

Apache Avro 与 Thrift 比较

十二月 27th, 2010 | by fankong in 云计算 , 高性能服务器

9 comments

Avro和Thrift都是跨语言，基于二进制的高性能的通讯中间件. 它们都提供了数据序列化的功能和RPC服务. 总体功能上类似，但是哲学不一样. Thrift出自Facebook用于后台各个服务间的通讯,Thrift的设计强调统一的编程接口的多语言通讯框架. Avro出自Hadoop之父Doug Cutting, 在Thrift已经相当流行的情况下Avro的推出，其目标不仅是提供一套类似Thrift的通讯中间件更是要建立一个新的，标准性的云计算的数据交换和存储的Protocol。这个和Thrift的理念不同，Thrift认为没有一个完美的方案可以解决所有问题，因此尽量保持一个Neutral框架，插入不同的实现并互相交互。而Avro偏向实用，排斥多种方案带来的 可能的混乱，主张建立一个统一的标准，并不介意采用特定的优化。Avro的创新之处在于融合了显式,declarative的Schema和高效二进制的数据表达，强调数据的自我描述，克服了以往单纯XML或二进制系统的缺陷。Avro对Schema动态加载功能，是Thrift编程接口所不具备的，符合了Hadoop上的Hive/Pig及NOSQL 等既属于ad hoc，又追求性能的应用需求。



语言绑定

目前阶段Thrift比Avro支持的语言更丰富.
Thrift: C++, C#, Cocoa, Erlang, Haskell, Java, Ocami, Perl, PHP, Python, Ruby, Smalltalk.
Avro: C, C++, Java, Python, Ruby, PHP.

数据类型

从常见的数据类型的角度来说，Avro和Thrift非常接近，功能上并没有什么区别。

	Avro	Thrift	
基本类型			true or false
	N/A		8-bit signed integer
	N/A	l16	16-bit signed integer
	int	l32	32-bit signed integer
	long	l64	64-bit signed integer
	float	N/A	32-bit floating point
	double	double	64-bit floating point
	bytes	binary	Byte sequence
	string	string	Character sequence
复杂类型			
	record	struct	用户自定义类型
	enum	enum	
	array<T>	list<T>	
	N/A	set<T>	
	map<string,T>	map<T1,T2>	Avro map的key必须是string
	union	union	
	fixed	N/A	固定大小的byte array e.g. md5(16);
RPC服务			
	protocol	service	RPC服务类型
	error	exception	RPC异常类型
	namespace	namespace	域名



分类目录

- C/C++ (2)
- dw架构 (19)
- ETL (6)
- greenplum (9)
- hadoop (22)
- Hive (14)
- java (13)
- 云计算 (15)
- 展现 (12)
- 开发技术 (1)
- 所有 (126)
- 招聘 (6)
- 推荐引擎 (10)
- 数据挖掘 (32)
- 高性能服务器 (15)

最近文章

- HS4J Kit 介绍
- jvm垃圾回收
- Hive-如何基于分区优化
- Hive源码解析-之-语法解析器
- 你的数据服务 OUT 了吗

文章归档

选择月份

近期评论

- 之奇 在 Apache Avro 与 Thrift 比较 上的评论
- 圆通 在 HS4J Kit 介绍 上的评论
- 瘦身茶 在 HS4J Kit 介绍 上的评论
- 之奇 在 Hadoop现有测试框架探幽 上的评论
- NanguoCoffee 在 Apache Avro 与 Thrift 比较 上的评论

标签

不包含 个性化推荐 关联营销 决策树 千人千面 否定式前瞻 所有 推荐引擎 收藏 数据压缩 数据, 营销 日志扫描 校园招聘 正则表达式 淘宝 淘宝网 聚类 高性能服务器 aop buffer dsl
executor gc hadoop Hive
http://rdc.taobao.com/blog/dw/ io ioc

开发流程

从开发者角度来说，Avro和Thrift也相当类似，

1) 同一个服务分别用Avro和Thrift来描述

Avro.idl:

```
protocol SimpleService {
  record Message {
    string topic;
    bytes content;
    long  createTime;
    string id;
    string ipAddress;
    map<string> props;
  }
  int publish(string context,array<Message> messages);
}
```

Thrift.idl:

```
struct Message {
  1: string topic
  2: binary content
  3: i64  createTime
  4: string id
  5: string ipAddress
  6: map<string,string> props
}
```

```
service SimpleService {
  i32 publish(1:string context,2:list<Message> messages);
}
```

