Homework lecture 6
Agreement and consensus II: Byzantine failures

Jaap-Henk Hoepman
jhh@cs.ru.nl

June 3, 2019

Question 1: Does the consensus protocol for Byzantine failures also work for non-binary input values? What about the consensus protocol for crash failures?

Answer: The consensus protocol for Byzantine failures also works for non-binary input values, as nowhere in the proofs the fact is uses that the values are binary.

You might think that the majority decision rule does not work when non-binary inputs are possible. For example, with one input 0 and one input 1 you might think the byzantine processor might create an intermediate set of collected values $\{1, 2, 3, 4, 1\}$ somewhere in the tree for one processor and another set $\{1, 2, 3, 4, 2\}$ (by sending 1 to the first and 2 to the other processor). In that case the majority outcome would be different for both processors. But that is actually something that cannot happen. Study the answer of question 2 to see why.

For the consensus protocol for crash failures this is also the case, provided the default decision value is a possible input value.

Question 2: Consider the consensus algorithm for Byzantien failures, and let $f = 1$ and $n = 4$. Let processor p_4 be faulty. Let processors p_1 and p_2 have input 0 and processors p_3 and p_4 have input 1. Write out the full execution of the protocol under these circumstances, assuming that p_4 sends the special value ? for each message it sends.

Answer: Let us write v^i for v^{p_i}.

Before round 1, all $v^j_\sigma = \perp$ for all $j \in \{1, 2, 3, 4\}$ and all σ of length 1 and 2. For $\sigma = \epsilon$ we have $v^1_\epsilon = 0$, $v^2_\epsilon = 0$, $v^3_\epsilon = 1$, and $v^4_\epsilon = 1$.

We have $f = 1$, so the protocol has 2 rounds.

In round 1 each processor p_i sends v^i_ϵ to all other processors; except p_4 that crashes and sends arbitrary values. Let us write $?_i$ for the value it sends to processor p_i. In other words, processor

- p_1 sends $m_1^1 = m_1^2 = m_1^3 = m_1^4 = 0$,
- p_2 sends $m_2^1 = m_2^2 = m_2^3 = m_2^4 = 0$,
- p_3 sends $m_3^1 = m_3^2 = m_3^3 = m_3^4 = 1$, and
- p_4 sends $m_4^1 = ?_1, m_4^2 = ?_2, m_4^3 = ?_3, m_4^4 = ?_4$.

Then after round 1 processor

- p_1 has $v_1^1 = 0$, $v_2^1 = 0$, $v_3^1 = 1$ and $v_4^1 = ?_1$
- p_2 has $v_1^2 = 0$, $v_2^2 = 0$, $v_3^2 = 1$ and $v_4^2 = ?_2$.
- p_3 has $v_1^3 = 0$, $v_2^3 = 0$, $v_3^3 = 1$ and $v_4^3 = ?_3$.
- p_4 has $v_1^4 = 0$, $v_2^4 = 0$, $v_3^4 = 1$ and $v_4^4 = ?_4$.

Now in round 2 processor

- p_1 sends $m_{21}^1 = m_{21}^2 = m_{21}^3 = m_{21}^4 = v_2^1 = 0$,
  $m_{31}^1 = m_{31}^2 = m_{31}^3 = m_{31}^4 = v_3^1 = 1$, and
  $m_{41}^1 = m_{41}^2 = m_{41}^3 = m_{41}^4 = v_4^1 = ?_1$.
- p_2 sends $m_{12}^1 = m_{12}^2 = m_{12}^3 = m_{12}^4 = v_1^2 = 0$,
  $m_{32}^1 = m_{32}^2 = m_{32}^3 = m_{32}^4 = v_3^2 = 1$, and
  $m_{42}^1 = m_{42}^2 = m_{42}^3 = m_{42}^4 = v_4^2 = ?_2$.
- p_3 sends $m_{13}^1 = m_{13}^2 = m_{13}^3 = m_{13}^4 = v_1^3 = 0$,
  $m_{23}^1 = m_{23}^2 = m_{23}^3 = m_{23}^4 = v_2^3 = 0$, and
  $m_{43}^1 = m_{43}^2 = m_{43}^3 = m_{43}^4 = v_4^3 = ?_3$.
- p_4 sends $m_{14}^1 = m_{14}^2 = m_{14}^3 = m_{14}^4 = ?$,
  $m_{24}^1 = m_{24}^2 = m_{24}^3 = m_{24}^4 = ?$, and
  $m_{34}^1 = m_{34}^2 = m_{34}^3 = m_{34}^4 = ?$.

