LSINF2345

LANGUAGES AND ALGORITHMS FOR DISTRIBUTED APPLICATIONS

Support for Atomic Transactions in Lasp Report

VAN GROOTENBRULLE Antoine
antoine.vangrootenbrulle@student.uclouvain.be
Noma : 2786-12-00
VAN SWIETEN Anne-France
anne-france.vanswieten@student.uclouvain.be
Noma : 8575-12-00

Assistant:
CHRISTOPHER MEIKLEJOHN
Professor:
PETER VAN ROY

May 5, 2017

Introduction

This project was based on an existing Erlang project called Lasp, which is a large-scale distributed application project. The goal was here to learn how to navigate an existing code base and to apply the concepts from the class on a practical and concrete case with the adding of a new functionality.

1 Implementation

A list of tasks describing the different phases of the implementation of the new functionality was provided as a guide.

1.1 `lasp:transaction`

The first thing to do was to create a new function. This function will be the new command to use to apply multiple updates. It then takes as parameters a list of tuples each containing an id and an operation to apply, and, as second argument, the actor. Because the actor is the same for each sequence of updates, we put it out of the list of updates to avoid repetition of this information.

The body if the function consists of a foreach loop on the list of pairs ¡id, operation¿ applying the core update function to each pair. After that comes a call to our method `store_in_buffer` which adds the list of updates in buffers (see subsection 1.3). It can be tested by launching ”make test” in terminal .We have added one test ”transaction_test” in the file ”lasp_SUITE.erl”.

1.2 Disabling normal synchronisation

We disabled the normal synchronisation by switching the `ShouldSync` variable from true to false in the `schedule_state_synchronization()` function of the file `lasp_state_based_synchronization_backend.erl`. It should thus lead to unexpected failures in the testsuite, because some tests depend on this synchronisation fonction.

1.3 Buffering the updates

We build a function, called `store_in_buffer`, to store the list of updates belonging to one transaction in one buffer for each of the peers the current nodes has. It does not create a buffer for itself because the updates are already applied for the current node. Unfortunately this function is not working despite our repeated attempts. It encounters a problem when dealing with the id of the current node when calling the `without_me` function.

2 Project Choices

As explained in the `lasp:transaction` section of the report, we chose the at-least once message delivery. We first apply the updates on the current node and then fill the buffers to be sent to the peers. Even if we did not implement all the given tasks, we took time at the beginning of the project to think about the approach we will follow. The drawbacks of at-least once is the need of an acknowledgement mechanism to ensure delivery of the message. Otherwise the message will be sent and resent infinitely. The repetition of messages can lead to multiple delivery which is here not a problem because we are working with sets (adding and removing). Set will not store more than once each value. At-most once has higher performances but does not ensure delivery while at-least once needs more work but ensures that every message will eventually be delivered.

3 Test suite

A test has been added in the `test/lasp_SUITE.erl` file to ensure that our version of `transaction` is working as planned. This test consists of the adding in an empty set of a list of updates and then verifies that the content of the set corresponds to the right values.

Conclusion

This project was interesting to make us face a real application of the theory seen during the class. It was a good thing to learn from a existing project despite it took much more time than expected. The Erlang language we needed to get used with in parallel made us lost a lot of time. These problems are the cause why we do not achieve the whole task, but we still learned a lot of things that will be useful for our future projects.

.