LSINF2345 Final Project
Support for Atomic Transactions in Lasp
Report

Victor Solana Roldan (0779-16-01) Robin Descamps (3325-13-00)

May 5, 2017

1 Transactions implementation

We followed all the tasks described in the instructions, and made some decisions since the way we must implement the atomic transactions is not imposed. First, we decide to apply only one `Actor` per transaction, in a matter of simplicity (this choice does not influence the transaction, it is not difficult to apply different `Actors` in the operations).

1.1 Synchronization

When we disabled the normal synchronization, we simply deleted all the lines that launched the `schedule_state_synchronization()` function, and then modify it. We set the `ShouldSync` to false (forever) in order to only execute our new transaction periodic synchronization. Note that we can already observe that we chose the **At-Least Once Message Delivery** approach. We will discuss this choice in the next section.

1.2 Buffer

Now, let's describe our buffer. We begin by creating a new file: `lasp_transaction_buffer.erl` where the buffer (that save a transaction, that is to say all the operations for all the peers) is handled, in a new process. The mechanism we use to make the buffer persistent is inspired from an Erlang counter example[1]. The structure of the buffer is the following:

$$[E_1, E_2, E_3, ..., E_n]$$

Where E_n is:

$$\{nodeN, [\{setX, operationA, Actor\}, \{setY, operationB, Actor\}, \{setX, operationC, Actor\}, ...]\}$$

Where

- $nodeN$ is some peer
- $setX$ is some CRDT state-based observed-remove set
- $OperationA$ is an operation to perform on the $setX$ in the peer $NodeN$
- $Actor$ is the Actor specified when we called `lasp:transaction()`.

The buffer will then try to send all theses updates in the order of the lists belonging to the nodes.

[1] http://erlang.org/pipermail/erlang-questions/2009-June/044893.html

1.3 Acknowledgements and FIFO order

Once a peer receive an update, it performs it and returns an acknowledgement. Once acknowledged, the corresponding update is removed from the buffer. To ensure the FIFO order of the transaction operations, we added a counter in the buffer structure. This counter is taken into account in the synchronization mechanism (see the `transaction_sync(N)` function) as all the operations are performed and confirmed in the order they were specified in the transaction.

2 Messages delivery approaches

Two approaches are possible, we will explain the advantages and the drawbacks of both:

- **At-Least Once Message Delivery**: This is the approach we use. The drawback here is that many messages can be sent in our network, and infinitely often in case of link failure between two peers. The advantage is that the message is guaranteed to be eventually delivered. **At-Most Once Message Delivery**: Here, we never "flood" the network, there is at most one message per operation to perform in our transaction, this is an advantage. The drawback is that if the message is lost, we do not re-send it. This is why we decided not to use this approach, in order to not have lost messages and to ensure a perfect synchronization in all of the peers in the system.

3 Difficulties

We encounter many difficulties with this project. First, we must learn and use a new language, but in addition to that, we must apprehend a complete system in order to complete it. This is not easy, and take a lot of time, before we can even begin the described tasks. Secondly, there is not a lot of support concerning the Lasp project. The community is small, and we cannot easily find answers to our questions. Besides, it takes a lot of time to perform/write tests on the Lasp system. One can notice that our project compiles, but does not work as expected: we struggled to debug a lot of problems, but not all of them, since we did not have the time to observe more code of the Lasp system in order to solve them. Finally, we performed some tests all along the development, but we were unable to test the transaction synchronization, since too many errors arisen.

Nevertheless, we are convinced that with the correct debugging, our solution (written in the code and described this report) can work on the Lasp system.