高性能NoSQL系统 BladeCube的设计与优化

Sohu北京研发中心

陈伟

BladeCube的设计与优化

应用背景

设计与细节

性能优化

应用背景

1、 云存储的元数据管理

为Sohu的大规模云存储系统管理元数据，比如文件版本信息，用户自定义元数据等，规模为单集群数千亿-万亿条记录。每条记录通常有十几个列，单列，多列的修改和读都非常频繁。对延时非常敏感。

应用背景

2、 权限管理

作为权限管理的数据库，数据量不太大，但访问频率非常高，热点数据集中，单个表的访问目前已可达3w tps。

应用背景

3、 常规的数据库场景

作为NoSQL数据库，相比于关系型数据库在一些事务，分析，跨行跨表的请求中有一些劣势，但是BladeCube通过提供一些常用的单行事务，服务端的计算功能，来满足一般应用的需求。

应用背景

Why Not MySQL/MongoDB/…?

1. 可扩展性
2. 高可用
3. 错误恢复

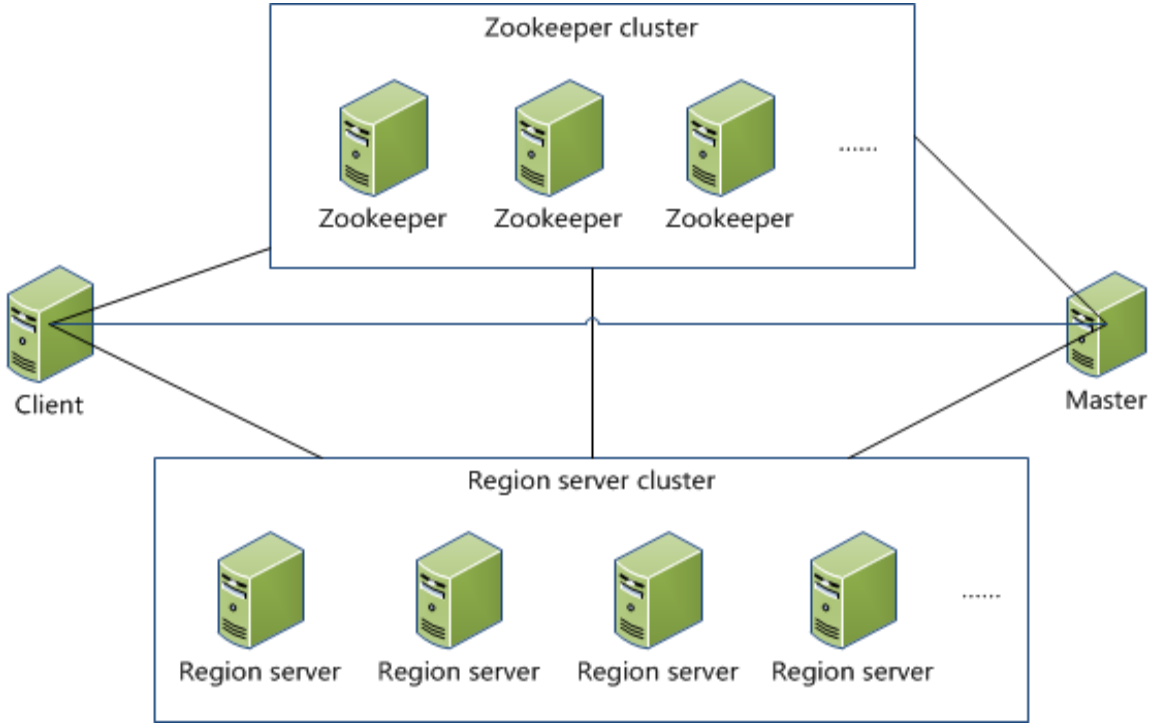
应用背景

Why Not HBase?

