

高手如何实践HBase？不容错过的滴滴内部技巧

HBase技术社区 诸葛子房的博客 7月17日

HBase和Phoenix的优势大家众所周知，想要落地实践却问题一堆？replication的随机发送、Connection的管理是否让你头痛不已？本次分享中，滴滴以典型的应用场景带大家深入探究HBase和Phoenix，并分享内核改进措施。

本文将围绕一下几个方面进行介绍：

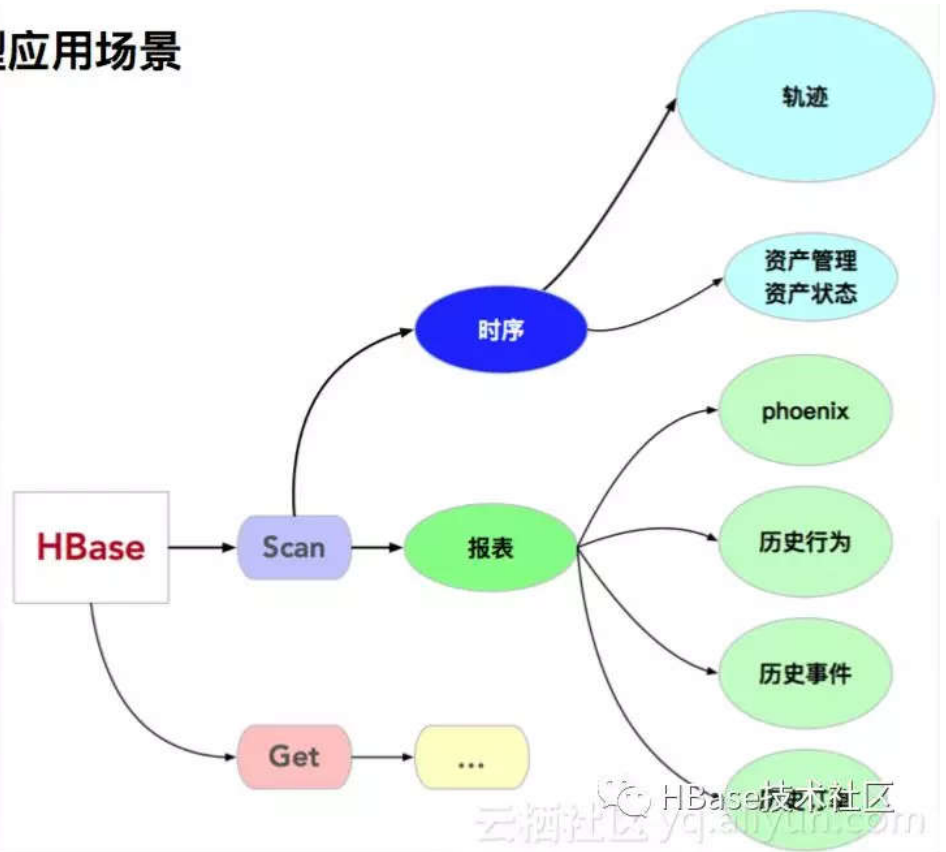
1. HBase特性应用与内核改进
2. Phoenix改进与实践
3. GeoMesa应用简介与展望
4. 稳定性&容量规划

一. HBase特性应用与内核改进

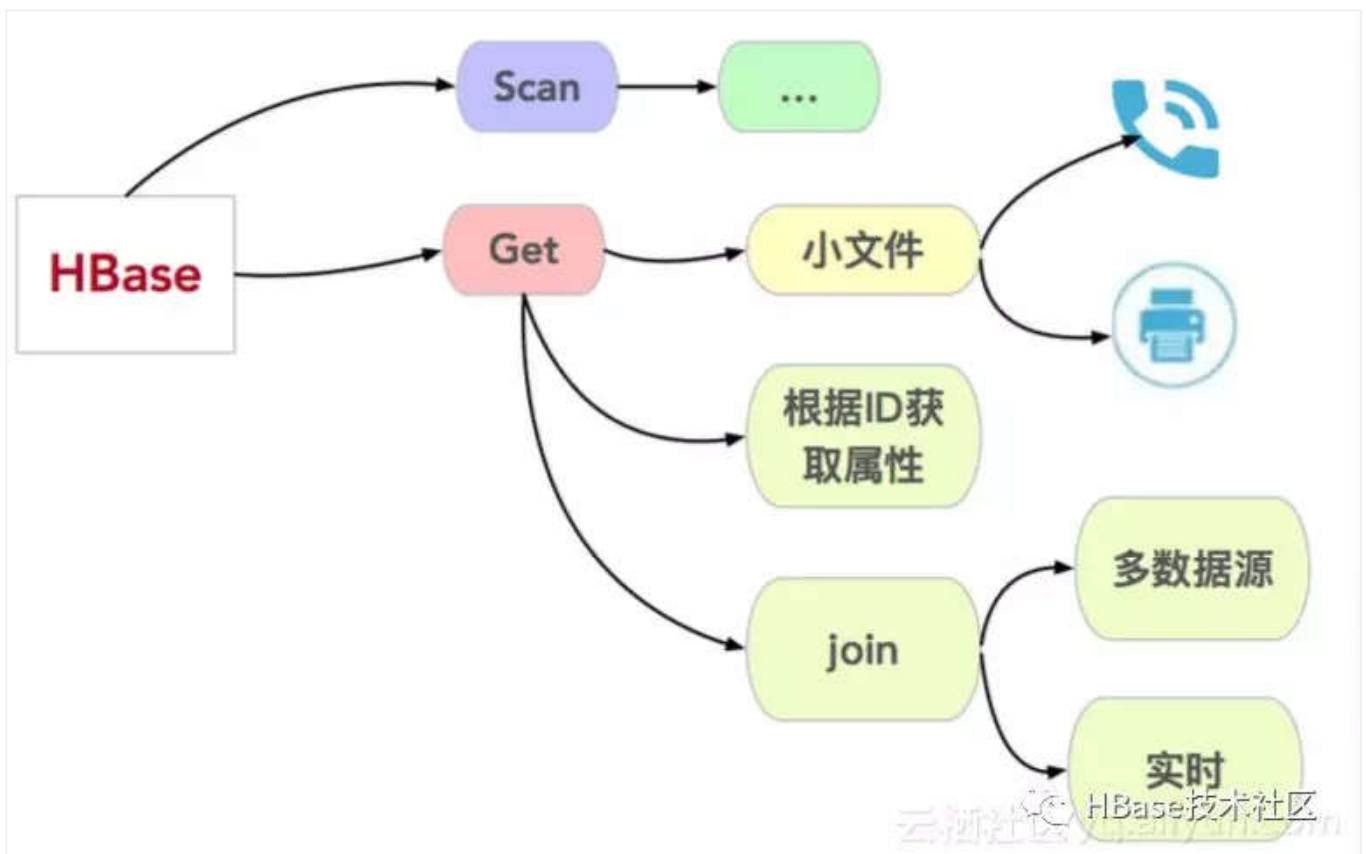
1. HBase在滴滴的典型应用场景

滴滴中有一些对HBase简单操作，例如Scan和Get。每一个操作可以应用于不同的场景，例如Scan可以衍生出时序和报表。时序可以应用到轨迹设计中，将业务ID、时间戳和轨迹位置作为整体建立时序。另外在资产管理中，将资产状态分为不同阶段，将资产ID、时间戳、资产状态等信息建立时序。Scan在报表中应用也非常广泛。其实现有多种方式，主流方法是通过phoenix，使用标准的SQL操作Hbase做联机事务处理，该方法中需要注意主键及二级索引设计。报表中会以用户历史行为、历史事件及历史订单为需求进行详细设计。