We don't care what p_4 sends here, so we just write ? for the value. But note that processors p_1, p_2 and p_3 are consistent with what they send as the value they received from processor p_4.

Then after round 2 processor

- p_1 has $v_1^1 = 0$, $v_2^1 = 0$, $v_3^1 = 1$ and $v_4^1 = ?$ as well as
  $v_{12}^1 = 0$, $v_{13}^1 = 0$, $v_{14}^1 = ?$, and
  $v_{21}^1 = 0$, $v_{23}^1 = 0$, $v_{24}^1 = ?$, and
  $v_{31}^1 = 1$, $v_{32}^1 = 1$, $v_{34}^1 = ?$, and
  $v_{41}^1 = ?_1$, $v_{42}^1 = ?_2$, $v_{43}^1 = ?_3$.

- p_2 has $v_1^2 = 0$, $v_2^2 = 0$, $v_3^2 = 1$ and $v_4^2 =?$ as well as
  $v_{12}^2 = 0$, $v_{13}^2 = 0$, $v_{14}^2 =?$, and
  $v_{21}^2 = 0$, $v_{23}^2 = 0$, $v_{24}^2 =?$, and
  $v_{31}^2 = 1$, $v_{32}^2 = 1$, $v_{34}^2 =?$, and
  $v_{41}^2 =?_1$, $v_{42}^2 =?_2$, $v_{43}^2 =?_3$.

- p_3 has $v_1^3 = 0$, $v_2^3 = 0$, $v_3^3 = 1$ and $v_4^3 =?$ as well as
  $v_{12}^3 = 0$, $v_{13}^3 = 0$, $v_{14}^3 =?$, and
  $v_{21}^3 = 0$, $v_{23}^3 = 0$, $v_{24}^3 =?$, and
  $v_{31}^3 = 1$, $v_{32}^3 = 1$, $v_{34}^3 =?$, and
  $v_{41}^3 =?_1$, $v_{42}^3 =?_2$, $v_{43}^3 =?_3$.

- p_4 has $v_1^4 = 0$, $v_2^4 = 0$, $v_3^4 = 1$ and $v_4^4 =?$ as well as
  $v_{12}^4 = 0$, $v_{13}^4 = 0$, $v_{14}^4 =?$, and
  $v_{21}^4 = 0$, $v_{23}^4 = 0$, $v_{24}^4 =?$, and
  $v_{31}^4 = 1$, $v_{32}^4 = 1$, $v_{34}^4 =?$, and
  $v_{41}^4 =?$, $v_{42}^4 =?$, $v_{43}^4 =?$.

Now each processor can (recursively) decide. Because $d_\sigma^p = v_\sigma^p$ for $|\sigma| = f + 1 = 2$ have

- p_1 has $d_{12}^1 = 0$, $d_{13}^1 = 0$, $d_{14}^1 =?$, and
  $d_{21}^1 = 0$, $d_{23}^1 = 0$, $d_{24}^1 =?$, and
  $d_{31}^1 = 1$, $d_{32}^1 = 1$, $d_{34}^1 =?$, and
  $d_{41}^1 =?_1$, $d_{42}^1 =?_2$, $d_{43}^1 =?_3$.

- p_2 has $d_{12}^2 = 0$, $d_{13}^2 = 0$, $d_{14}^2 =?$, and
  $d_{21}^2 = 0$, $d_{23}^2 = 0$, $d_{24}^2 =?$, and
  $d_{31}^2 = 1$, $d_{32}^2 = 1$, $d_{34}^2 =?$, and
  $d_{41}^2 =?_1$, $d_{42}^2 =?_2$, $d_{43}^2 =?_3$.

