Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed Computing

Systems@ETH Zürich

Prof. T. Roscoe, Prof. R. Wattenhofer

Computer Systems

— Solution to Assignment 8 —

1 Synchronous Model

1.1 Synchronous Consensus in a Grid

a) We can establish consensus on the minimum value. In every round a node broadcasts the smallest value it has received so far to its neighbors. Since the longest distance between any two nodes on a grid is $w + h$, it will take at most $w + h$ rounds until every node learns the minimum value in the grid.

b) All nodes store from which nodes they already received their initial value. In the first round, every node sends its value to all of its neighbors. In all consecutive rounds, every node only forwards values: If it receives a tuple (u, x) containing the initial value x of a node u, it only forwards the tuple to all neighbors, if this is the first time the node hears the value from u. As soon as there is a round in which a node does not hear a new tuple, the node decides for the minimum of all received values and terminates.

c) Since the longest path from any node to any other node is an upper bound on the duration until a node receives a value, we know that the algorithm terminates after at most $w+h+1$ rounds. The nodes on the corners of the grid require exactly $w + h$ time until they learn the value of the corner on the opposite side. Therefore, the runtime is exactly $w + h + 1$ rounds.

d) One Byzantine node, placed next to a corner. In that case, the corner node has one Byzantine neighbor, and one correct neighbor. If the Byzantine node pretends to send messages in such a way, that it looks like a completely normal execution of the algorithm, but with wrong initial values, the corner node gets two different pictures, and cannot determine which one is right. Any algorithm must choose one of the nodes to be Byzantine and listen to the other, and since there is no good way to do that, any algorithm for the corner node will violate the agreement property in at least 50% of the executions.

1.2 Synchronous Consensus in a Grid - Crash Failures

a) This is the algorithm described in Exercise 1.1**b**).

The goal of the algorithm is that every node learns the initial values of all nodes. Each node stores the received values in a set “allValues”. Every round, all newly received information is forwarded to all neighbors, until no new information is received anymore. In the first round, every node sends its own value to its neighbors.

Correctness: Let us look at a particular node u. Note that all messages are transported on the shortest path to u. Hence, in round 1, u receives all messages from its direct neighbors.

In round 2, all values from nodes in distance 2. And so on. Hence, u receives a new value every round, until it received all values.

Termination: The longest path from any node to any other node is an upper bound on the duration until a node receives a value. We know that any node will have received all values by round l (see Correctness). Therefore, the algorithm terminates after at most $l + 1$ rounds. (The last new information could be received in round l, and in round $l + 1$ the node realizes that no new values will arrive and terminates.)

Algorithm 1 Simple Consensus in a Grid

```
1:  allValues = {(myId, myValue)}
2:  recv = {(myId, myValue)}
3:  for Round 1 to $\infty$ do
4:      Send values(recv) to all neighbors
5:      recv = receive tuples from neighbors
6:      remove all tuples from recv which are already in allValues
7:      if recv = $\emptyset$ then
8:          No new tuple received
9:          return minimum value in allValues
10:     else
11:         allValues = allValues $\cup$ recv
12:     end if
13: end for
```

b) In our example ($w = 7$,$h = 6$), the crashing nodes could be arranged as in Figure 1). The longest shortest path has the length of 32.

For every w and h it is possible to arrange the faulty nodes in a pattern as shown in Figure 2. In that case, a longest shortest path of $l \approx 2 \cdot (w + h)$ can be achieved. For our special case this only gives a length of $l = 25$.

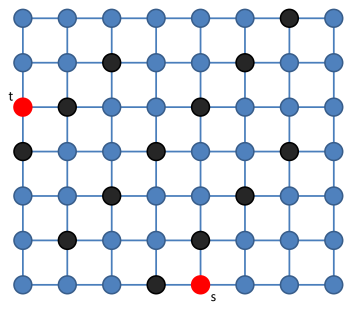


Figure 1: Strategy for $w = 7$,$h = 6$, with faulty nodes marked as black. The longest shortest path l leads from t to s.