2) Avro和Thrift都支持IDL代码生成功能

```
java idl avro.idl idl.avro
```

```
java org.apache.avro.specific.SpecificCompiler idl.avro avro-gen
```

目标目录生成Message.java和SimpleService.java

```
thrift -gen java thrift.idl
```

同样的,目标目录生成Message.java和SimpleService.java

3) 客户端代码

Avro client :

```
URL url = new URL ( "http", HOST, PORT, "/");
```

```
Transceiver trans = new HttpTransceiver(url);
```

```
SimpleService proxy=
```

```
= (SimpleService)SpecificRequestor.getClient(SimpleService.class, transceiver);
```

```
...
```

Thrift client :

```
TTransport transport = new TFramedTransport(new TSocket(HOST,PORT));
```

```
TProtocol protocol = new TCompactProtocol(transport);
```

```
transport.open();
```

```
SimpleService.Client client = new SimpleService.Client(protocol);
```

```
...
```

4) 服务器端 Avro和Thrift都生成接口需要实现:

Avro server:

```
public static class ServiceImpl implements SimpleService {
  ..
}
```

```
Responder responder = new SpecificResponder(SimpleService.class, new ServiceImpl());
```

```
Server server = new HttpServer(responder, PORT);
```

itaobao, 关联推荐, 淘宝, 购物路径, java

logging mrunit mysql netty Nginx

nio node node.js nodejs

PostgreSQL python refactor test

zerocopy

Tech@Alibaba

淘宝DBA

淘宝QA

淘宝UED

淘宝核心系统

友情链接

CNodeJS.ORG

www.dbanotes.net

数据魔方官方网站

量子官方博客

阿里巴巴数据仓库团队blog

功能

登录

文章 RSS

评论 RSS

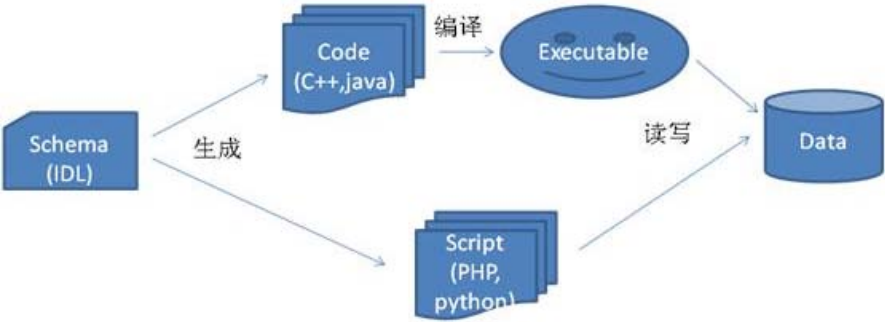
WordPress.org

Thrift server:

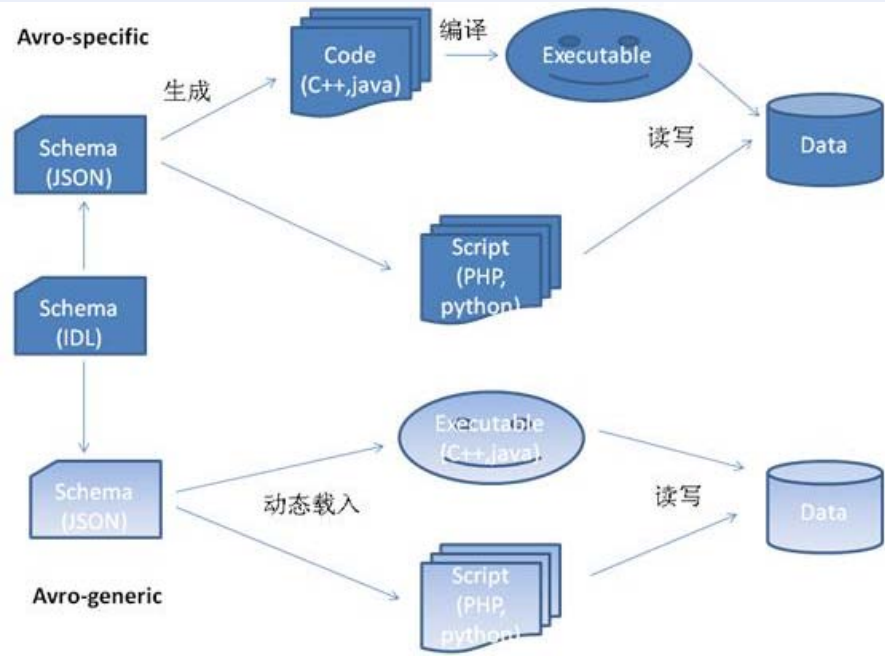
```
public static class ServerImpl implements SimpleService.Iface {  
    ..  
}  
TServerTransport serverTransport=new TServerSocket(PORT);  
TServer server=new TSimpleServer(processor,serverTransport,new TFramedTransport.Factory(), new  
TCompactProtocol.Factory());  
server.serve();
```

