Assignment 3 – Paxos

**Team:**

Livar Cunha

Luciano Mandryk

Sergio Clemente

# Introduction

For this assignment we chose to build up on assignment 1 and improve the reliability of the Storage Server by using Paxos to coordinate a number of replicas and have them agree on a total order of execution for commands sent to any storage server node. The choice of improving the storage server instead of the facebook server was mostly arbitrary; we could just as easily have used the facebook server (or both) by changing a single line of code that delivers the chosen Paxos value for each slot to a particular component for execution.

# Compiling and Executing

Just run ./compile.sh or ./execute.sh

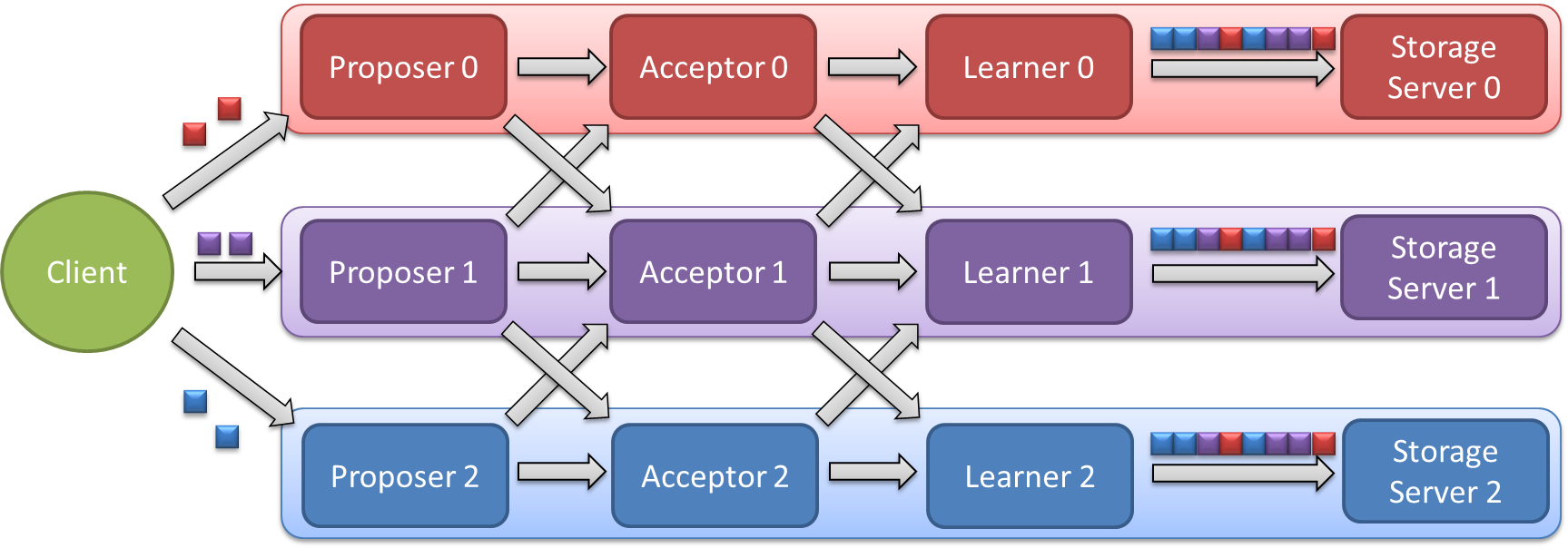
If you run into any issues make sure you set your java home to something like the following:

[~] JAVA\_HOME=/Library/Java/JavaVirtualMachines/1.7.0.jdk/Contents/Home/bin

[~] export JAVA\_HOME

Also, make sure the classes folder exist.

# High-Level Design



We chose to build a 3-node system (nodes 0, 1, 2) where each node implements the roles of proposer, acceptor, and learner, as well as the storage server. This is not a hard requirement, it was chosen for simplicity and debugability. The Paxos roles can be easily separated, mixed and matched across many different nodes by editing the address arrays at the top of src/node/paxos/PaxosNode.java.

The roles are what one would expect from Paxos, with nodes being able to send messages to all other nodes (some arrows are purposely missing in the diagram to keep it readable).

