Foundations of distributed systems

Magí Tell Bonet (magi.tall@estudiants.urv.cat)
Guillem Frisach Pedrola (guillem.frisach@estudiants.urv.cat)
URV, Tarragona, Campus Sescelades
Sistemes Distribuïts

Index

Introduction

The goal of this task is to implement a synchronization system by using Cloud Serverless technologies. In concrete, by Using IBM-PyWren for running serverless functions and RabbitMQ as a message passing interface.

The proposed exercises were:

1. First spawn a leader, responsible to receive write requests and decide which function is allowed to write in each moment (mutual exclusion). To receive the write requests you can use a single queue (e.g. Direct Exchange). All the slaves must publish its ID to this queue. To allow a slave to write, you have to randomly choose on of the Ids and then notify the slave(s). Master functions will not return anything.

2. Slaves functions must generate a random integer number (for example between 0 and 1000). Then synchronize this number with the rest of the functions by putting (append) the number into a list (e.g. Fanout Exchange). At the end, all the slave functions must return the same list.

Architecture

Our program consists in one master, and a number of slaves. The master, acts as a leader and has a various number of functionalities that will be explained in a moment. The number of slaves can be specified when the function multiple_queue_cs.py is called.

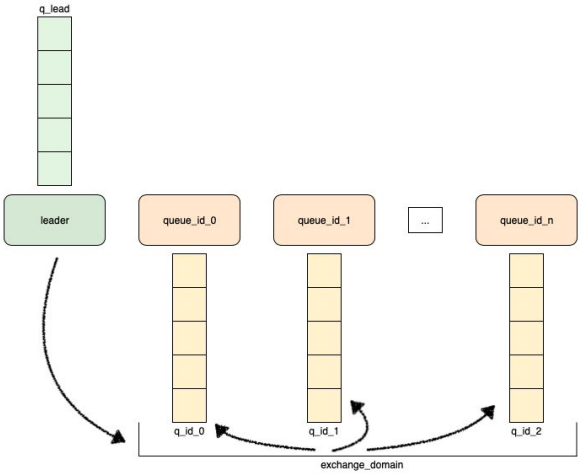
Creating the master and slaves

First, we use the pywren-ibm-cloud repository in order to create and invoke the functions on the IBM-Cloud, in which the *master* and the *slaves* are created.

Once, the functions are invoked successfully, each *slave* creates its own queue into the common `exchange_domain` with the fanout mode and it starts listening (consuming). The *master* creates its queue as well, but outside the `exchange_domain` as he will be the only one reading from it.

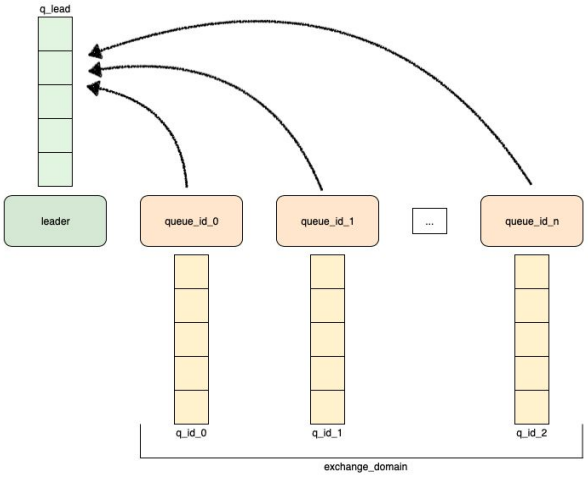
Node communication

The first sent message is by the *master*, and it is the number of *slaves* participating. The *slaves*, already listening, receive the value and store it in a variable named `num_slaves`. This value will allow the *slaves* to keep track of the remaining messages and when do they have to stop consuming from its queues.



Now that the *slaves* have received the `num_slaves`, they send their ID to the *master*. The *slave* only sends its ID if he is active. The `active` flag is up if the slave has never been chosen by the *master*, if it isn't so, the *slave* will be inactive.

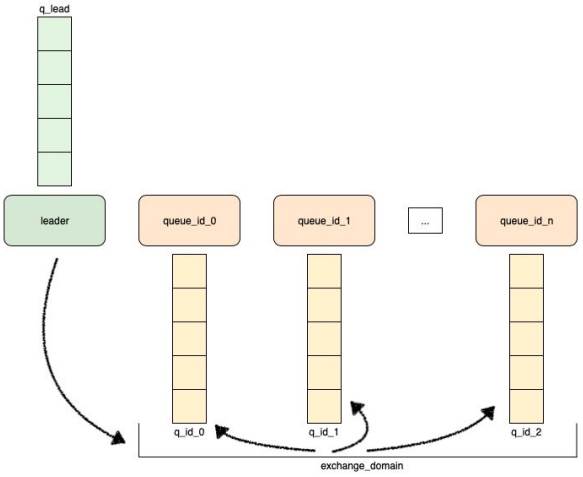
The *master* is in charge of receiving one message from each and every active *slave* in the system and randomly, choosing one of this messages. As the messages sent by the *slaves* are their IDs, when the master is randomly choosing a message, what really is happening is that he is electing a *slave* to work.



