Byzantine Agreement

CS255

Chris Pollett

Mar. 20, 2006.

Outline

- Byzantine Agreement Problem

Byzantine Agreement Problem

- Similar to Choice Coordination Problem.
- We want to agree on one of two possible values (say, heads or tails [0,1]).
- We have n processors t of which may be faulty.
- We require that the decision reached by a protocol should have:
  1. All good processors finish with the same decision.
  2. If all the good processors begin with the same value v, then they should all finish with same value v.

More on the set up of the Byzantine Problem

- The set of faulty processors is fixed before the computation begins.
- The good processors do not know which processors are faulty.
- During a round, each processor may send one message to each other processor.
- Each processor receives a vote from each of the remaining processors, before the following round begins.
- A processor is allowed to send different messages to different processors.
- Good processors will be assumed to follow our algorithm exactly.
- It is known any deterministic algorithm for this problem needs at least t+1 rounds.
- We will present a randomized algorithm with O(1) expected runtime.

Some More Remarks before we Begin

- Last day, our solution to the choice coordination problem was only for two processors choosing between two values.
- Neither of these processors was faulty.
- The algorithm we present today works for n processors choosing between two values, so is already more general, ignoring the allowance for faulty processors.
- The original choice coordination problem was for n processors to choose among m choices.
- Notice by repeating the Byzantine procedure $\log_2 m$ times we can have our processors agree on a first bit of the number between 1 to m, then a second bit of the number between 1 to m, etc.
- If each such single bit agreement can be done in constant time as a function of n. Then, agreeing on a number between 1 and m can be done in O(log m) time. Notice this does not depend on the number of processors.

Randomized Algorithm for Byzantine Agreement

- We will assume that at the start of each round a trusted third party flips a fair coin.
- Any of the processors have access to this coin.
- We will assume that the number of faulty processors is a number $t < n/8$.
- Each round a good processor sends the same vote to all the other processors.
- A faulty processor may send arbitrary or even inconsistent votes to each other processor.
- Let $L = (5n/8) + 1$, $H = (3n/4) + 1$, and $G = 7n/8$.

What the *i*th Processor does during a round (if it is good).

Input: A value for b_i.

Output: A decision d_i.

1. vote = b_i.
2. For each round, do
    3. Broadcast vote;
    4. Receive votes from all the other processors.
    5. Set maj = majority (0 or 1) value among the votes cast
    6. Set tally = the number of votes that maj received.
    7. if coin = heads then set threshold = L; else set threshold = H
    8. if tally >= treshold then set vote = maj; else vote = 0
    9. if tally >= G then set d_i=maj permanently.

Analysis

- First, if all processors begin the round with the same vote, then 9 will apply and so this value will be the value eventually settled upon.
- Suppose the processors begin the round with different values for the vote.
- If two processors compute different values for maj in step 5, then tally does not exceed threshold regardless of whether L or H was chosen as threshold. So all good processors would set their votes to 0 and an agreement would be reached.
- We say a faulty processor **foils** a threshold x in {L,H} in a round if, by sending different messages to the good processors, they cause tally to exceed x for at least one good processor, an to be no more than x for at least one good processor.
- Since the difference between the two possible thresholds is at least t, the faulty processor can foil at most one threshold in a round.
- Since the threshold is chosen with equal probability from {L,H}, it is foiled with probability at most 1/2.
- Thus, the expected number of rounds before we have an unfoiled threshold is at most 2. If the threshold is not foiled then all good processors compute the same value v in step 8.