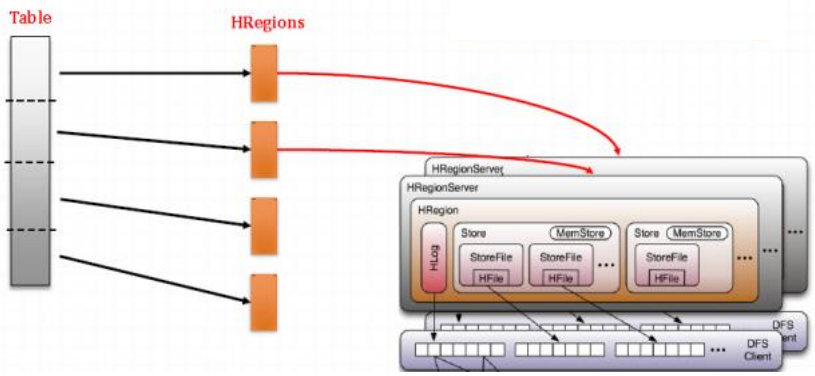
HBase Coprocessor 优化与实验

郭磊涛

2012年12月

HBase 简介

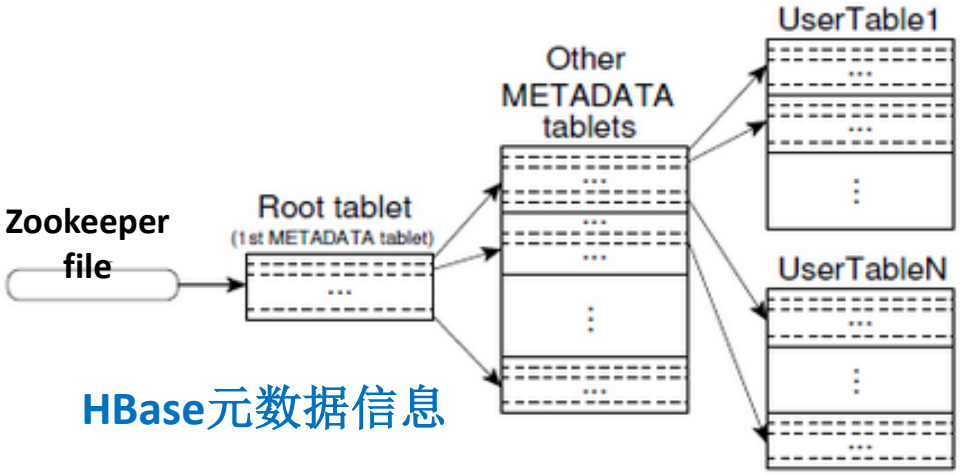
- HBase是在Hadoop之上构建的非关系型、面向列存储的开源分布式结构化数据存储系统
- HBase表分区与索引管理