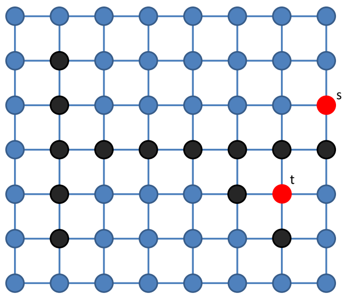
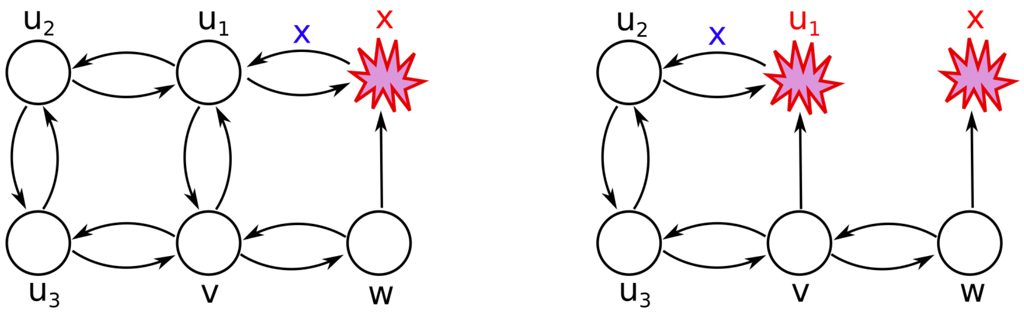


Figure 2: Strategy for any grid that achieves a longest shortest path of $l \approx 2 \cdot (w + h)$.

c) Consider a 1×2 grid with six nodes. We show that two nodes, v and w, in the bottom of the grid will terminate too early:



(a) In the first round of the execution we assume that node x crashes before forwarding its input value to the node w. Only node u_1 receives the value x.

(b) In the second round, the node u_1 crashes before forwarding its value to v. Again, only the node u_2 learns the value x.

By the end of the second round node v will have learned the values from the nodes u_1, u_2, u_3 and w. Since two nodes crashed during the execution of the algorithm, the value x will take the longest path from x to v on the grid, which takes exactly 4 rounds. In round 3, node v will not learn any new information and will therefore terminate too early. After node v terminates, node w becomes isolated and terminates within one round as well.

2 Asynchronous Model

2.1 What is the Average?

a) Assume we have a crash failure in the system. A node might crash before broadcasting its own input value. For the given input values any node that crashes would make other nodes decide on a value that is not 0.

b) In the worst case, two nodes with either the largest or the smallest input values will crash. We therefore expect the consensus value to be inside the interval $[-1, 1]$.

c) Note that a byzantine node is not restricted to send a value inside the interval $[-3, 3]$ as the correct nodes do. Since byzantine values can be arbitrarily small or large, the consensus value is expected to be unbounded.

d) A node could remove the largest and the smallest f values upon receiving them and compute the average of the remaining values.

e) The two boundary cases are: if byzantine nodes send values that are too large, a correct node will remove the byzantine values and the two smallest correct values; if the byzantine nodes send values that are too small, a correct node will remove the byzantine values and the two largest correct values. Therefore the approximations will be inside $[-1, 1]$.

f) A valid value is a value inside the interval

$$[\text{average without the largest } f \text{ correct values, average without the smallest } f \text{ correct values}].$$

g) The two boundary cases are: if byzantine nodes send values that are too large plus the two smallest correct values do not arrive at the node due to scheduling, a correct node will remove the byzantine values, the third and the fourth smallest correct values; if the byzantine nodes send values that are too small plus the two largest correct values do not arrive at the node due to scheduling, a correct node will remove the byzantine values, the third and the fourth largest correct values. Therefore the approximations will be inside $[-2, 2]$.

h) A valid value is a value inside the interval

$$[\text{average without the largest } 2f \text{ correct values, average without the smallest } 2f \text{ correct values}].$$

2.2 Computing the Average Synchronously

a) Algorithm 2 shows a possible implementation.

b) Since the byzantine nodes try to prevent other nodes from converging, they will choose their input values in such a way that the new input values are as far away from each other as possible. Assume that in each of the rounds the byzantine nodes send a value smaller than -3 to three correct nodes, a value larger than 3 to other three correct nodes, and no values at all to the remaining one correct node. Then, the new input values of the correct nodes after the first round are $\{-1, -1, -1, 0, 1, 1, 1\}$, after the second round $\{-2/5, -2/5, -2/5, 0, 2/5, 2/5, 2/5\}$ and after the third round $\{-4/25, -4/25, -4/25, 0, 4/25, 4/25, 4/25\}$.

