Homework lecture 5
Agreement and consensus I: concepts and protocols for crash failures

Jaap-Henk Hoepman
jhh@cs.ru.nl

May 13, 2019

Question 1: Suppose a protocol can tolerate (i.e. works when confronted with) byzantine failures. Will the same protocol tolerate (the same number of) crash failures?

Answer: Yes. A crash failure is a special kind of byzantine failure, namely one where the arbitrary action of the processor is the action of not doing anything anymore.

Question 2: How many messages does the consensus protocol for crash failures exchange if there are no failures?

Can you somehow optimize this?

Answer: To answer the first part of the question, let n be the number of processors and f be the maximal number of fautly ones. The protocol specified that in each round r, with $1 \leq r \leq f+1$, a processor p does the following. It sends, for all σ with $|\sigma| = r-1 \wedge p \notin \sigma$, a message to all q including p.

How many σ with $|\sigma| = r-1 \wedge p \notin \sigma$ are there? Remember that σ never contains the same processor more than once. Hence for the first element in σ we have $n-1$ choices (remember: $p \notin \sigma$), for the next we have $n-2$ choices, etc. That means there are

$$\frac{(n-1)!}{((n-1)-(r-1))!} = \frac{(n-1)!}{(n-r)!}$$

such σ with $|\sigma| = r-1 \wedge p \notin \sigma$. For these, n messages are sent (to all processors q including p). I.e. processor p sends

$$n\frac{(n-1)!}{(n-r)!} = \frac{n!}{(n-r)!}$$

messages in round r. In total all processors then send

$$n\sum_{r=1}^{f+1}\frac{n!}{(n-r)!}$$

messages.

To answer the second part of the question, recall the decision rule. Let $V_p = \{v \mid v = v_\sigma^p \in T_p \wedge v \neq \perp\}$. The decision rule says that p decides on v if $V_p = \{v\}$ and on a default value v_{def} otherwise. In other words, as soon as $|V_p| > 1$, i.e. as soon as the tree contains two different values (also different from $\perp$), then p decides on the default. This means that as soon as the tree contains two such different values, p knows enough to decide. Moreover, if p is non faulty, it will have sent these two values to all other processors. This means all other non-faulty nodes have received these two different values and hence will also decide on the default.

This means the protocol can be modified in the following manner. Processors keep a set of sent values S_p, initially empty. In each round r processor p does the following. It sends, for all σ with $|\sigma| = r - 1 \wedge p \notin \sigma$, a value v_σ^p to all q including p, provided $v_\sigma^p \notin S_p$. It adds v_σ^p to S_p.

This drastically reduces the message complexity. Each processor sends at most 2 messages to all other processors. The total number of messages sent is therefore never more than $2n^2$.

Question 3: Consider an asynchronous system of n processes, f of which may fail by crashing (only). Let each process p have an input value $C[p].in \in \{0, 1\}$. Consider the following protocol for process p.

> **forall** q (including p) **send** $C[p].in$ to q.
> **receive** $n - f$ values and store them in the multiset V.
> **decide** on $C[p].decision = majority(V)$

(where $majority(V)$ computes the majority of values in the multiset V, returning 1 if there is a tie). Now answer the following questions.

a) Why can the algorithm only consider $n - f$ received values (and no more) to compute the majority, even if no processes crashed?

b) Why can different processes decide on different values using this protocol?

c) How many 0 (or 1) valued inputs should there be initially, to guarantee that all correct processors decide on the same value?

Answer:

a) Even if no processes crash, there is no way for a process to know this in advance. If it waits for more than $n - f$ values to receive before computing a decision, it may wait forever (in an execution in which f processes *do* crash).

b) Suppose n is even, and let $f = 1$. Consider a scenario where the first $n/2$ processes have input 0, while the last $n/2$ processes have input 1.

If no process crashes, there are $n/2$ zeros and $n/2$ ones being sent to each process. However, each process receives at most $n - f = n - 1$ values into V. Because the system is asynchronous, there is no guaranteed order in which messages are delivered. Therefore in some cases V may contain $n/2$ zeros and $n/2 - 1$ ones (deciding 0) or vice versa.

c) A process decides 1 if it receives at least $\lceil (n - f)/2 \rceil$ ones, and 0 if it receives at least $\lfloor (n - f)/2 \rfloor + 1$ zeros (note that 0 and 1 are the only possible decision value). Suppose at least one process p decides 1. To ensure no other process receives $\lfloor (n - f)/2 \rfloor + 1$ or more zeros, the number of processes having input 0 must be less than $\lfloor (n - f)/2 \rfloor + 1$. So there must be at least $n - \lfloor (n - f)/2 \rfloor + 1$ processes having input 1. (Or, the other way around, the number of processes having input 1 must be less than $\lceil (n - f)/2 \rceil$.)

Alternative answer: a process needs to receive at least $\lfloor (n - f)/2 \rfloor + 1$ copies of the same value to ensure this is the majority, and thus the value decided. Of all input values sent a process receives only $n - f$, i.e. it looses an arbitrary f of the input values. hence if at least $\lfloor (n - f)/2 \rfloor + 1 + f \sim n/2 + f/2 + 1$ of the input values are the same, all processes receive at least $\lfloor (n - f)/2 \rfloor + 1$ copies of that value and decide on it.