- 将Table中的数据根据rowKey字段划分为多个HRegion
- HRegion分配给RegionServer管理

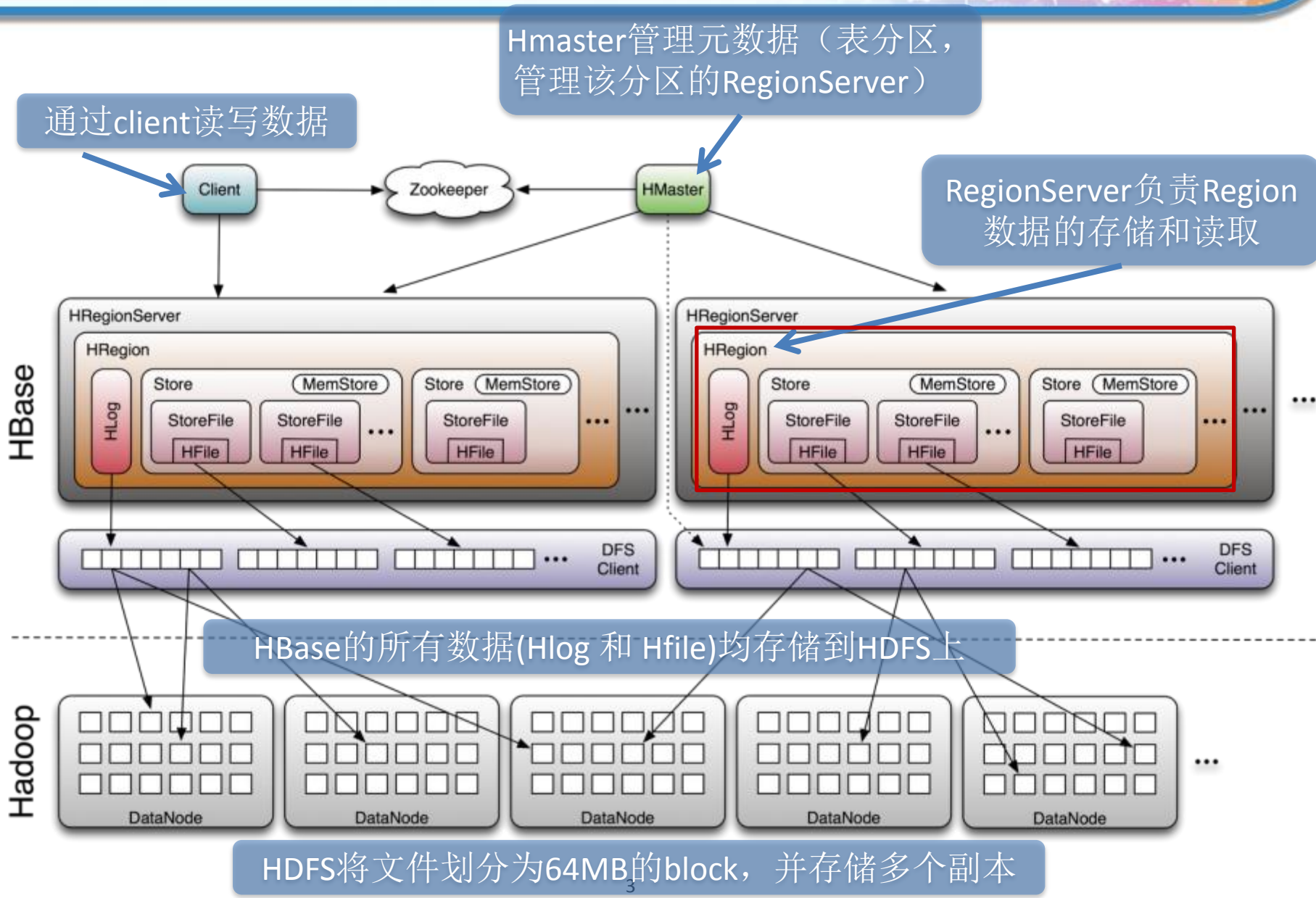
三级元数据：

- **MetaTable**：Region与RegionServer的映射信息
- **RootTable**：MetaTable与RegionServer的映射信息
- **Zookeeper file**： RootTable的存储位置



HBase元数据信息

HBase 系统架构


Hmaster管理元数据（表分区，管理该分区的RegionServer）
通过client读写数据
Client
Zookeeper
HMaster
RegionServer负责Region数据的存储和读取
HBase
HRegionServer
HRegion
HLog
Store
MemStore
StoreFile
HFile
DFS Client
HBase的所有数据(Hlog 和 Hfile)均存储到HDFS上
Hadoop
DataNode
HDFS将文件划分为64MB的block，并存储多个副本

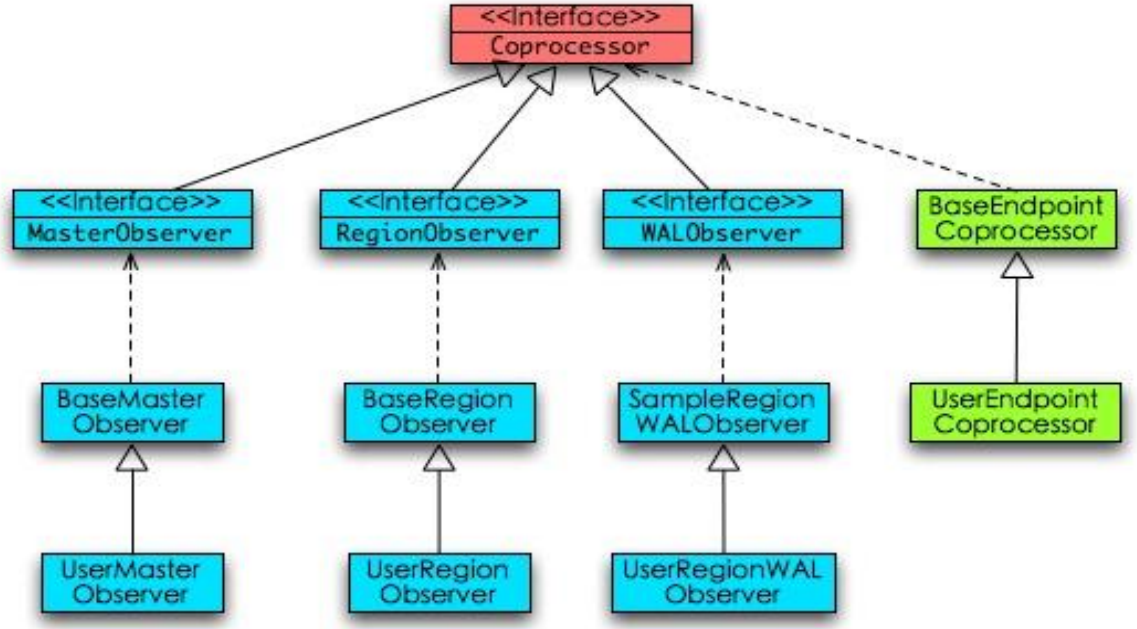
HBase Coprocessor 简介

- HBase Coprocessor受启发于Google的Jeff Dean在LADIS’09 上的报告
  - Google BigTable的Coprocessor特点
    - 在每个表服务器的任何tablet上均可执行用户代码
    - 提供客户端调用接口 （coprocessor客户端lib将可定位每个row/range的位置；多行读写将自动分片为多个并行的RPC调用）
    - 提供可构建分布式服务的灵活的编程模型
    - 可以自动扩展，负载均衡等
  - 与Google Bigtable Coprocessor相比
    - Bigtable coprocessor 以独立的进程执行，可以更好的控制CP计算所需资源
    - HBase coprocessor是一个在Master/RegionServer进程内的框架，通过在运行时执行用户的代码，在HBase内实现灵活的分布式数据处理功能
- HBase Coprocessor的主要应用场景
  - secondary indexing
  - complex filtering
  - access control

