2018

Distributed Algorithms Project

RÉMI BOUKHELOUA
SIMON CHANG

Table of contents

I. High level description of the system

The main goal of this system is to design a fault-tolerant distributed system. The system is constituted of N processes. Every process stores its own key-value store and can communicate with all process (including himself).

1. What does a process contain?

First, a process is composed of:

- Its id to differentiate it from others
- A list of every process' reference to communicate with them (mem)
- A state showing if he is available, faulty or waiting
- A variable showing what operation it is performing: none, get or put (activeop)
- A HashMap storing every value of the key-value store (dict)
- A variable recording the number of operations performed so far
- M the number of operations to do
- Two buffers to store every response the process receives

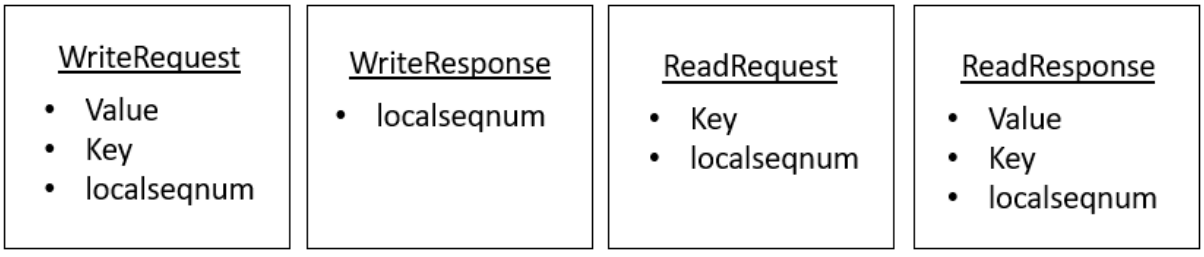
A process has also two different functions defined:

- put(k, v) which put the value v at the key k in the HashMap of all processes
- get(k) which gets the most recent at the case k stored over all processes

Finally, the process has some listeners that make him perform different actions regarding what type of message it receives.

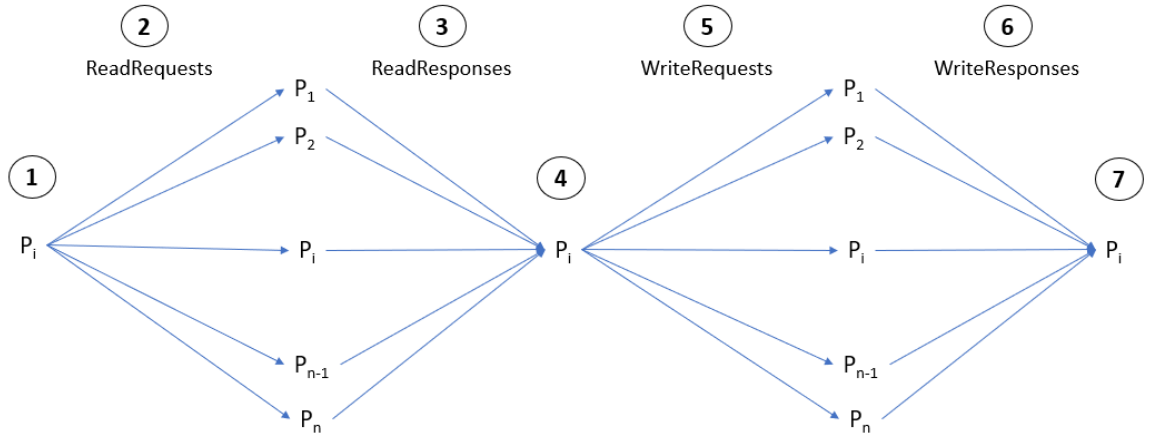
2. What are the different types of messages going through the system?

We have four types of messages going through the system:



We will see later how we use them to pass information.
Additionally, processes can receive Member that is the list of all process' reference.

