D7024E Lab report
Distributed File Synchronization

Linus Hedenberg,
Andreas Hägglund

October 29, 2013

1 Requirements and assumptions

The manager is currently assumed to be run locally. It would probably not be hard to implement the possibility to add your local computer as machine and allow ssh to that machine to run docker containers (the nodes) but since we did not see this as an interesting problem we just ignored it.

For a node to be online it needs to have connection to the Manager-program. This means that if the manager is run locally with a local and an external node and the local system goes offline it is the external node that is considered to be offline. By extension since all communication goes through a RabbitMQ broker if this communication fails every node is considered offline and if a single nodes connection to the broker fails then obviously that node is considered offline.

2 Frameworks and tools

Vagrant

Used to create and configure lightweight, reproducible, and portable development environments. We used this with virtualbox and ubuntu 13.10.

Docker

Used to create lightweight, portable, self-sufficient containers from any applications. In this project each node is run in its own docker-container using Ubuntu.

OpenStack and boto

OpenStack is a cloud operating system that controls compute, storage, and networking resources in a datacenter. Here it is simply used to create a complete virtual machine on demand for our application. The user creates a virtual machine to create nodes on. Boto is simply a python interface for OpenStack (Actually made for Amazon but since both implement EC2 it works for OpenStack just as well).

Flask

Flask is a lightweight web application framework based on Werkzeug WSGI toolkit and Jinja2 template engine. It has a very basic and simple to use RESTful request dispatching used in this project to handle all RESTful requests.

SQLAlchemy

Python SQL toolkit.

RabbitMQ, pika and json

RabbitMQ is an open source message broker software that implements AMQP. For all synchronization we use construct simple json messages sent through RabbitMQ. Pika is a python client library we use for both our node and manager clients.

3 System architecture and overview

Nodes

The manager runs on a vagrant-VM. Each node it creates locally gets its own docker-container to run in. Each node will send messages to the master through an update-channel and receive messages from the master through a master-channel. The manager will therefore receive messages from all nodes in the update-channel and be able to broadcast messages to all nodes through the master-channel.

Manager

The manager can create virtual machines (VM) on OpenStack with a simple command it will receive an IP-address and an identifier it will use later to kill the VM. A node will be created on the VM in the same way it is created locally.

Synchronization

When a node comes online it will send a sync-message to the manager. This message includes all blobs currently on that node, in this case it will most likely be empty. The manager will first select a random node to use as source since it doesn't actually store the blobs itself. It will then go through the list of blobs the node sent and send send information about any blob the node is missing.

When a node adds a blob it will send a message to the master and the master will store meta-data about the blob and send the info to all other nodes along with a last-sync time-stamp the source-node so that all nodes can pull the new file from it through HTTP-requests. A similar appropriate message is sent when a node updates or deletes a blob as well. The original node will also receive the message from the master and will update the blobs last-sync time-stamp. If a node is offline when it adds or updates a file it will not receive a last-sync time-stamp.

When a node comes online after being offline the user needs to press the Sync-button. This will send a sync-message as described before. If the node has updated any files while it was offline the last-changed time-stamp will have changed but the last-sync time-stamp will not. The manager will compare these time-stamps. If the last-sync time-stamp is identical to the nodes then no conflict will have happened and the manager can ask all nodes to update their blob from the source-node that just came online. If the last-sync time-stamp is not the same then the file has been updated on both an online node and the offline node and a conflict-file is created on the node that just came online. This conflict-file is pushed to all nodes and the node that just came online is asked to replace its old file with the file the online-system had.

4 Collaborations

How to use tools and frameworks have been discussed openly in the cellar/ dungeon/ hole they call our project room. We've also discussed the problems of the lab assignment to better understand it but we've solved those problems individually.

Obviously our main source when looking to solve a problem has been to simply do a simple google-search to see if anyone has already solved this problem. These are hard to quote since a small problem is solved so quickly this way.

Before we all understood the basics of Docker we had a lot of trouble trying to get it to work. There were a lot of discussions between our group and David Eriksson- and Anton Wennberg's group trying to figure it out.

When researching on how to use OpenStack in our manager we were recommended boto from Mikael Åhlen and Peter Sanfridssons group, they also got us started by explaining the extremely cryptic connection command.

Discussions and research together with David Eriksson and Anton Wennberg was the main reason we chose to work with RabbitMQ. We also looked into using ZeroMQ but we figured RabbitMQ would be enough.

5 Limitations and possibilities for improvements

Due to unexpected problems with rabbitmq, external nodes are currently experiencing problems connecting to the rabbitmq server. We've tried looking in to the source of this problem and can only assume that it has something to do with either running the client in a second thread or the amount of messages we send at startup. The problem is that after creating a new virtual machine the first node will often fail to connect to the rabbitmq server and a second node created shortly after will always fail. Trying to create a node over and over again will result in some successes to connect and sometimes waiting for an extended period of time will allow for one node to be created and successfully connect to the rabbitmq server. Just creating a docker-container and running a simple program that creates a new channel and sends a short message will always succeed.

References

[1] *Vagrant*. http://www.vagrantup.com/

[2] *Docker*. http://www.docker.io/

[3] *OpenStack*. http://www.openstack.org/

[4] *Flask*. http://flask.pocoo.org/

[5] *SQLAlchemy*. http://www.sqlalchemy.org/

[6] *RabbitMQ*. http://www.rabbitmq.com/

[7] *Pika*. https://github.com/pika/pika/

[8] *VirtualBox*. https://www.virtualbox.org/

[9] *boto*. https://github.com/boto/boto