Coprocessor 在以下将简称为 **CP**

HBase Coprocessor 的实现类型

- HBase Coprocessor的实现分为Observer和Endpoint两种
  - Observer类似于触发器，工作在服务器端。可以实现权限管理、监控等
  - Endpoint类似于存储过程，工作在服务器端和客户端。可以实现min/max等计算
- Coprocessor的作用范围
  - System coprocessor: 对所有table的所有region
  - Table coprocessor：对某个table的所有region

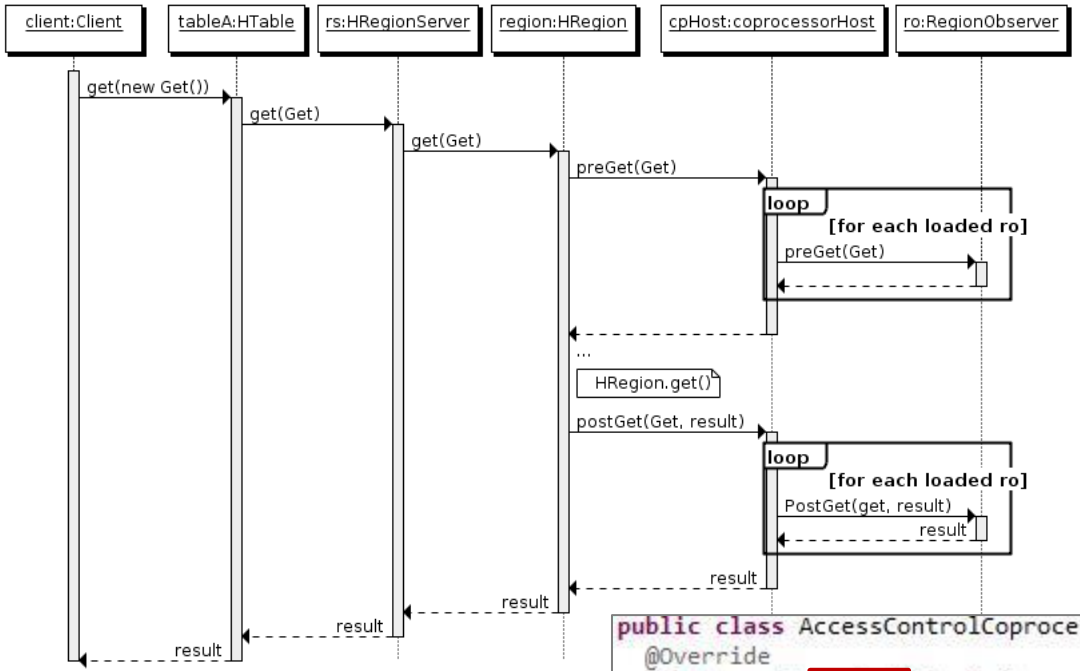


- **RegionObserver**：提供表数据操作事件的钩子函数：Get、Put、Scan等的pre/post处理。
- **WALObserver**：提供WAL相关操作钩子。
- **MasterObserver**：提供DDL类型的操作钩子。如创建、删除、修改数据表等。
- **Endpoint**：只适用于RegionServer, 对应于每个table的Region的处理。