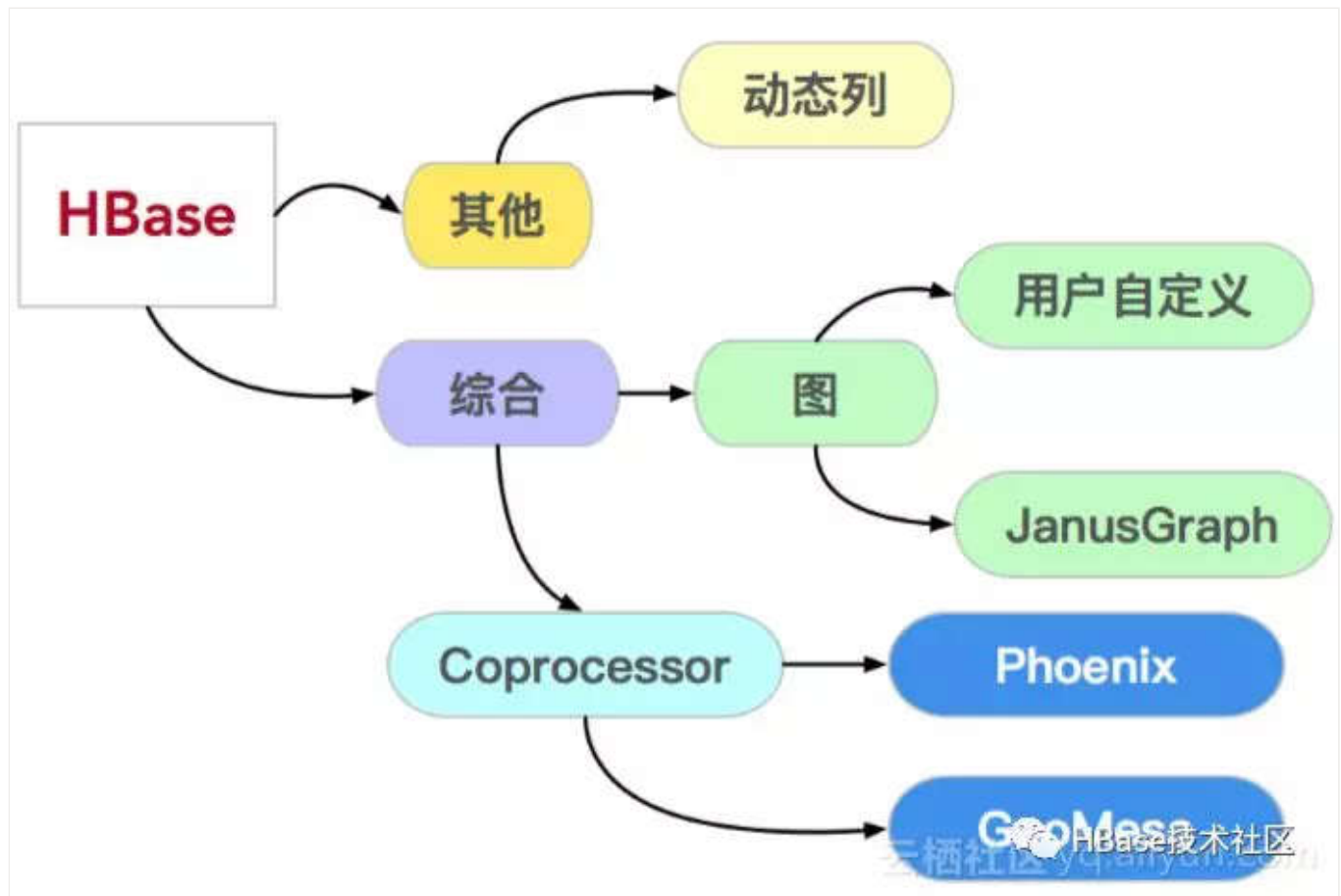
1.HBase在滴滴的典型应用场景



Get操作可以应用于HBase中存储的语音和滴滴发票等小文件中。最基本的应用方法为根据ID获取实体属性。更深入的例如可以应用于join操作，例如在实时计算中有多个数据流需要合并，此时的ID即为HBase中的rowkey。另例如业务上游存在多个数据源，需要将这多个数据源数据聚合至一个表中。

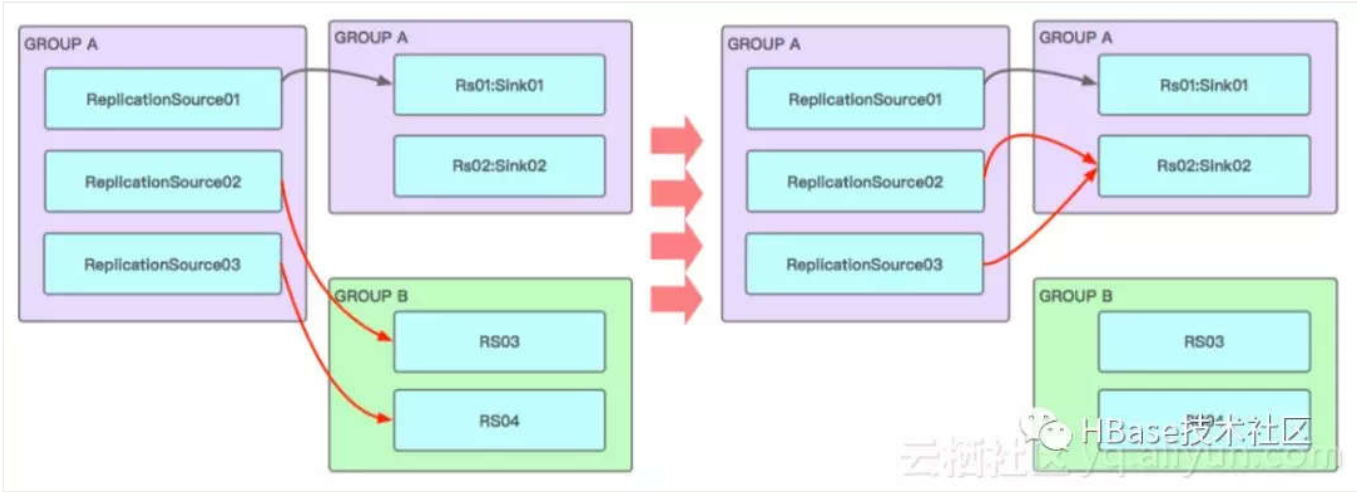


此外，HBase中仍衍生出一些其他操作。互联网公司需求变化快速，介入业务方众多，可以通过动态列帮助实现这类需求。另有一些综合应用，例如图和Coprocessor。图包括用户自定义的图，可以自定义数据来源与数据分配。HBase集群中也接入了JanusGraph。Coprocessor主要应用于Phoenix和GeoMesa。



2. replication 的应用与优化

假设原集群有三个主机：ReplicationSource01, ReplicationSource02, ReplicationSource03，目标集群有四个：RS01, RS02, RS03, RS04。若原集群发送replication请求，传统的逻辑会随机发送该请求。若目标集群的表存储在GROUP A中的两个主机上，但随机发送却有可能导致这两个主机接收不到replication请求，而是发送至和该业务无关的GROUP中。因此这里对此作出优化，对执行策略进行适当修改，将可能发送到其他集群的请求在原集群中进行匹配，获取目标集群GROUP的分配，使得请求可以发送至对应的GROUP主机中，防止影响其他业务。



此外，未来希望在replication中增加table级别的信息统计，统计请求的连接错误信息，从用户角度进行优化。

3. Connection的管理与使用

基于滴滴如今的HBase版本，用户使用中会出现关于Connection的一些问题，例如建立了多个Connection后，对应的ZK也会非常多，因此需要对Connection进行管理。这里采用在RS中创建ClusterConnection来尽量减少Connection的创建。这会应用在Phoenix二级索引场景中，详情在Phoenix部分介绍。

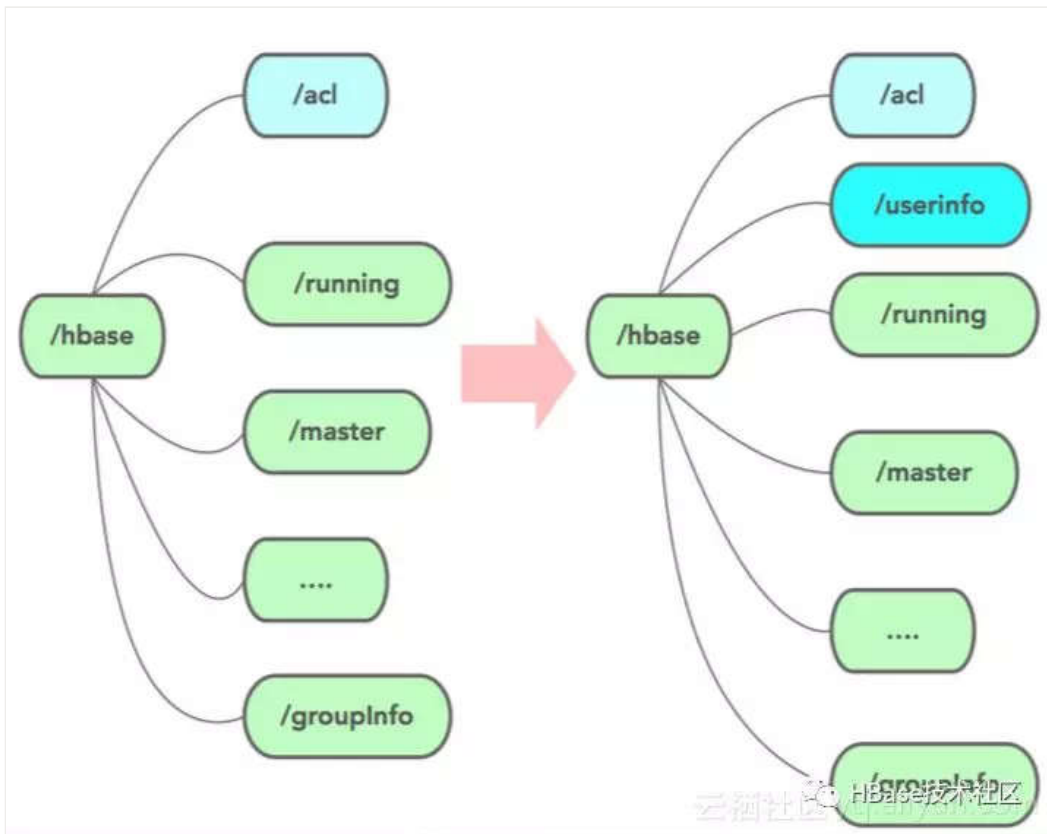
4. ACL权限认证的优化

ACL主要实现用户的用户名、密码与IP匹配的功能。此处创建了userinfo来存储用户密码信息，rowkey为用户名，CF(Column Family)为对应的密码。HBase:ACL这个表，在ZK中有/acl节点，类似的建立了userinfo节点。

