Memorandum

To: Instructor
From: Craig Peterson
Date: May 18, 2014
Subject: Research proposal

Introduction

My research will revolve around a software framework called Thrift. Specifically my question will be “What is the optimal platform and configuration for a running a thrift service?”

Question seems a bit open ended. You are not testing EVERY possible configuration. Just a select group. Maybe limit to protocols? Maybe reword it to match your scope better.

Background

Apache Thrift is an open-source framework for developing interfaces between components in a distributed system.

This whole section is a bit wordy. Try to keep it simpler:
1: Why web services are hard.
2: Why/how thrift solves these problems.
3: Why thrift is hard to configure.
Make sure everything here is relevant to your particular research.

A common problem in software engineering is how to communicate between processes running on different systems, and how to transfer data between them. This problem is made difficult by a variety of factors:

1. Systems written in different programming languages need to agree on a common format for data passed between them. It can be problematic to make representations of data types that are consistent across all client languages.
2. Communications over a network can be slow or lossy. Messages may be garbled or lost in transit, and latency may decrease the performance of the system.

3. Servers have finite resources. Any program which consumes too many CPU, memory, or network resources can decrease the performance of the system. If a program is not efficient in it's usage of these resources, it will not scale well.
4. Systems must be documented well so that people can use them as intended.
5. Servers and Clients should strive to maintain backwards and forwards compatibility to the fullest extent possible. That is, newer server versions should work with older client versions, and vice versa.

Using these factors as the primary considerations, it would follow that two of the most important measures of the quality of a web service are:

1. How well does it perform?
2. How easy is it for a client to interact with?
3. How easy is it to create and maintain?

Over time, the software industry has come up with a variety of solutions to this problem, including technologies such as REST, SOAP, JSON, XML, CORBA, JSON-RPC, and many others. My research will not be an in depth study of these technologies and their strengths and weaknesses. Suffice it to say they each have unique benefits and also shortcomings and pain points that make them difficult to work with for one reason or another.

Apache Thrift is a framework for remote procedure calls that aims to make web service development simple and efficient. It solves the problems listed above by:

- Defining all data types in a standard Interface Description Language (IDL) that is entirely language independent. This allows language-specific code to be generated that perfectly matches the specification, so that all users are guaranteed compatibility with each other.
- Using efficient transfer and serialization protocols to keep network and cpu utilization low.
- Automatically generated code for client and server allows for rapid creation of services, and extremely easy implementation for clients of a service.
- Generated documentation is automatically kept up to date as the code changes.

Thrift is appealing because it decouples the interface definition from the implementation of the service, and from the specific details of what a message looks like, or how it is transmitted. This flexibility gives the programmer the ability to choose independently:

1. The programming language the service is implemented in.
2. The protocol used to format data for communication.

3. The transport mechanism used to move data from one server to another.

No matter what choices are made, a thrift service will be fully compatible with any client that wishes to communicate with it.

There is difficulty in making these choices, as there are numerous options for all three pieces. My research will aim to test various configurations and decide which is best according to the criteria for web service quality listed above, namely: performance, usability, and maintainability in that order.

Audience

My target audience is professional software engineers interested in thrift. They will have significant experience already developing web services using other technologies, and should need very little introduction to the relevant concepts.

Methodology

My report will exclusively rely on primary research.

I plan to analyze various server configurations by writing real test code in multiple configurations, running it, and analyzing the results. For all tests I will focus on three programming languages which fairly well represent major families of languages. From the 20 or so languages that thrift supports, I have chosen:

- **Go** to represent low-level compiled languages.
- **Python** to represent higher order dynamic languages.
- **C#** to represent enterprise level, semi-compiled, virtual machine languages.

I hope that these choices will cover a wide range of languages, and will give enough variety to eliminate the possibility that an individual language's implementation is not an accurate representation of the thrift ecosystem.

Good justification for choices.

My first set of tests will be to choose which protocol is most efficient. A protocol is the method by which a given message is converted into a sequence of bytes to be transmitted. Thrift offers three main options on any platform: JSON, Binary, and Compact protocols. I will create a sample set of objects and test each of these protocols in each of my target languages. My criteria for judging them will be:

1. Speed of serializing data (converting an object to a stream of bytes)
2. Speed of deserializing data (converting bytes to objects)
3. Final size of serialized byte stream.

After performing these tests I hope to have a clear winner of a protocol that is both fastest and most compact on any platform. Once that is complete I will move to performance analysis of various server configurations.

Thrift allows for several different transport modes. Most common are direct tcp connections, or else embedded in standard http requests. For tcp servers, some platforms even have multiple server modes that use different threading models to achieve different performance characteristics. It can be very confusing trying to pick the proper configuration for your particular use case.

I plan to create a sample service and create multiple server implementations on each platform (2-3 unique configurations per programming language). I will then run a suite of load tests on each configuration to asses the performance characteristics of each one.

For all of my tests I plan to write my own code, and will run the tests on identical virtual machines running in a stable cloud environment to minimize variability due to hardware differences or network fluctuations.

Qualifications

I am qualified to perform this research because I am a professional software engineer with experience using thrift, as well as the programming languages I will be using. I am proficient at writing and deploying programs, and have experience measuring runtime performance characteristics of my programs.

Why is this bold? Formatting error?

Plan

"What is the optimal platform and configuration for a running a thrift service?"

Possibly: "Which thrift protocol and server technology choices make the most efficient service platform?"

1) Introduction to Thrift
    a) Brief overview of technology
    b) Possible platforms and configuration options
    c) Important performance characteristics for a web service
2) Protocol

    a) Multiple possible protocols
    b) Overview of tests performed
    c) Results of tests
3) Server platforms
    a) Explanation of chosen platforms
    b) Breakdown of test configurations
    c) Test results
4) Conclusions
    a) Lessons learned
    b) Recommendations

Schedule

Jun 22 - Proposal submitted
Jun 23-25 - Perform thorough protocol testing
Jun 26-30 - Perform server platform testing
Jun 30-Jul 4 - Write up report
Jul 5 - Finish initial draft
Jul 6 - Submit for intructor comment
Jul 6-13 - Polish final draft
Jul 14- Submit final report

You are already behind? Does this need to be revised?