该图参考了《HBase Coprocessor的分析》http://walkoven.com/?p=77

HBase Coprocessor Observer示例

- RegionObservers与HBase的交互流程：



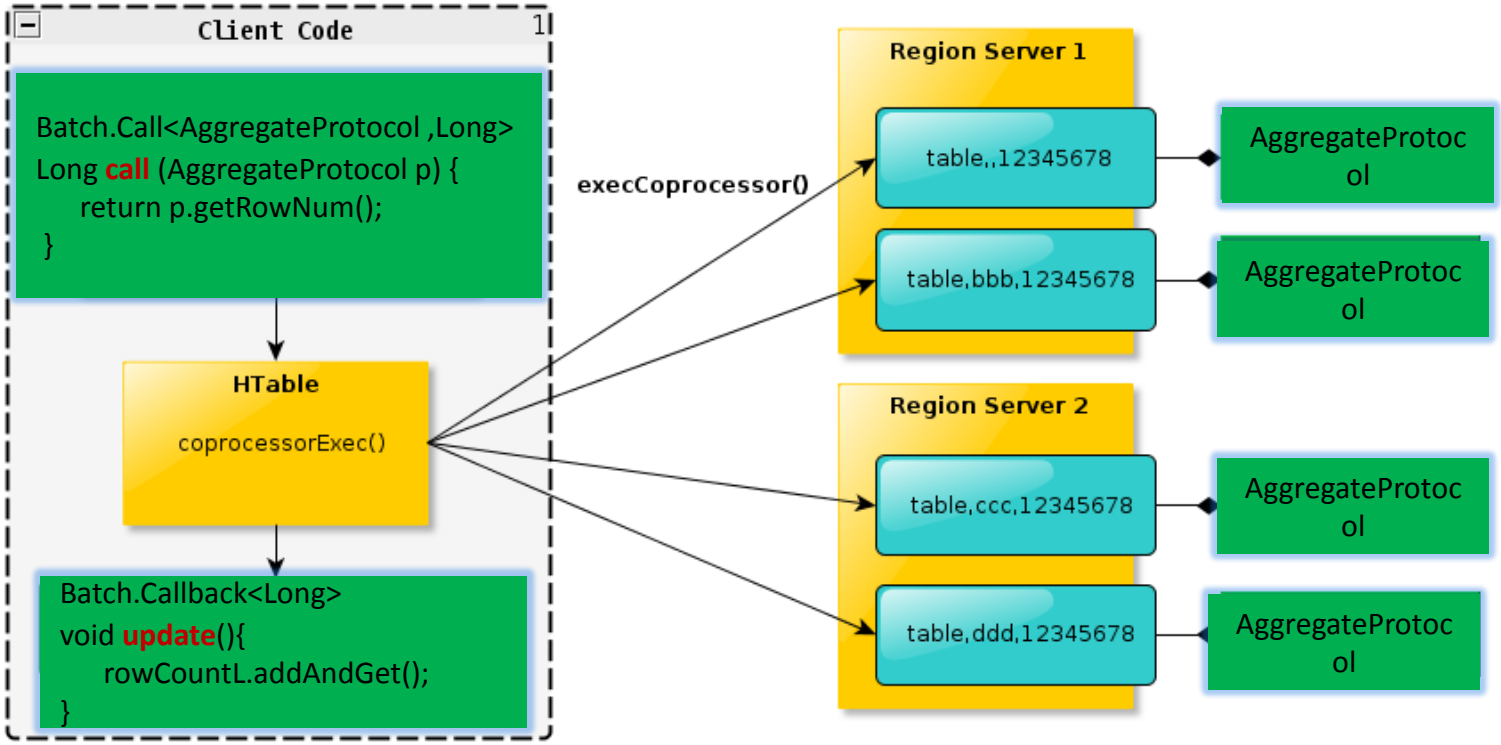
使用RegionObserver，在preGet()中判断用户是否具有数据读权限：

```
public class AccessControlCoprocessor extends BaseRegionObserver {
  @Override
  public void preGet(final ObserverContext<RegionCoprocessorEnvironment> c, final Get get,
      final List<KeyValue> result) throws IOException {

    // check permissions..
    if (!permissionGranted()) {
      throw new AccessDeniedException("User is not allowed to access.");
    }
  }

  private boolean permissionGranted() {
    // TODO Auto-generated method stub
    return false;
  }
  // override prePut(), preDelete(), etc.
}
```

该页参考了Coprocessor Introduction,
https://blogs.apache.org/hbase/entry/coprocessor_introduction

HBase Coprocessor Endpoint示例

Client通过API调用对某行/行范围的处理，CP框架自动将行范围映射到Region，并将计算请求发送给每个Region并行执行



① public interface AggregateProtocol **extends** CoprocessorProtocol

② public class AggregateImplementation **extends** BaseEndpointCoprocessor **implements** AggregateProtocol

③ public class AggregationClient

HBase Coprocessor 配置

① 在hbase-site.xml中配置CP class （System coprocessor）

```
<property>
        <name>hbase.coprocessor.region.classes</name>
        <value>org.apache.hadoop.hbase.coprocessor.AggregateImplementation</value>
</property>
```