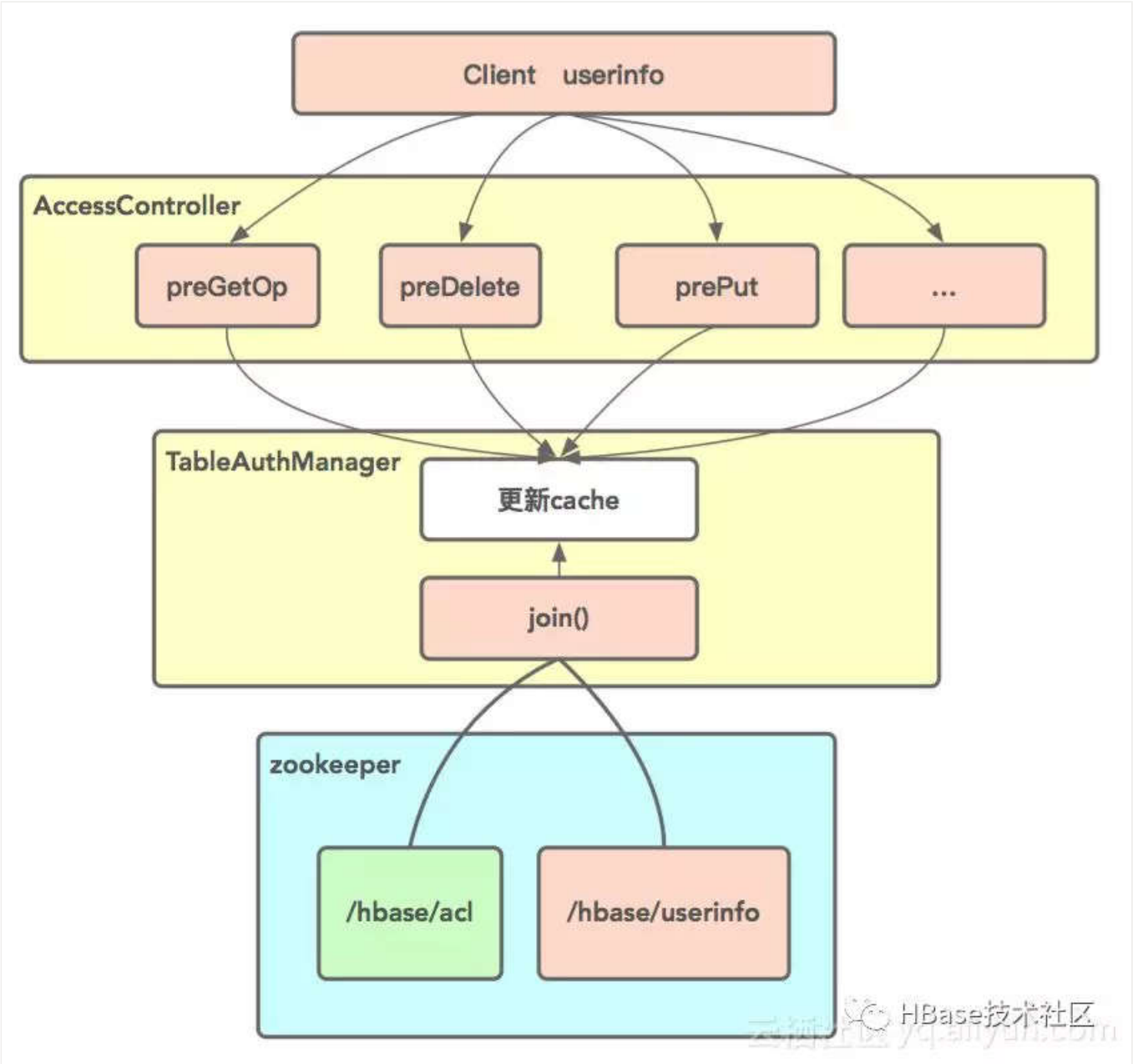
hbase:userinfo

rowkey(username)	CF: l:p
hadoop	hadoop_password
username01	user01_password
username02	...
username03	...

A watermark 'HBase技术社区' is visible at the bottom right of the table.



下图所示为userinfo接入流程。最上方是用户客户端封装后的userinfo序列化信息，发送至服务器端，服务器通过AccessController进行一些操作，例如preGetOp, preDelete和prePut等。然后TableAuthManager进行权限判断。TableAuthManager类中会存储权限信息cache，当接收到新的userinfo时，首先会更新cache，供客户端访问时调用。传统cache只包含/hbase/acl信息，滴滴优化后增添了/hbase/userinfo信息。那么此时更新cache需要/acl信息和/userinfo信息进行Join()操作。

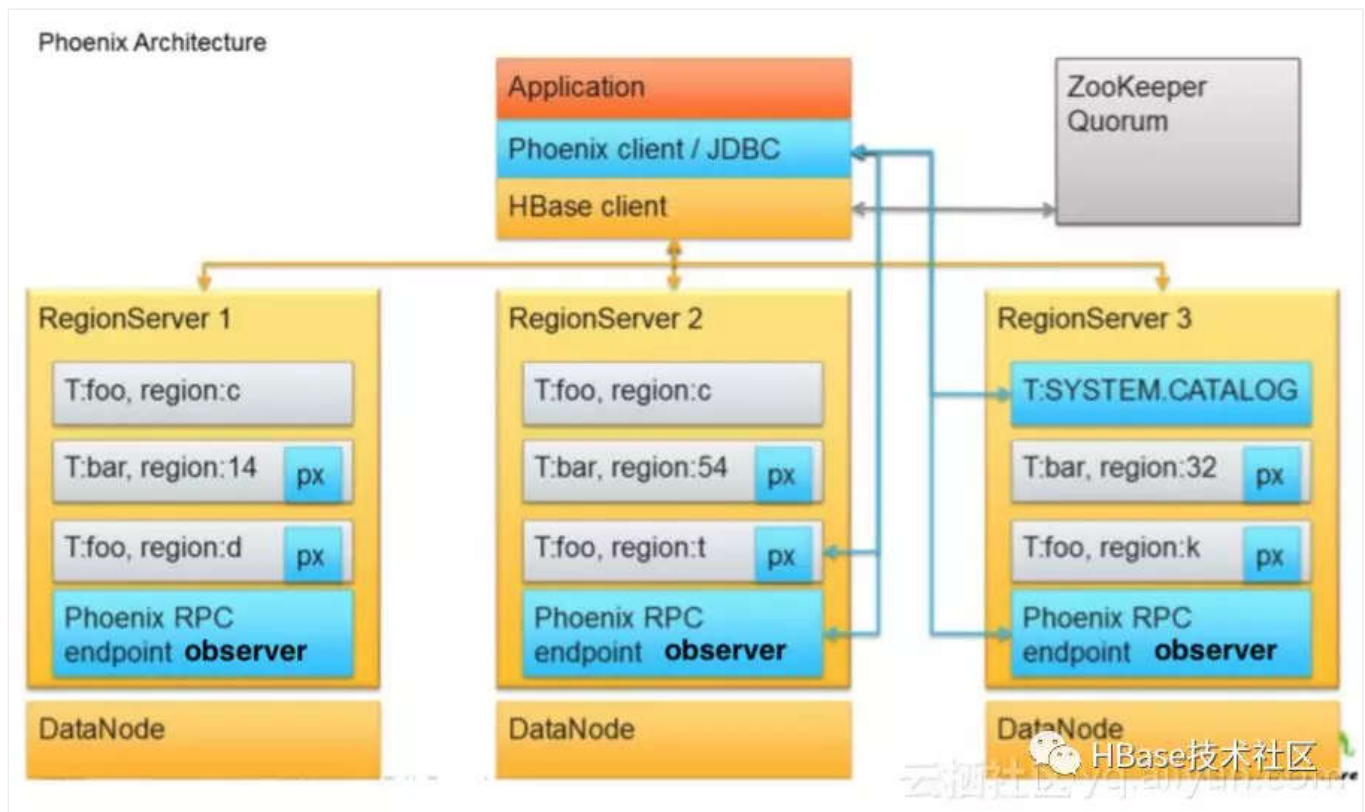


但是在使用原生/acl时也遇到一些问题，究其原因，还是需要减轻对zookeeper的依赖。此外，滴滴还对其他方面进行了优化，包括RPC audit log, RSGroup, quota, safe drop table等。RPC audit log可以对用户请求量及错误信息进行分析。quato可以限制用户流量，例如限制用户对某个表每秒内可以执行多少次put操作。另外在drop table时，传统方式为直接清理表格内容，现优化为先存储一份快照再删除，防止误删操作。