### Commands:

The client sends any number of commands to any of the proposers via the execute\_command command. For example:

0 execute\_command create login\_names.txt

Commands sent to the proposers are kept in a per-proposer queue. Since storage server commands may have dependencies (e.g. a command to append data to a file must not be reordered and executed before the command that created the file), in order to preserve the serialization, commands sent to the same proposer are negotiated (and therefore executed) in the order they are received (FIFO). We make no guarantees as to the order of execution of commands sent to different proposers. One consequence of this design decision is that each proposer will negotiate a slot for only one command at a time ( is 1), although every proposer may be proposing a value at the same time.

### Proposers:

Upon receiving a command, the proposer starts a Paxos round for the next available slot. Other proposers may be proposing other commands for the same slot, and eventually the algorithm reaches a consensus on which command should be associated with that particular slot. Along with the command, the Paxos value also contains the identity of the proposer, making the {proposer\_id,command} tuple unique even when multiple proposers are proposing the same command for the same slot. This also prevents a bug where if multiple proposers propose the same command at the same time, the command would be chosen and executed only once despite the intent of it being executed as many times as it was proposed by different proposers.

Once a value has been chosen for that particular slot, the proposer learns the chosen value by tracking the accept responses from the acceptors. If the chosen value was the command that it proposed, the proposer moves on to the next slot and proposes the next command in its queue, if there is any. If a different command was chosen for that slot, the proposer initiates a Paxos round for the next available slot, repeating this process as many times as necessary until its command is chosen.

If the proposal round fails (i.e. the proposer doesn’t receive enough responses by some timeout period or a majority of acceptors did not promise to accept the proposal), the proposer backs out for a random number of ticks before retrying the proposal. The goal with the random back out is to perturb the determinism of the messaging layer and allow one of the competing proposers to eventually win the round.

### Acceptors and Learners:

Acceptors and learners behave like any vanilla Paxos implementation. Upon accepting a value (obviously subject to the conditions imposed by the algorithm), the acceptor notifies all learns of the accepted value. The learners learn that a value has been chosen when it detects that a majority of acceptors accepted the same proposal.

Once a learner learns that a value has been chosen, it stores the chosen command in a data structure. If there are no gaps in the slots, the command is passed to the storage server component, which executes it. If there happened to be gaps, the learner waits until the gap is filled before passing the commands to the storage server component.

The proposers, acceptors and learners properly persist the necessary state to ensure that slot numbers, chosen values and accepted proposal numbers are never forgotten even in case of total failure. Additionally, learners persist their state before passing the commands to the storage server component in order to maintain that at-most-once semantics that was required in assignment 1.

Because of Paxos, all learners learn the same values for the same slots, so eventually all storage servers end up executing the same commands in the same other, therefore achieving the same state.

# Examples

Here we show a series of execution scenarios and their outcomes. For brevity, we only show the executed commands and the final result produced. The complete output of each scenario containing all the relevant message traffic and debug spew is provided in separate files (some are roughly 1Mb) and can be used to verify the correctness of the execution.

### Scenario 1: consensus over a single command

In this scenario we send a command to create a file called scenario1\_file.txt to one of the proposers. All nodes (proposers, acceptors, learners) are online and there is no competition for slots, so this is Paxos at its best case scenario. The expectation is that all learners will learn and execute the command. We use the dump\_values command to have each node display its view of the commands chosen for each slot.

The sequence of commands provided to the program is:

**# initialize the nodes**

**start 0**

**start 1**

**start 2**

**# tell node 0 to propose the command “create scenario1\_file.txt”**

**0 execute\_command create scenario1\_file.txt**

**# after consensus is reached, ask each node to dump the chosen values**

**0 dump\_values**

**1 dump\_values**

**2 dump\_values**

The relevant output snippets are shown below:

Time: 0

Please input a sequence of commands terminated by a blank line or the TIME command:

**start 0**

**start 1**

**start 2**

0: START 0

1: START 1

2: START 2

In what order should the events happen? (enter for in-order)

[...]

Time: 1

Please input a sequence of commands terminated by a blank line or the TIME command:

**0 execute\_command create scenario1\_file.txt**

Live nodes: 0, 1, 2

Crash which nodes? (space-delimited list of addresses or just press enter)

0: COMMAND 0 executes execute\_command create scenario1\_file.txt

[...]