Schema处理

Avro和Thrift处理Schema方法截然不同。
Thrift是一个面向编程的系统, 完全依赖于IDL->Binding Language的代码生成。 Schema也“隐藏”在生成的代码中了, 完全静态。为了让系统识别处理一个新的数据源, 必须走编辑IDL, 代码生成, 编译载入的流程。

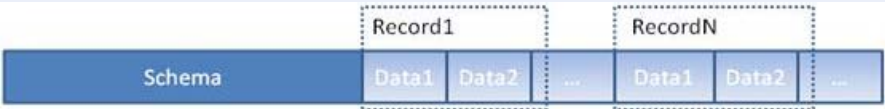


与此对照, 虽然Avro也支持基于IDL的Schema描述,但Avro内部Schema还是显式的,存在于JSON格式的文件当中, Avro可以把IDL格式的Schema转化成JSON格式的。
Avro支持2种方式。Avro-specific方式和Thrift的方式相似, 依赖代码生成产生特定的类, 并内嵌JSON Schema. Avro-generic方式支持Schema的动态加载, 用通用的结构(map)代表数据对象, 不需要编译加载直接就可以处理新的数据源。



Serialization

对于序列化Avro制定了一个协议, 而Thrift的设计目标是一个框架, 它没有强制规定序列化的格式。
Avro规定一个标准的序列化的格式, 即无论是文件存储还是网络传输, 数据的Schema(in JASON)都出现在数据的前面。数据本身并不包含任何Metadata(Tag). 在文件储存的时候, schema出现在文件头中。在网络传输的时候Schema出现在初始的握手阶段.这样的好处一是使数据self describe,提高了数据的透明度和可操作性, 二是减少了数据本身的信息量提高存储效率, 可谓一举二得了

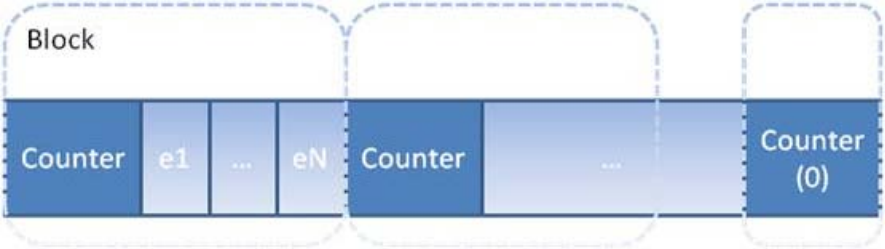


Avro的这种协议提供了很多优化的机会：

- 对数据作**Projection**，通过扫描**schema**只对感兴趣的部分作反序列化。
- 支持**schema**的**versioning**和**mapping** ,不同的版本的**Reader**和**Writer**可以通过查询**schema**相互交换数据(**schema**的**aliases**支持**mapping**),这比**thrift**采用的给每个域编号的方法优越多了

Avro的**Schema**允许定义数据的排序**Order**并在序列化的时候遵循这个顺序。这样的话不需要反序列化就可以直接对数据进行排序,在**Hadoop**里很管用。

另外一个Avro的特性是采用**block**链表结构，突破了用单一整型表示大小的限制。比如**Array**或**Map**由一系列**Block**组成，每个**Block**包含计数器和对应的元素，计数器为0标识结束。



Thrift提供了多种序列化的实现：

TCompactProtocol: 最高效的二进制序列化协议，但并不是所有的绑定语言都支持。

TBinaryProtocol: 缺省简单二进制序列化协议。

与Avro不同，Thrift的数据存储的时候是每个**Field**前面都是带**Tag**的，这个**Tag**用于标识这个域的类型和顺序ID（IDL中定义，用于**Versioning**）。在同一批数据里面，这些**Tag**的信息是完全相同的，当数据条数大的时候这显然就浪费了。



