Apache Thrift

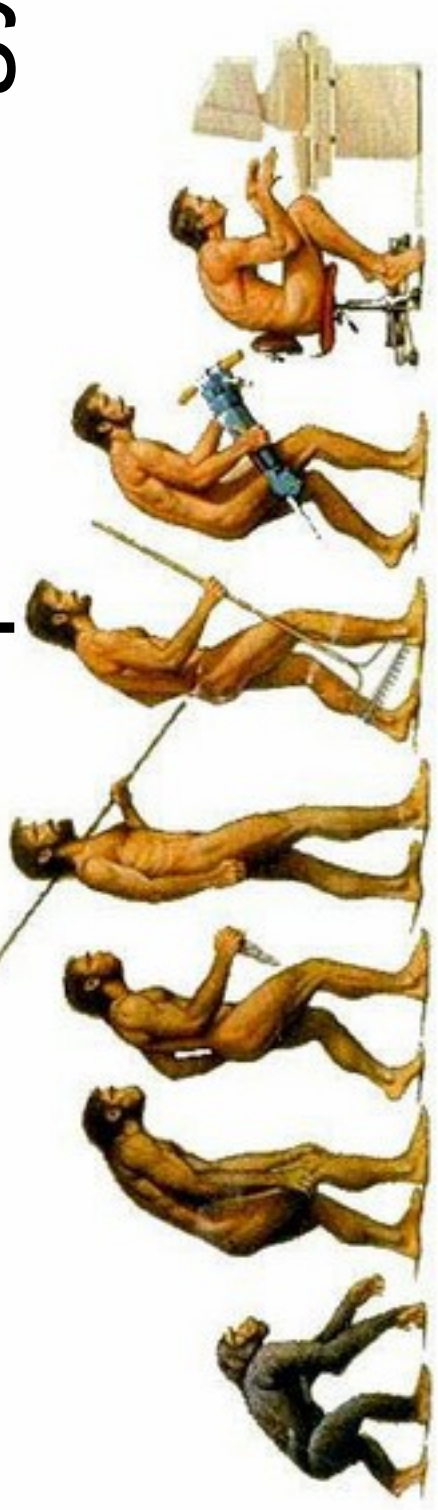
http://thrift.apache.org

Dave DiFranco
david.difranco@oracle.com
ddif@alum.mit.edu

Goals of Thrift

- Scalable, cross-language services development
  - Allow developers to use the right language for their problem
  - But talk to code written in other languages
    - (across the wire or not)
  - Without lots of performance overhead
  - Without lots of developer pain

Evolution of interop technology?



2007+: Thrift, Protocol Buffers, Avro...

2000: SOAP

1991: CORBA

1970: RPC

Other technologies

- **RPC** Implementations on UNIXes and other OSes starting in the 70s. Not cross platform.
- **CORBA** Comprehensive. Object-centric. Complicated and heavyweight.
- **SOAP** XML-based => parsing + network overhead
- **Protocol Buffers** Similar open source framework from Google
  - Doesn't include server infrastructure
  - Someone should do a presentation
- **Avro** (another) Apache framework
  - Schemas defined in JSON
  - Code gen not required
  - Someone should do a presentation

History of Thrift so far

- developed at Facebook
- open sourced in April 2007
- entered the Apache Incubator in May, 2008
- 2010-02-08 Thrift 0.6 released

