Thrudb: Document Oriented Database Services

T Jake Luciani
http://thrudb.googlecode.com — http://3.rdrail.net
jake [at] 3.rdrail.net

Abstract

Thrudb is a set of simple services built on top of Facebook's Thrift framework that provides indexing and document storage services for building and scaling websites. Its purpose is to offer web developers flexible, fast and easy-to-use services that can enhance or replace traditional data storage and access layers.

This paper explains the services Thrudb provides as well as the underlying libraries and choices made when designing it.

1. Introduction

Relational databases are the backbone of modern websites. Along with web servers, they enable the world wide web by allowing anyone with a web browser to extract and interact with data and documents. The traditional LAMP stack has provided everything needed to build websites, but with the onset of web 2.0 there are numerous programming frameworks and web services that can simplify the development effort, as well as countless techniques and enhancements to improve the speed and scalability of your site once deployed, but there is still one area that is difficult to design and deploy correctly: the data layer.

Data on the web is often fluid and loosely structured and it is becoming increasingly difficult to fit this data into a fixed database schema which is amended over time. A simple example of this is tagging. The many-to-many relationship of tags is difficult to query efficiently using tables and SQL, such that ad-hoc solutions are required.

Also, web data is often ”mashed up” and viewed together (e.g. Facebook profile) or viewed spatially (e.g. Google maps + event data).

In order to provide this new kind of data flexibility the web is moving towards a document-oriented data model, where records aren't grouped by their structure but by their attributes.

Object relational mapping libraries (e.g. Hibernate, Active Record) attempt to hide the underlying relational model but the fact remains that sparse and unnormalized data was not intended for these systems.

There are also standard data-oriented issues like indexing, caching, replication and backups, which are left for ”later” but are never easy to implement when it's time to do it. There are a number of great of open source solutions to these problems, but they require proper integration and configuration. These components end up being learned over time and learned by trial and error.

Thrudb, therefore, is an attempt to simplify the modern web data layer and provide the features and tools most web-developers need. These features can be easily configured or turned off.

Thrudb can be used on traditional hosting setups but is also intended to be deployed on Amazon's EC2 using Amazon S3 as its storage backend.

When designing Thrudb the following requirements were identified:

- *Modular.* Each service should be valuable on its own and impose no requirements on the others.
- *Simple.* The APIs must be obvious and work in most languages.
- *Fast.* The services must be written in a compiled language, be multi-threaded when necessary and use asynchronous processing when necessary.
- *Flexible.* The services should be capable of handling any type of document.
- *Easy to manage.* The configuration of these services should be simple and fairly obvious, in contrast to sendmail.cf.
- *Redundant.* The data should be replicable and provide multi-mastership writes.
- *Recoverable.* The services should be ACID compliant and provide redo logs and incremental backups.
- *Scalable.* The system should be horizontally scalable and provide write-through caching.

The resulting package includes the following services:

- *Thrudoc* - Document storage service. See Section 2.
- *Thrucene* - Document indexing service. See section 3.
- *Throxy* - Service proxy and partition aggregator. See section 4.

These services are built with the help of the following open source projects:

- *Thrift* - Facebook's framework for providing cross-language services. This is the backbone of Thrudb. [1]
- *Lucene* - Specifically CLucene, which provides high performance indexing. [2]
- *Spread* - Guaranteed messaging bus with reliable membership updates. [3]
- *Amazon Services* -Not open source but almost free scalable computing and storage platform. [4]
- *Memcached* - Highly scalable distributed caching service. [5]
- *Brackup* - Nifty backups to disk or S3. [6]

[1] See http://developers.facebook.com/thrift

[2] See http://clucene.sf.net

[3] See http://spread.org

[4] See http://www.amazonaws.com

[5] See http://danga.com/thrift

[6] See http://search.cpan.org/ bradfitz/Brackup-1.06/brackup

2. Thrudoc

Thrudoc is a simple key value storage system designed to work on a number of underlying storage backends, such as files on disk or S3 records. Its purpose is to provide simple scalable access to any stored object.

Let's start with a simple example in Perl. Since Thrudb is built on top of Thrift this client API is available natively in most languages.

```
######################################################
my $socket    = new Thrift::Socket('localhost',9091);
my $transport = new Thrift::FramedTransport($socket);
my $protocol  = new Thrift::BinaryProtocol($transport);
my $client    = new ThrudocClient($protocol);

$transport->open();

my $id = $client->store("a serialized doc");

print "Recieved: ".$client->fetch($id);

$client->store("an updated doc",$id);

$client->remove($id);
```

The *value* parameter can be a string, JSON object, XML object, or a serialized Thrift object.

It is recommend that you use Thrift objects because they offer backwards compatible object versioning.

Briefly, this means if your website launches and you forgot to include "address" in the user definition object but already have thousands of users, you don't recreate your old objects. Instead, simply add the address field to your Thrift object definition; it will support both old and new versions of your serialized object. [7]

Thrudb's identifiers are UUIDs by default. This removes the dependency on centralized or incremental identifiers (which are not scalable). You can specify a key identifier of your own, but a new ID will be returned that is the MD5 value of the key. This keeps the storage area balanced.

Note: there is no concept of different storage locations: everything is stored together in what appears to be one big bucket.

Thrudb supports multiple storage backends. Initially these include local disk storage or Amazon S3, but this could be easily extended.

Thrudb also offers caching, partitioning, and replication with multi-master writes, redo logs and backups. Each of these will be described later in this document.

3. Thrucene

Thrucene exposes the Lucene Search API as a Thrift service.

