Friend search for distributed social networks

Sebastian Probst Eide, St Edmunds College

Originator: Sebastian Probst Eide

4 October 2010

Special Resources Required

Personal laptop for development and initial testing (1.86 Ghz, 2GB Ram)
CL machine with the Erlang VM installed as a backup
Virtual Server infrastructure, the likes of Amazon EC2, for parts of the evaluation

Project Supervisors: Dr David Eyers and Dr David Evans

Director of Studies: Dr Robert Harle

Project Overseers: Prof. Jon Crowcroft and Dr Simone Teufel

Introduction

It is hard to get started using independent, distributed online social networks as it is frequently difficult to find and connect with your existing friends, not knowing in which social networking system(s) they have their profiles and where those networks are hosted. The purpose of this project is to lay the foundation for a decentralised and distributed friend search engine that can be offered alongside installations of independent social networks allowing users to easily reconstruct their social graph in the online social network(s) of their choice. The focus of the project will be on the data storage layer of the search engine. I will compare and contrast different Distributed Hash Tables that I implement in Erlang. Erlang is chosen because it is known to be well suited for developing concurrent and distributed systems.

A front end, allowing basic searches to be performed, will also be created, but mainly to provide a way to rapidly exercise the data storage layer. More user-oriented functionality needed to allow the project to be used on a larger scale will be left out so as to limit the scope of the project.

Work that has to be done

There are a number of distributed hash table designs available.[1] I have decided to implement and compare the following three: Chord, Kademilia and Pastry. They were chosen because they are well documented algorithms, but differ in the way they perform their routing. As an example, consider how Pastry allows for heuristics based on anything from ping to available bandwidth or combinations thereof, Kademilia uses XOR arithmetic to determine the distance between nodes as a routing heuristic, while Chord has none of the above. These different approaches to the same problem make the algorithms excellent candidates to compare.

The main parts of the project are to:

- Implement the data storage layer of the search engine in Chord, Kademilia and Pastry using a uniform API that allows the system to use any one of the three without additional changes
- Implement infrastructure that facilitates testing and monitoring of the system. More specifically it should allow:
  - Starting and stopping virtual servers across the different service providers to minimize server rental costs between testing sessions. The servers will be used to test different aspects of the distributed hash tables

[1] Wikipedia currently lists 8 different protocols

  - Start and stop search nodes across the physical servers
  - Display how many search nodes are available in the system, and potentially some metric for how they are interconnected in terms of latency and bandwidth
  - Add and remove test data from the system. The system will be tested with *Database of names* from Facebook which I have access to. It is of significant size and importantly, contains keys with non-random distributions making for a more realistic dataset
  - Perform repeatable load testing on the system where tests as an example could compare read/write performance for different key and value sizes and different numbers of key-value pairs
  - Count the number of jumps and the time a key lookup needs in order to find a data item, in addition other appropriate descriptive statistical measures for writes and lookups in fixed key-value datasets
  - Being able to eliminate and add subsets of storage nodes in a repeatable fashion to test how the different Distributed Hash Tables cope with nodes disappearing and appearing

- Setup a Linux image that can be run across Infrastructure as a Service providers[2]

- Implement a web front end to allow users to perform basic searches across the data storage layer

Please note that while the following aspects of the search server are secondary to the project and will not initially be implemented, they all, should time permit, serve as excellent project extensions:

- Fuzzy searches allowing the user to misspell names.

- Support composite keys to allow searching for different attributes of a record

- Predictive searches

- Searches taking knowledge about social circles from online social networks, or similar metadata, into account in order to more intelligently prioritise and order the search results returned to the user

- Protections against malicious use of the storage network like broadcasting data, attempting to overload nodes with requests or using the network to store spam or pollute the namespace with spammy records

[2]Vagrant seems like a likely candidate to help automate this (http://vagrantup.com)

Starting Point

I have a reasonable working knowledge of Erlang and Linux and development of web based systems. The algorithms that will be implemented have all previously been implemented in other languages, and are used in production systems, so finding information about them should be possible. I have not yet used Amazon EC2 or any of the other Infrastructure as a Service (IaaS) providers that could be used to perform testing on the system in a distributed manner.

Success criterion

I regard the project as successful if I have working implementations of the three Distributed Hash Tables that allow me to set and retrieve values based on keys across a distributed network of machines, and also metrics for how the performance of key-lookup varies by node-, key-count and distributed hash table type and recommendations for future work based on the metrics collected.

The search component of the project, which is the application part that uses the distributed hash tables as a datastore to allow users to find their friends, has been left out of the success criterion due to it being harder to quantify and evaluate well.

Difficulties to Overcome

The following main learning tasks will have to be undertaken before the project can be started:

- To learn and fully understand the Chord, Kademilia and Pastry algorithms.
- To learn how to do network communication in Erlang other than the built in message passing. I prefer the individual nodes to communicate over TCP or UDP as that frees the design from assumptions regarding the language of implementation. Additionally there are security issues when allowing Erlang VMs to connect directly in untrusted networks as any node is allowed to execute arbitrary code on any other connected node
- To find a way to test the system on geographically distributed nodes.

Resources

Some aspects of this project (amongst others the heuristics in Pastry that take locality into account when routing) are more interesting to test in nodes that are geographically distributed. For this reason using server instances from a provider

like Amazon AWS or PlanetLab seem like a good idea. Ways of getting access to time on such infrastructure for academic purposes is currently being looked into. For the majority of the development cycle local testing will be just as interesting and can be done on my development machine.

This project requires no additional file space on University machines. I will be hosting the project source code and dissertation files in a repository on github.[3] If my machine breaks down, the development can be continued on any Unix based machine that has VIM, git and the Erlang VM installed.

Work Plan

Planned starting date is 15/10/2010.

Below follows a list of tasks that need to be done:

- Work through the theory behind Chord, Kademilia and Pastry and other items listed under *difficulties to overcome* (2 weeks).
- Implement initial version of Chord, Kademilia and Pastry in Erlang (4 weeks)
- Implement test harness to perform testing of the system (4 weeks)
- Implement a web frontend for search alongside a search server written in Erlang, that uses the distributed data storage layer for its data (2 weeks).
- Write dissertation (6 weeks).

Michaelmas Term

By the end of this term I intend to have completed the research and learning tasks and have finished the first implementations of the Distributed Hash Tables in Erlang. In the vacation that follows I intend to make a good start on the testing harness and do a little work on the search server.

Lent Term

In the first half of this term I intend to finish the test harness and search server and spend time testing the system and solving problems. The tests could follow a factorial design where distributed hash table type, number of nodes, key-size, payload size, number of entries in the system and geographical distribution are all factors worth considering. What levels should be considered for each factor is yet to be determined and will become clearer as the project develops.

[3] http://github.com/sebastian/Part-2-project

In the second half of this term I tend to get an initial draft of my dissertation written.

Easter Term

In this term I plan to polish the dissertation. The estimated completion date is the 15th of May, leaving a couple of days to let the dissertation rest before giving it a final read and correcting last minute mistakes before the due date on the 20th of May.