When the random message is selected, the *master* sends in fanout mode, a message to every *slave* containing the chosen ID as a negative number. This way, the *slaves* are able to differentiate between an ID message from the *master* and a message sent by a *slave*:

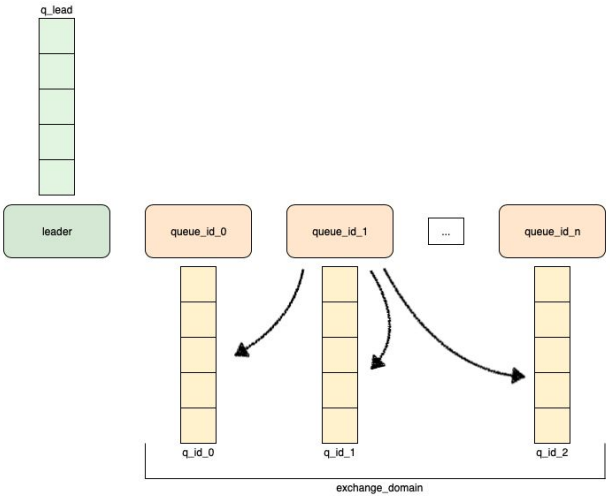
-If the message received is negative → ID message.

-If the message received is positive → Message from the chosen slave with a random number in it.



As explained, every *salve* receives a message containing an ID in a negative fashion. The *slave* checks if the ID in the *master's* message is the same as they have. If they do, they send a random number between 1 and 100 to the `exchange_domain` and then the chosen *slave* sets its `active` flag to `False`, as he will be inactive from now on.

When a *slave* receives an ID, they also add `+1` to a variable that helps keeping track of the number of *slaves* remaining.



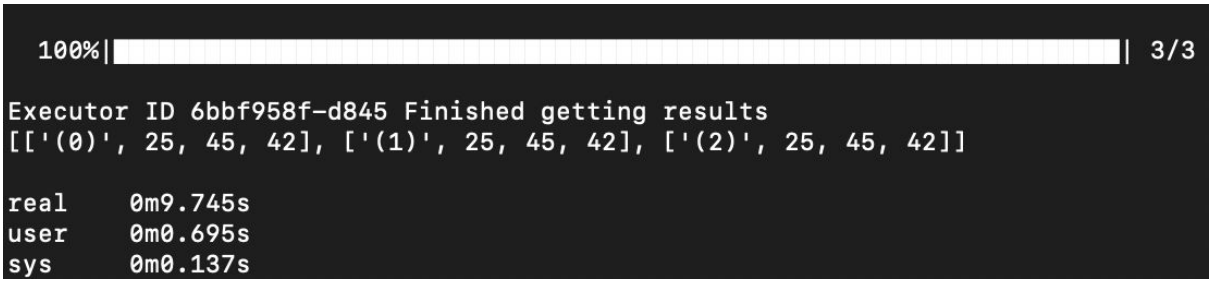
The sent message to the `exchange_domain` is received by all the *slaves* including the sender. Now, they check the message and as the message is positive, they know that it isn't an ID, so they proceed to append the value received in a variable named `result`.

The process is repeated until there is no active *slave* to be chosen. Once the process is completed, the `result` variable from each *slave* is shown.
If every `result` from every *slave* is the same, the process was successfully executed.