RPC服务

Avro提供了

HttpServer : 缺省,基于**Jetty**内核的服务。

NettyServer: 新的基于**Netty**的服务。

Thrift提供了：

TThreadPolServer: 多线程服务

TNonBlockingServer: 单线程 non blocking 的服务

THsHaServer: 多线程 non blocking 的服务

Benchmarking

测试环境：2台4核 Intel Xeon 2.66GHz， 8G memory, Linux, 分别做客户端，服务器。

Object definition:

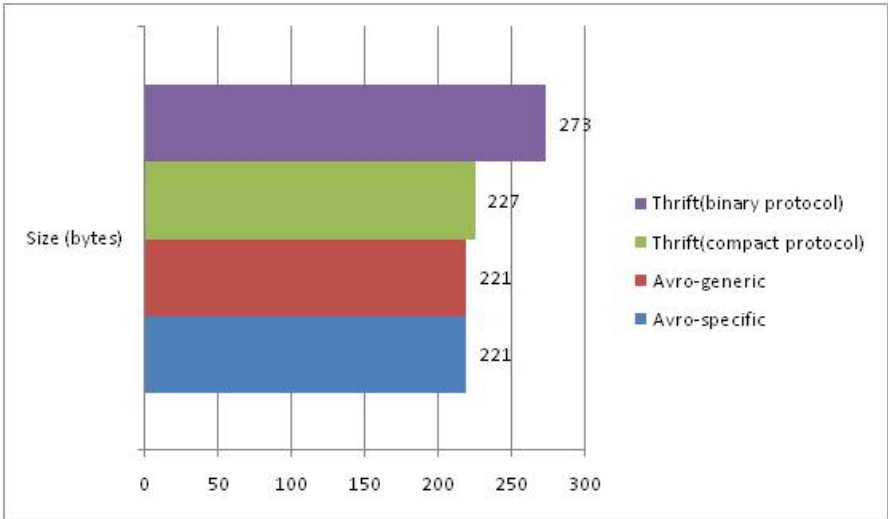
```
record Message {  
  string topic;  
  bytes payload;  
  long createdTime;  
  string id;  
  string ipAddress;  
  map<string,string > props;  
}
```

Actual instance:

```
msg.createdTime : System.nanoTime();  
msg.ipAddress : "127.0.0.1";  
msg.topic : "pv";
```

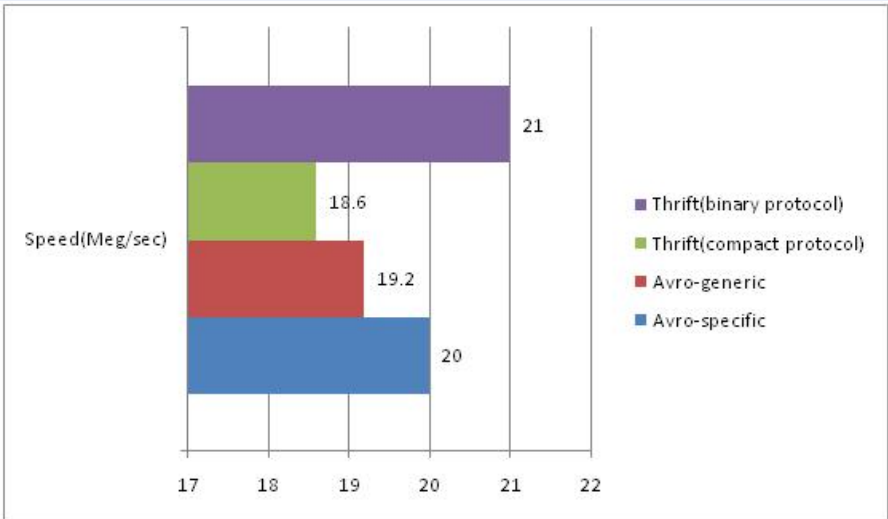
```
msg.payload : byte[100]
msg.id : UUID.randomUUID().toString();
msg.props : new HashMap<String,String>();
msg.props.put("author", "tjerry");
msg.props.put("date", new Date().toString());
msg.props.put("status", "new");
```

