State whether the following statement is true or false.
In Google File System (GFS), when a client needs to access data, it contacts the master and gets the data from the master.

Q2. Distributed File System,Map-Reduce and Hadoop **[3+7=10]**

(a) What is the role of a mapper and reducer in Hadoop Map-reduce?

(b) Consider a twitter data-set which contains data of the form:

postid <tab> status-message

In Twitter, some status-messages are repeats of earlier status messages. These repeated messages are preceded by "RT" followed by the original status message. For example:

142412141 I need puppies but youtube is blocked.
584923144 Puppies are awesome!
929483834 RT Puppies are awesome!
872935923 RT Puppies are awesome!
423135125 Meow mix meow mix please retwitter
723917388 RT I need puppies but youtube is blocked.
629359237 RT Puppies are awesome!
562983948 RT Meow mix meow mix please retwitter
729385724 RT Meow mix meow mix please retwitter

In this problem, your goal is to identify the most "retweeted" status messages using the MapReduce style of programming. You must write one or more Map and Reduce functions whose final output contains a sorted list of post ids that have been "retweeted"/repeated, sorted by the number of times they have been retweeted.

If you run on the above example, the output would be:

1 I need puppies but youtube is blocked.
2 Meow mix meow mix please retwitter
3 Puppies are awesome!

Only retweeted statuses should appear in the output. The output should be sorted by the number of retweets from fewest to most. You can use any pseudocode you choose (Java-like or C++-like): You can use two types: Integers and Strings. You can assume the existence of STL-like string functions for your convenience. You can iterate over lists using the for construct as shown below:

```
for item in list {
        (do something with item)
    }
```

You should use the "emit(key,value)" pseudocode for the output of functions. The skeleton of the code has been provided for you. Fill in the appropriate types and the map and reduce functions.

```
// Assume line has been parsed into post id and status already
Map(int postid, String status) {
}


// Fill in the appropriate types in the brackets
Reduce(<      > key, List<      > values) {

}

Map2(String status, int count) {


}

Reduce2(int count, String status) {

}
```

Q3. Resource Sharing in Distributed Systems - Naming Services **[10]**

(a) What is navigation in context of naming service such as DNS

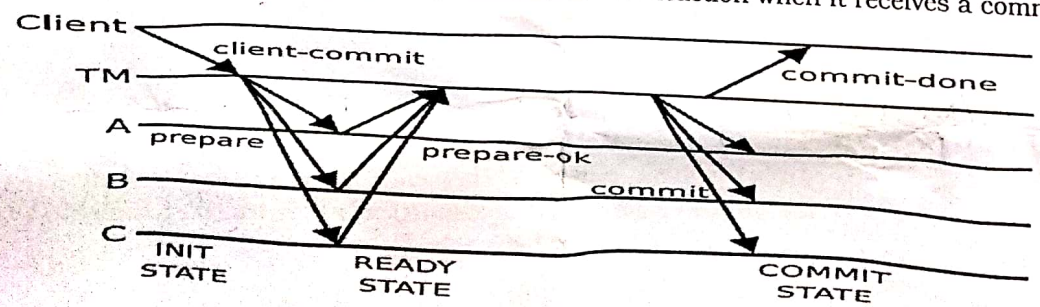
(b) Discuss the iterative and the recursive navigation model

(c) How does caching help a naming service's availability?

(d) When might a DNS server provide multiple answers to a single name look up and why?

Q4. Distributed Database - Two-Phase Commit **[10]**

The figure below shows a successful execution of two-phase commit. The client asks the transaction manager (TM) to organize an atomic transaction involving servers A,B,and C. A transaction is atomic if either all three servers execute their part of the transaction, or none of them execute it. The TM sends prepare messages over the network to the three servers asking them if they are willing to perform the transaction. Each server that is willing (in this case all of them) sends a prepare-ok message to the TM. If the TM collects prepare-ok messages from all three servers, the TM sends a commit message to each server, and then sends a commit-done message to the client. A server executes the transaction when it receives a commit message



If the server receives a prepare message and thinks it would not be able to commit, it returns a prepare-abort meassge to TM, and the TM sends abort meassages to all servers and an abort-done message to client.

The tricky part in two phase commit is to handle server failures. Answer the following questions regarding fault tolerance of a two phase commit protocol. Assume there is never any confusion about which transaction a message is part of (since message carries transaction ids). Additionally, assume all participants follow the protocol to the best of their ability.

(a) What are the posssible symptoms of a server or TM failure?
(b) Propose an action to resolve the transaction (either commit or abort) in case of server failure.
(c) Suppose TM has sent all the prepare messages but has not received a prepare-ok (prepare-abort) from server A. Would it be correct for the TM to abort the transaction at this point - sending abort messages to servers and abort-done to client? Why or Why not?