为所有table加载了一个cp class，可以用”,”分割加载多个class

② 从hbase shell中加载CP class （Table coprocessor）

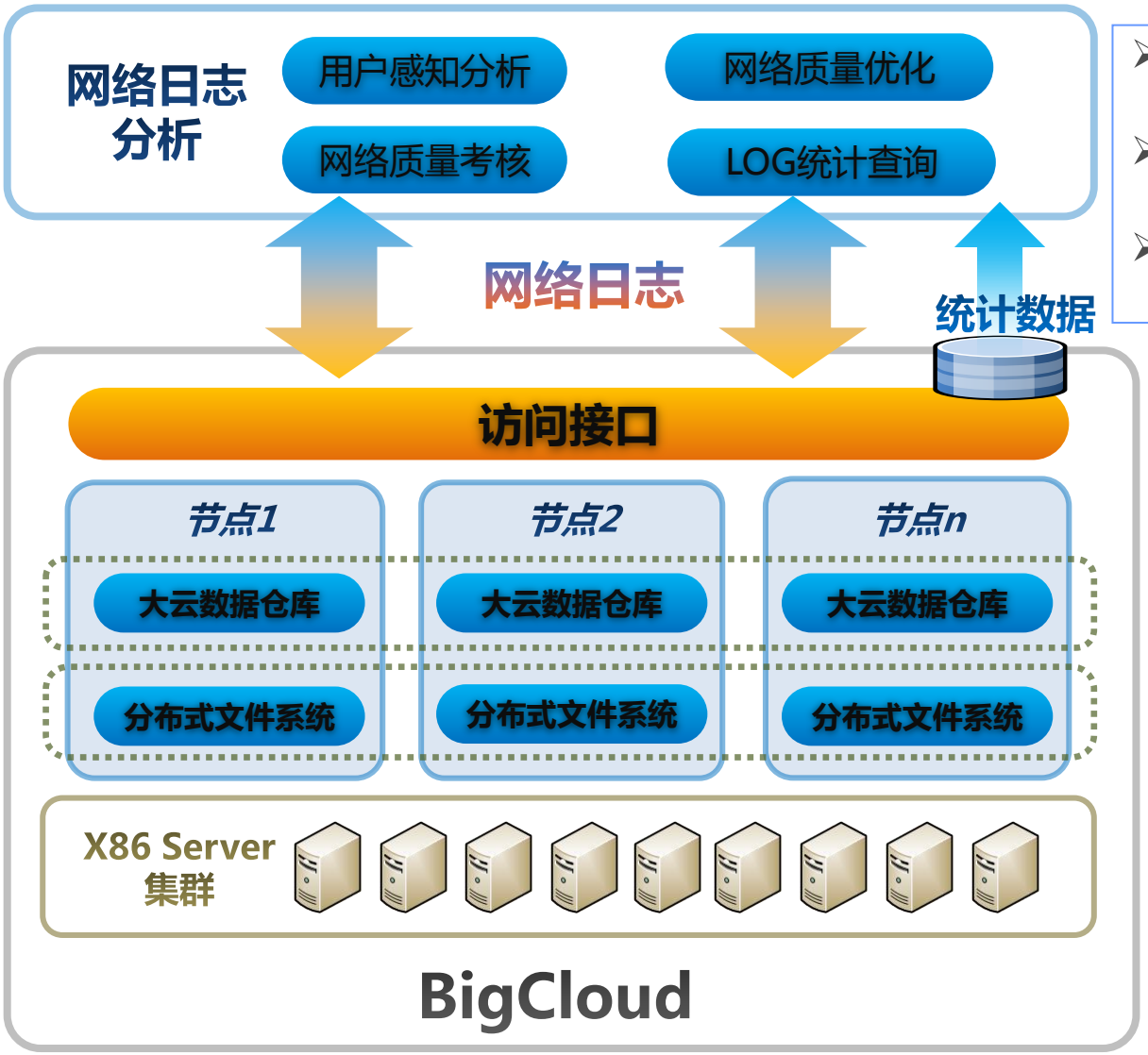
```
> alter ‘testTable', METHOD => 'table_att',
'coprocessor'=>'hdfs:///foo.jar|com.foo.FooRegionObserver|1001|arg1=1,arg2=2‘
> enable ' testTable '
```

为testTable加载一个RegionObserver

```
> alter 'testTable ', METHOD => 'table_att_unset', NAME => 'coprocessor$1‘
```

为testTable卸载一个cp class

我们的应用场景-网络日志分析查询

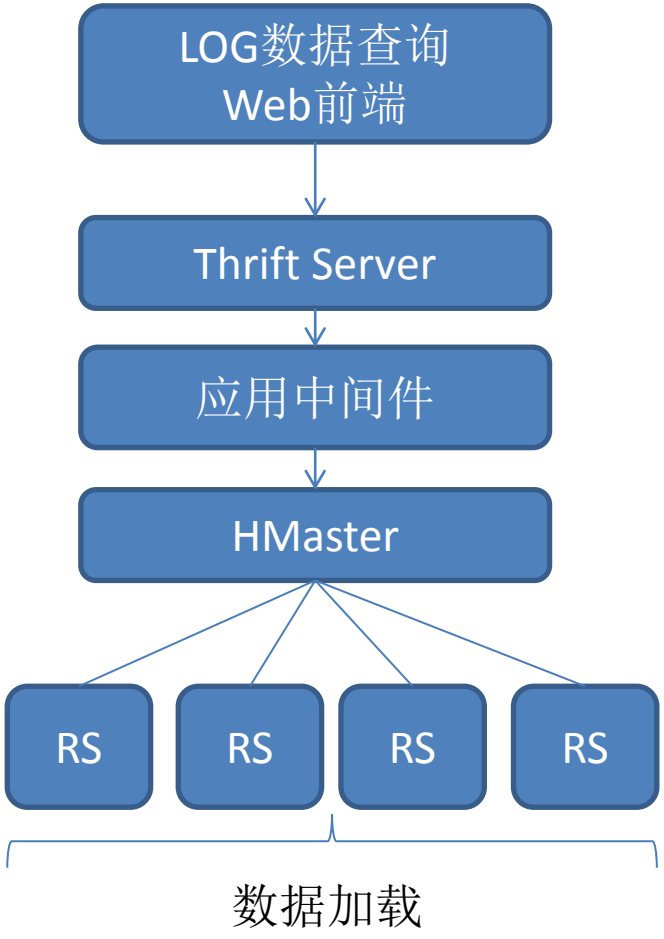


- 使用大云实现海量网络日志数据的实时加载和实时复杂查询
- 数据量：中等规模子公司记录约30亿条/时
- 使用大云实现60个统计指标的复杂查询一分钟内返回结果

LOG统计查询：

```
select intProtocol, callingIMSI,
calledIMSI, intDelayAltering,
intDelayConnect,
intTermEvent, …
 from
vwXDRQuery_1350619391458 t
where (t.dtStime>… and …) and
t.intProtocol in (32,97)
order by callingIMSI ASC
limit 10000
```