Serialization size



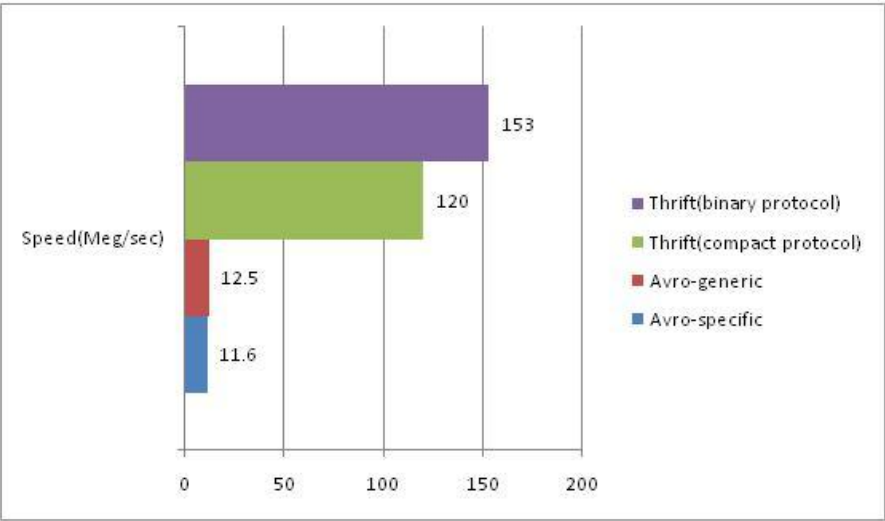
Avro的序列化产生的结果最小

Serialization speed



Thrift-binary 因为序列化方式简单反而看上去速度最快.

Deserialization speed



这里 Thrift的速度很快， 因与它内部实现采用zero-copy的改进有关.不过在RPC综合测试里这一优势似乎并未体现出来.

序列化测试数据采集利用了<http://code.google.com/p/thrift-protobuf-compare/>所提供的框架, 原始输出:

Starting

	Object create,	Serialize, /w Same Object,	Deserialize, and Check Media,	and Check All,	Total Time, Serialized Size
avro-generic	, 8751.30500,	10938.00000,	1696.50000,	16825.00000,	16825.00000,
	16825.00000,	27763.00000,	221		
avro-specific	, 8566.88000,	10534.50000,	1242.50000,	18157.00000,	18157.00000,
	18157.00000,	28691.50000,	221		
thrift-compact	, 6784.61500,	11665.00000,	4214.00000,	1799.00000,	1799.00000,
	1799.00000,	13464.00000,	227		
thrift-binary	, 6721.19500,	12386.50000,	4478.00000,	1692.00000,	1692.00000,
	1692.00000,	14078.50000,	273		

RPC测试用例:

客户端向服务器发送一组固定长度的message,为了能够同时测试序列和反序列, 服务器收到后将原message返回给客户端.

array<Message> publish(string context, array<Message> messages);

测试使用了Avro Netty Server和 Thrift HaHa Server因为他们都是基于异步IO的并且适用于高并发的环境。 结果

Message	payload	10 bytes	(total 130 bytes)				payload	100 bytes	(total 130 bytes)
1 Tx =	100 Msgs	(RT)	~13KB					~21 KB	
Load	1 Thread						Load	1 Thread	
	TPS	Server	Client	Server	Client			TPS	Server
		CPU%	CPU%	Memory(M)	Memory(M)				Client
Avro	920	8	8	88	93		Avro	760	
Thrift	510	9	10	64	57		Thrift	550	
Load	5 Thread						Load	5 Thread	
	TPS	Server	Client	Server	Client			TPS	Server
		CPU%	CPU%	Memory(M)	Memory(M)				Client
Avro	3400	31	33	166	167		Avro	2800	
Thrift	2400	32	40	203	140		Thrift	2100	
Load	10 Thread						Load	10 Thread	
	TPS	Server	Client	Server	Client			TPS	Server
		CPU%	CPU%	Memory(M)	Memory(M)				Client
Avro	5500	48	54	278	271		Avro	4400	
Thrift	3500	45	47	222	223		Thrift	3100	
Load	20 Thread						Load	20 Thread	
	TPS	Server	Client	Server	Client			TPS	Server
		CPU%	CPU%	Memory(M)	Memory(M)				Client
Avro	6900	60	72	279	288		Avro	5100	
Thrift	4400	59	75	220	233		Thrift	4000	
Load	50 Thread						Load	50 Thread	
	TPS	Server	Client	Server	Client			TPS	Server
		CPU%	CPU%	Memory(M)	Memory(M)				Client
Avro	6600	58	78	291	328		Avro	5200	
Thrift	4700	65	84	222	252		Thrift	4300	
Load	100 Thread						Load	100 Thread	
	TPS	Server	Client	Server	Client			TPS	Server
		CPU%	CPU%	Memory(M)	Memory(M)				Client
Avro	6100	53	79	312	475		Avro	5100	
Thrift	4600	63	82	255	323		Thrift	4100	

