# **TASK REPORT**

## **Project Demo & Test**

Project Test sử dụng cho toàn bộ Thrift task được đưa lên github, server và client đều sử dụng java:

[**https://github.com/rotstitan/ThriftServerTest.git**](https://github.com/rotstitan/ThriftServerTest.git)

File Số liệu Test: [**ThriftServerTest.xlsx**](https://vngms-my.sharepoint.com/:x:/g/personal/haihq_vng_com_vn/EUCNAwMvAhFFrwv5y5iucEYBnS0qJqCz3sI2M2XoMP_Reg?e=gf6wuy)

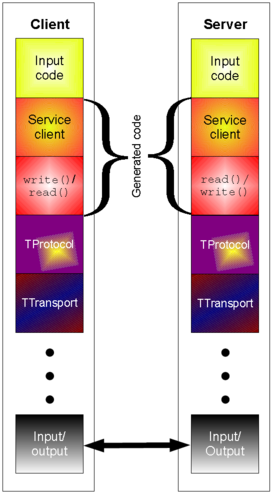
## **Tìm hiểu Apache Thrift và demo**

### **Giới thiệu Apache Thrift**

Thrift là một interface definition language và là một binary communication protocol. Thrift cung cấp chuẩn RPC (remote proceduce call) dùng để giao tiếp cho các service thông qua network (TCP). Thrift ban đầu được phát triển bởi facebook, và hiện tại là mã nguồn mở thuộc Apache project.

### **Kiến trúc**

Thrift sẽ cho phép chúng ta định nghĩa datatypes và service interface bằng ngôn ngữ trung lập (source file định dạng .thrift). Sau đó, Thrift compiler sẽ build source file này thành các file class tùy theo ngôn ngữ đã chọn. Ở server, chúng ta sẽ implement method từ file class Thrift, project client có thể “gọi” method mà server cung cấp thông qua service (interface) được định nghĩa sẵn.

**Hiện tại Thift hỗ trợ các ngôn ngữ bao gồm:** ActionScript, C, C++, C#, Cappuccino, Cocoa, Delphi, Erlang, Go, Haskell, Java, Node.js, Objective-C, OCaml, Perl, PHP, Python, Ruby, Smalltalk. **Protocol được hỗ trợ:** TBinaryProtocol, TCompactProtocol, TDebugProtocol, TDenseProtocol, TJSONProtocol, TSimpleJSONProtocol.

**Các giao thức tầng Transport được hỗ trợ:** TFileTransport, TFrameTransport, TMemoryTransport, TSocket, TZLibTransport.

**Thrift cung cấp một số mô hình Server:**

* TNonblockingServer: multi-threaded server sử dụng non-blocking I/O. Server này phải sử dụng TFrameTransport để có thể hoạt động.
* TSimpleServer: single-threaded server, sử dụng blocking I/O. Sử dụng vào mục đích test.
* TThreadPoolServer: multi-threaded server sử dụng blocking I/O

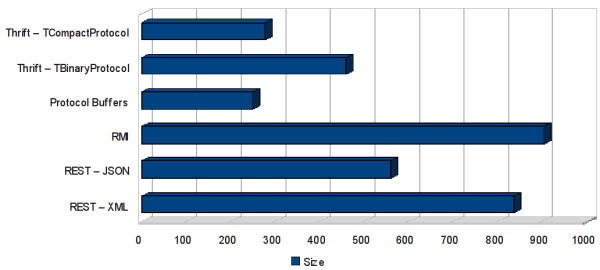
### **So sánh Thrift với một số công nghệ khác**

Các công nghệ được so sánh: Thrift và Probuf, XML, JSON, RMI

Cách thức thực hiện, kết quả bài so sánh được lấy từ nguồn:

<http://jnb.ociweb.com/jnb/jnbJun2009.html>

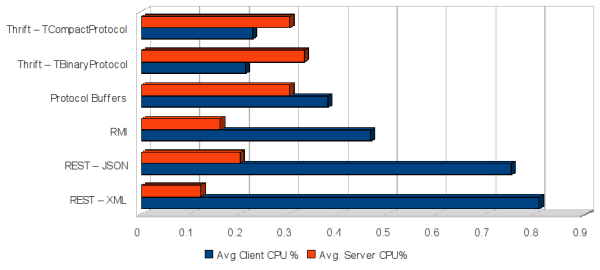
**Kích thước gói tin (object serialized size):**

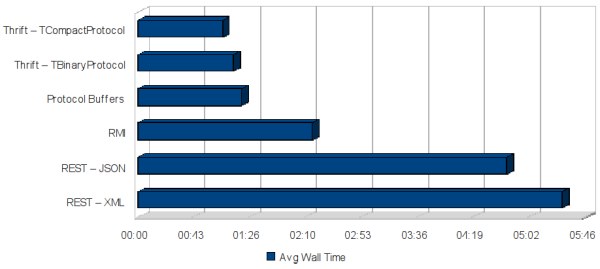


|  |  |  |
| --- | --- | --- |
|  | size | % lớn hơn TCompactProtocol |
| Thrift - TCompactProtocol | 278 | N/A |
| Thrift - TBinaryProtocol | 460 | 65.47% |
| Protocol Buffers | 250 | -10.07% |
| RMI | 905 | 225.54% |
| REST — JSON | 559 | 101.08% |
| REST — XML | 836 | 200.72% |

So với các công nghệ được so sánh thì Thrift có kích thước object truyền thấp thứ 2, chỉ sau Protocol Buffer.

**Performance:**





|  |  |  |  |
| --- | --- | --- | --- |
|  | Server CPU% | Avg Client CPU% | Avg Wall Time |
| REST - XML | 12.00% | 80.75% | 05:27.45 |
| REST - JSON | 20.00% | 75.00% | 04:44.83 |
| RMI | 16.00% | 46.50% | 02:14.54 |
| Protocol Buffers | 30.00% | 37.75% | 01:19.48 |
| Thrift - TBinaryProtocol | 33.00% | 21.00% | 01:13.65 |
| Thrift -TCompactProtocol | 30.00% | 22.50% | 01:05.12 |

Bài so sánh cho thấy: Thrift và Protocol buffers tiêu tốn nhiều xử lý cho server CPU nhất, và các công nghệ còn lại làm cho client CPU phải xử lý nhiều nhất. về tổng thời gian hoàn thành thì Thrift dẫn đầu tương ứng với hiệu năng ấn tượng nhất, tiếp theo đó là Protocol Buffers.

## **Nghiên cứu lựa chọn mô hình Thrift Server phù hợp nhất**

### **So sánh các loại mô hình thrift server**

* TSimpleServer: không hỗ trợ nhiều connection đồng thời nên chỉ nên sử dụng chủ yếu để test chứ không sử dụng cho sản phẩm thực tế.
* TNonblockingServer: Hỗ trợ nhiều connection đồng thời, mỗi thời điểm nhất định chỉ có 1 connection được select và execute nên chưa tối ưu được perfomance khi sử dụng trong các hệ thống lớn.
* THsHaServer (Half-Sync/Half-Async server): Sử dụng 1 thread cho I/O và nhiều thread cho execute nên chưa tối ưu được perfomance khi sử dụng trong các hệ thống lớn.
* TThreadedSelectorServer: Cho phép thiết lập nhiều Thread cho IO. Cung cấp 2 threadpool, 1 cho I/O và 1 cho Execute. Khắc phục được nhược điểm của TSimpleServer, THsHaServer và TNonblockingServer. Đặc biệt đạt được best performance khi Network I/O bị ngẽn cổ chai. Có thể được config để tối ưu hóa cho hầu hết các hệ thống lớn. Khả năng xảy ra lỗi thấp nhất trong bài test.
* TThreadPoolServer: Có nhiều Thread cho I/O không config được), và 1 threadPool cho Execute. 1 connection sẽ có 1 thread được dành riêng cho nó cho đến khi connection bị close. Do đó mô hình này sẽ kém hiệu quả khi tài nguyên hệ thống bị hạn chế. ThreadPoolServer đạt được best performmance trong bài test.