Algorithm 2 Simple Synchronous Approximate Agreement

```
1: Lest $x_u$ be the input value of node $u$
2: repeat:
3: Broadcast $x_u$
4: $I :=$ all received values $x_v$ without the largest and the smallest $f$ values
5: Set $x_u := mean(I)$
```

2.3 Computing the Average Asynchronously

a) Algorithm 3 shows a possible implementation.

b) Byzantine strategy is similar to Question 2.2**b)**: to the first three correct nodes, byzantine nodes send values smaller than -3 and additionally delay the values 3 and 2 until the end of the round; to the second three correct nodes, byzantine nodes send values larger than 3 and additionally delay the values -3 and -2 until the current round is completed; the remaining correct node does not receive any byzantine values at all. Then, the new inputs after the end of the first round are $\{-2, -2, -2, 0, 2, 2, 2\}$. Since the intervals I of the nodes do not intersect in any value, the input values at the end of the second round remain the same. This way, the algorithm does not converge.

c) All local intervals I of the correct nodes can be shown to intersect in at least one value for $f < n/5$: from the $n - f$ correct values, the nodes can hide at most $2f$ too large or too small values. After the removal, the intervals should intersect, i.e. $(n - f) - 2f - 2f = n - 5f > 0$ should be satisfied.

d) In every round of the algorithm, all nodes will have at least one common value inside their intervals I. We need to show that the new input values of the correct nodes will span a smaller interval than the current input values. Consider therefore the extreme case where the local intervals of two nodes intersect in exactly one value. Let the smaller interval be spanned by the values v_1 and v_2, while the larger interval is spanned by v_2 and v_3, where $v_1 \leq v_2 \leq v_3$. The mean of the first interval is strictly larger than v_1, unless $v_1 = v_2$; while the mean of the second interval is strictly smaller than v_3, unless $v_2 = v_3$. If the values have not converged yet ($v_1 = v_2 = v_3$), then at least one of the two nodes will choose a new input value that is strictly inside the interval of its current values. This is true for any pair of correct nodes, which concludes the proof.

e) If FIFO broadcast is used instead of best-effort broadcast, byzantine nodes are prevented from sending different input values to different nodes. Therefore, the nodes see the same n values, unless the byzantine nodes use scheduling to hide f values. As before, byzantine nodes can make the correct nodes ignore the smallest and the largest $2f$ values from the interval. Since $n - 2f - 2f = n - 4f$, the FIFO broadcast improves the number of tolerated byzantine nodes to $f < n/4$.

Algorithm 3 Simple Asynchronous Approximate Agreement

```
1: Lest $x_u$ be the input value of node $u$
2: Let $r := 1$ denote the round
3: repeat:
4: Broadcast $(x_u, r)$
5: Wait until received $n - f$ messages of the form $(x_v, r)$
6: $I$ := all received values $x_v$ in round $r$ without the largest and the smallest $f$ values
7: Set $x_u := mean(I)$ and $r := r + 1$
```

3 Broadcast

3.1 Simplifying Reliable Broadcast?!

a) This algorithm does not implement reliable broadcast, as the totality property is violated.

Validity still holds. If a correct node v_s broadcasts a message x_s, every correct node will echo x_s. Since we have $n - f$ correct nodes by assumption, every node will eventually receive at least $n - f$ echo messages for x_s. Hence, every correct node will eventually accept x_s.

Integrity and *Agreement* are preserved as well. Assume for the sake of contradiction that some nodes v and u (where possibly $v = u$) are the first ones to accept $\texttt{msg}_{<i,v_s>}(x)$ and $\texttt{msg}_{<i,v_s>}(y)$, respectively, where $x \neq y$. Since v and u were the first ones to accept $\texttt{msg}_{<i,v_s>}(x)$ and $\texttt{msg}_{<i,v_s>}(y)$, respectively, they could not have accepted these messages as the result of receiving $f + 1$ ready messages. Hence, v has received $\texttt{echo}_{<i,v_s>}(x)$ from $n - f$ nodes, while u has received $\texttt{echo}_{<i,v_s>}(y)$ from $n - f$ nodes. This means that there are at least $(n - f) + (n - f) - n = n - 2f > f$ nodes, and hence at least one correct node, that echoed both x and y. This is a contradiction, because a correct node echoes only one value. Therefore, we conclude that there exists some x such that the correct nodes can only accept $\texttt{msg}_{<i,v_s>}(x)$.