- p_3 has $d_{12}^3 = 0$, $d_{13}^3 = 0$, $d_{14}^3 =?$, and
  $d_{21}^3 = 0$, $d_{23}^3 = 0$, $d_{24}^3 =?$, and
  $d_{31}^3 = 1$, $d_{32}^3 = 1$, $d_{34}^3 =?$, and
  $d_{41}^3 =?_1$, $d_{42}^3 =?_2$, $d_{43}^3 =?_3$.

- p_4 has $d_{12}^4 = 0$, $d_{13}^4 = 0$, $d_{14}^4 =?$, and
  $d_{21}^4 = 0$, $d_{23}^4 = 0$, $d_{24}^4 =?$, and
  $d_{31}^4 = 1$, $d_{32}^4 = 1$, $d_{34}^4 =?$, and
  $d_{41}^4 =?$, $d_{42}^4 =?$, $d_{43}^4 =?$.

Using $d_\sigma^p = \mathit{Majority}(\{d_{\sigma;q}^p \mid q \notin \sigma\})$ we have (note that it does not matter whether ? equals 0 or 1!)

- p_1 has $d_1^1 = 0$, $d_2^1 = 0$, $d_3^1 = 1$, and $d_4^1 =?_{123}$.

- p_2 has $d_1^2 = 0$, $d_2^2 = 0$, $d_3^2 = 1$, and $d_4^2 =?_{123}$.

- p_3 has $d_1^3 = 0$, $d_2^3 = 0$, $d_3^3 = 1$, and $d_4^3 = ?_{123}$.
- p_4 has $d_1^4 = 0$, $d_2^4 = 0$, $d_3^4 = 1$, and $d_4^4 = ?$.

(where $?_{123} = \mathit{Majority}(?_1, ?_2, ?_3)$.)

Hence $d_\epsilon^1 = v$, $d_\epsilon^2 = v$, $d_\epsilon^3 = v$ and $d_\epsilon^4 = ?$. I.e. all correct processors agree on the value $v = \mathit{Majority}(0, 0, 1, ?_{123})$.

Question 3: The consensus algorithm for crash failures can be optimised to send a lot less messages: a processor stops forwarding messages as soon as it sent 2 different values to all other processors. Does the same optimisation also work for the protocol tolerating Byzantine failures?

Answer: This optimisation does *not* work in the case of Byzantine failures. The reason is that the decision rule on the crash failure case is fundamentally different from the Byzantine failure case.

In the crash failure case, processors decide on a default value as soon as their tree T contains two different values, i.e. if $|V_p| = 2$. In the Byzantine case, processors determine the decision value recursively. The decision at level k equals the majority over all decision values of the children at level $k + 1$. This majority depends on the actual values received, and not just on the number of different values.

Question 4: Suppose we would adapt the consensus protocol for *crash* failures by attaching signatures to all values and outgoing messages. We assume that all nodes know the correct public keys of all other nodes.

I.e. let $\sigma = p_1, \ldots, p_k$. Let us write $[m]_\sigma$ for $[\ldots[[m]_{p_1}]_{p_2}\ldots]_{p_k}$, i.e. the message m signed by all processors in σ in succession. Values v_σ^p in the tree are actually stored with their signature $[v_\sigma^p]_\sigma$ so that if node p sends $m_{\sigma;p}^q$ to node q it can actually send it with its signature $[v_\sigma^p]_p$. Messages with invalid signatures are rejected, of course.

This clearly makes it impossible for Byzantine processors to lie when forwarding messages to other processors. Does this mean that the protocol, modified this way, can tolerate an arbitrary number of Byzantine, instead of crash, failures?

Answer: No. The reason is that a faulty processor can still lie about its input value when broadcasting it to the other processors in the very first round. It can even send different values (with valid signatures!) to different processors in an attempt to confuse them even further.

Consider the following situation: suppose all processors hold the value 1, but there is one faulty processor that sends 0 (with valid signature!) to all other processors, then all processors will see two values in their tree, hence $|V_q| = 2$ and so they will all decide on the default value $v_{def} = 0$. This violates the validity condition that demands that if all inputs are equal, then the decision value equals that value. This can be fixed by having a slightly less strict validity condition that reads that "there is an execution in which 0

is decided, and there is an execution in which 1 is decided". This condition *is* satisfied by the protocol. (You could verify for yourself that the protocol with signatures outlined above does satisfy the agreement condition even under Byzantine failures.)