Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed Computing

Systems@ETH Zürich

Prof. T. Roscoe, Prof. R. Wattenhofer

Computer Systems

Assignment 8

1 Synchronous Model

Quiz

1.1 Synchronous Consensus in a Grid

In the lecture you learned how to reach consensus in a fully connected network where every process can communicate directly with every other process. Now consider a network that is organized as a 2-dimensional grid such that every process has up to 4 neighbors. The width of the grid is w, the height is h. Width and height are defined in terms of edges: A 2×2 grid contains 9 nodes! The grid is big, meaning that $w + h$ is much smaller than $w \cdot h$. We use the synchronous time model; i.e., in every round every process may send a message to each of its neighbors, and the size of the message is not limited.

a) Assume every node knows w and h. Write a short protocol to reach consensus.

b) From now on the nodes do not know the size of the grid. Write a protocol to reach consensus and optimize it according to runtime.

c) How many rounds does your protocol from **b)** require?

Assume there are Byzantine nodes and that you are the adversary who can select which nodes are Byzantine.

d) What is the smallest number of Byzantine nodes that you need to prevent the system from reaching agreement, and where would you place them?

Basic

1.2 Synchronous Consensus in a Grid - Crash Failures

Consider the same network as in Question 1.1. Assume that some of the nodes crashed at the beginning of the algorithm such that any two correct processes are still connected through at least one path of correct processes.

Let l be the length of the longest shortest path between any pair of nodes in the grid; i.e., l is the number of edges between those two nodes which are "farthest away" from each other. If there are no failures, l is the distance between two corners, i.e. $l = w + h$.

a) Modify the algorithm from 1.1**b)** to solve consensus in $l + 1$ rounds with this special type of crash failures. Show that your algorithm works correctly; i.e., a node does not terminate before it learned the initial value of all nodes.

b) As an adversary you are allowed to crash up to $w + h$ many nodes at the beginning of the algorithm. Let $w = 7, h = 6$. What is the largest l you can achieve?

c) Assume that you run the algorithm with any type of crash failures; i.e, nodes can crash at any time during the execution. Show that with such failures the algorithm does not always work correctly anymore, by giving an execution and a failure pattern in which some nodes terminate too early!

2 Asynchronous Model

Quiz

2.1 What is the Average?

Assume that we are given 7 nodes with input values $\{-3, -2, -1, 0, 1, 2, 3\}$. The task of the nodes is to establish agreement on the average of these values. As always, our system might be faulty - nodes could crash or even be byzantine.

a) Show that in the presence of even one failure (crash or byzantine), the nodes cannot agree on the average of all input values.

Since we cannot establish agreement on the exact value, it would be great to understand how close we can get to the average value. Let us begin by only considering crash failures in the system. Assume that at most 2 of the 7 given nodes can crash.

b) In which range do you expect the consensus value to be?

From now on, we will consider byzantine failures as well. Assume that we have 9 nodes in total. 7 of these nodes are correct and have the input values specified above. The remaining two nodes are byzantine. We will start with a synchronous system.

c) Show that the consensus values can be basically anything now.

d) Suggest a rule that a node could use to locally choose a value as an approximation to the average.

e) What is the range of all possible local approximations of the average?

f) Suggest a validity condition that can be used to determine a consensus value.

Now assume that the system is asynchronous. Keep in mind that the scheduling is worst-case.

g) How does the range of all possible local approximations of the average change in this case?

h) Suggest a new validity condition that can be used to determine a consensus value.

Basic

2.2 Computing the Average Synchronously

In the lecture, we have focused on a class of algorithms which satisfy termination and agreement. In the following, we drop the termination condition and relax the agreement assumption:

Agreement The interval size of the input values of all correct nodes converges to 0.

a) Suggest a simple synchronous algorithm satisfying the agreement property defined above. Use the strategy from Question 2.1**d)**.

b) Apply your algorithm to the input from Question 2.1, again with 9 nodes in total, two of which are byzantine. Compute 3 iterations assuming that byzantine nodes try to prevent the nodes from converging.

Advanced

2.3 Computing the Average Asynchronously

Consider the algorithm you derived in Question 2.2 in an asynchronous system. Assume that each node broadcasts the current round together with the current input value.

a) Sketch your algorithm in the asynchronous setting.

b) Show that byzantine nodes can prevent this algorithm from converging if we apply it to the input sequence from Question 2.1.

c) What is the largest value of f for which your algorithm will work in the asynchronous system?

d) Show that with the chosen bounds on the number of byzantine nodes each of the algorithms will converge.

e) Explain how the bounds change in the asynchronous case if FIFO broadcast is used instead of best-effort broadcast or vice versa?