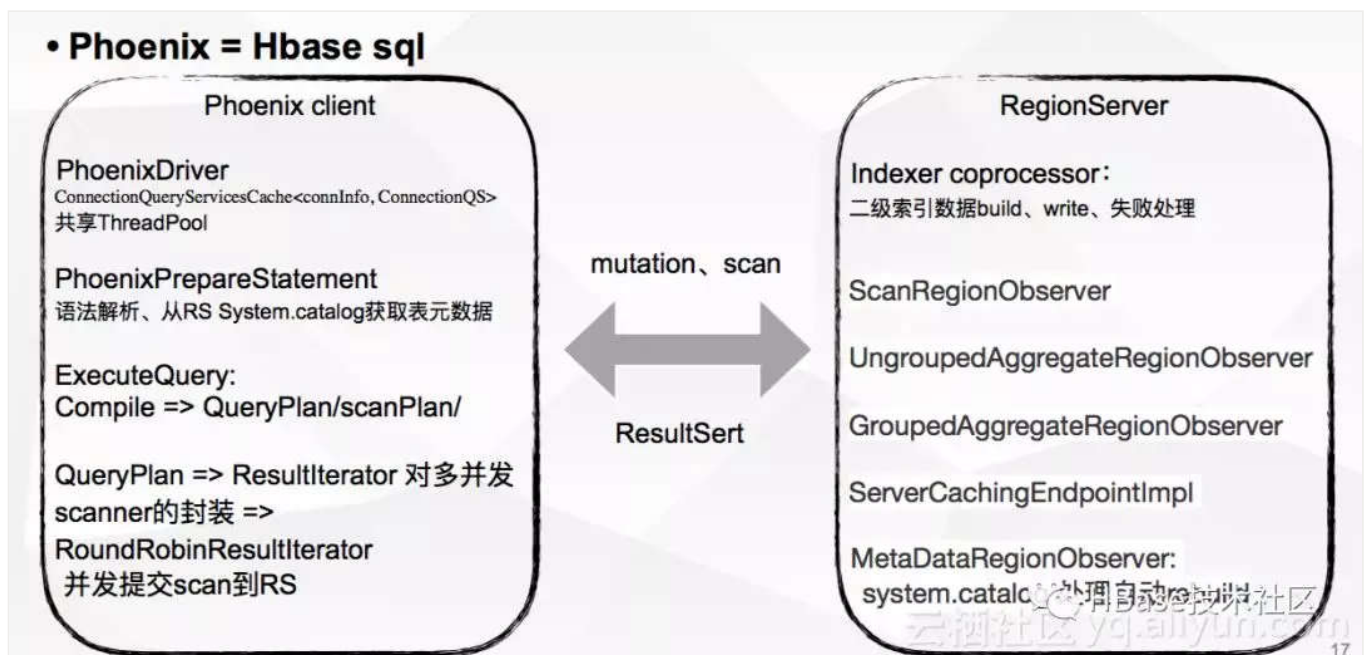
二. Phoenix改进与实践

1. Phoenix原理与架构

Phoenix提供基于HBase的sql操作的框架，主要进行源数据的管理。在RegionServer3上存储着SYSTEM.CATALOG表，在每个RegionServer上有Coprocessor进行查询、聚合、二级索引的建立等操作。



Phoenix client主要进行Connection管理，源数据管理，sql语法物理执行计划，并行Scan的发送与查询，对scanner进行封装。RegionServer中使用coprocessor较多。



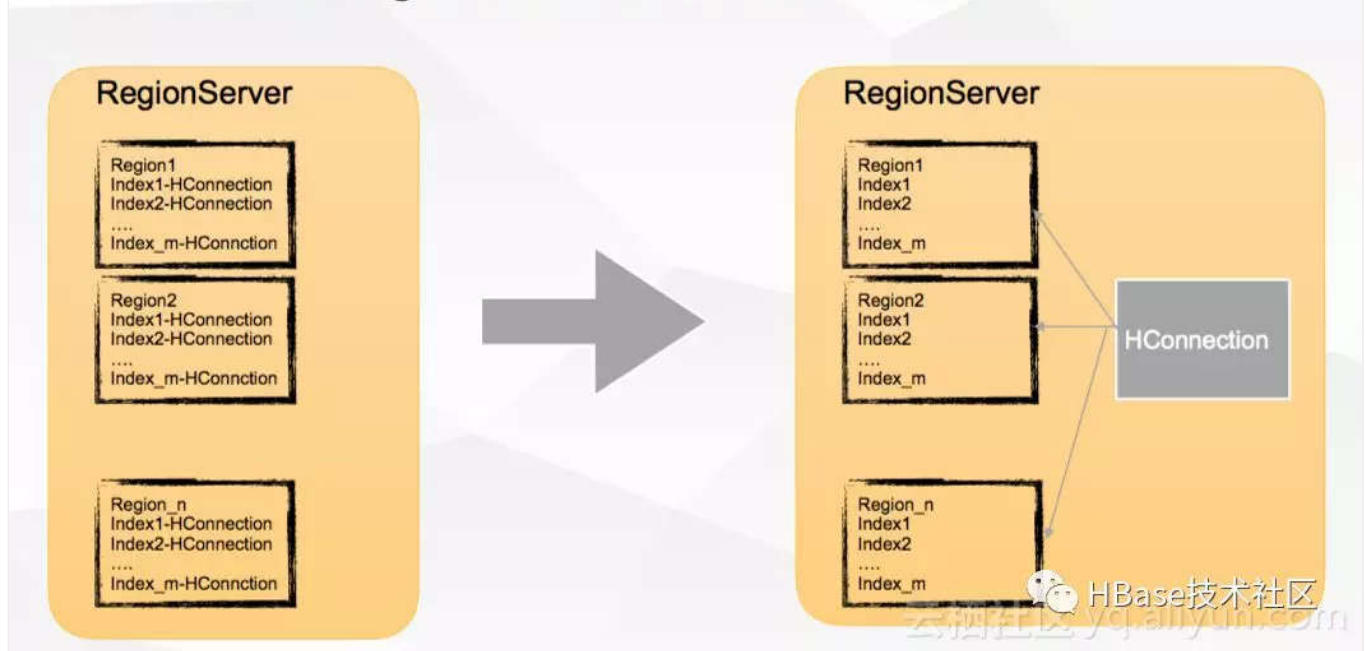
2. Phoenix重要业务支持——全量历史订单

接下来主要介绍滴滴基于一个Phoenix重要端上任务，历史订单查询，作出的优化改进。滴滴的APP端改进前可以查询三个月内的历史订单数据，如今已经达到全量历史订单数据，即理论上可以查询所有订单数据。系统稳定性可以达到SLA99.95%。除却HBase集群自身保证的监控和恢复措施外，二级索引写失败处理和业务增加二级索引问题也作出了较好的改进。并且，滴滴对查询延迟要求也比较高，现使用JDBC客户端P99延迟已达到了30-40ms。在功能上也实现了在每个column都可以声明default值。

2.1 Index coprocessor改进

在稳定性的改进中，connection的管理至关重要。在HBase中，ZK较为脆弱，connection的数量过多会对ZK造成较大的压力。Phoenix传统写二级索引都是由Index coprocessor完成，当主表声明的Index较多时，会产生较多Region。ZK的连接数即为region数乘以index数量，这会导致可能一个表会包含几千个ZK。因此，为了解决这个问题，在RegionServer内部建立ClusterConnection，所有Region都复用该ClusterConnection。