Powered by Thrift

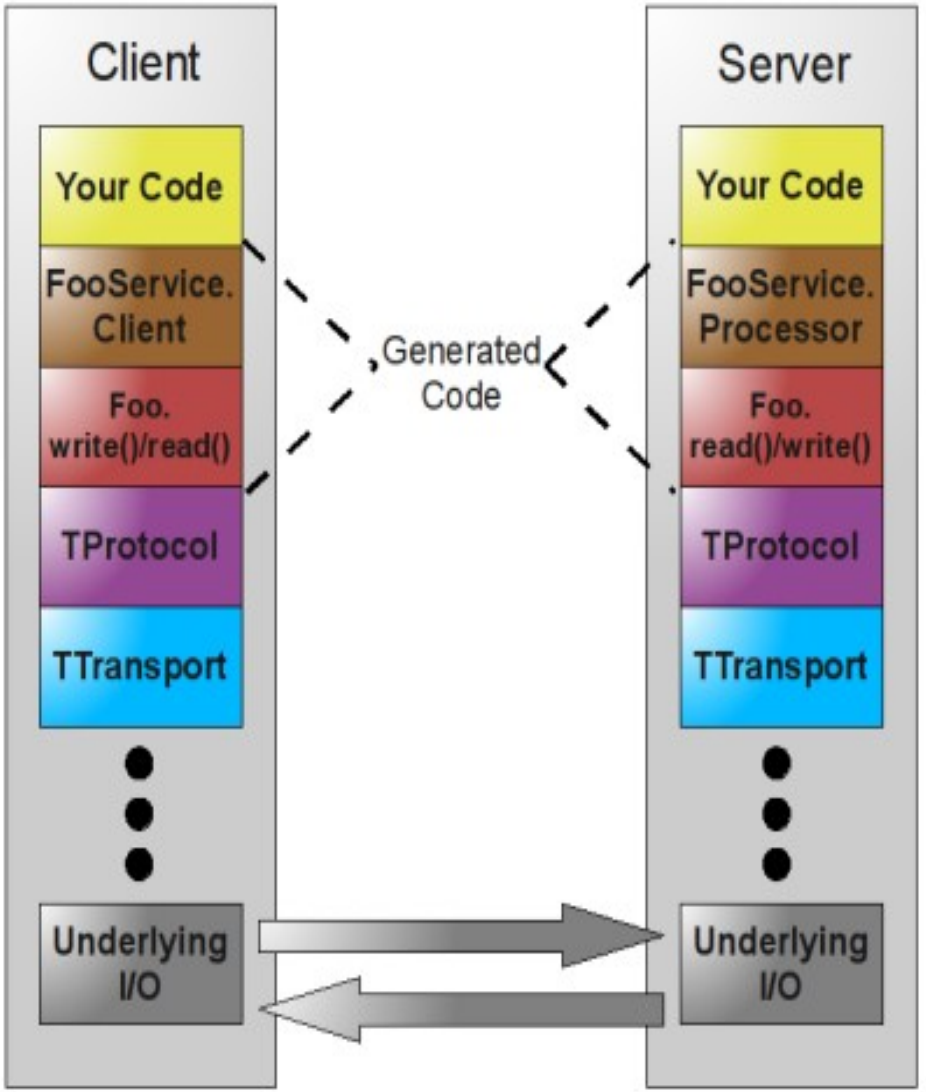
- www.facebook.com
- Apache Cassandra DB (client API)
- www.last.fm
- www.powerset.com
- www.recaptcha.com
- www.rapleaf.com
- www.amiestreet.com
- www.evernote.com
- www.esportnetwork.com
- www.openx.org/

Language Bindings

- C++
- Java
- Python
- PHP
- Ruby
- Erlang
- Perl
- Haskell
- C#
- Cocoa
- JavaScript
- Node.js
- Smalltalk
- OCaml

- But:
  - Not all language bindings support all protocols
  - Documentation, tutorials are spotty for some languages

Architecture



- Pluggable protocols:
  - Compact
  - Binary
  - JSON
  - Debug
  - ...
- Pluggable transports
  - Socket
  - File
  - Memory
  - ...

Server Infrastructure

- Provided servers:
  - Simple
  - ThreadPool
  - NonBlocking
- One service per server

Dev process

- Write service specification
- Generate language-specific server classes from the spec
- Write the server
- Generate language-specific client classes from the spec
- Write the client

Example: service spec

```
/**
 * FaceSpace profile
 */
struct User {
  1: i32 id,
  2: string name,
  3: list<i32> friendIDs
}


/**
 * FaceSpace service
 */
service Service  {

   /**
    * Create a user
    */
   void create(1:User user),

   /**
    * Return the number of users in all of FaceSpace
    */
   i32 countUsers()

}
```

Example: code generation

- thrift --gen erl facespace.thrift
- thrift --gen rb facespace.thrift

Example: generated code (Erlang)

```
-module(facespace_types).

-include("facespace_types.hrl").

-export([struct_info/1]).
%% struct user

% -record(user, {id, name, friendIDs}).

struct_info('user') ->
  {struct, [{1, i32},
  {2, string},
  {3, {list, i32}}]}
;

struct_info('i am a dummy struct') -> undefined.
```

```
-module(service_thrift).
-behaviour(thrift_service).


-include("service_thrift.hrl").

-export([struct_info/1, function_info/2]).

struct_info('i am a dummy struct') -> undefined.
%%% interface
% create(This, User)
function_info('create', params_type) ->
  {struct, [{1, {struct, {'facespace_types', 'user'}}}]}
;
function_info('create', reply_type) ->
  {struct, []};
function_info('create', exceptions) ->
  {struct, []}
;
% countUsers(This)
function_info('countUsers', params_type) ->
  {struct, []}
;
function_info('countUsers', reply_type) ->
  i32;
function_info('countUsers', exceptions) ->
  {struct, []}
;
function_info(xxx, dummy) -> dummy.
```