1. 性能 Java vs C++
2. 功能迭代
3. 定制化需求

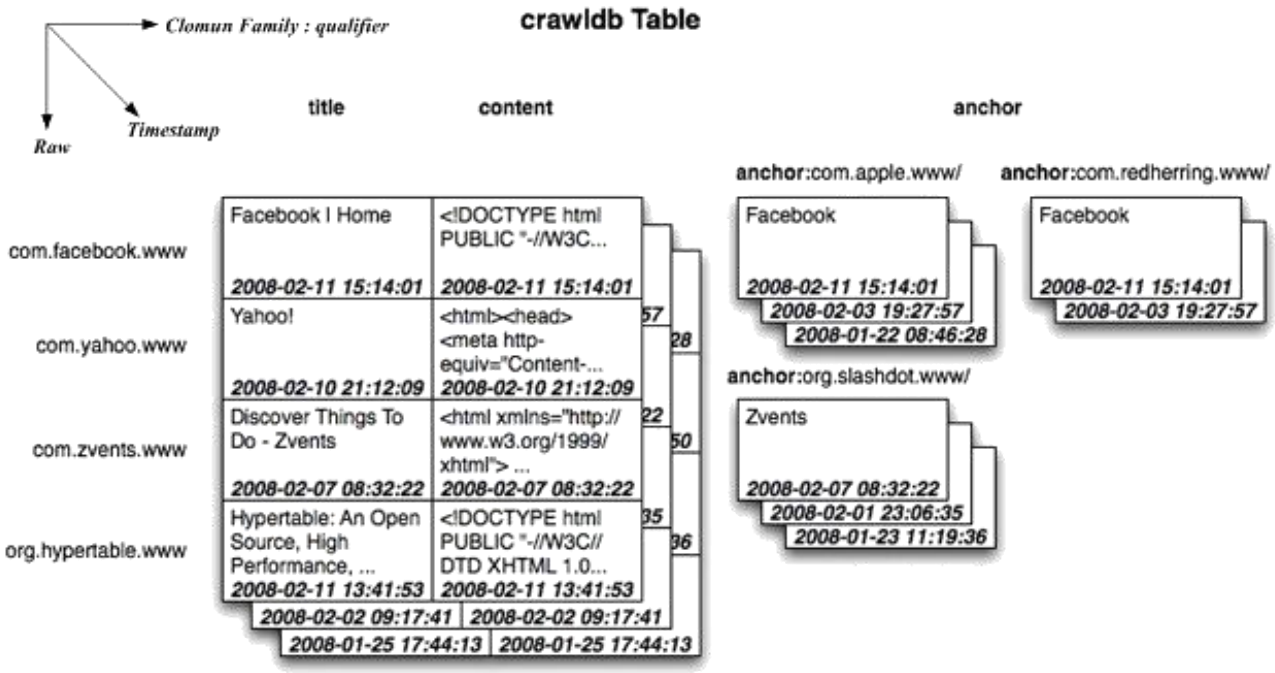
设计与细节

BladeCube的设计参考了BigTable和HBase



设计与细节

BladeCube多维数据模型



设计与细节

底层存储系统：

是采用我们独立开发的BladeStore系统，支持三备份的冗余存储，高可用。自动恢复的NameServer。

支持安全写接口，在GFS的流式写入之外，提供了更低延时的安全写接口，并发的向多个客户端发送请求，在写WAL时更有优势

设计与细节

数据存储模型：

内存数据：MemStore

持久化数据：SSTable

SSTable中有bloomfilter和前缀压缩

网络包采用google的protobuf

设计与细节

网络模型：

基于epoll的异步通讯库AmFrame

支持单或多IO线程，高吞吐量，能够在各种大小的数据包下压满网络带宽

数据安全：

支持WAL，数据刷入到底层存储之后再操作。

设计与细节

负载均衡：

基于读写请求量，内存占用量等指标计算出的综合数据评定。

Region分配，Split：

依赖于Zookeeper模块提供的全局视图保证一致性。

通过进度通报来降低出错后的恢复时间

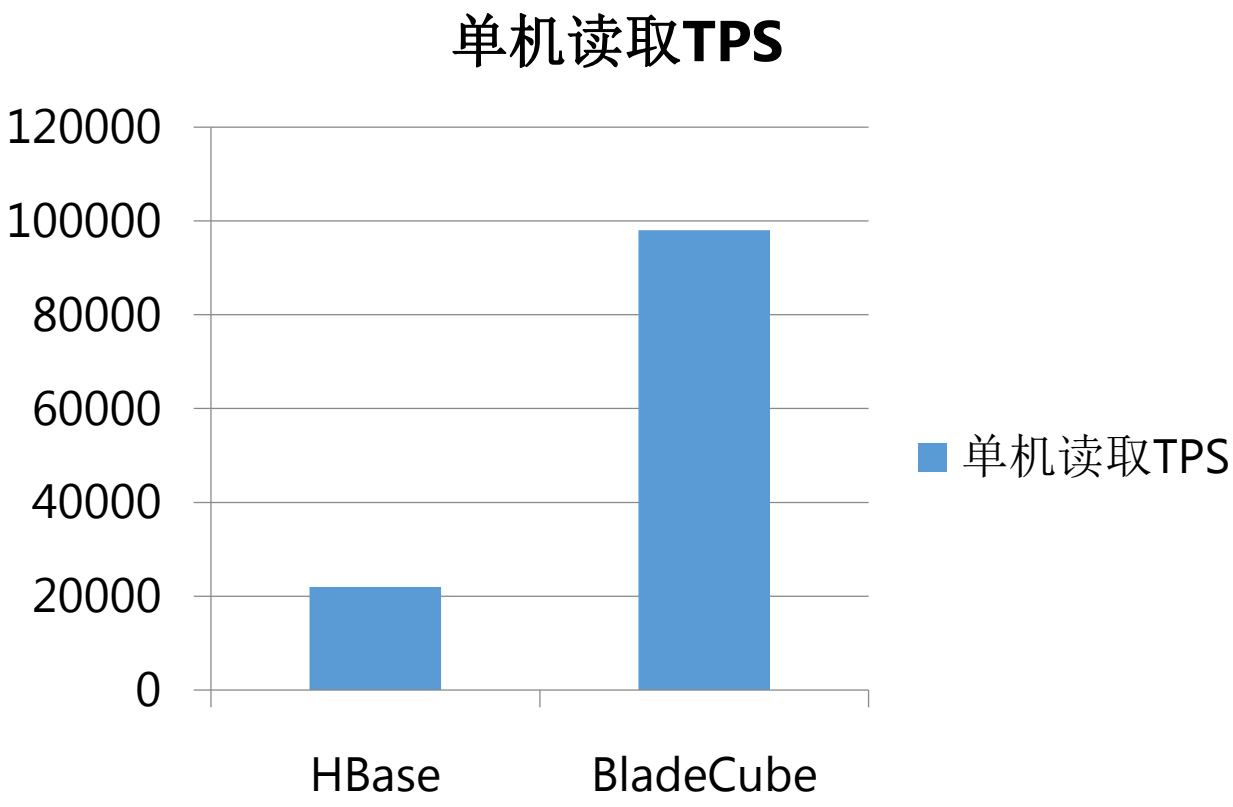
性能优化

优化效果：

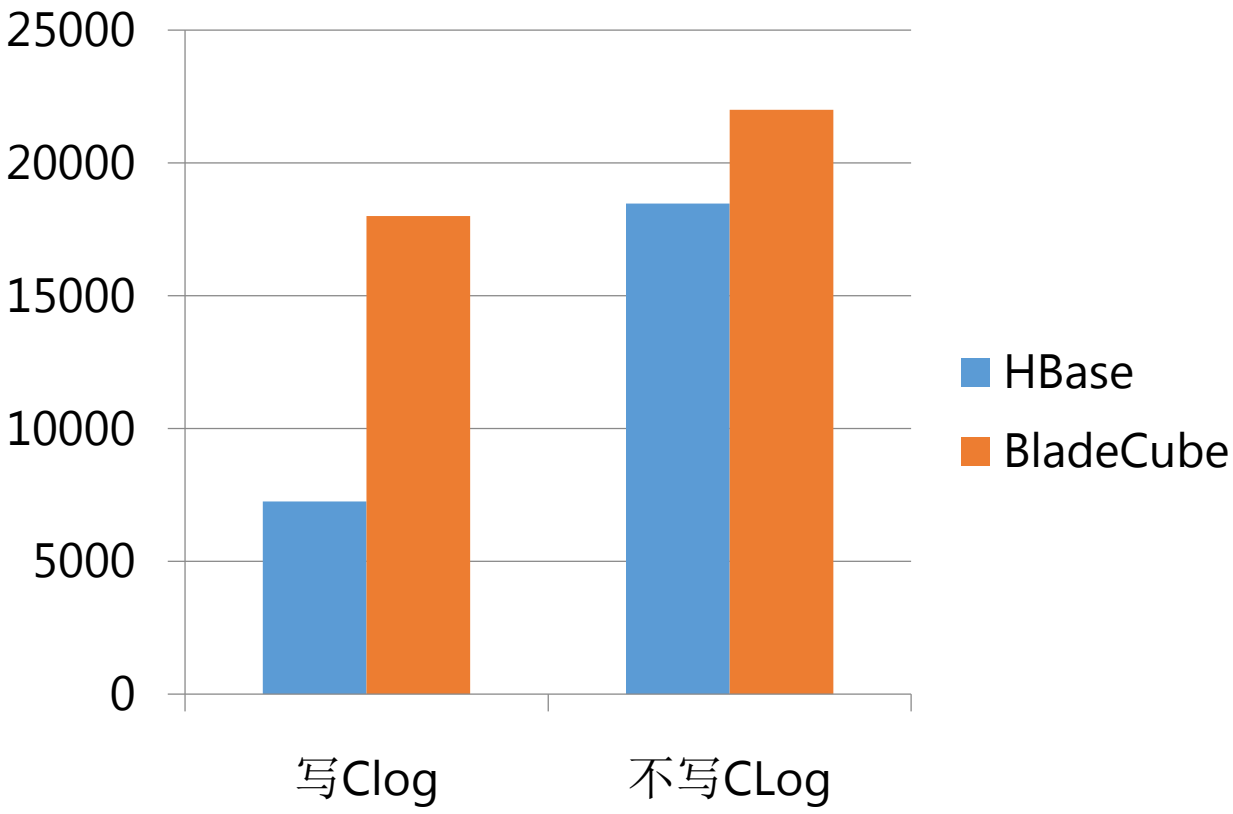
测试数据：

单个表，单个列族，10列数据，每次读写都是同时访问这10列数据，每列的value长度在10字节，rowkey在100字节以内。

性能优化


单机读取TPS
120000
100000
80000
60000
40000
20000
0
HBase
BladeCube