利用CP实现对LOG数据的查询



在应用开发中遇到的主要问题

- 加载CP class，造成region分布错乱，性能降低
- 客户端的网络成为瓶颈
- 在较复杂应用逻辑下，CP不够稳定

HBase Coprocessor优化 – Region数据本地化

- 测试和实验环境
  - LOG数据2亿条，预分为720个Region，平均每节点48个Region
    - 1 Master + 15 RegionServer （32GB RAM， 千兆以太网）
  - 利用CP在每个region查询符合条件的记录，并在CP客户端汇总结果
    - 每个region返回1w条记录，每条记录1KB，即每个region返回10MB数据
  - 执行结果（平均值）：127.7秒
  - 原因分析：
    - Region数据非本地化存储

Total Regions = 720 ;avgDataLocality =	0.2113607

    - CP客户端网络成为瓶颈
      - 客户端需要处理 10MB*720 = 7GB数据

Regions by Region Server

Region Server	Region Count	Avg. Data Locality
http://compute-H-12-04:60030/	48	0.10666581
http://compute-G-16-09:60030/	48	0.15927553
http://compute-H-17-06:60030/	48	1.0
http://compute-H-12-06:60030/	48	0.14705287
http://compute-H-16-01:60030/	48	0.23983204
http://compute-G-16-05:60030/	48	0.19384198
http://compute-H-13-02:60030/	48	0.101728536
http://compute-H-16-06:60030/	48	0.1452163
http://compute-G-16-07:60030/	48	0.1424232
http://compute-H-13-03:60030/	48	0.117355354
http://compute-G-16-06:60030/	48	0.1271935
http://compute-H-17-03:60030/	48	0.192312
http://compute-H-12-05:60030/	48	0.14130123
http://compute-G-16-08:60030/	48	0.12304617
http://compute-G-16-04:60030/	48	0.23316501

HBase Coprocessor优化 – Region数据本地化

- HBase支持的默认Region分配策略 (*AssignmentManager*)
  - retainAssignment （default: true)
  - roundRobinAssignment
  - randomAssignment
- 当集群较为稳定时，默认Region策略即可实现Region数据本地读写。
- 当节点退出/上线，或者table dis/enable时，会造成Region数据非本节点存储
  - 在对table增加CP时，需要dis/enable table
  - 在enable table时，会对region采用 randomAssignment（default: true） 或 roundRobinAssignment 分配策略
- 我们的改进：对Region可以通过配置采用 ***toplocalizedAssignment*** 策略
  - 在制定Region分配计划时，通过 *HRegion.computeHDFSBlocksDistribution()* 计算其数据在RegionServer的分布情况，将Region调度至具有最多数据的RegionServer

HBase Coprocessor优化 – Region数据本地化

- 采用 *toplocalizedAssignment* 的Region分配策略后， LOG查询的执行结果为： 59.95秒 ，性能提升了113%

Regions by Region Server

Region Server	Region Count	Avg. Data Locality
http://compute-H-12-04:60030/	48	1.0
http://compute-H-12-06:60030/	50	1.0
http://compute-H-17-06:60030/	51	1.0
http://compute-G-16-09:60030/	48	1.0
http://compute-H-16-01:60030/	51	1.0
http://compute-H-13-02:60030/	49	1.0
http://compute-G-16-05:60030/	46	1.0
http://compute-H-16-06:60030/	49	1.0
http://compute-G-16-07:60030/	45	1.0
http://compute-G-16-06:60030/	48	1.0
http://compute-H-13-03:60030/	49	1.0
http://compute-H-17-03:60030/	54	1.0
http://compute-H-12-05:60030/	46	1.0
http://compute-G-16-08:60030/	45	1.0
http://compute-G-16-04:60030/	41	1.0