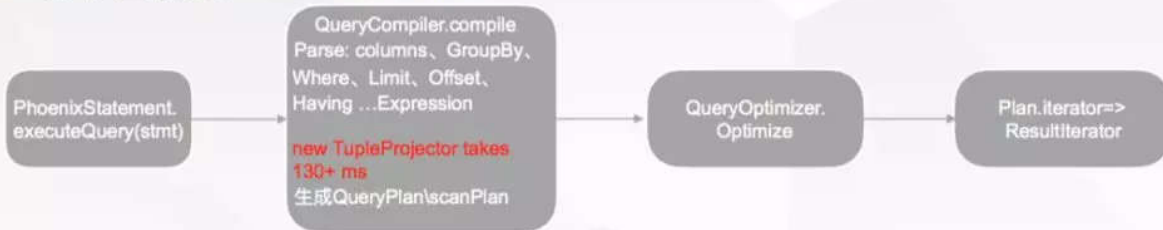
• 二级索引Indexer coprocessor 对Hconnection的改进 zk 连接数 = region num * Index num 512*5 => 1



2.2 TupleProjector性能优化

为了对客户P99延迟进行优化，这里针对TupleProjector进行了改进工作。初始phoenix 进行一次Query compilePlan耗时150ms左右，这主要由于订单业务表多达130多个column，对每一个column进行get column expression都会耗时1ms，因此这总耗时对系统来说是难以容忍的消耗。对于上述类型的宽表，最有效的优化措施为配置并行处理。优化后P99可以达到35ms左右。

- phoenix Query compilePlan 一次 >150 ms
- 订单业务表 >130 column



```
expressions[i++] = ((ProjectedColumn) column).getSourceColumnRef().newColumnExpression();
```

- Get column expression 1ms/column
- 优化: serial to parallel、可配置项、宽表parallel处理

2.3 二级索引设计

Phoenix的Salting功能非常有效，但对延迟影响较大，因此若延迟要求较高，那么Salting则并不合适，所以这里在主表与索引表中不使用Salting功能，而是采用reverse将主键列散列。索引中使用Function Index和Function Index减少查询延迟。示例代码如下所示：

```
CREATE TABLE TEST.TEST_DOS_ORDER (
  order_id BIGINT,
  reversed_order_id VARCHAR NOT NULL,
  global_order_id VARCHAR DEFAULT '',
  passenger_id VARCHAR DEFAULT '0',
  driver_id VARCHAR DEFAULT '0',
  status TINYINT DEFAULT 1,
  departure_time TIMESTAMP DEFAULT TIMESTAMP '1970-12-31 08:00:00.000',
  _create_time TIMESTAMP DEFAULT TIMESTAMP '1970-12-31 08:00:00.000'
) CONSTRAINT DOS_PK PRIMARY KEY (reversed_order_id,global_order_id)
DATA_BLOCK_ENCODING='FAST_DIFF',COMPRESSION='SNAPPY',BLOOMFILTER='row',UPDATE_CACHE_FREQUENCY=3000000;

CREATE INDEX IDX_PASSENGER_CTIME ON DOS.DOS_ORDER(REVERSE(passenger_id), create_time DESC, status)
INCLUDE(order_id,district,channel,extra_type,starting_name,dest_name,departure_time,tip,order_status,type,is_pay,pay_type,combo_type,
_modify_time,product_id,starting_poi_id,dest_poi_id,require_level,area,country_iso_code) ASYNC
DATA_BLOCK_ENCODING='FAST_DIFF',COMPRESSION='SNAPPY',BLOOMFILTER='row',UPDATE_CACHE_FREQUENCY=3000000,UM
N_ENCODED_BYTES = NONE SPLIT ON ('001953'.....);
```

2.4 Default值的坑

一般业务不会使用Default值，并且传统的Default设计存在很多bug。例如在声明Default列的二级索引中的写入错误，即试图写入新值时，真正写入的仍然是Default值。示例如下：

• 申明Default 列的二级索引写入错误

```
CREATE TABLE PHOENIX.TEST_DEFAULT (
  A INTEGER NOT NULL,
  B VARCHAR NOT NULL DEFAULT 'AB',
  C BIGINT DEFAULT 200,
  D BIGINT DEFAULT 200,
  E BIGINT DEFAULT 200,
  F VARCHAR DEFAULT 'AB',
  CONSTRAINT DOS_PK PRIMARY KEY (A,B));

CREATE INDEX IDX_TEST_DEFAULT ON PHOENIX.TEST_DEFAULT(C, F) INCLUDE(D,E);

upsert into PHOENIX.TEST_DEFAULT(A,C,F) values(1000, 100, 'mn');
```

索引表中Result: 200,'AB',1000,'AB',200,200

Upsert 实际值索引表中全部填入default值

云栖社区 yq.aliyun.com HBase技术社区

这里有两种解决方案。一是在Index进行build之前，即在客户端时进行一些特殊处理。方法二为在生成索引表的源数据时对错误填写的值进行修改。

另一处bug例如在建立异步二级索引生成rowkey和Indexer build rowkey不一致，导致在索引查询数据时double。具体原因是PTableImpl.newKey对Default值的列逻辑上存在bug。示例如下：

```
000000000211282\x00\x80\x00\x00\x06\x7F\xFF\xFE\x9F column=0:_0, timestamp=1515165074289, value=_0
\x81L6?\xFF\xFF\xFF\xFF0306702239474292511
000000000211282\x00\x80\x00\x00\x06\x7F\xFF\xFE\xA0 column=0:_0, timestamp=1513338534031, value=x
}\xB6!\xBF\xFF\xFF\xFF\xFF850553014337406675\x00
000000000211282\x00\x80\x00\x00\x06\x7F\xFF\xFE\xA0 column=0:_0, timestamp=1515716288467, value=_0
\xE9\xE4\xD5\x7F\xFF\xFF\xFF\xFF992362910863435675
000000000211282\x00\x80\x00\x00\x06\x7F\xFF\xFE\xA0 column=0:_0, timestamp=1513338534031, value=x
\xE9\xE4\xD5\x7F\xFF\xFF\xFF\xFF992362910863435675\
x00
```

云栖社区 yq.aliyun.com HBase技术社区

2.5 性能压测

在进行上述优化后，分别通过Java-JDBC和Java-queryServer查询进行性能压测。结果如下：

• 分别通过Java-JDBC和 Java-queryServer 查询的性能压测结果

offset	并发	P99 ms	P75m s	QPS 每秒结果集
0	8	15	11	544
	16	21	12	781
	32	23	13	1307
	64	24	13	1283
300	8	22	16	361
	16	27	17	539
	32	27	17	944
	64	28	17	944
600	8	35	23	276
	16	33	22	447
	32	34	22	715
	64	34	22	718

offset	并发	P99	P75	QPS
0	8	26	19	659
	16	31	22	896
	32	53	37	969
	64	106	71	967
300	8	49	12	528
	16	57	21	779
	32	114	41	910
	64	111	71	1001
600	8	71	12	463
	16	80	20	705
	32	106	36	962
	64	157	71	962

其他非Java语言会通过Go-queryServer查询，压测P99结果会比Java语言稍慢。

2.6 二级索引策略

在传统逻辑中，若Indexer二级索引写失败，则直接disable二级索引表，然后rebuild，但这在线上是不可用的。因此，滴滴采取关闭自动rebuild和disable进行改进。那么关闭自动rebuild和disable后如何感知写失败呢？此时需要定时查询RegionServer log实时收集的ES，然后调用异步二级索引Partial rebuild来进行修补。当主表数据量较大时，进行异步全量二级索引时可能会split大量的maptask，超出master的承受范围。因此需要改进split逻辑，对某些task进行合并。当这些改进完成，就可以提供较为稳定的支持。

• Indexer 二级索引写失败策略关闭disable及自动rebuild

- A. 自动Rebuild关闭 `phoenix.index.failure.handling.rebuild`、`REBUILD_INDEX_ON_WRITE_FAILURE`
- B. Disable关闭 `phoenix.index.failure.disable.index`、`DISABLE_INDEX_ON_WRITE_FAILURE`
- C. Disable索引表造成端上查询不可用 业务不可接受
- D. 自动rebuild 功能在`system.catalog`的`MetaDataReaderObserver`中处理, `coprocessor` 内存有限、影响`system.catalog`的稳定性, 实际作用有限

• 异步二级索引Partial rebuild

- E. RegionServer log 实时收集到ES
- F. 定时查询ES 索引写失败`keyword(SingleIndexWriteFailureException)`、提取索引表信息、索引写失败起止时间点、调度Partial rebuild MR job

• 异步全量二级索引 Snapshot build

- G. 离线集群build、bulkload到hbase集群

云栖社区 yq.aliyun.com HBase技术社区

2.7 集群升级对二级索引写入影响

当出现集群升级时, 二级索引写入肯定会失败, 该如何将该影响降至最低呢? 如果将主表和索引表部署在一起, 那么一台机器升级, 所有机器都会出现写失败。前文中也提到, 滴滴在HBase中增添了GROUP功能, 那么便可以将主表和索引表分开, 部署在不同GROUP中, 降低集群升级的影响。如下图所示:



三. GeoMesa应用简介与展望

1. GeoMesa架构原理

滴滴最近开始对GeoMesa展开调研，暂未取得丰富的线上经验。GeoMesa是基于分布式存储、计算系统上的大规模时空数据查询分析引擎。即在存储时可以选择存储区域，数据输入也可以包含多种形式，如spark kafka等。架构如下图所示：



GeoMesa对地理时空信息进行索引有着极大的优势。以HBase为例，GeoMesa可以将二维或三维数据映射至一维存储，Geohash是较为常见的编码方式。这种索引方法属于点索引，如今使用较为广泛。具体示例如下图所示：

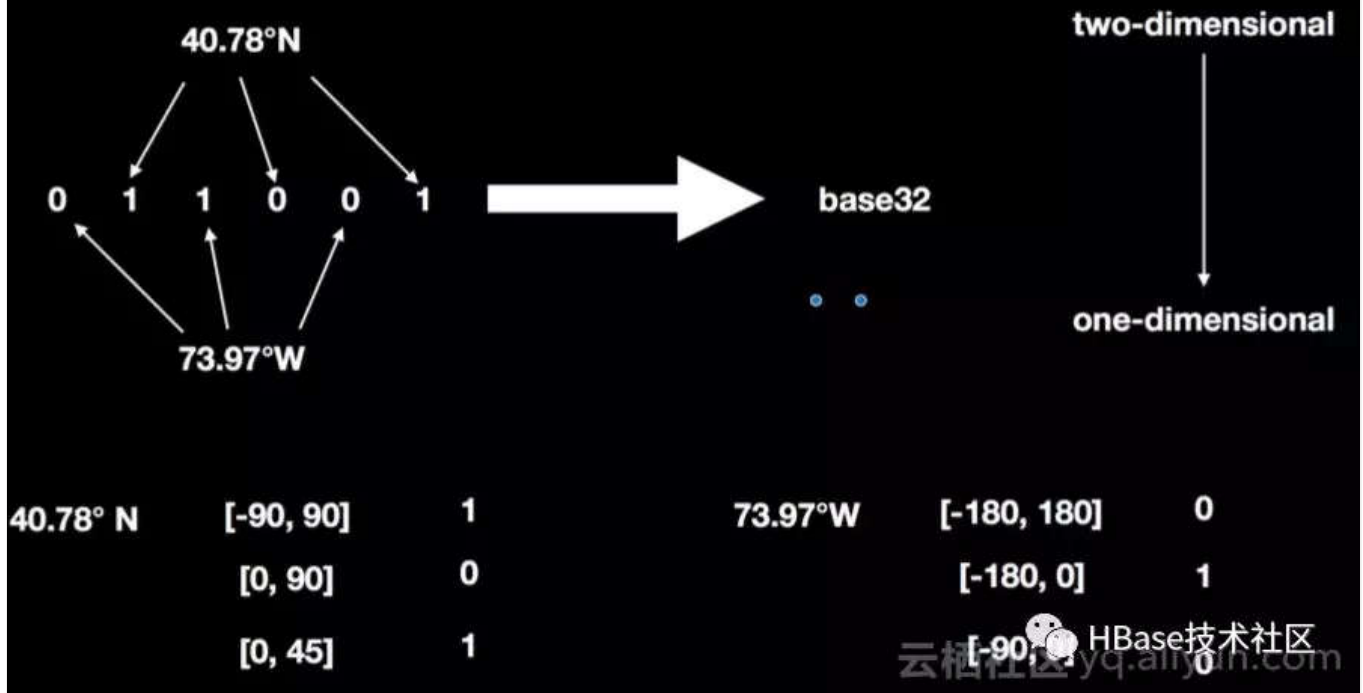
- **Geomesa Z3 Index: Salting + Geohash(spatial + temporal) + ID**
- 点的索引

KEY						VALUE	
ROW			COLUMN		TIMESTAMP	VIZ	Byte-encoded SimpleFeature
			COLUMN FAMILY	COLUMN QUALIFIER			
Epoch Week 2 bytes	Z3(x,y,t) 8 bytes	Unique ID (such as UUID)	"F"	-	-	Security tags	

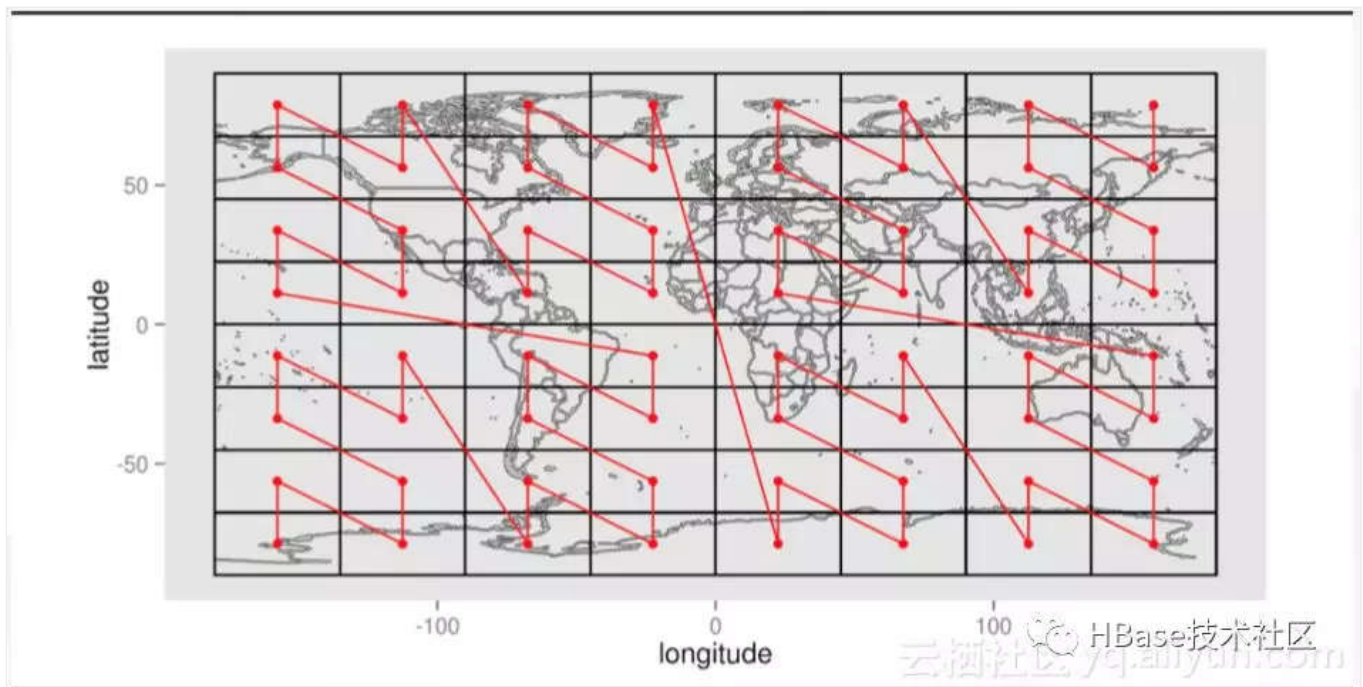
A watermark 'HBase技术社区' is visible at the bottom right of the table.

Geohash方法是将经纬度等高维地理时空数据进行01编码映射到一维。首先将维度置于奇数位，经度为偶数位，通过base32生成字符串，降为一维，具体过程见下图：

GeoHash

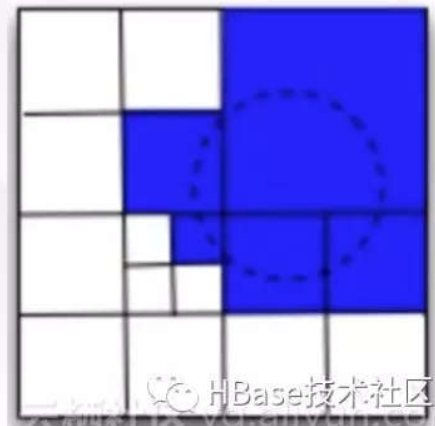


Geohash的理论基础为Z-order曲线。但Z-order曲线存在突变性，即当地理位置距离非常远，但编码后的逻辑位置可能会产生连续的现象，如下图。因此，通过Geohash查询到的结果会比真正需要的结果数量差异较大，会有很多无效结果。