3 Broadcast

Basic

3.1 Simplifying Reliable Broadcast?!

In the script, you have seen Algorithm 18.11, which implements reliable broadcast in the asynchronous communication model for $n > 3f$ byzantine nodes. In this task, we attempt to simplify the algorithm. Concretely, we have removed the last three lines of the Algorithm 18.11, and now a node accepts $\texttt{msg}_{<i,v_S>}(x)$ immediately after broadcasting $\texttt{ready}_{<i,v_S>}(x)$.

Algorithm 1 Reliable Broadcast: Iteration i, Sender v_S

```
1:  Code for sender $v_S$ with input $x_S$:
2:  Send $\texttt{msg}_{<i,v_S>}(x_S)$ to everyone.
3:
4:  Code for node $v$:
5:  // Ignore any messages tagged with different values $i, v_S$.
6:  upon receiving $\texttt{msg}_{<i,v_S>}(x)$ from $v_S$:
7:     If no $\texttt{echo}_{<i,v_S>}$ message was sent in this instance:
8:          Send $\texttt{echo}_{<i,v_S>}(x)$ to everyone.
9:  end upon
10:
11: upon receiving $\texttt{echo}_{<i,v_S>}(x)$ from $n - f$ distinct nodes or
                   $\texttt{ready}_{<i,v_S>}(x)$ from $f + 1$ distinct nodes:
12:    Send $\texttt{ready}_{<i,v_S>}(x)$ to everyone.
13:    Accept $\texttt{msg}_{<i,v_S>}(x)$.
14: end upon
```

a) Does the algorithm still implement reliable broadcast? That is, does it still achieve validity, integrity, agreement and totality? For each property, either explain why it still holds, or provide a counterexample execution where it fails.

3.2 Broadcast With Erasure Coding

In this exercise, we are studying a simplified broadcast algorithm which, like the efficient reliable broadcast algorithms you have seen in the lecture, makes use of erasure coding to reduce communication complexity. Comparing this algorithm to the efficient reliable broadcast algorithms from the lecture, most notably, the Merkle Proofs are missing.

Unsurprisingly, this algorithm is not capable of providing reliable broadcast in a setting with byzantine nodes. Furthermore, one can show that this algorithm is also incapable of providing reliable broadcast in the crash failure model. We thus consider an even weaker model for this exercise. Nodes may still fail, but each broadcast operation adheres to the following model.

Definition 1 (Crash Failure Model With All-Or-Nothing Broadcast)**.** *In this model, nodes may still crash at arbitrary points in the execution of the algorithm (like in the Crash Failure Model), but we require broadcast operations to happen with all-or-nothing semantics. That is, if a node fails, it either fails before a broadcast operation, or after completing it.*

You might ask: In this model, why do we need anything other than best-effort broadcast? After all, if a sender just broadcasts its input by sending it to everyone, then this broadcast either succeeds completely or fails completely. The reason is that the sender could have a large input, and it could be too costly for it to just send this whole input to everyone.

Algorithm 2 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, \ldots, f_n) := $ get_fragments($x_S$)
for $j \in \{1, \ldots, n\}$ do
    Send msg$_{<i,v_S>}(f_j)$ to $v_j$
end for

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

upon receiving echo$_{<i,v_S>}(f_k)$ from any node $v_k$:
    $F := F \cup \{(k, f_k)\}$
    if $|F| = f + 1$ then
        $m := $ recover_message($F$)
        $(f_1, \ldots, f_n) := $ get_fragments($m$)
        if not broadcast echo$_{<i,v_S>}(f_j)$ before then
            Broadcast echo$_{<i,v_S>}(f_j)$
        end if
        Accept msg$_{<i,v_S>}(m)$.
    end if
end upon
```

a) In a network of n nodes, for which values of f does this broadcast algorithm achieve validity, totality and weak integrity against f crash faults under the crash failure model with all-or-nothing broadcast? Explain your answer.

b) Which of the broadcast properties (validity, totality and weak integrity) are violated when running this algorithm under the crash failure model without all-or-nothing broadcast?

c) Can we make this algorithm work without all-or-nothing broadcast in the crash failure model? If so, explain the necessary changes to the algorithm. How many crashes does your algorithm tolerate?