Time: 21

Please input a sequence of commands terminated by a blank line or the TIME command:

**0 dump\_values**

**1 dump\_values**

**2 dump\_values**

Live nodes: 0, 1, 2

Crash which nodes? (space-delimited list of addresses or just press enter)

0: COMMAND 0 executes dump\_values

1: COMMAND 1 executes dump\_values

2: COMMAND 2 executes dump\_values

In what order should the events happen? (enter for in-order)

**\*\*\* 0: PAXOS: node 0: slot 0: 0.create scenario1\_file.txt**

**\*\*\* 1: PAXOS: node 1: slot 0: 0.create scenario1\_file.txt**

**\*\*\* 2: PAXOS: node 2: slot 0: 0.create scenario1\_file.txt**

Note in the output above that all nodes (0, 1, 2) reached consensus on the command to run for slot 0. The chosen command was “**create scenario1\_file.txt**”, which was proposed by proposer 0, as indicated by the “**0.**” prefix.

The following output shows that the command has been executed and the file has been created in the storage area of each node:

$ ls -FalR .

.:

total 4

drwxr-xr-x 5 Luciano Administ 0 Jun 6 01:00 ./

drwxr-xr-x 11 Luciano Administ 8192 Jun 6 01:00 ../

drwxr-xr-x 2 Luciano Administ 0 Jun 6 01:01 0/

drwxr-xr-x 2 Luciano Administ 0 Jun 6 01:01 1/

drwxr-xr-x 2 Luciano Administ 0 Jun 6 01:01 2/

./0:

total 1

drwxr-xr-x 2 Luciano Administ 0 Jun 6 01:01 ./

drwxr-xr-x 5 Luciano Administ 0 Jun 6 01:00 ../

-rw-r--r-- 1 Luciano Administ 290 Jun 6 01:00 acceptor\_state.txt

-rw-r--r-- 1 Luciano Administ 168 Jun 6 01:01 node.txt

-rw-r--r-- 1 Luciano Administ 0 Jun 6 01:01 scenario1\_file.txt

./1:

total 1

drwxr-xr-x 2 Luciano Administ 0 Jun 6 01:01 ./

drwxr-xr-x 5 Luciano Administ 0 Jun 6 01:00 ../

-rw-r--r-- 1 Luciano Administ 290 Jun 6 01:00 acceptor\_state.txt

-rw-r--r-- 1 Luciano Administ 168 Jun 6 01:01 node.txt

-rw-r--r-- 1 Luciano Administ 0 Jun 6 01:01 scenario1\_file.txt

./2:

total 1

drwxr-xr-x 2 Luciano Administ 0 Jun 6 01:01 ./

drwxr-xr-x 5 Luciano Administ 0 Jun 6 01:00 ../

-rw-r--r-- 1 Luciano Administ 290 Jun 6 01:00 acceptor\_state.txt

-rw-r--r-- 1 Luciano Administ 168 Jun 6 01:01 node.txt

-rw-r--r-- 1 Luciano Administ 0 Jun 6 01:01 scenario1\_file.txt

The full output of this scenario execution is included in the file scenario1.txt, located in the same directory as this document.

### Scenario 2: multiple commands issued to multiple proposers simultaneously

This scenario sends 4 commands simultaneously to each proposer, asking the storage servers to create a file and append 3 messages to each file. The expectation is that the proposers will agree on a total order of execution for all commands, preserve the ordering of the commands sent to the same proposer, have every learner learn the same total order for all commands and consequently have each storage server create all 3 files, each file with its 3 messages in the proper order.

The sequence of commands provided to the program is:

# start all nodes

**start 0**

**start 1**

**start 2**

# ask node 0 to create a file and append 3 messages

**0 execute\_command create scenario2\_file0.txt**

**0 execute\_command append scenario2\_file0.txt first message from proposer 0**

**0 execute\_command append scenario2\_file0.txt second message from proposer 0**

**0 execute\_command append scenario2\_file0.txt third message from proposer 0**

# ask node 0 to create a file and append 3 messages

**1 execute\_command create scenario2\_file1.txt**

**1 execute\_command append scenario2\_file1.txt first message from proposer 1**

**1 execute\_command append scenario2\_file1.txt second message from proposer 1**

**1 execute\_command append scenario2\_file1.txt third message from proposer 1**

# ask node 0 to create a file and append 3 messages

**2 execute\_command create scenario2\_file2.txt**

**2 execute\_command append scenario2\_file2.txt first message from proposer 2**

**2 execute\_command append scenario2\_file2.txt second message from proposer 2**

**2 execute\_command append scenario2\_file2.txt third message from proposer 2**

# after **consensus is reached, ask each node to dump the chosen values**

**0 dump\_values**

**1 dump\_values**

**2 dump\_values**

The relevant output snippets are shown below. After ~100 ticks, the nodes reach consensus on the order of execution of all commands.