3. How does it work?



How put and get are performed ?

a. How is put performed?

Step 1: First, P_i receives the (k, v) tuple. It puts the value in the HashMap at the key k with the current timestamp in the case (aka seqnum).

Step 2: P_i sends ReadRequests to every process. It asks everyone to give them its value at a given case (key k).

Step 3: Every process sends back the value they have at case k.

Step 4: Once P_i has received a majority of responses, it goes through every process' response to get the highest timestamp ht (seqnum) stored in the case overall the HashMap. P_i changes the seqnum into the value ht + 1.

Step 5: P_i sends a WriteRequest to every process sending them its new case.

Step 6: Every process sends back WriteResponse to P_i saying they have received the request.

Step 7: Once P_i has received a majority of responses from all processes, it moves to its next operation.

b. How is get performed?

Step 1: First, P_i receives the key k of value it has to load

Step 2: P_i sends ReadRequests to every process. It asks everyone to give them its value at a given case (key k).

Step 3: Every process sends back the value they have at case k.

Step 4: Once P_i has received a majority of responses, it goes through every process' response to get the value with the highest timestamp stored in the case overall the HashMap. P_i changes its value to this value.

Step 5: P_i sends a WriteRequest to every process sending them its "new" case. It might not be a new value if it has already the highest value.

Step 6: Every process sends back WriteResponse to P_i saying they have received the request.

Step 7: Once P_i has received a majority of responses from all processes, it moves to its next operation.

II. A pseudocode of the implementation

Here you can find a pseudo code of functions we used:

```
get()
    localseqnum++
(2) send ReadRequest[k, localseqnum] to all
(3) wait until received a majority of ReadResponse[(value, seqnum, id), K, localseqnum]
(4) (v, seqnum, p) = (v, seqnum, p) with the highest seqnum in ReadRequests
(5) send WriteRequest[(v, seqnum, p), k,  localseqnum]
    wait until received a majority of WriteResponse[localseqnum]
    return

put()
    localseqnum++
(2) send ReadRequest[k, localseqnum] to all
(3) wait until received a majority of ReadResponse[Value, k, localseqnum]
(4) seqnum = highest seqnum in ReadResponse
(5) send WriteRequest[(value, seqnum, id), k,  localseqnum]
    wait until received a majority of WriteResponse[localseqnum]
    return

On receive of WriteRequest [(value', seqnum', id'), k',  localseqnum']
    if (seqnum', id') >> (seqnum, id)
        (value, seqnum, id) = (value', seqnum', id') at key k'
Between (5) and (6)
    send back WriteResponse[Value, k, localseqnum]
    return

On receive of ReadRequest[k, localseqnum']
Between (2) and (3)
    send back ReadResponse[(value, seqnum, id), k, localseqnum']
```

Additional fact: we kept in mind the idea of a key-value storage, so we decided to have a HashMap instead of a single register.

III. Linearizability

This is a linearizability of the instance N = 3 and M = 3

Let 3 processes P0, P1 and P2 concurrently execute 3 operations each, one at a time, with P1 being faulty. The following output describes the order in which processes invokes a new operation (both put and get), and when it finishes.

When an operation finishes, it shows the value in the register at the time it finishes, making it easier to linearize it as it helps placing the linearization points.