Payload	500 bytes (total 620 bytes)					Payload	1000 byte (total 1200 bytes)				
	~61KB						~109 KB				
Load	1 Thread					Load	1 Thread				
	TPS	Server	Client	Server	Client		TPS	Server	Client	Server	Client
		CPU%	CPU%	Memory(M)	Memory(M)			CPU%	CPU%	Memory(M)	Memory(M)
Avro	438	7	7	96	91	Avro	280	7	7	96	91
Thrift	370	7	7	62	50	Thrift	260	7	7	62	50
Load	5 Thread					Load	5 Thread				
	TPS	Server	Client	Server	Client		TPS	Server	Client	Server	Client
		CPU%	CPU%	Memory(M)	Memory(M)			CPU%	CPU%	Memory(M)	Memory(M)
Avro	1600	25	28	231	225	Avro	1000	25	28	231	225
Thrift	1400	26	25	210	175	Thrift	1000	26	25	210	175
Load	10 Thread					Load	10 Thread				
	TPS	Server	Client	Server	Client		TPS	Server	Client	Server	Client
		CPU%	CPU%	Memory(M)	Memory(M)			CPU%	CPU%	Memory(M)	Memory(M)
Avro	1800	27	33	300	300	Avro	1000	27	33	300	300
Thrift	1800	31	34	234	213	Thrift	1000	31	34	234	213
Load	20 Thread					Load	20 Thread				
	TPS	Server	Client	Server	Client		TPS	Server	Client	Server	Client
		CPU%	CPU%	Memory(M)	Memory(M)			CPU%	CPU%	Memory(M)	Memory(M)
Avro	1800	27	36	280	315	Avro	1000	27	36	280	315
Thrift	1800	31	34	246	259	Thrift	1000	31	34	246	259
Load	50 Thread					Load	50 Thread				
	TPS	Server	Client	Server	Client		TPS	Server	Client	Server	Client
		CPU%	CPU%	Memory(M)	Memory(M)			CPU%	CPU%	Memory(M)	Memory(M)
Avro	1800	27	36	270	360	Avro	1000	27	36	270	360
Thrift	1800	31	36	215	227	Thrift	1000	31	36	215	227
Load	100 Thread					Load	100 Thread				
	TPS	Server	Client	Server	Client		TPS	Server	Client	Server	Client
		CPU%	CPU%	Memory(M)	Memory(M)			CPU%	CPU%	Memory(M)	Memory(M)
Avro	1800	27	39	287	433	Avro	1000	27	39	287	433
Thrift	1800	36	39	255	284	Thrift	1000	36	39	255	284

从这个测试来看，再未到达网络瓶颈前，Avro Netty比Thrift HsHa服务提供了更高的吞吐率和更快的响应，另外 avro占用的内存高些。

通过进一步实验,发现不存在绝对的Avro和Thrift服务哪一个更快,决定于给出的test case,或者说与程序的用法有关,比如当前测试用例是Batch模式，大量发送fine grained的对象(接近后台tt,hadoop的用法),这个情况下Avro有优势. 但是对于每次只传一个对象的chatty客户端,情况就出现逆转变成Thrift更高效了.还有当数据结构里blob比例变大的情况下,Avro和Thrift的差别也在减小.

Conclusion

- Thrift适用于程序对程序静态的数据交换，要求schema预知并相对固定。
- Avro在Thrift基础上增加了对schema动态的支持且性能上不输于Thrift。
- Avro显式schema设计使它更适用于搭建数据交换及存储的通用工具和平台,特别是在后台。
- 目前Thrift的优势在于更多的语言支持和相对成熟。



◀ Previous Post

Next Post ▶

Comments (9)



raymond 09:26下午 十二 28,2010



interesting review, which one Taobao select ?



fankong 03:55下午 十二 29,2010
现在用的是thrift



peter 10:33上午 一 6,2011
总结的非常不错，有一点疑问， thrift HsHa服务器性能不应该有这么差， 小数据量，100线程，应该有4w+。



kangsg219 09:52上午 四 13,2011
和pb比较他们性能如何？



chenxiaojian 01:26上午 五 22,2011
能告诉我thrift有没有可以在win32运行的版本？我在thrift官网上下了一个，在我的xp系统下结果不好使



NanguoCoffee 11:17上午 六 21,2011
博主可以都用Netty Server来测试Avro和Thrift的序列化性能呀



之奇 10:14上午 六 23,2011
有趣的比较，知道两者的区别，才好选择。

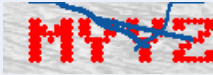
Add a Comment

Your Name

Email

Website

验证码



Submit