Example: server code (in Erlang)

```
-module(server).
-include("service_thrift.hrl").
-export([start/0, start/1, handle_function/2,
         stop/1, create/1, countUsers/0]).

debug(Format, Data) ->
    error_logger:info_msg(Format, Data).

%service methods
create(User) ->
    {_, UserID, _, _} = User,
     ets:insert('Users', {UserID, User}),
    debug("create(~p)",[User]),
    ok.

countUsers() ->
     ets:info('Users', size).

%%infrastructure methods
start() ->
    start(9999).
start(Port) ->
    Handler   = ?MODULE,
     ets:new('Users', [public, named_table]),
    debug("start",""),
    thrift_socket_server:start([{handler, Handler},
                                {service, service_thrift},
                                {port, Port},
                                {name, facespace_server}]).

stop(Server) ->
    thrift_socket_server:stop(Server).

handle_function(Function, Args) when is_atom(Function), is_tuple(Args) ->
    case apply(?MODULE, Function, tuple_to_list(Args)) of
        ok -> ok;
        Reply -> {reply, Reply}
    end.
```

Example: generated code (C++)

```
#ifndef facespace_TYPES_H
#define facespace_TYPES_H

#include <Thrift.h>
#include <protocol/TProtocol.h>
#include <transport/TTransport.h>

class User {
 public:
  static const char* ascii_fingerprint; // = "43193034EC8FD29153371776AF655
  static const uint8_t binary_fingerprint[16]; // = {0x43,0x19,0x30,0x34,0x
8F,0xD2,0x91,0x53,0x37,0x17,0x76,0xAF,0x65,0x5A,0x70};

  User() : id(0), name("") {
  }

  virtual ~User() throw() {}

  int32_t id;
  std::string name;
  std::vector<int32_t>  friendIDs;

  struct __isset {
    __isset() : id(false), name(false), friendIDs(false) {}
    bool id;
    bool name;
    bool friendIDs;
  } __isset;

  bool operator == (const User & rhs) const {
    if (!(id == rhs.id))
      return false;
    if (!(name == rhs.name))
      return false;
    if (!(friendIDs == rhs.friendIDs))
      return false;
    return true;
  }
  bool operator != (const User &rhs) const {
    return !(*this == rhs);
  }

  bool operator < (const User & ) const;

  uint32_t read(apache::thrift::protocol::TProtocol* iprot);
  uint32_t write(apache::thrift::protocol::TProtocol* oprot) const;
};
#endif
```

Example: generated code (C++)

```
#ifndef facespace_TYPES_H
#define facespace_TYPES_H

#include <Thrift.h>
#include <protocol/TProtocol.h>
#include <transport/TTransport.h>

class User {
 public:
  static const char* ascii_fingerprint; // =
"43193034EC8FD29153371776AF655
  static const uint8_t binary_fingerprint[16]; // =
{0x43,0x19,0x30,0x34,0x
8F,0xD2,0x91,0x53,0x37,0x17,0x76,0xAF,0x65,0x5A,0x70};

  User() : id(0), name("") {
  }

  virtual ~User() throw() {}

  int32_t id;
  std::string name;
  std::vector<int32_t>  friendIDs;

.....
```

```
#ifndef Service_H
#define Service_H

#include <TProcessor.h>
#include "facespace_types.h"



class ServiceIf {
 public:
  virtual ~ServiceIf() {}
  virtual void create(const User& user) = 0;
  virtual int32_t countUsers() = 0;
};

....
```

Example: server code (in C++)

```
class UserStorageHandler : virtual public UserStorageIf {
 public:
  UserStorageHandler() {
    // Your initialization goes here
  }

  void create(const UserProfile& user) {
    // Your implementation goes here
    printf("store\n");
  }

  int32_t count() {
    // Your implementation goes here
    printf("retrieve\n");
    return -1;
  }
};

int main(int argc, char **argv) {
  int port = 9999;
  shared_ptr<UserStorageHandler> handler(new UserStorageHandler());
  shared_ptr<TProcessor> processor(new UserStorageProcessor(handler));
  shared_ptr<TServerTransport> serverTransport(new TServerSocket(port));
  shared_ptr<TTransportFactory> transportFactory(new TBufferedTransportFactory());
  shared_ptr<TProtocolFactory> protocolFactory(new TBinaryProtocolFactory());
  TSimpleServer server(processor, serverTransport, transportFactory, protocolFactory);
  server.serve();
  return 0;
}
```