Totality does not hold. To see why, consider a network with $n = 4$ nodes, of which $f = 1$ are byzantine. Let node 4 be the byzantine node. The byzantine node sends $msg_{<i,4>}(x)$ to nodes 1 and 2. Consequently, the correct nodes 1 and 2 both send $echo_{<i,4>}(x)$. Furthermore, the byzantine node sends $echo_{<i,4>}(x)$ to node 3. Node 3 thus receives $n - f = 3$ echo messages from distinct nodes, and accepts x. Node 1 and 2 will, however, never accept this message, as they will neither receive $n - f$ echo messages from distinct nodes, nor receive more than the $f = 1$ ready message sent by Node 3.

3.2 Broadcast With Erasure Coding

a) This algorithm can tolerate up to $f < n/2$ failures in this model. We first show why this is an upper bound, and then proceed with proving the correctness for $f < n/2$. Let $f \geq n/2$. In this case, the faulty nodes can crash before sending any message, which prevents correct nodes from ever accepting a message (since we only have $n - f < n/2$ correct nodes). Hence, the validity property does not hold.

We now show that this algorithm works for $f < n/2$.

Validity holds, because if the sender is correct, then $n - f$ correct nodes v_j receive the fragment f_j from the sender, and broadcast f_j. This ensures that every correct node receives $n - f \geq f + 1$ fragments, causing the node to accept x.

Totality also holds. Consider a node v that accepted $\texttt{msg}_{<i,v_s>}(m)$. This node must have received fragments from at least $f + 1$ other nodes that broadcast their fragments. Because we assume all-or-nothing broadcast, the other correct nodes will also eventually receive the fragments broadcast by these nodes, and thus also accept m.

Weak integrity is also preserved. It is easy to see that in the absence of byzantine nodes, if the correct v_s does not send and message in iteration i, then no correct node will accept any message in iteration i.

b) The totality property is violated. To see this, consider a case in which a node v accept $msg_{<i,v_s>}(m)$ as consequence of receiving $f + 1$ echoed fragments. Since we no longer require the broadcast operation to be all-or-nothing, the nodes that broadcast the $f + 1$ fragment could have failed while broadcasting, leading to some correct node u receiving only $1 < f + 1$ fragment. Hence, our proof of totality no longer works. Note that validity and weak integrity still hold under the crash failure model, as shown in a).

c) For this algorithm to work without all-or-nothing broadcast, we need to restore the totality property. To do so, we change the algorithm into Algorithm 4. The change is that now, a node accepts a message not when it has $|F| = f + 1$ fragments, but when it has $|F| = 2f + 1$ fragments. This means that we now need $n > 3f$, since otherwise $n - f < 2f + 1$, and the correct node never receive $2f + 1$ fragments if f nodes crash immediately. Suppose some correct node has accepted a message in Algorithm 4. By increasing the message accepting threshold to $2f + 1$ fragments, we have made sure that this node has accepted a message after receiving at least $(2f + 1) - f = f + 1$ fragments from correct nodes. Because these nodes are correct, eventually, all nodes receive $f + 1$ echoed fragments. Each correct node consequently broadcasts its fragment, which ensures that every correct node receives $n - f \geq 2f + 1$ correct fragments, and accepts a message.

Algorithm 4 Efficient Broadcast: Iteration i, Sender v_S. We use an $(n, f+1)$-erasure code.

```
Code for sender $v_S$ with input $x_S$:
$(f_1, \dots, f_n) := \texttt{get\_fragments}(x_S)$
for $j \in \{1, \dots, n\}$ do
   Send $\texttt{msg}_{<i,v_S>}(f_j)$ to $v_j$
end for

Code for node $v_j$:
$F := \{\}$
upon receiving $\texttt{msg}_{<i,v_S>}(f_j)$ from the sender $v_S$:
   if not broadcast $\texttt{echo}_{<i,v_S>}(f_j)$ before then
      Broadcast $\texttt{echo}_{<i,v_S>}(f_j)$
   end if
end upon

upon receiving $\texttt{echo}_{<i,v_S>}(f_k)$ from any node $v_k$:
   $F := F \cup \{(k, f_k)\}$
   if $|F| = f + 1$ then
      $m := \texttt{recover\_message}(F)$
      $(f_1, \dots, f_n) := \texttt{get\_fragments}(m)$
      if not broadcast $\texttt{echo}_{<i,v_S>}(f_j)$ before then
         Broadcast $\texttt{echo}_{<i,v_S>}(f_j)$
      end if
   end if
   if $|F| = 2f + 1$ then
      Accept $\texttt{msg}_{<i,v_S>}(m)$, where $m$ is the message recovered when $|F| = f + 1$.
   end if
end upon
```