EE 360P TEST 2 Vijay K. Garg Spring'18

NAME:

UT EID:

Honor Code: I have neither cheated nor helped anybody cheat in this test.
Signature:

Time Allowed: 75 minutes
Maximum Score: 75 points
Instructions: This is a CLOSED book exam. Attempt all questions.

Q. 1 **(20 points)** Please be concise in your answers.
(a, 4 points)
(i) Give one advantage and one disadvantage of using Java RMI compared to using TCP.
Advantages: easier to program (no parsing of messages). Disadvantages: Additional overhead (e.g. serialization of parameters)

(ii) What is the role of stub routines generated by Java compiler for remote objects?

marshalling all the parameters and then communicating them

(iii) How are non-remote objects passed as parameters for java remote method invocation?

by serialization

(b, 3 points) Let E be the set of events in a distributed computation. Define Lamport's happened-before relation ($\rightarrow$) on E.

(c, 4 points) Two events e and f are defined to be *concurrent* if neither event happened-before the other. Show that concurrency relation defined on the set of events is not transitive.

(d, 5 points) Let $(E, \rightarrow)$ be a distributed computation on n processes with E as the set of events and $\rightarrow$ as the happened-before relation.
(i) Let C be a logical clock function that maps each event to a natural number. Give the property satisfied by the logical clock function.

$\forall e, f \in E : e \rightarrow f \Rightarrow C(e) < C(f)$

(ii) Let V be a vector clock function that maps each event to a vector of size n. Give the property satisfied by the vector clock function.

$\forall e, f \in E : e \rightarrow f \text{iff} V(e) < V(f)$

(iii) Let G be a subset of E. Give the property satisfied by G to be a consistent global state (i.e. define a consistent global state in the event-based model)

$\forall e, f \in E : (f \in G) \wedge (e \rightarrow f) \Rightarrow (e \in G)$

(e, 4 points) Consider a distributed system with n processes $U = \{P_1, \ldots, P_n\}$.
(i) Suppose that P_i always requests permission from $C_i \subseteq U$ to enter the critical section. Give the condition on request sets (C_i's) that guarantees that two processes can never be in the Critical Section at the same time.

$$\forall i, j : i \neq j : C_i \cap C_j \neq \{\}$$

(ii) Suppose that P_i always requests permission from $R_i \subseteq U$ for read operations and requests permission from $W_i \subseteq U$ for the write operations. Give the conditions on the request sets so that there are no read-write conflicts and no write-write conflicts.

$$\forall i, j : i \neq j : R_i \cap W_j \neq \{\}$$

$$\forall i, j : i \neq j : W_i \cap W_j \neq \{\}$$

Q. 2**(5 points, Dining Philosopher's Algorithm)**
Consider the dining philosopher for resource allocation. The forks are placed initially so that they are all dirty and the the conflict resolution graph induced by the forks is acyclic.

(a, 1 point) Give the condition under which a hungry philosopher should release the fork on getting the request for the fork.

(b, 4 points) Show that a thinking philosopher can hold only dirty forks in the dining philosopher algorithm.

Initially the invariant holds because all forks are dirty. Only a hungry philosopher can request forks and therefore get clean forks. Once the philosopher eats, all forks get dirtied and whenever philosopher stops eating (gets into thinking state), all forks that he holds are dirty.

Q. 3 **(10 points, Maekawa's Algorithm)**
(a, 5 points) Recall that Maekawa's grid based quorum algorithm requires every process to request votes from all processes in its row and its column before entering the critical section. Is it guaranteed that Maekawa's algorithm allows processes to enter only in the increasing order of timestamps? Justify your answer. Give a brief proof if the statement is true. Otherwise, show a counter-example.

There are four processes P1, P2, P3, P4 arranged in two by two grid. At time 1, P1 sends a request to P2 and P3. At time 2 P4 sends a request to P2 and P3. P4's request reaches P2 and P3 before P1's request. Both P2 and P3 give their vote to P4 which enters the critical section before P1.