### **Lựa chọn công nghệ phù hợp**

* TThreadedSelectorServer được lựa chọn cho hầu hết các trường hợp phát triển hệ thống lớn. Bởi khả năng linh hoạt, dễ dàng tinh chỉnh cho mọi cấu hình hệ thống, khả năng bị lỗi thấp nhất trong bài test hiệu năng trong tất cả các mô hình server. Perfomance chỉ đứng sau TThreadPoolServer.
* TThreadPoolServer sẽ được lựa chọn trong trường hợp nắm được có bao nhiêu CCU và không quan tâm đến việc có bao nhiêu Thread ở Server được tạo ra. Bởi khả năng đạt được Best Perfomance của nó.

## **Test Thrift Server đáp ứng 100k req/s**

Sử dụng ThriftServerTest Project, với việc cho Server listen ở localhost và 1 client có nhiệm vụ tạo nhiều thread, mỗi thread sẽ gọi hàm liên tục lên Server, sau một khoảng thời gian nhất định, server có thể đạt được 100k req/s

## **Hiện thực connection pool, hiểu rõ các tham số**

### **Mô tả công việc**

Tham khảo các source code mẫu và tự xây dựng Client Connection Pool để quản lý connection trong client. Có các chức năng:

* Tạo một số lượng connection nhất định lúc ban đầu (ClientConfigs.InitConnection = 5).
* Mỗi khi client cần connection, connection sẽ được get từ pool.
* Nếu pool đã hết connection, pool sẽ tiến hành tạo thêm connection rồi sử dụng luôn connection đó.
* Nếu số lượng connection đạt max (ClientConfig.MaxConnection = 10) Thì sẽ không thể tạo connection và gửi đi nữa. Function đó sẽ bị reject.
* Khi có 1 connection thực hiện công việc xong. Connection đó sẽ được trả về lại Pool để được tái sử dụng.
* Khi không kết nối được với Server, Client sẽ tự động ping sau mỗi 1s. Khi kết nối thành công, Connection Pool sẽ được khởi tạo lại.
* Update 23/5: Sau khi lấy connection, nếu tổng số connection chưa đạt max và số lượng connection pool còn lại, nhỏ hơn min (ClientConfig.MinConnection) thì sẽ tiến hành tạo thêm 1 connection (connection sẽ được tạo ở thread khác nhằm làm tăng hiệu suất cho pool).

## **So sánh Thrift và HTTP**

### **Giống nhau**

Đều là công nghệ hỗ trợ communication giữa các entries qua internet.

Đều hỗ trợ kết nối liên tục (persistent connections), http hỗ trợ từ phiên bản 1.1

Đều hỗ trợ giao tiếp giữa các nền tảng, framework, ngôn ngữ lập trình khác nhau.

### **Khác nhau**

* Thrift có perfomance tổng thể tốt hơn HTTP
* Thrift gen client và server code tự động, bao gồm cả cấu trúc dữ liệu, không cần phải viết phần handler và invoke ở client. Các paramaters ở thrift sẽ tự động được validated và parsed.
* Thrift hỗ trợ truyền và xử lý Exceptions, Http phải tự define lại.
* Dữ liệu được gửi bởi thrift gọn nhẹ hơn, dễ dàng tích hợp mã hóa, nén, non-blocking IO...
* Thrift có thể được setup để sử dụng nhiều protocol, bao gồm cả http và json nếu nhà phát triển muốn.
* HTTP+JSON dễ debug hơn so với hầu hết các protocol được sử dụng bởi thrift.

### **Serializers performance giữa Thrift và JSON:**

Số liệu được lấy ở project:

<https://github.com/eishay/jvm-serializers/wiki>

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | create | ser | deser | total | size |
| thrift | 95 | 1455 | 731 | 2186 | 349 |
| thrift-compact | 97 | 1280 | 808 | 2088 | 240 |
| json/gson/manual | 53 | 2936 | 3641 | 6577 | 468 |