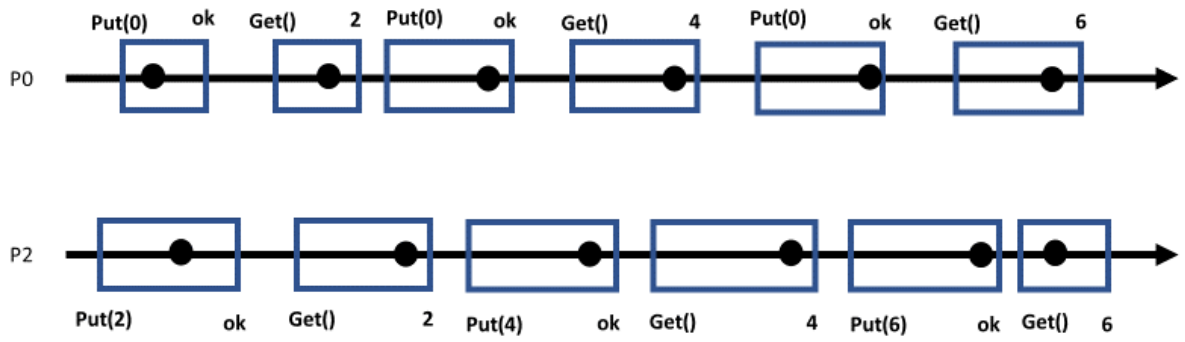
```
cher-3] [akka://system/user/P1] P1 is faulty
cher-2] [akka://system/user/P2] P2: invokes put operation 1 replacing 0 by 2
cher-4] [akka://system/user/P0] P0: invokes put operation 1 replacing 0 by 0
cher-4] [akka://system/user/P0] P0: updated the local value with value (2,1,2)
cher-5] [akka://system/user/P2] P2: updated the local value with value (2,1,2)
cher-4] [akka://system/user/P0]               P0: completes PUT operation 1, value = 2
cher-5] [akka://system/user/P2]               P2: completes PUT operation 1, value = 2
cher-4] [akka://system/user/P0] P0: invokes get operation 2
cher-5] [akka://system/user/P2] P2: invokes get operation 2
cher-2] [akka://system/user/P0]               P0: completes GET operation 2, value = 2
cher-2] [akka://system/user/P0] P0: invokes put operation 3 replacing 2 by 0
cher-7] [akka://system/user/P2]               P2: completes GET operation 2, value = 2
cher-7] [akka://system/user/P2] P2: invokes put operation 3 replacing 2 by 4
cher-2] [akka://system/user/P0] P0: updated the local value with value (0,2,0)
cher-7] [akka://system/user/P2] P2: updated the local value with value (0,2,0)
cher-9] [akka://system/user/P0]               P0: completes PUT operation 3, value = 0
cher-9] [akka://system/user/P0] P0: invokes get operation 4
cher-3] [akka://system/user/P2] P2: updated the local value with value (4,3,2)
cher-9] [akka://system/user/P0] P0: updated the local value with value (4,3,2)
cher-3] [akka://system/user/P2]               P2: completes PUT operation 3, value = 4
cher-3] [akka://system/user/P2] P2: invokes get operation 4
cher-8] [akka://system/user/P0]               P0: completes GET operation 4, value = 4
cher-8] [akka://system/user/P0] P0: invokes put operation 5 replacing 4 by 0
cher-3] [akka://system/user/P2]               P2: completes GET operation 4, value = 4
cher-3] [akka://system/user/P2] P2: invokes put operation 5 replacing 4 by 6
cher-9] [akka://system/user/P0] P0: updated the local value with value (0,4,0)
cher-3] [akka://system/user/P2] P2: updated the local value with value (0,4,0)
cher-9] [akka://system/user/P0]               P0: completes PUT operation 5, value = 0
cher-9] [akka://system/user/P0] P0: invokes get operation 6
cher-3] [akka://system/user/P2] P2: updated the local value with value (6,5,2)
cher-9] [akka://system/user/P0] P0: updated the local value with value (6,5,2)
cher-3] [akka://system/user/P2]               P2: completes PUT operation 5, value = 6
cher-3] [akka://system/user/P2] P2: invokes get operation 6
cher-9] [akka://system/user/P0]               P0: completes GET operation 6, value = 6
cher-9] [akka://system/user/P0] Process 0 execution time: 0.019 second(s).
cher-9] [akka://system/user/P2]               P2: completes GET operation 6, value = 6
cher-9] [akka://system/user/P2] Process 2 execution time: 0.019 second(s).
```