单机读取TPS

性能优化


25000
20000
15000
10000
5000
0
写Clog
不写CLog
HBase
BladeCube

性能优化

高并发写入：

采用并发跳表，基于C++11的实现

写入性能相比于加锁的数据结构有质的飞跃

在32个线程并发写入的情况下，tps从11w提升到500w。

性能优化

读取性能：

减少内存拷贝

C++11的右值引用

Intel指令集优化（64位大小端转换，memcmp）

性能优化

读取性能：

C++11的右值引用：

```
template <class T> swap(T& a, T& b)
    {
        T tmp(a);   // tmp对象创建后，我们就拥有了a的两份拷贝
        a = b;      // 现在我们拥有b的两份拷贝
        b = tmp;    // 现在我们拥有a的两份拷贝
    }
```

性能优化

读取性能：

C++11的右值引用：

```
T&& move(T&& a){
    return a;
}
template <class T> void swap(T& a, T& b){
    T tmp(move(a)); // 对象a被移动到对象tmp，a被清空
    a = move(b);    // 对象b被移动到对象a，b被清空
    b = move(tmp);  // 对象tmp被移动到对象b
}
```

性能优化

读取性能：

Intel指令集优化（64位大小端转换，SSE，memcmp）

对于频繁调用的这种数值运算，采用intel指令集优化之后性能会有明显提升

性能优化

内存分配：

避免内存碎片，Arena

通过内存池的分配来减少内存分配消耗的时间

精确控制生命周期，Slice

C++11的智能指针，shared_ptr,unique_ptr