Total Regions = 720 ;avgDataLocality = 1.0

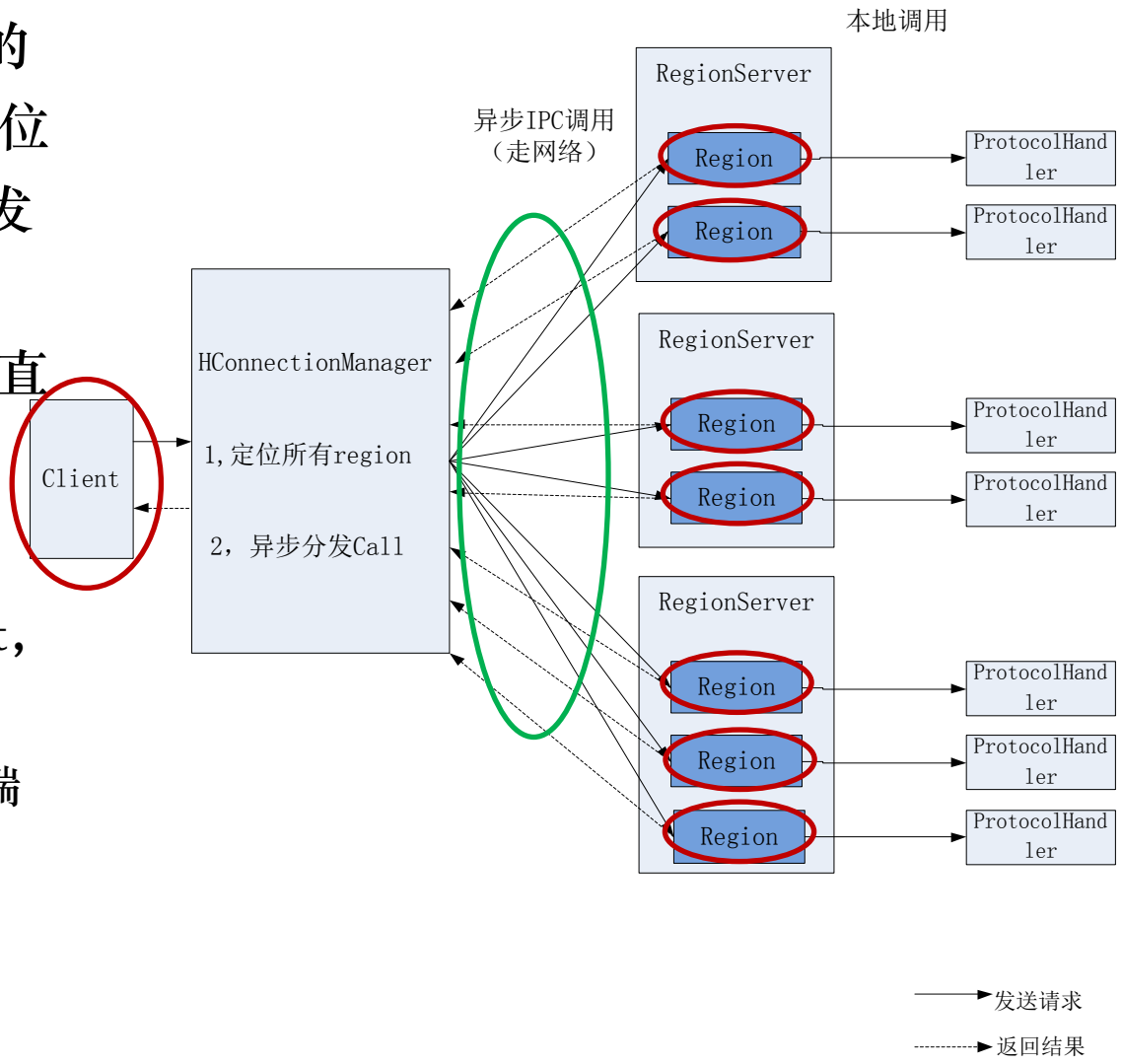
- 但是，客户端节点的网络仍然是瓶颈，CPU和内存占用也较高

total-cpu-usage　　net/total

usr	sys	idl	wai	hiq	siq	recv	send
0	0	100	0	0	0	0	0
10	0	90	0	0	0	52M	376k
20	1	79	0	0	0	0	0
23	1	76	0	0	0	165M	963k
18	1	82	0	0	0	0	0
21	0	78	0	0	0	152M	960k
21	0	79	0	0	0	0	0
16	0	83	0	0	0	96M	596k
11	0	88	0	0	0	0	0
14	1	86	0	0	0	108M	769k
17	0	82	0	0	0	0	0
22	0	78	0	0	0	86M	497k
11	0	89	0	0	0	0	0
14	0	86	0	0	0	68M	373k
14	0	86	0	0	0	0	0