History: P2-Put(2); P0-Put(0); P0-ok; P2-ok; P0-Get(); P2-Get(); P0-2; P0-Put(0); P2-2;

P2-Put(4); P0-ok; P0-Get(); P2-ok; P2-Get(); P0-4; P0-Put(0); P2-4; P2-Put(6); P0-ok;

P0-Get(); P2-ok; P2-Get(); P0-6; P2-6;

We get the following schema:

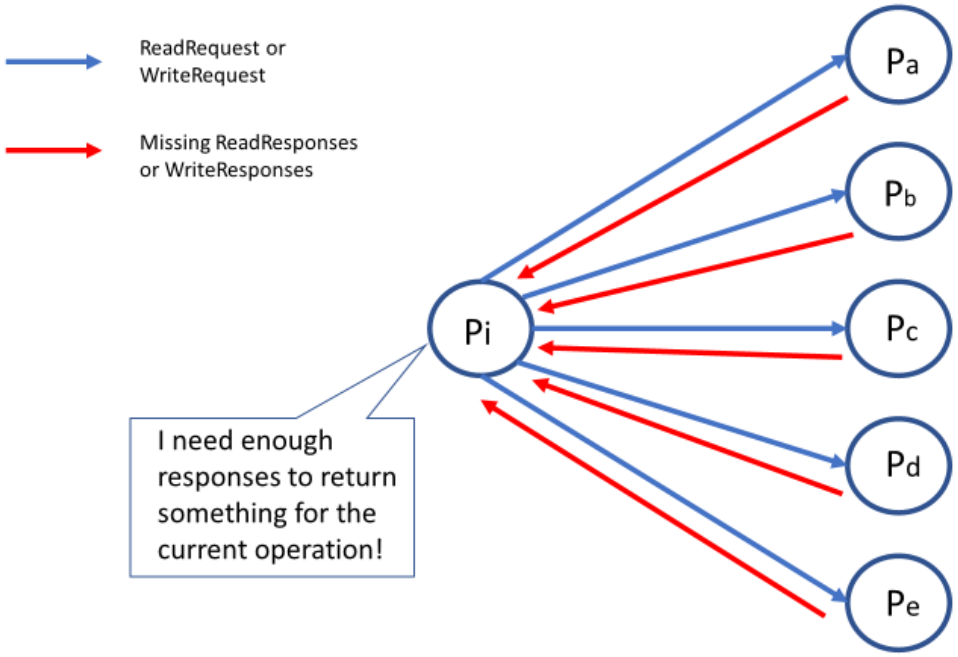


The resulting execution is linearizable.

IV. Liveness and Safety

We assume that we will always have a majority of non-faulty process.

1. Liveness: Can an operation invoked by a correct process never return?



This is the only situation where a process might not return anything for a given operation, but is it possible to stay in this situation indefinitely?
The answer is no since during the time the process "waits" for responses, it is not in a busy waiting state and it can still respond to other requests. If they could not, we could have faced deadlock phases where every process wait for an answer. As such, every process can respond to requests while they are waiting responses to their requests. So, the longest time a process P could wait is the time where processes answer first the requests of all other processes. After all those responses, the processes will have the P request in their buffer in first position, so they will answer it.

So, a process will never wait forever in the system.

2. Safety: The corresponding history is linearizable with respect to the sequential specification above

Safety makes sure the operations never return anything incorrect.
During a get, the process will request the values of everyone and pick among them the one with the highest timestamp. So, the only way the process has not the correct value is if a process is executing a put right after the process collected the value.

During a put, the process will request the values of everyone and pick the highest timestamp among them. As it has the last value (the one it wants to write), he can't have the wrong value.

Otherwise, the process receives request to change its value. If the received value is older than what the receiver gets (lower timestamp), it will not update. If the received value is more recent (timestamp higher), it will update it.

So, the process always has the correct value, safety is guarantee.