Time: 0

Please input a sequence of commands terminated by a blank line or the TIME command:

**start 0**

**start 1**

**start 2**

0 execute\_command create scenario2\_file0.txt

0 execute\_command append scenario2\_file0.txt first message from proposer 0

0 execute\_command append scenario2\_file0.txt second message from proposer 0

0 execute\_command append scenario2\_file0.txt third message from proposer 0

1 execute\_command create scenario2\_file1.txt

1 execute\_command append scenario2\_file1.txt first message from proposer 1

1 execute\_command append scenario2\_file1.txt second message from proposer 1

1 execute\_command append scenario2\_file1.txt third message from proposer 1

2 execute\_command create scenario2\_file2.txt

2 execute\_command append scenario2\_file2.txt first message from proposer 2

2 execute\_command append scenario2\_file2.txt second message from proposer 2

2 execute\_command append scenario2\_file2.txt third message from proposer 2

0: START 0

1: START 1

2: START 2

3: COMMAND 0 executes execute\_command create scenario2\_file0.txt

4: COMMAND 0 executes execute\_command append scenario2\_file0.txt first message from proposer 0

5: COMMAND 0 executes execute\_command append scenario2\_file0.txt second message from proposer 0

6: COMMAND 0 executes execute\_command append scenario2\_file0.txt third message from proposer 0

7: COMMAND 1 executes execute\_command create scenario2\_file1.txt

8: COMMAND 1 executes execute\_command append scenario2\_file1.txt first message from proposer 1

9: COMMAND 1 executes execute\_command append scenario2\_file1.txt second message from proposer 1

10: COMMAND 1 executes execute\_command append scenario2\_file1.txt third message from proposer 1

11: COMMAND 2 executes execute\_command create scenario2\_file2.txt

12: COMMAND 2 executes execute\_command append scenario2\_file2.txt first message from proposer 2

13: COMMAND 2 executes execute\_command append scenario2\_file2.txt second message from proposer 2

14: COMMAND 2 executes execute\_command append scenario2\_file2.txt third message from proposer 2

In what order should the events happen? (enter for in-order)

[...]

Time: 100

Please input a sequence of commands terminated by a blank line or the TIME command:

**0 dump\_values**

**1 dump\_values**

**2 dump\_values**

Live nodes: 0, 1, 2

Crash which nodes? (space-delimited list of addresses or just press enter)

0: COMMAND 0 executes dump\_values

1: COMMAND 1 executes dump\_values

2: COMMAND 2 executes dump\_values

In what order should the events happen? (enter for in-order)

