CALIFORNIA INSTITUTE OF TECHNOLOGY
Computing and Mathematical Sciences

CS 142

R. Murray, M. Chandy — **Homework Set #7** — Issued: 20 Nov 2019
Fall 2019 — Due: 27 Nov 2019

Note: Please keep track of the number of hours that you spent on this homework set (including reading) and enter this when you turn in your homework on Moodle.

1. Consider the Byzantine agreement problem in which a source processor broadcasts its initial value to all other processors. All nonfaulty processors should agree on a single common value. We assume that message delivery is synchronous, the processors are fully connected, receivers know the identity of the sender processor of a message, and the communication medium is reliable (so only processors are prone to failure).

   (a) Show that the Byzantine agreement problem cannot always be reached among four processors if two processors are faulty and messages are not signed.

   (b) Show that if you can solve the Byzantine agreement problem, you can solve the consensus problem (each processor has its own initial value and all nonfaulty processors must agree on a single common value, if there is one.)

   Detailed proofs are not required, but you should provide enough information to capture the key ideas behind your answer.

2. The Byzantine Generals problem discussed in class is a form of a fault-tolerant "consensus problem" in which (1) all non-faulty officers come to a consensus, and (2) if the general is non-faulty then this consensus value is the same as the value set by the general. The problem discussed in class deals with consensus on a single bit (advance or retreat).

   Consider a generalization of the algorithm to two bits. Consider only the case of non-forgeable (encrypted) messages. The general sets two bits (one of the values 00, 01, 10, 11) instead of the single (attack/retreat) bit. For example, the first bit represents artillery and the second cavalry. The same officers command both artillery and cavalry units. For example, the general issues the command 1,0 to indicate that all units of artillery should attack and all units of cavalry should retreat.

   The obvious solution to solve a 2-bit problem is to solve two separate 1-bit problems: run two completely separate Byzantine agreement problems, one for the artillery and one for the cavalry.

   Somebody proposes the following algorithm to reduce the number of messages:

   - Officers (including the general) send messages with two bits. (Recall that a faulty officer also has the option of sending no messages at all.)
   - Each officer sends a message $x[0], x[1]$ where $x[j] = 1$ to indicate that this officer is committed to set bit j to 1. It sets $x[j] = 0$ to indicate that it has not yet committed to set bit j to 1. For example, an officer sends 10 to indicate that the officer's artillery unit

is committed to attack, and the officer hasn't yet committed her cavalry unit to attack or not.

The proposed algorithm with two bits is identical to the one-bit algorithm discussed in class and the text except that messages have two bits.

Provide an argument that the algorithm works or give a counterexample. Detailed proofs are not required, but you should provide enough information to capture the key ideas behind your answer.

3. Consider the Byzantine Generals problem and algorithm discussed in class with unforgeable messages. Consider a problem with a very large number of officers, say 1000002 of officers (including the general), of which at most 2 can be faulty. We showed in class that the problem can be solved in 3 synchronous rounds. Will exactly the same algorithm work in the case *where messages can be forged*? Provide an argument that the algorithm works or give a counterexample. (Detailed proofs are not required, but you should provide enough information to capture the key ideas behind your answer.)

   Note: In the non-encrypted case, a faulty agent i can send a message to another agent j of what i claims to be a copy of a message from a third agent k even though k sent no such message.

4. Consider a simple agreement protocol that consists of a set of processors that are attempting to agree on whether to commit a piece of data to a database. Under initiation of a commit request by a master process P0, each processor makes a local determination about whether a transaction should be committed (represented by the value 1) or aborted (represented by the value 0). We assume that messages are delivered without error and that all communications are *synchronized* (all messages delivered at the same time). A processor may fail at any time, but otherwise performs properly.

   We have the following specifications for the commitment protocol:
   - S1: No two processes should decide on different values.
   - S2: If any process starts with 0 (abort), then 0 is on the only possible decision value.
   - S3: If all processes start with 1 and there are no failures, then 1 is the only possible decision value.

   Termination comes in two flavors:

   - Weak termination: if there are no failures, then all processes eventually decide.
   - Strong termination: all nonfaulty processes eventually decide.

   Consider the "two phase commit" algorithm, with process 0 (P0) acting as a distinguished process (master server):

   - Round 1:
     - All processes except for P0 send their initial value to P0.
     - Any process whose initial value is 0 decides 0.
     - P0 collects all values, plus its own initial value, into a vector. If all positions in the vector are 1, then P0 decides 1. Otherwise, P0 decides 0.

- Round 2:
  - P0 broadcasts its decision to all of the processes.
  - Any process other than P0 that receives a message at round 2 and has not already decided in round 1 decides on the value it receives in the message from P0.

(a) Show that the two phase commit algorithm solves S1, S2, and S3 with the weak termination condition (i.e., S1–S3 hold in the presence of failure and in the absence of failure, the algorithm terminates). You do not have to give a detailed proof, but you should provide enough detail to cover all cases correctly.

(b) Give a counterexample showing that the two phase commit algorithm does not satisfy the strong termination condition.

Now consider the "three phase commit" algorithm:

- Round 1:
  - All processes except for P0 send their initial value to P0.
  - Any process whose initial value is 0 decides 0.
  - P0 collects all values, plus its own initial value, into a vector. If all positions in the vector are 1, then P0 becomes *ready*, but does not yet decide. Otherwise, P0 decides 0.
- Round 2:
  - If P0 has decided 0, then it broadcasts *decide*(0) to all of the processes. Otherwise, it broadcasts *ready* and decides 1.
  - Any process other than P0 that receives a *decide*(0) message at round 2 decides 0. Any process that recieves *ready* becomes *ready*.
- Round 3:
  - If P0 has decided 1, it broadcasts decides(1).
  - Any process that receives *decide(1)* decides 1.

(c) Show that the three phase commit algorithm solves S1, S2, and S3 with the strong termination condition as long as P0 does not fail. You do not have to give a detailed proof, but provide enough information to capture the key cases.

(d) If P0 fails, show that the three phase commit algorithm does *not* solve the commitment problem with strong termination.