性能优化

Scan管理：

HBase的scan性能很差

延时非常不稳定，在长期的几百-几千tps的scan操作时会经常出现较长时间的访问。

优化了的scan_id管理，大部分情况无锁。实际性能远超HBase。

采用动态扩展的scan_id列表，scan超时检测延时生效等，可以做到scan_id的管理几乎不对性能有影响。

性能优化

Region上下线管理：

并发读取多个sstable，2-5倍的性能提升

将sstable列表读出后，分成多个任务，多线程同时进行读取和分析。

缩短region server宕机之后恢复时间

性能优化

Split流程：

HBase的Split流程是各大运营hbase的团队优化的重点，甚至通过预分配直接取消自动触发split的机制。

在bladecube中，对Split进行了特殊的处理，做到了不影响在线服务。在split过程中，首先创建子region，并同时向父region和子region写入数据，后台进行创建ref等操作，最后在meta中进行切换

总结

C++是双刃剑，用好了很犀利，用不好很悲催

性能总是还可以继续优化，不过要看性价比

百度技术沙龙

关注我们：t.baidu-tech.com

资料下载和详细介绍：infoq.com/cn/zones/baidu-salon

“畅想•交流•争鸣•聚会”是百度技术沙龙的宗旨。 百度技术沙龙是由百度与InfoQ中文站定期组织的线下技术交流活动。目的是让中高端技术人员有一个相对自由的思想交流和交友沟通的的平台。主要分讲师分享和OpenSpace两个关键环节，每期只关注一个焦点话题。

讲师分享和现场Q&A让大家了解百度和其他知名网站技术支持的先进实践经验，OpenSpace环节是百度技术沙龙主题的升华和展开，提供一个自由交流的平台。针对当期主题，参与者人人都可以发起话题，展开讨论。