Example: server code (in Erlang)

```
-module(server).
-include("service_thrift.hrl").
-export([start/0, start/1, handle_function/2,
         stop/1, create/1, countUsers/0]).

debug(Format, Data) ->
    error_logger:info_msg(Format, Data).

%service methods
create(User) ->
    {_, UserID, _, _} = User,
     ets:insert('Users', {UserID, User}),
    debug("create(~p)",[User]),
    ok.

countUsers() ->
     ets:info('Users', size).

%%infrastructure methods
start() ->
    start(9999).
start(Port) ->
    Handler   = ?MODULE,
     ets:new('Users', [public, named_table]),
    debug("start",""),
    thrift_socket_server:start([{handler, Handler},
                                {service, service_thrift},
                                {port, Port},
                                {name, facespace_server}]).

stop(Server) ->
    thrift_socket_server:stop(Server).

handle_function(Function, Args) when is_atom(Function), is_tuple(Args) ->
    case apply(?MODULE, Function, tuple_to_list(Args)) of
        ok -> ok;
        Reply -> {reply, Reply}
    end.
```

Example: client code (in ruby)

```
#!/usr/bin/env ruby
$:.push('../gen-rb')
$:.unshift '~/workspace/thrift-instant-r760184/lib/rb/lib'

require 'thrift'
require 'thrift/protocol/binaryprotocol'
require 'service'

begin
  host = ARGV[0] || 'localhost'
  port = ARGV[1] || 9999

  puts "Enter a username to create"
  while (username = readline)
     #connect
     transport = Thrift::BufferedTransport.new(Thrift::Socket.new(host, port))
     protocol = Thrift::BinaryProtocol.new(transport)
     client = Service::Client.new(protocol)
     transport.open()

     #create user
     user = User.new()
     user.name = username
     user.id = rand(999999)
     user.friendIDs = [1111]
     print 'creating ', user.name
     client.create(user)

     #count users
     count = client.countUsers()
     print "count=", count, "\n"

     #close
     transport.close()

     puts "Enter a username to create"
  end
end
```

Example: client code (in python)

```
# Connect to the service (TCP sockets with binary protocol)
transport = TSocket.TSocket("localhost", 9999)
transport.open()
protocol = TBinaryProtocol.TBinaryProtocol(transport)
service = UserStorage.Client(protocol)

# Call the service to store a something
user = UserProfile(uid=1,
                   name="Zark Muckerberg",
                   nfriends="3755307”)
service.store(user)

# Call our service API to retrieve something
n = service.countUsers()
```

demo

Try it:
http://www.facebook.com/careers/puzzles.php

Do you like puzzles? So do we.

If you love puzzles like we do, become a fan of the new Puzzle Master Facebook Page . Notes are regularly posted to answer questions, explain puzzles, and announce new things. While you're here, try your hand at the following puzzles. The larger the difficulty, the harder it gets (hors d'oeuvres are simple tests to help you out).

Puzzles	Difficulty	Keyword
Hoppity Hop!	Hors d'oeuvre	hoppity
Meep meep!	Hors d'oeuvre	meepmeep
Liar, Liar	Snack	liarliar
Breathalyzer	Snack	breathalyzer
Gattaca	Snack	gattaca
Simon Says	Snack	simonsays
Dance Battle	Snack	dancebattle
It's A Small World	Snack	smallworld
User Bin Crash	Snack	usrbincrash
Rush Hour	Meal	rushhour
Battleship	Meal	battleship
Refrigerator Madness	Meal	fridgemadness
Peak Traffic	Meal	peaktraffic
We Are The Swarm	Meal	swarm

Submission directions

All submissions must execute in a *NIX type environment (sorry, no Windows specific solutions are accepted). You are not guaranteed any libraries or plugins beyond what is part of the language/interpreter itself. The following languages are accepted:

GNU C/C++ 4.2.3
Ericsson Erlang 5.5.5
GHC Haskell 6.8.2
Sun Java 1.5.0_15
INRIA OCaml 3.10.0
Perl 5.8.8
PHP 5.2.4
Python 2.5.2
Ruby 1.8.6

Some puzzles may require being solved with the following libraries:

Thrift r760184

Solved and got hired

Jonathan Hsu

Is the proud owner of the Puzzle Python.

Alan McConnell

I've seen puzzles you people wouldn't believe.

Tooling support

- Ant task: http://code.google.com/p/thriftc-task/
- Maven plugin: https://github.com/dtrott/maven-thrift-plugin
- Eclipse plugin: http://sourceforge.net/projects/thrift4eclipse/

References

- http://thrift.apache.org/
  - thrift download includes tutorial projects
- Technical paper: http://thrift.apache.org/static/thrift-20070401.pdf
- OCI article by Andrew Prunicki: http://jnb.ociweb.com/jnb/jnbJun2009.html
- Blog article by Alex Miller http://tech.puredanger.com/2011/05/27/serialization-comparison/

questions / comments