Test and validation

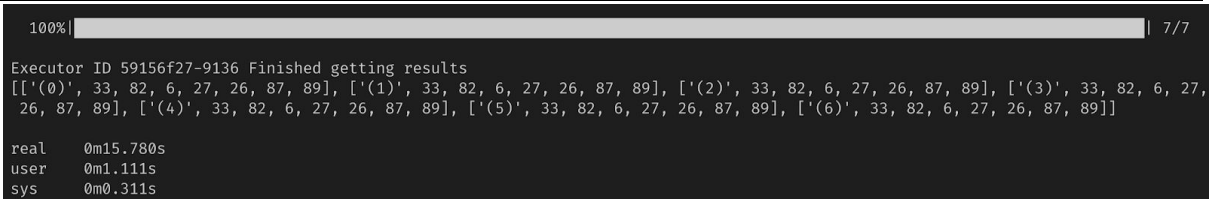
> Test with 3 slaves

	Expected	Got
Number of slaves created matches the one specified	3	3
Same result for everyone	YES	YES



> Test with 7 slaves

	Expected	Got
Number of slaves created matches the one specified	7	7
Same result for everyone	YES	YES



> Test with 10 slaves (Default value)

	Expected	Got
Number of slaves created matches the one specified	10	10
Same result for everyone	YES	YES

```
100%|████████████████████████████████████████| 10/10

Executor ID 3800dba5-1cfc Finished getting results
[['(0)', 16, 99, 79, 37, 73, 32, 60, 20, 22, 65], ['(1)', 16, 99, 79, 37, 73, 32, 60, 20, 22, 65], ['(2)', 16, 99, 79, 37, 73, 32,
60, 20, 22, 65], ['(3)', 16, 99, 79, 37, 73, 32, 60, 20, 22, 65], ['(4)', 16, 99, 79, 37, 73, 32, 60, 20, 22, 65], ['(5)', 16, 99,
79, 37, 73, 32, 60, 20, 22, 65], ['(6)', 16, 99, 79, 37, 73, 32, 60, 20, 22, 65], ['(7)', 16, 99, 79, 37, 73, 32, 60, 20, 22, 65],
['(8)', 16, 99, 79, 37, 73, 32, 60, 20, 22, 65], ['(9)', 16, 99, 79, 37, 73, 32, 60, 20, 22, 65]]

real    0m16.020s
user    0m1.239s
sys     0m0.341s
```

> Test with 17 slaves

	Expected	Got
Number of slaves created matches the one specified	17	17
Same result for everyone	YES	YES

```
100%|████████████████████████████████████████| 17/17

Executor ID 7364aed0-beae Finished getting results
[['(0)', 10, 40, 7, 88, 3, 73, 25, 7, 58, 24, 63, 64, 16, 90, 95, 100, 14], ['(1)', 10, 40, 7, 88, 3, 73, 25, 7, 58, 24, 63, 64, 16
, 90, 95, 100, 14], ['(2)', 10, 40, 7, 88, 3, 73, 25, 7, 58, 24, 63, 64, 16, 90, 95, 100, 14], ['(3)', 10, 40, 7, 88, 3, 73, 25, 7,
 58, 24, 63, 64, 16, 90, 95, 100, 14], ['(4)', 10, 40, 7, 88, 3, 73, 25, 7, 58, 24, 63, 64, 16, 90, 95, 100, 14], ['(5)', 10, 40, 7
, 88, 3, 73, 25, 7, 58, 24, 63, 64, 16, 90, 95, 100, 14], ['(6)', 10, 40, 7, 88, 3, 73, 25, 7, 58, 24, 63, 64, 16, 90, 95, 100, 14]
, ['(7)', 10, 40, 7, 88, 3, 73, 25, 7, 58, 24, 63, 64, 16, 90, 95, 100, 14], ['(8)', 10, 40, 7, 88, 3, 73, 25, 7, 58, 24, 63, 64, 1
6, 90, 95, 100, 14], ['(9)', 10, 40, 7, 88, 3, 73, 25, 7, 58, 24, 63, 64, 16, 90, 95, 100, 14], ['(10)', 10, 40, 7, 88, 3, 73, 25,
7, 58, 24, 63, 64, 16, 90, 95, 100, 14], ['(11)', 10, 40, 7, 88, 3, 73, 25, 7, 58, 24, 63, 64, 16, 90, 95, 100, 14], ['(12)', 10, 4
0, 7, 88, 3, 73, 25, 7, 58, 24, 63, 64, 16, 90, 95, 100, 14], ['(13)', 10, 40, 7, 88, 3, 73, 25, 7, 58, 24, 63, 64, 16, 90, 95, 100
, 14], ['(14)', 10, 40, 7, 88, 3, 73, 25, 7, 58, 24, 63, 64, 16, 90, 95, 100, 14], ['(15)', 10, 40, 7, 88, 3, 73, 25, 7, 58, 24, 63
, 64, 16, 90, 95, 100, 14], ['(16)', 10, 40, 7, 88, 3, 73, 25, 7, 58, 24, 63, 64, 16, 90, 95, 100, 14]]

real    0m25.494s
user    0m1.933s
sys     0m0.466s
```

Code

The code can be found in: https://github.com/Plumoll/SD_pyWren