V. Performance analysis

We will make a performance analysis based on the total computation time, which in our system is **the execution time of the last finishing process**.

N = 3 and M = 3

```
system/user/P2] P2: invokes put operation 5 replacing 4 by 6
system/user/P0] P0: updated the local value with value (0,4,0)
system/user/P2] P2: updated the local value with value (0,4,0)
system/user/P0]                P0: completes PUT operation 5, value = 0
system/user/P0] P0: invokes get operation 6
system/user/P2] P2: updated the local value with value (6,5,2)
system/user/P0] P0: updated the local value with value (6,5,2)
system/user/P2]                P2: completes PUT operation 5, value = 6
system/user/P2] P2: invokes get operation 6
system/user/P0]                P0: completes GET operation 6, value = 6
system/user/P0] Process 0 execution time: 0.019 second(s).
system/user/P2]                P2: completes GET operation 6, value = 6
system/user/P2] Process 2 execution time: 0.019 second(s).
```

The total computation time for 3 processes and 3 operations is 0.019 seconds.

N = 3 and M = 10

```
m/user/P2] P2: updated the local value with value (0,12,0)
m/user/P0]                P0: completes PUT operation 19, value = 0
m/user/P0] P0: invokes get operation 20
m/user/P2] P2: updated the local value with value (20,13,2)
m/user/P0] P0: updated the local value with value (20,13,2)
m/user/P2]                P2: completes PUT operation 19, value = 20
m/user/P2] P2: invokes get operation 20
m/user/P0]                P0: completes GET operation 20, value = 20
m/user/P0] Process 0 execution time: 0.025 second(s).
m/user/P2]                P2: completes GET operation 20, value = 20
m/user/P2] Process 2 execution time: 0.026 second(s).
```

The total computation time for 3 processes and 10 operations is 0.026 seconds.

N = 3 and M = 100

```
tem/user/P2] P2: updated the local value with value (100,197,1)
stem/user/P1]              P1: completes PUT operation 199, value = 100
tem/user/P2] P2: updated the local value with value (200,198,2)
stem/user/P1] P1: invokes get operation 200
stem/user/P1] P1: updated the local value with value (200,198,2)
stem/user/P2]              P2: completes PUT operation 199, value = 200
stem/user/P2] P2: invokės get operation 200
tem/user/P1]               P1: completes GET operation 200, value = 200
tem/user/P1] Process 1 execution time: 0.095 second(s).
stem/user/P2]              P2: completes GET operation 200, value = 200
stem/user/P2] Process 2 execution time: 0.095 second(s).
```

The total computation time for 3 processes and 100 operations is 0.095 seconds.

N = 10 and M = 3

```
system/user/P4] P4: invokes get operation 6
ystem/user/P3]               P3: completes PUT operation 5, value = 9
ystem/user/P3] P3: invokes get operation 6
system/user/P6]              P6: completes GET operation 6, value = 9
ystem/user/P5]               P5: completes GET operation 6, value = 9
system/user/P7]              P7: completes GET operation 6, value = 9
system/user/P1]              P1: completes GET operation 6, value = 9
system/user/P6] Process 6 execution time: 0.047 second(s).
ystem/user/P5] Process 5 execution time: 0.047 second(s).
system/user/P7] Process 7 execution time: 0.047 second(s).
system/user/P1] Process 1 execution time: 0.047 second(s).
system/user/P3]              P3: completes GET operation 6, value = 9
system/user/P3] Process 3 execution time: 0.048 second(s).
system/user/P4]              P4: completes GET operation 6, value = 9
system/user/P4] Process 4 execution time: 0.049 second(s).
```

The total computation time for 10 processes and 3 operations is 0.049 seconds.