(d) Suppose server A has received the prepare, sent the prepare-ok but has not yet received a commit or abort message. Server A contacts Server B and discovers that server B has received a commit message. Would it be ok for server A to execute its part of the transaction at this point, as if it had too received a commit message? Why or why not?

(e) Suppose A has received the prepare, sent the prepare-ok but has not yet received a commit or abort message. This time Server A contacts Server B and C, and discovers both of them have received prepare, sent prepare-ok but have nor received commit or abort. Would it be correct for server A to execute its part of the transaction at this point - as if it had too received a commit message? Why or why not?

Q5. Distributed Algorithms- Distributed Mutual Exclusion **[15]**

a. You want to choose a way to provide mutual exclusion for the following scenario: You have 2 very high quality, fast computers both connected to the same Ethernet switch. Requests are sent to both computers, and only one should act upon the request. When the request arrives, it should be assigned to one or the other computer, which then processes it. Exactly one computer should handle each request. What mutual exclusion protocol would you use – token ring or permission based? Justify your answer in terms of its performance, implementation complexity, and other factors you feel important.

b. Instead of 2 computers, you now want to build the same system using 20 computers. These computers are cheap, pretty unreliable, and all have varying speeds. Don't worry about a computer crashing once a request was received, but now what mutual exclusion protocol would you use? Justify numerically in terms of the number of messages, the latency, reliability, etc., compared to the protocol you picked for the previous part:

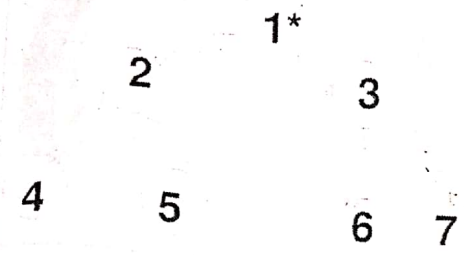
c. A friend proposes a new protocol for you. He wants to get the efficiency of Maekawa's quorum system but with robustness to node failures. He calls it the *Cluster Mutex, Ultra*, and it works like this: Arrange the nodes in a grid, just like Maekawa's algorithm. Assume there are an exact power of two number of nodes for simplicity.
Mutex acquire : The requesting node r picks two rows and two columns to send its requests to. (Recall that Maekawa just sent to one row and one column). It broadcasts a message to the $2\sqrt{N}$ nodes in its rows and the $2\sqrt{N}$ nodes in its columns, saying REQUEST(time, r).

recv REQUEST : If node hasn't granted its VOTE to anyone yet, send VOTE(time, r) back to r. Otherwise it's voted for s, so send back SORRY(time, r, s).

recv VOTE: When a node receives at least $\sqrt{N}$ column votes (from nodes in the columns it picked) and $\sqrt{N}$ row votes (from nodes in the rows it picked), it has the lock and can proceed.
Mutex release: Broadcast RELEASE(r) to all of the row and column nodes in its quorum.

Explain whether the protocol provides mutual exclusion.

d. An instance of the logical tree structure for the token based mutual exclusion algorithm protocol has been shown in the figure below. Intially all request queues are empty and the token is at node 1. At this moment, request to claim the token come every one unit time apart in the order of {7,6,5,4,3,2,1}. Assume that the communication delay between each pair of nodes is also one-unit time. Show the distribution of requests in the request queue after node 7 has claimed the token and has been in the critical section for five units of time. What is the order of the execution sequence? Calculate the synchronization delay.



Q6. Distributed Algorithms - Leader Election

(a) For a synchronous ring all but one processor have the same identifier, is it possible to have a leader election? Either give an algorithm or prove an impossibility result.

(b) Explain the bully algorithm with an example. **[10]**

Q7. Distributed Consensus, Fault Tolerance – Byzantine Generals Problem and Crash failures

The difficulty of byzantine general problem lies in the ability of a traitor lieutenant to lie about the commander's order, thus if we can restrict this ability by making the following assumptions, the 3 - general problem is solvable with any number of traitors. **[10]**

i. Byzntine generals send signed messages to each other.
ii. A loyal general's signature cannot be forged, any alteration can be detected. -> can drop a message, but can't change it
iii. Any one can verify the authenticity of a signature. -> no one can fool a general
iv. One traitor can forge the signature of another traitor.

Provide an algorithm that gurantees interactive consistency for any number of traitors (n generals can tolerate any number of traitors). If there are less than two loyal generals, interactive consistency is vacuously satisfied.
