Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed Computing

Systems@ETH Zürich

Autumn Term 2018 Prof. T. Roscoe, Prof. R. Wattenhofer

Computer Systems

— Solution to Assignment 5 —

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 ∞ 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 = ∅ then
8:        No new tuple received
9:        return minimum value in allValues
10:    else
11:       allValues = allValues ∪ recv
12:    end if
13: end for
```

b) If $w \approx h$ one can arrange the faulty nodes in two diagonals (see Figure 1). The longest shortest path has a length of $l \approx 3 \cdot (w+h)$. For the special case $w = 7, h = 6$, l has a length of 27.

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 $l = 25$.

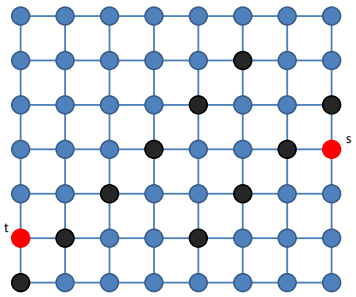


Figure 1: Strategy for $w \approx h$ with faulty nodes marked as black. The longest shortest path l leads from t to s

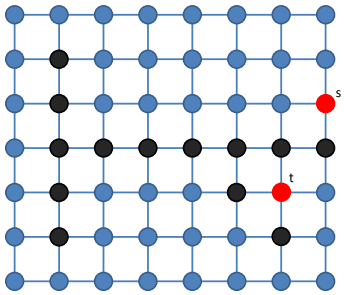
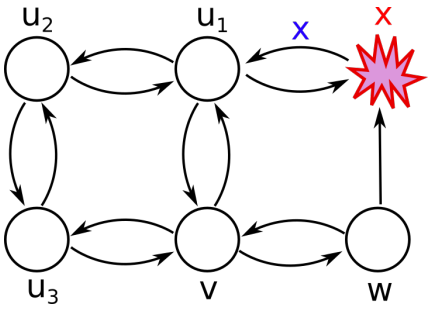
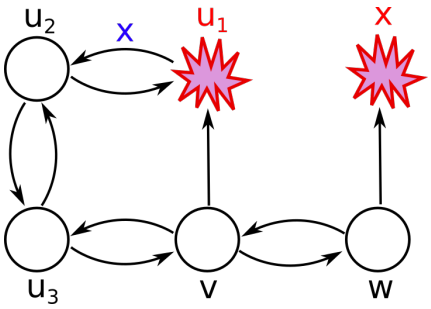


Figure 2: Strategy for any grid that achieves a longest shortest path of $l \approx 2 \cdot (w + h)$

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.

1.3 Consensus in a Grid... again!

a) Our algorithm is very simple: The node in the center is the leader, and it sends its value to all neighbors. All other neighbors relay the value to all neighbors and terminate.

Since w and h are even (recall that w, h denote the number of *edges* and not nodes) the center is well defined; and since the center knows its coordinates and w and h, it can initiate the broadcast. Analogous to exercises **1** and **2** we know that the runtime of the algorithm is the distance from the center to the farthest away node, which is the corner. Hence, the runtime is exactly $(w + h)/2$.

b) We study three different possible distributions of initial values:

All Zero All nodes start with initial value 0.

All One All nodes start with initial value 1.

Half-Half The bottom left half (including the "middle diagonal") starts with 0, the top right quarter (without the "middle diagonal") starts with 1.

The "middle diagonal" is defined by all nodes which have distance $(w + h)/2$ from the lower left corner, respectively the upper right corner. Without loss of generality, assume that $w \leq h$ (otherwise, simply turn the grid by 90°). The coordinates of these nodes are $(i, (w + h)/2 - i)$, for $i \in [0, w]$. The diagonal includes the center of the grid and is shown below.

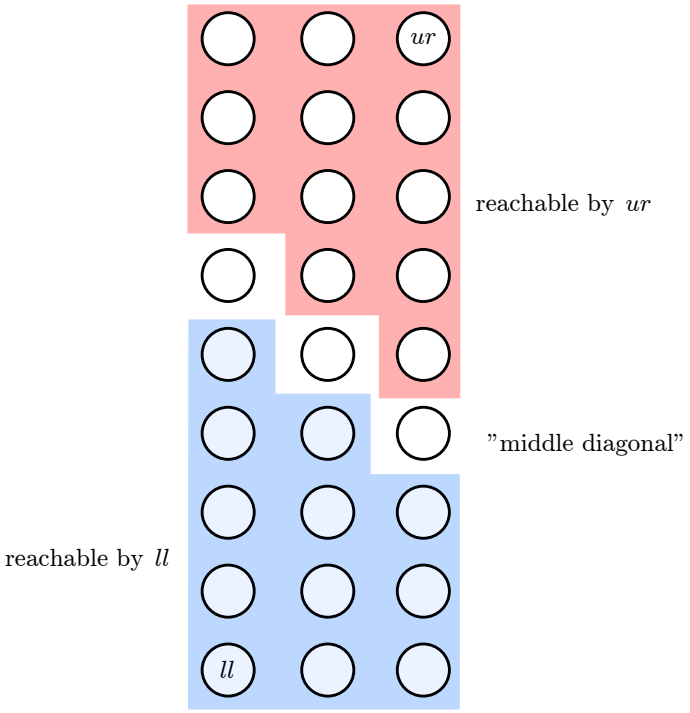


Figure 4: A grid with $w = 2$ and $h = 8$.

Assume that there is an algorithm which terminates after $(w+h)/2-1$ many rounds. Observe that after r rounds, any algorithm can only forward information up to a distance of r. Thus, neither the lower left corner, nor the upper left corner will receive any information from the diagonal or the nodes on the opposing side of the diagonal, since these nodes are at least in distance $(w + h)/2$.

Hence we study the information received by the lower left corner ll and the upper right corner ur in all three scenarios:

	ll	ur
All Zero	only 0 values	only 0 values
All One	only 1 values	only 1 values
Half-Half	only 0 values	only 1 values

Recall that the validity property of consensus requires the decision value to be one of the initial values. Thus, ll and ur must decide for 0 in the scenario "All Zero", and 1 in the scenario "All One". Since ll cannot distinguish "All Zero" and "Half-Half", and we require a deterministic algorithm, ll will decide for 0. However, with an analogous argument, ur cannot distinguish "All One" and "Half-Half" and must therefore decide for 1.

Hence, if the validity property is satisfied in both "All Zero" and "All One", the agreement property will be violated in the scenario "Half-Half". Thus, no deterministic algorithm with runtime less than $(w + h)/2$ can exist.

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 ny 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 $[-5, 5]$ 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

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

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

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

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. $(n - f) - 2f - 2f = n - 5f > 1$ 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 $r := r + 1$