• GeoHash编码 Z-order curve

- 缺点：曲线的突变性造成编码相似但距离可能相差很大的问题
- 空间上相邻但编码上不相邻需要切分成更多的scan



这会造成上图中，某些点编码前缀相同，但实际地理位置却相差较远，而与实际地理位置较近的编码前缀并不相同。

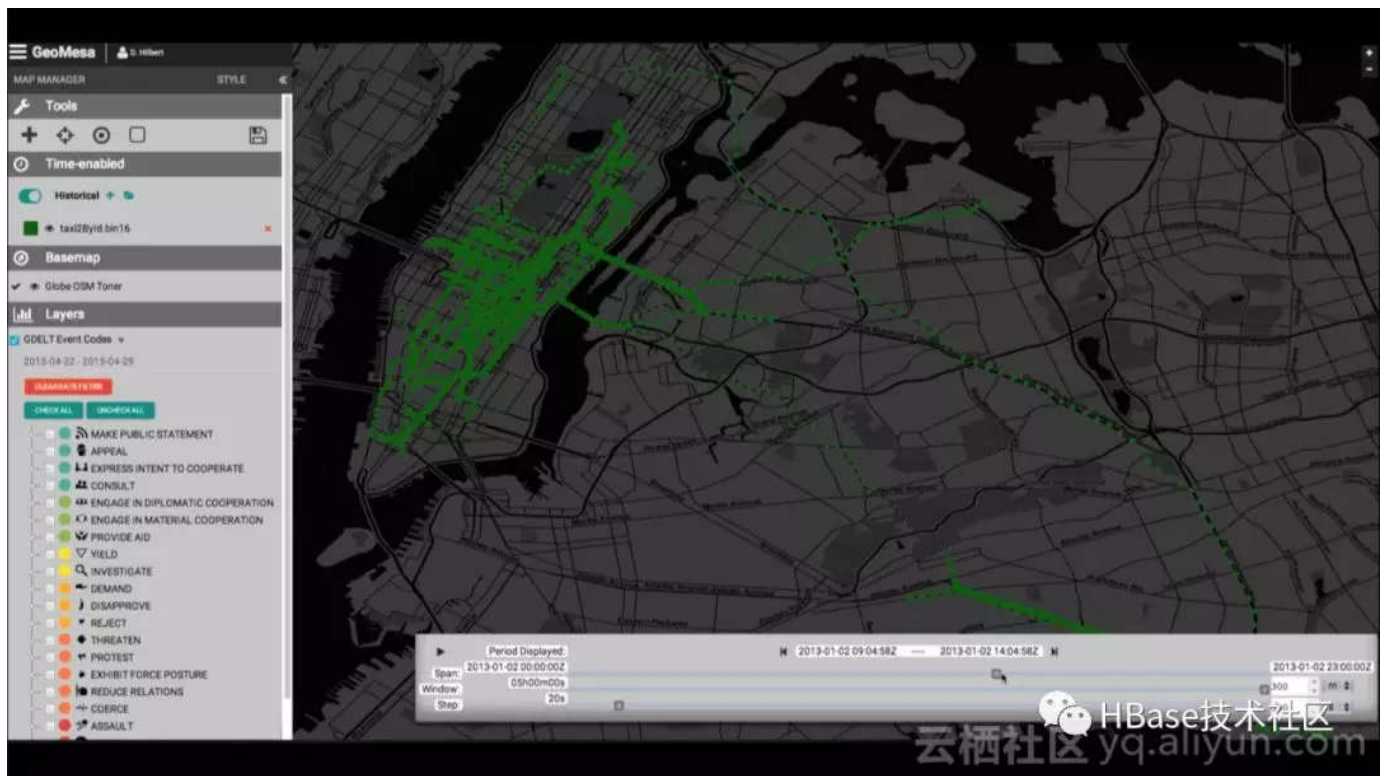
2. GeoMesa应用

GeoMesa经常应用于热力图绘制。滴滴使用其进行行车轨迹记录，即查询某时间段某区域内经过的车辆。以及对轨迹点加工，通过矢量路径分析拥堵等。日后希望将XZ-order线和多边形索引应用至滴滴的业务场景中去。

• Z-order 点的索引

- 行车记录轨迹:查询某时间段某区域内经过的车辆
- 轨迹点加工成矢量路径分析拥堵
- XZ-order 线、多边形的索引

滴滴也将GeoMesa进行了可视化，具体详见视频。



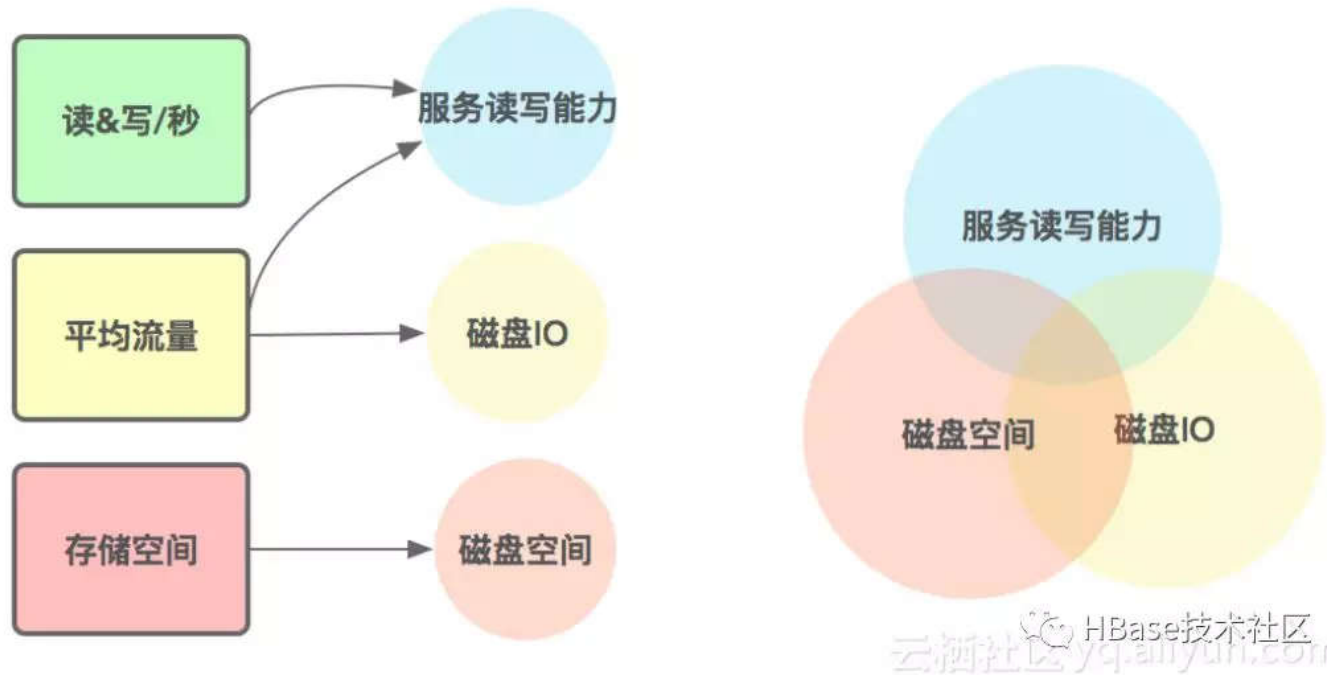
3. GeoMesa未来展望

基于上述Z-order曲线的突变性问题，未来希望可以基于希尔伯特空间填充曲线编码生成索引。因为希尔伯特空间填充曲线含有地理空间局部性特征，因此一维曲线上离得越近的点在2D空间离得越近。这个特性对Scan具有较好的适用性。此外，GeoMesa中的plan耗时较长，平均每次耗时90ms，未来希望可以进行优化。