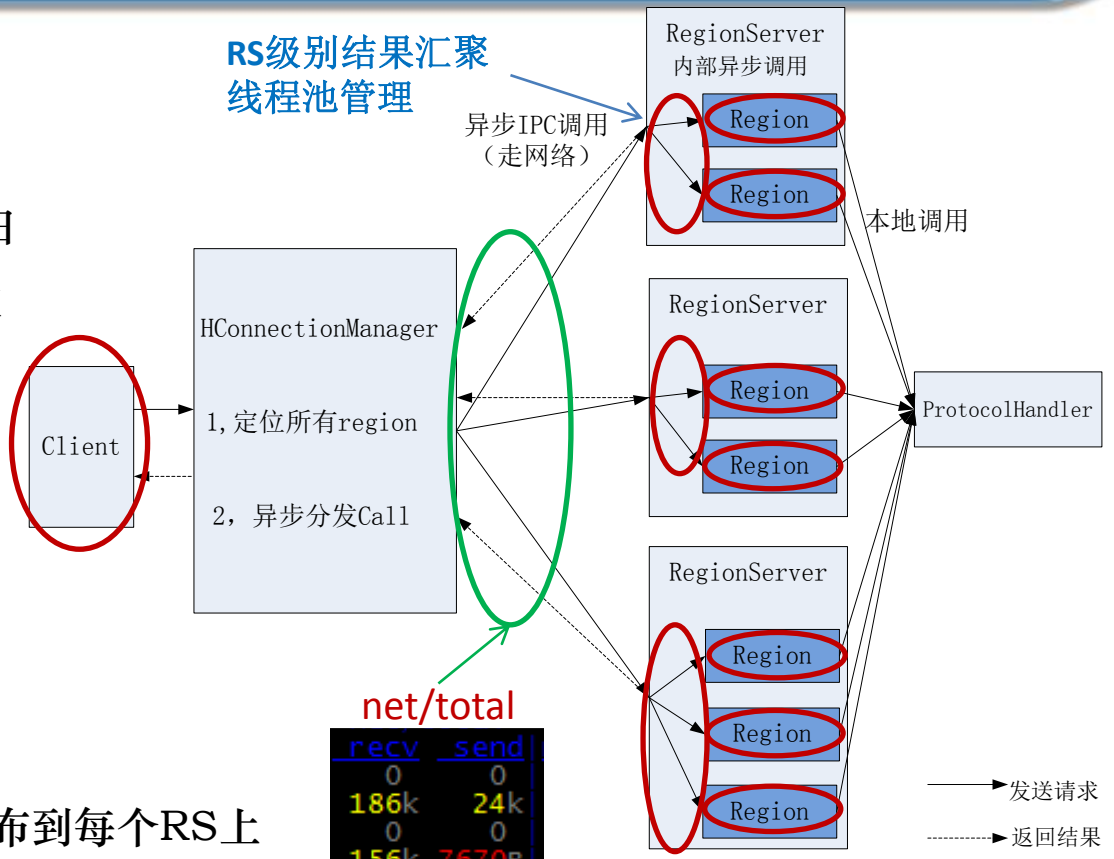
HBase Coprocessor优化 – CP本地汇聚

- 目前Apache Hbase社区的实现机制是以Region为单位执行请求，每个请求直接发送到Region上，每个Region执行处理后将结果直接返回给Client
- 存在的问题
  - 所有汇总计算集中在client，其CPU和内存成为瓶颈
  - 当返回数据量大时，客户端网络成为瓶颈



HBase Coprocessor优化 – CP本地汇聚

- CP本地汇聚
  - 以RS为单位发出CP计算请求
  - 每个RS对其管理的Region并行扫描，扫描结果在RS节点先做一次汇总
  - 当RS上所有Region均计算完毕，则RS将其本地汇聚结果返回给Client
  - Client将各RS返回结果进行汇总
- 优点
  - **计算分摊**：Client端的计算被分布到每个RS上
  - **减轻网络负载**： 减轻Client端的网络负载
    - Client只需要接收10MB*15 = 150MB的数据
  - **编程灵活**：可以分别定义RS和Region级别的处理函数



net/total

recv	send
0	0
186k	24k
0	0
156k	7670B
0	0
2315k	35k
0	0
4280k	110k
0	0
4226k	25k
0	0
246k	23k
0	0
10M	109k
0	0
2245k	18k
0	0

采用 ***CP本地汇聚*** 后，
LOG 查询的执行结果为：
34.86秒 ， 性能提升了**72%**

HBase Coprocessor优化 – CP本地汇聚

- API：原有CP API没有变动，增加了新的接口可使用CP本地汇聚
  - Client端采用*Htable.coprocessorExecRS()*
  - 接口定义与原CP相同
  - 接口实现需做判断，以便在Region上调用对Region的处理函数，在RegionServer上调用对整个RegionServer数据的处理函数

```
if(result==null&&oneRegionValue==null&&serverName==null)
        return getRegionTop(ci,scan);
 else
        return getRSTop(ci,scan,serverName,result,oneRegionValue);
```

serverName：RS的名称
result, 要返回的结果
oneRegionValue： 一个region上的结果，用这个值去更新result

总结

- 经验总结
  - HBase Coprocessor提供Observer和Endpoint两种实现方式，Endpoint方式可用于在HBase内部实现分布式并行计算
  - Region的本地化调度策略，可以很好的提高HBase数据读写效率
  - Coprocessor的本地汇聚功能，可以降低Client端网络、CPU和内存开销，将计算分摊到各个RegionServer
- 遇到的主要问题
  - Coprocessor是HBase进程内的分布式处理框架，当CP计算较复杂且中间结果过多时，会占用大量内存，造成HBase内存不足而发生故障 (HBASE-4047 Generic external process host)
  - 没有容错机制，速度最慢的RS节点会拖慢整个计算执行进度 (HBASE-5047 Implement child failure strategies for external coprocessor host)

谢 谢

谢谢项目组各位同事，特别是钱岭、张宝海和邓鹏对本工作的贡献！