Its purpose is to provide powerful searching capabilities for your documents. This is intended to be used with the Thrudoc service but can be used to provide search for traditional database records as well.

The idea of your index living separately from your document isn't new. Google is built this way since an index store can scale very differently than your document store. For example, you may only be interested in searching a few fields within your documents or wish to keep multiple indexes on the same set of documents. This also makes it possible to perform parallel operations like MapReduce [8]

Hence, with Thrucene you can create multiple indexes and use them to search any relevant information about your documents.

You can build a complete index from a document store or you can incrementally update your index as you update your documents to simulate a traditional database layer.

Here's an example:

```
##########################################
#Create a bookmark
my $b = new Bookmark(); #thrift obj

$b->name("Thrudb: Document Services");
$b->url("http://thrudoc.googlecode.com");
$b->tags("database thrift web2.0");

my $bm_id = $thrudoc->store( $b->write() );

##########################################
#Index it
my $d = new Thrucene::DocMsg();

$d->domain( "tests"      );
$d->index (  "bookmarks" );
$d->docid ( $bm_id       );

my $field = new Thrucene::Field();

$field->name( "tags" );
$field->value( $b->tags );

push(@{$d->{fields}}, $field);

$thrucene->add( $d );
$thrucene->commitAll();
```

In this example we created a bookmark and indexed it, now we'll show how to search bookmarks. [9]

```
########################################
#Find some bookmarks
my $q = new Thrucene::QueryMsg();

$q->domain( "test" );
$q->index ( "bookmarks" );
$q->query ( "tags:(+database +web2.0)" );

my $r = $thrucene->query( $q );

print "Found documents".join(",",@{$r->{ids}});
```

The query returns the matching document IDs which can then be fetched from Thrudoc.

This service supports replication with multi-master writes, redo logging, partitioning, and backups.

Note, one great aspect of Thrucene is it's indexes can always be rebuilt from the document source. This allows us to comfortably

[7] See Appendix A for a further discussion on thrift objects.

[8] See http://code.google.com/edu/parallel/mapreduce-tutorial.html

[9] See http://lucene.apache.org/java/docs/queryparsersyntax.html for query syntax

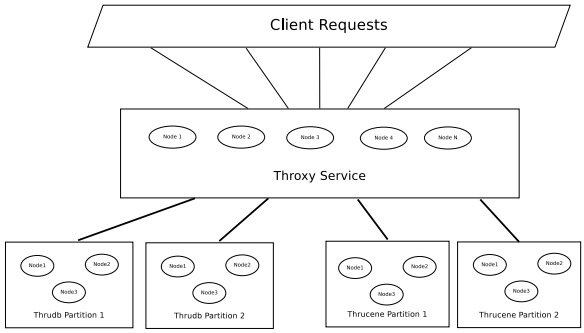
deploy on Amazon's Compute Cloud using Thrudoc's S3 backend for document storage with no fear of data loss.

4. Throxy

The Throxy service enables other Thrudb services to be partitioned and scaled horizontally. It sits between client and servers, partitioning writes and aggregating reads.

This system is completely asynchronous and takes advantage of Thrift's distinct send and receive methods for each service call.

Below is a diagram of all Thrudb services working together.



As mentioned earlier, all Thrudb services are optional. You can pick and choose the services and features you would like to integrate. By no means is this the required setup.

5. Replication and Multi-Mastership

Thrudoc and Thrucene both provide replication, for redundancy, and multi-master replication, for client simplicity, using Spread. Spread was written to simplify building services that can synchronize with other instances to offer high availability.

If Spread usage is enabled, your Thrudb services will communicate with each other via a Spread group and keep synchronized using two-phased commits.

You define a Spread group to join and the minimum number of cohorts required for writes to succeed. As usual, this feature can be turned off.

The details of the communication protocol are outside the scope of this document.

6. Caching

Thrudoc employs write-through caching of all documents, while Thrucene lets you cache particular queries.

The caching service is provided by Memcached.

Memcached is well known as the best distributed caching service, period. It lets the client distribute documents across instances rather than try to maintain document locations itself. It also offers compression and great stability.

The Memcached client used by Thrudb supports multiple hashing algorithms for distributing documents around the cache. Consistent hashing[10] is recommended since it provides highest hit rate.

Also, Memcached is currently used as the transport mechanism for replication as large documents would bog down Spread.

[10] See `http://en.wikipedia.org/wiki/Consistent_hashing`

7. Redo Logging and Backup

Redo logs are used for recovery purposes. For example, if you have three instances of Thrudoc replicating in the same group and one instance goes down, when it comes back up, one of the other instances will replay its redo log from the point the instance died to bring it back up to sync before it is officially online.

Thrift has a built-in file logging facility that makes it simple to create redo logs.

Complete backups of your documents or indexes are also taken systematically using Brackup.[11]

Brackup has implemented a number of storage backends, but the backup tool itself is abstracted and can be swapped altogether with something else.

8. Appendix

8.1 Thrift

To learn more about thrift please visit `http://developers.facebook.com/thrift`

8.2 Bloom filters

Thrudoc maintains a bloom filter[12] for quickly detecting if a document exists in its store. This becomes extremely important since reads affect performance, and in the case of S3 you get charged for them. If corrupted, this bloom filter can be recreated from the document source.

9. Thanks

I'd like to thank the open source community for providing such great software. It has made an ambitious project like this something one person can accomplish.

I'd also like to thank Amazon web services for providing affordable and scalable computing resources and storage.

[11] http://search.cpan.org/ bradfitz/Brackup-1.06/brackup

[12] See `http://en.wikipedia.org/wiki/Bloom_filter`