LSINF2345 Final Project: Support for Atomic Transactions in Lasp

Christopher Meiklejohn (christopher.meiklejohn@gmail.com)

April 19, 2017

1 Overview

Our project is based on research that's happening at Université catholique de Louvain, part of just-finished research project SyncFree, on CRDTs and synchronization-free computing, and LightKone, a new European project on edge computing. Instead of building a greenfield[1] project, like in previous classes, we focus on adapting an existing system in Erlang to add new functionality.

1.1 Motivation

Most of the time when you're working an engineering job, you don't have the luxury to start a new project to solve a problem! Therefore, we're going to focus on how you can do a small adaptation to an existing Erlang project and learn how to navigate an existing code base of a large-scale distributed application.

This will leverage a lot of the concepts you've learned in the class: how do we send updates reliably, how do we ensure the updates were received, and how do we ensure they were received in the correct order.

That said, if your work is extremely good, you might end up getting your code into the main version of Lasp, which is used by a few companies in the Silicon Valley!

1.2 Description of work

This project aims to add support for "atomic transactions" in the Lasp programming system.

Lasp, outside of being a programming system, is just a distributed database: it offers two main operations: query, to get the current value of an object, and update, to update the value of an object based on a strict API that ensures updates won't conflict with each other. Most of the time, updates are applied independently, where each update is applied to an object independently. However, sometimes you need updates to appear on each machine at the same time;

[1] A project that is created from scratch.

for example, if I decrement the quantity of available stock, and create an invoice for someone who bought an item, these two actions should become visible together. Another example is if you were to model a graph in Lasp with two sets: you do not want to have an edge appear to a vertex you do not know about yet. Therefore, you want to send these updates together.

This is precisely what we're going to implement. We're going to build a system to ensure that updates appear **at the same time**, or, what is commonly referred to as **atomic transactions.**[2]

1.3 Project description

We're going to extend the model to have a new command that let's us specify a group of updates to be applied at the same time. Then, we have to ensure we only send the group of updates together (by disabling the behavior where we try to send around individual changes) and ensure changes are delivered together in all of the machines in the distributed systems.

2 Project work (85 possible points, 15 possible bonus points)

2.1 Groups

The project should be done in groups of **2** students.

2.2 Where to find the code?

Head over to GitHub, and download the Lasp project at http://github.com/lasp-lang/lasp. Follow the instructions on https://lasp-lang.readme.io/docs on how to get started.

2.3 Project tasks

Outline of project tasks:

- **[10 points] Create a new API call named `lasp:transaction()`**
  This call should take a list of updates to be performed in the form of the parameters to the update call. Look at the existing update API to see the format.
  (for example: [{ObjectId, Operation, Actor}])

- **[5 points] Disable the normal synchronization mechanism**
  Periodically, the state synchronization backend will transmit the current value of objects to all of the nodes it knows about. This has to change, because updates that are done in the same transaction need to be sent together.

[2]If you've ever heard of ACID transactions, this is the A part of ACID transactions.

- **[20 points] When the API is called, buffer the updates**
  Store the changes in a buffer in the state synchronization backend, one per peer the system knows about.
- **[10 points] Periodically, synchronize with peers**
  Every X seconds, send the buffered events, in the right order to the node's peers. Don't clear the buffer until you receive an acknowledgement from the other node.
- **[5 points] On receive, acknowledge them and apply them locally**
  For each update, apply the updates locally using the `?CORE:update` API and then acknowledge the update.
- **[5 points] Ensure atomicity and FIFO**
  The system shouldn't interleave any other requests: the backend needs to ensure the updates are applied all at the same time before resuming; these updates should also be delivered in the correct (read: send) order.
- **[10 points] Adapt the test suite**
  Using what's already existing in the test suite, write a test that verifies that each node receives all of the updates, in the correct order.

2.4 Project choices

You have two choices for messages delivery; either approach is fine, and has its own tradeoffs.

- **At-Least Once Message Delivery**
  You can execute the updates on the node where the updates originated immediately, then buffer the effect of those updates and send the values in the buffer: these changes will be idempotent, which means you won't have to ensure they are delivered at-most once and can keep sending them until you receive an acknowledgement.
- **At-Most Once Message Delivery**
  You can send the update commands to all nodes and tag them uniquely to ensure that they are only ever applied at the target node once.

2.5 Bonus points

- **[5 points] Trade-offs**
  Explain why you made the choice you did and what are the trade-offs.
- **[10 points] Configurability**
  Make it configurable, so we can try both approaches based on a configuration parameter.

2.6 Deliverables

- **Project deliverable**
  A zip file or fork of the Lasp repository with your changes so they can be easily compared to the mainline of Lasp through GitHub.

- **Documentation**
  A document justifying your decisions that outlines how to run the project and ensure that it's working properly. Your application should work with Erlang 19, the standard distribution that's used for Lasp.

- **Justification**
  If you didn't finish certain components, you can explain why they don't work, and partial credit will be awarded.

3 Further information

Deadline: May 5th, 23:59 AoE (Anywhere on Earth)

3.1 Help

Where to find help:

- **Slack**
  http://lasp-lang.slack.com

- **GitHub**
  https://github.com/lasp-lang/lasp

- **Lasp Documentation**
  https://lasp-lang.readme.io

- **Email**
  christopher.meiklejohn@gmail.com