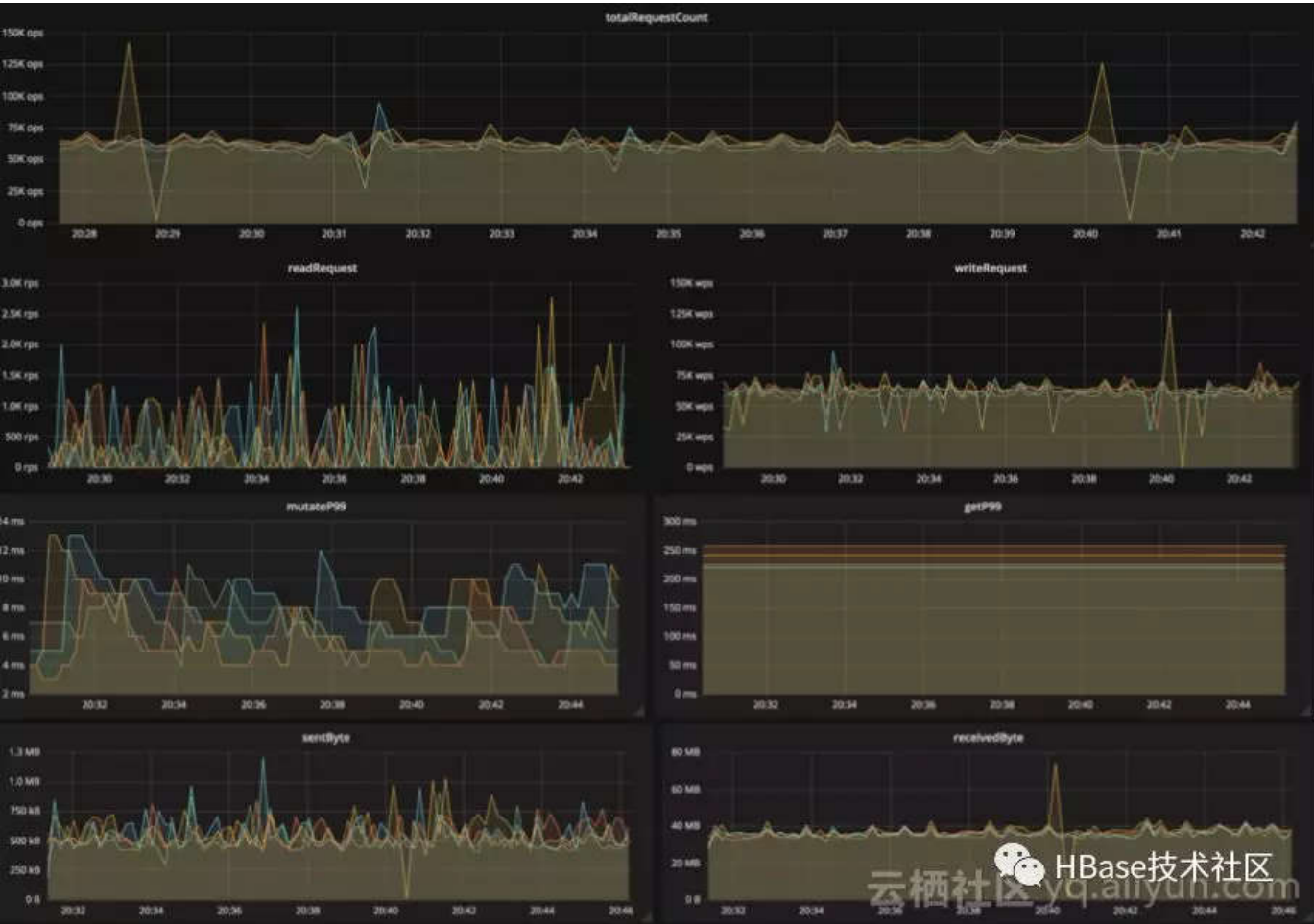
四．稳定性&容量规划

为了达到稳定性与容量规划，滴滴主要进行了以下工作。

首先是机器规划。其中主要从三个维度进行考虑：每秒读写量、平均流量和存储空间。每秒的读写量影响服务读写能力，平均流量会影响服务读写能力和磁盘IO，而存储空间对应其磁盘空间。若每秒读写量太大，会影响该服务的GC及网络流量等。若需要的存储空间比磁盘的总存储空间要大，那么服务也会受到影响。因此这里可以通过压力测试和DHS统计，将这三个维度进行综合比较，计算机器容量规划。同时，也可以根据上述信息，完成集群的GROUP划分规划。

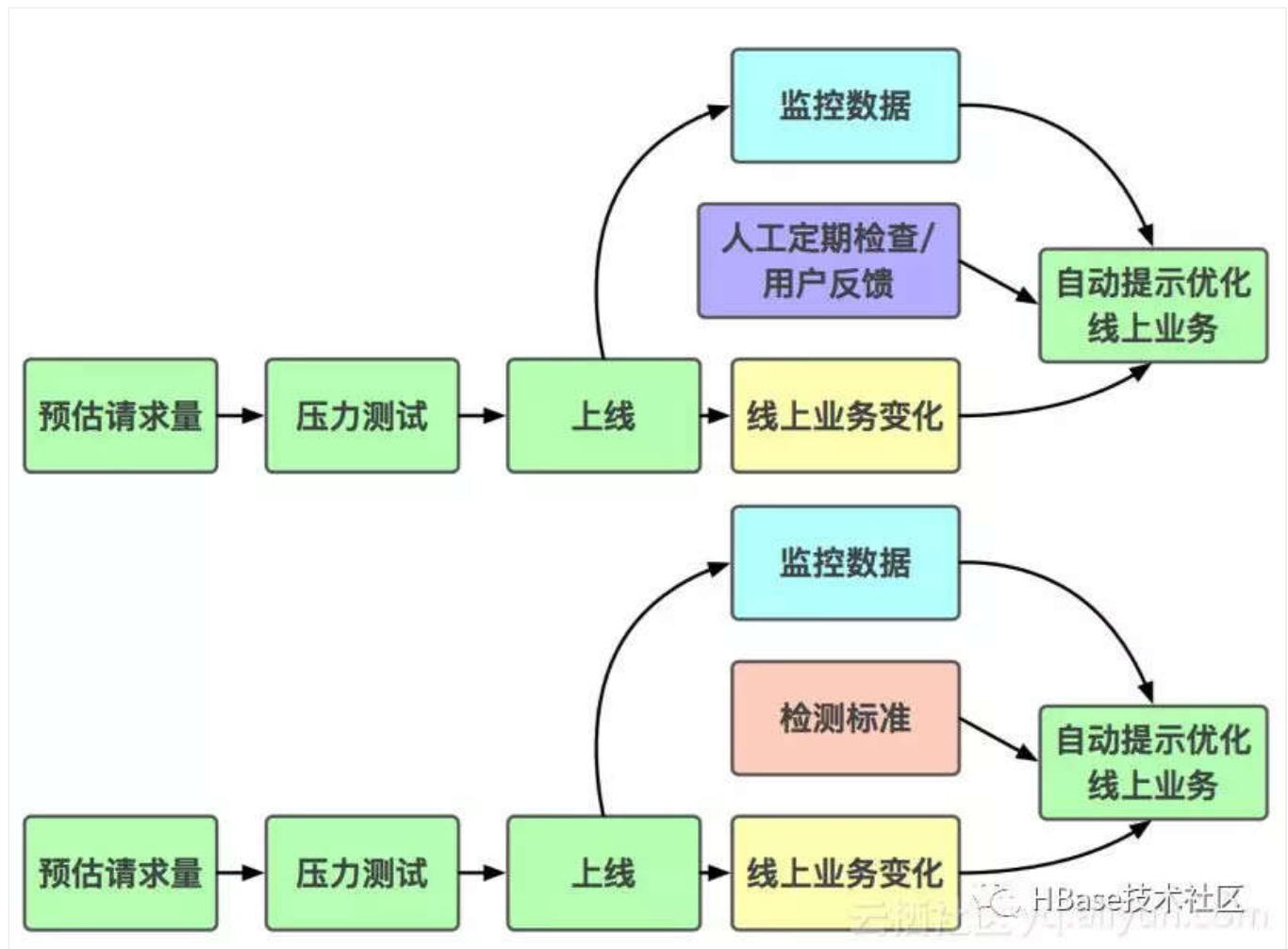


其次，还对HBase的一些指标进行监控，根据监控情况判断用户服务是否处于健康状态。出现故障时，也可以根据监控排查问题。这里不再赘述，其中一监控效果如下：



滴滴目前的工作流程大致为，当用户接入时，先预估请求量，进行压力测试，检查后进行上线。上线后业务经常会发生变化，例如发展较好数据量增加等，此时会循环进行一些操作，如根据监控数据自

动提示优化线上业务，或者人工定期检查或用户反馈来进行优化。后续工作是希望可以达到自动化模式，即利用监控数据，优化线上服务，自动发现空闲节点和可优化的GROUP。



同时，滴滴也建立了DHS(Didi HBase Service)平台，接入了用户需求，可以可视化信息、自动验证，帮助用户了解服务状态。如今正致力于以更少的人工计算，统计上述更细致的服务相关信息，帮助管理员了解用户使用方式。

3.DHS(Didi HBase Service)平台：

更多推荐值、可视化信息、自动验证，帮助用户了解服务状态

更少的人工计算

更细致的统计，帮忙管理员了解用户使用方式

更紧密的与集群管理和运维互动



下面是我的公众号，感谢大家关注。也想大家推荐一下hbase相关学习，大家工作学习遇到HBase技术问题，把问题发布到HBase技术社区论坛hbase.group，欢迎大家论坛上面提问留言讨论。想了解更多HBase技术关注HBase技术社区公众号(微信号:hbasegroup)，非常欢迎大家积极投稿。

长按下面的二维码关注我的公众号



长按下面的二维码关注HBase技术社区公众号