**\*\*\* 0: PAXOS: node 0: slot 0: 2.create scenario2\_file2.txt**

**\*\*\* 0: PAXOS: node 0: slot 1: 2.append scenario2\_file2.txt first message from proposer 2**

**\*\*\* 0: PAXOS: node 0: slot 2: 2.append scenario2\_file2.txt second message from proposer 2**

**\*\*\* 0: PAXOS: node 0: slot 3: 2.append scenario2\_file2.txt third message from proposer 2**

**\*\*\* 0: PAXOS: node 0: slot 4: 1.create scenario2\_file1.txt**

**\*\*\* 0: PAXOS: node 0: slot 5: 1.append scenario2\_file1.txt first message from proposer 1**

**\*\*\* 0: PAXOS: node 0: slot 6: 1.append scenario2\_file1.txt second message from proposer 1**

**\*\*\* 0: PAXOS: node 0: slot 7: 1.append scenario2\_file1.txt third message from proposer 1**

**\*\*\* 0: PAXOS: node 0: slot 8: 0.create scenario2\_file0.txt**

**\*\*\* 0: PAXOS: node 0: slot 9: 0.append scenario2\_file0.txt first message from proposer 0**

**\*\*\* 0: PAXOS: node 0: slot 10: 0.append scenario2\_file0.txt second message from proposer 0**

**\*\*\* 0: PAXOS: node 0: slot 11: 0.append scenario2\_file0.txt third message from proposer 0**

**\*\*\* 1: PAXOS: node 1: slot 0: 2.create scenario2\_file2.txt**

**\*\*\* 1: PAXOS: node 1: slot 1: 2.append scenario2\_file2.txt first message from proposer 2**

**\*\*\* 1: PAXOS: node 1: slot 2: 2.append scenario2\_file2.txt second message from proposer 2**

**\*\*\* 1: PAXOS: node 1: slot 3: 2.append scenario2\_file2.txt third message from proposer 2**

**\*\*\* 1: PAXOS: node 1: slot 4: 1.create scenario2\_file1.txt**

**\*\*\* 1: PAXOS: node 1: slot 5: 1.append scenario2\_file1.txt first message from proposer 1**

**\*\*\* 1: PAXOS: node 1: slot 6: 1.append scenario2\_file1.txt second message from proposer 1**

**\*\*\* 1: PAXOS: node 1: slot 7: 1.append scenario2\_file1.txt third message from proposer 1**

**\*\*\* 1: PAXOS: node 1: slot 8: 0.create scenario2\_file0.txt**

**\*\*\* 1: PAXOS: node 1: slot 9: 0.append scenario2\_file0.txt first message from proposer 0**

**\*\*\* 1: PAXOS: node 1: slot 10: 0.append scenario2\_file0.txt second message from proposer 0**

**\*\*\* 1: PAXOS: node 1: slot 11: 0.append scenario2\_file0.txt third message from proposer 0**

**\*\*\* 2: PAXOS: node 2: slot 0: 2.create scenario2\_file2.txt**

**\*\*\* 2: PAXOS: node 2: slot 1: 2.append scenario2\_file2.txt first message from proposer 2**

**\*\*\* 2: PAXOS: node 2: slot 2: 2.append scenario2\_file2.txt second message from proposer 2**

**\*\*\* 2: PAXOS: node 2: slot 3: 2.append scenario2\_file2.txt third message from proposer 2**

**\*\*\* 2: PAXOS: node 2: slot 4: 1.create scenario2\_file1.txt**

**\*\*\* 2: PAXOS: node 2: slot 5: 1.append scenario2\_file1.txt first message from proposer 1**

**\*\*\* 2: PAXOS: node 2: slot 6: 1.append scenario2\_file1.txt second message from proposer 1**

**\*\*\* 2: PAXOS: node 2: slot 7: 1.append scenario2\_file1.txt third message from proposer 1**

**\*\*\* 2: PAXOS: node 2: slot 8: 0.create scenario2\_file0.txt**

**\*\*\* 2: PAXOS: node 2: slot 9: 0.append scenario2\_file0.txt first message from proposer 0**

**\*\*\* 2: PAXOS: node 2: slot 10: 0.append scenario2\_file0.txt second message from proposer 0**

**\*\*\* 2: PAXOS: node 2: slot 11: 0.append scenario2\_file0.txt third message from proposer 0**

As we can see, all nodes agreed on the same order of execution. Given that all proposers start competing for the same slot at the same time and message delivery is deterministic, since proposer 2 has a slightly higher proposal number then all others, it has the advantage and ended up having its proposed value chosen for the first slot. That allowed proposer 2 to move ahead to the next command and propose it for the next slot before the other proposers, thus winning the subsequent slots.

The list of files produced by each storage server and their contents is shown below:

$ l -FalR

.:

total 10

drwxr-xr-x 5 Luciano Administ 0 Jun 6 12:10 ./

drwxr-xr-x 12 Luciano Administ 8192 Jun 6 12:10 ../

drwxr-xr-x 2 Luciano Administ 4096 Jun 6 12:12 0/

drwxr-xr-x 2 Luciano Administ 4096 Jun 6 12:12 1/

drwxr-xr-x 2 Luciano Administ 4096 Jun 6 12:12 2/

./0:

total 6

drwxr-xr-x 2 Luciano Administ 4096 Jun 6 12:12 ./

drwxr-xr-x 5 Luciano Administ 0 Jun 6 12:10 ../

-rw-r--r-- 1 Luciano Administ 2577 Jun 6 12:12 acceptor\_state.txt

-rw-r--r-- 1 Luciano Administ 1490 Jun 6 12:12 node.txt

-rw-r--r-- 1 Luciano Administ 151 Jun 6 12:12 scenario2\_file0.txt

-rw-r--r-- 1 Luciano Administ 151 Jun 6 12:11 scenario2\_file1.txt

-rw-r--r-- 1 Luciano Administ 151 Jun 6 12:10 scenario2\_file2.txt

./1:

total 6

drwxr-xr-x 2 Luciano Administ 4096 Jun 6 12:12 ./

drwxr-xr-x 5 Luciano Administ 0 Jun 6 12:10 ../

-rw-r--r-- 1 Luciano Administ 2577 Jun 6 12:12 acceptor\_state.txt

-rw-r--r-- 1 Luciano Administ 1490 Jun 6 12:12 node.txt

-rw-r--r-- 1 Luciano Administ 151 Jun 6 12:12 scenario2\_file0.txt

-rw-r--r-- 1 Luciano Administ 151 Jun 6 12:11 scenario2\_file1.txt

-rw-r--r-- 1 Luciano Administ 151 Jun 6 12:10 scenario2\_file2.txt

./2:

total 6

drwxr-xr-x 2 Luciano Administ 4096 Jun 6 12:12 ./

drwxr-xr-x 5 Luciano Administ 0 Jun 6 12:10 ../

-rw-r--r-- 1 Luciano Administ 2577 Jun 6 12:12 acceptor\_state.txt

-rw-r--r-- 1 Luciano Administ 1490 Jun 6 12:12 node.txt

-rw-r--r-- 1 Luciano Administ 151 Jun 6 12:12 scenario2\_file0.txt

-rw-r--r-- 1 Luciano Administ 151 Jun 6 12:11 scenario2\_file1.txt

-rw-r--r-- 1 Luciano Administ 151 Jun 6 12:10 scenario2\_file2.txt

$ cat 0/scenario2\_file0.txt

scenario2\_file0.txt first message from proposer 0

scenario2\_file0.txt second message from proposer 0

scenario2\_file0.txt third message from proposer 0

$ cat 0/scenario2\_file1.txt

scenario2\_file1.txt first message from proposer 1

scenario2\_file1.txt second message from proposer 1

scenario2\_file1.txt third message from proposer 1

$ cat 0/scenario2\_file2.txt

scenario2\_file2.txt first message from proposer 2

scenario2\_file2.txt second message from proposer 2

scenario2\_file2.txt third message from proposer 2

$ cat 1/scenario2\_file0.txt

scenario2\_file0.txt first message from proposer 0

scenario2\_file0.txt second message from proposer 0

scenario2\_file0.txt third message from proposer 0

$ cat 1/scenario2\_file1.txt

scenario2\_file1.txt first message from proposer 1

scenario2\_file1.txt second message from proposer 1

scenario2\_file1.txt third message from proposer 1

$ cat 1/scenario2\_file2.txt

scenario2\_file2.txt first message from proposer 2

scenario2\_file2.txt second message from proposer 2

scenario2\_file2.txt third message from proposer 2

$ cat 2/scenario2\_file0.txt

scenario2\_file0.txt first message from proposer 0

scenario2\_file0.txt second message from proposer 0

scenario2\_file0.txt third message from proposer 0

$ cat 2/scenario2\_file1.txt

scenario2\_file1.txt first message from proposer 1

scenario2\_file1.txt second message from proposer 1

scenario2\_file1.txt third message from proposer 1

$ cat 2/scenario2\_file2.txt

scenario2\_file2.txt first message from proposer 2

scenario2\_file2.txt second message from proposer 2

scenario2\_file2.txt third message from proposer 2

### Scenario 3: failures and recovery

This scenario is similar to scenario 2, except that it starts with one of the nodes down (node 1). Since a majority of nodes (0 and 2) is still up, those nodes agree on the total order of execution for the commands sent to them. We then crash all living nodes, so that their memory state is lost, start them again so that they recover from the persisted state, then we issue commands to node 1 – the node that was down all the time and therefore didn’t see any previously chosen value – to show that node 1 successfully learns and executes all previously chosen values and properly drives consensus for its commands across all nodes for the subsequent slots.