N = 10 and M = 10

```
ystem/user/P8]               P8: completes GET operation 20, value = 40
stem/user/P1]                P1: completes PUT operation 19, value = 40
ystem/user/P4]               P4: completes PUT operation 19, value = 40
ystem/user/P0] Process 0 execution time: 0.085 second(s).
ystem/user/P4] P4: invokes get operation 20
ystem/user/P8] Process 8 execution time: 0.085 second(s).
stem/user/P1] P1: invokes get operation 20
ystem/user/P3]               P3: completes GET operation 20, value = 40
ystem/user/P3] Process 3 execution time: 0.086 second(s).
ystem/user/P4]               P4: completes GET operation 20, value = 40
ystem/user/P4] Process 4 execution time: 0.086 second(s).
ystem/user/P1]               P1: completes GET operation 20, value = 40
ystem/user/P1] Process 1 execution time: 0.086 second(s).
```

The total computation time for 10 processes and 10 operations is 0.086 seconds.

N = 10 and M = 100

```
em/user/P1] P1: updated the local value with value (700,416,7)
tem/user/P7] P7: updated the local value with value (700,416,7)
tem/user/P5] P5: updated the local value with value (700,416,7)
tem/user/P6] P6: updated the local value with value (700,416,7)
tem/user/P1]              P1: completes GET operation 200, value = 700
em/user/P7]               P7: completes PUT operation 199, value = 700
em/user/P7] P7: invokes get operation 200
tem/user/P1] Process 1 execution time: 0.107 second(s).
em/user/P7]               P7: completes GET operation 200, value = 700
em/user/P7] Process 7 execution time: 0.107 second(s).
```

The total computation time for 10 processes and 100 operations is 0.107 seconds.

N = 100 and M = 3

```
a://system/user/P92] Process 92 execution time: 0.150 second(s).
://system/user/P17] Process 17 execution time: 0.150 second(s).
a://system/user/P75] Process 75 execution time: 0.150 second(s).
://system/user/P2]              P2: completes GET operation 6, value = 66
a://system/user/P36]            P36: completes GET operation 6, value = 66
a://system/user/P22]            P22: completes GET operation 6, value = 66
a://system/user/P36] Process 36 execution time: 0.150 second(s).
://system/user/P2] Process 2 execution time: 0.150 second(s).
a://system/user/P22] Process 22 execution time: 0.150 second(s).
```

The total computation time for 100 processes and 3 operations is 0.150 seconds.

N = 100 and M = 10

```
system/user/P62]            P62: completes GET operation 20, value = 250
/system/user/P20]           P20: completes GET operation 20, value = 250
/system/user/P25]           P25: completes GET operation 20, value = 250
/system/user/P75]           P75: completes GET operation 20, value = 250
system/user/P62] Process 62 execution time: 0.217 second(s).
/system/user/P25] Process 25 execution time: 0.217 second(s).
/system/user/P20] Process 20 execution time: 0.217 second(s).
/system/user/P75] Process 75 execution time: 0.217 second(s).
```

The total computation time for 100 processes and 10 operations is 0.217 seconds.

N = 100 and M = 100

```
em/user/P52] Process 52 execution time: 1.186 second(s).
em/user/P23] Process 23 execution time: 1.186 second(s).
em/user/P45]              P45: completes GET operation 200, value = 2300
em/user/P59] Process 59 execution time: 1.186 second(s).
em/user/P45] Process 45 execution time: 1.186 second(s).
em/user/P11]              P11: completes GET operation 200, value = 2300
em/user/P11] Process 11 execution time: 1.186 second(s).
```

The total computation time for 100 processes and 100 operations is 1.186 seconds.

Computation time (in seconds) summary:

Number of processes \ Number of operations	3	10	100
3	0.019	0.026	0.095
10	0.049	0.086	0.107
100	0.150	0.217	1.186

As we can see, for a same number of operations, it is much faster to have a lower number of processes with a higher number of operations than the inverse.
We can take for instance 1000 operations, 10 processes take 0.107 seconds while 100 processes take 0.217 seconds, which is two times greater.