(b, 5 points) Your friend has suggested that Maekawa's algorithm can be extended by arranging processes in a three dimensional cube. We will assume a processor at each of the (x, y, z) coordinate where $x, y, z \in \{0..N^{1/3} - 1\}$ (you may assume that N is a perfect cube). Each process needs vote from all processes that differ in at most one coordinate. Either show that your friend's algorithm guarantees safety property or give a counter-example.

Consider a cube on 8 processes. We will assume a processor at each of the (x, y, z) coordinate where x, y, z are either 0 or 1. Consider two concurrent requests by P(0,0,0) and P(1,1,1). The request set for P(0,0,0) contains all processes with at least two zero coordinates. The request set for P(1,1,1) contains all processes with at least two coordinates as 1. Hence their request sets do not intersect.

Q. 4 **(10 points, Map-Reduce)** Suppose that there is a large matrix M of dimension $m \times n$ stored in a distributed system. The matrix is so large that it cannot be stored on any one machine and its entries are stored across different machines. Each machine has the copy of a column vector v of size n. The vector v can fit in the main memory of any machine. We would like to compute Mv using the map-reduce.
(a, 5 points) Give the pseudo-code of your mapper function. Show how the matrix entries are stored at various nodes and the computation of the mapper function.

The matrix would be stored as the key-value pair $((i, j), M[i, j])$

The mapper function is as follows:
function Mapper
input: key-value ((i,j), a) // a is $M[i, j]$

write(j, a*v[j]);

(b, 5 points) Show the pseudo-code of your reducer function.

function reducer
input: key, list of values
sum := 0;
for (val: values)
sum := sum + val;

write(key, sum)

Q. 5 **(15 points)** Consider a distributed computation in which every process is initially blue. A blue process can send blue messages and a brown process can send brown messages. A process can turn from blue to brown whenever it wants. It can turn from brown to blue only on receiving a blue message.
(a, 3 points) Let B be defined as " All processes are blue." Is B stable? Justify your answer.

No. A process that is blue may turn brown on its accord making B false.

(b, 2 points) Let predicate B be defined as "Every process has turned brown at least once." Is this predicate stable? Justify your answer.

Yes. Once a predicate has turned brown at least once, the predicate continues to hold.

(c, 10 points) Let B be defined as "Every process is blue and has turned brown at least once." Briefly give a method to detect this predicate. What is the message complexity of your detection algorithm? Assume that there are n processes in the system and any process sends at most m messages in the computation. You do not have to give the pseudo code; just invoke appropriate algorithms discussed in the class.

Each process sends its vector clock to the checker process whenever (1) it is blue (2) it has turned brown at least once and (3) it has not sent its vector clock since the last external event. The checker process invokes conjunctive predicate detection algorithm to detect B.

Q. 6 **(15 points, Consistent Global States)** A consistent global state G in a computation on n processes is a vector of local states such that $G[i]$ is concurrent with (or incomparable) to $G[j]$ for all $i, j \in \{1..n\}$, where $G[i]$ is the local state of P_i in G. Show that if G and H are consistent global states, then K defined as $min(G, H)$ (i.e., $K[i] = min(G[i], H[i])$, for all i) is also a consistent global state.

Answer: We need to show that for all $i, j : K[i]||K[j]$. This is equivalent to showing for all $i, j : K[i] \not\rightarrow K[j]$. Without loss of generality assume that $K[i] = G[i]$. Since $K[j] = min(G[j], H[j])$, we have two cases:
Case 1 : If $K[j] = G[j]$ then $K[i] \not\rightarrow K[j]$ because $G[i] \not\rightarrow G[j]$ (G is a consistent global state)
Case 2: If $K[j] = H[j]$ then $K[i] \rightarrow K[j]$ is equivalent to $G[i] \rightarrow H[j]$. This implies $G[i] \rightarrow G[j]$ because $H[j] \leq G[j]$). However, this is a contradiction because G is a consistent global state.