玄陵

HBase 和 Cassandra的浅谈

11月前 5897

一：简介

关于hbase和cassandra的探讨，主要从多个方面来说，从最基本的一步步往深入的说，从最基本的部署运维，到支持特性，使用功能以及各自的优势；

先从比较宏观的角度分析，HBase是基于Google的bigtable的论文实现的列式数据库，cap理论中更倾向于强调c（副本数据一致性）和p（分区容错性）。而Cassandra是号称dynamo 加上 bigtable（数据模型）的实现，cap中强调a（可用性） 和 p。而且底层都是使用lsm-tree 来做存储引擎的核心数据结构；

HBase基本上就是bigtable的开源实现，对标的bigtable的chubby（分布式锁服务）HBase有对应的zookeeper，HBase用regionserver管理region里的数据，实际上的数据副本容错的机制是交付给底层的分布式文件系统hdfs，类比bigtable的gfs，HBase有hmaster做元信息管理。Cassandra在架构上更多借鉴了dynamo，一种完全的区中心对等的分布式数据库，她的每个节点维护一份元信息，每一个节点在集群中的身份完全一样。

二：部署运维

单纯的就部署和运维hbase以及Cassandra来说，部署hbase前，需要部署的组件有zookeeper，hdfs，然后才是hbase。对应的Cassandra就比较简单很多，编译完成一个jar包，单台服务器启动一个Cassandra进程即可。

在部署hbase的时候，可能需要规划好，哪些机器跑hmaser，rs,zk,hdfs的相关进程等，还有可能为了集群的性能，还要预先规划好多少个rs。自己人工去部署这么一个hbase集群还是比较麻烦的，更别提自己维护（阿里云ApsaraDB-HBase你值得拥有）。

Cassandra部署的时候比较简单，一个tar包搞定，由于cassandra数据落本地盘，需要人为的配置一些参数比如是否需要虚拟节点（vnode）以及多少vnode；需要基于业务的场景选择特定的key的放置策略（partitioner），这个放置策略的选择以及一些参数的配置需要一定的门槛。

简单总结下：部署运维的话，hbase依赖组件多，部署麻烦一点，但是相关资料很多，降低了难度；cassandra部署依赖少，但是配置参数多，相关资料较少。

特别是使用云HBase完全避免了部署造成的各种麻烦，比手工部署运维任何大数据数据库都方便太多。

三.特性对比

3.1.特性概述

HBase的社区文章里面有介绍hbase的特性点，翻译过来有如下的几个点：

1.强一致性的读写：HBase不是一个最终一致

2.自动sharding：HBase的table在集群种

3.regionserver的failover；

4.Hadoop/HDFS的集成；

5.MapReduce：支持大数据的并行处理；

6.JAVA Client 以及Thrift/RESR API 访

7.Block Cache 以及Bloom filter;

8.操作管理。

就上述HBase的特性，C*也有相应的自己的特性：

1.C*借鉴Dynamo的架构思想，把自己叫做一

2.C*的sharding方式：一致性hash，有2种

3.可以容忍：replicator_number - (rea

4.支持MapReduce;

5.Thrift、CQL访问;

6.大数据处理的bloom filter 必备;

7.自己有jmx等常见管理，且datastax 公司

从上述特性来看，各个数据库各有千秋，但是有各自的适合场景，如果业务对数据一致性要求比较苛刻，那么HBase可能更合适，毕竟C 还是存在一定的问题，比如删除数据可能复现；C推荐放置策略用Random以及Murmur3等方式，这样把key打散的很随机，以此做节点负载均衡，这样做scan业务自己的需求数据，可能数据库要全集群都访问，当然可以使用OrderPreserving 和ByteOrdered可以不用全集群都访问，可是负载不是很均匀，HBase这点却支持的不错；就易用性来说，Cassandra做的还是不错的，在数据库内部提供cql，一种类sql的语句。HBase还是主要JAVA API/Thrift的接口支持，但是后续HBase这方面应该会提高，前段时间的HBase 2017 Asia 阿里也介绍了sql on HBase。

后续在易用性这一块HBase应该是越来越好，会弥补上在易用性这一块的欠缺。试想下，如果Nosql都支持大部分的SQL的功能和语法，那是多完美的事情。

3.2.特性对比

本小结就 HBase 和 C*在部分相关特性上进行细一步的分析对比，大概对比的部分选择为：一致性；sharding方式；用户使用方式；数据复制；

3.2.1 一致性：

一般的一致性在这里是指副本数据的一致性，我们常用的是3副本用来做冗余。HBase的话主要是依赖底层的HDFS来支持副本冗余，一次写入被regionserver收到以后，发送给底层的HDFS,而HDFS会对应的给这个数据在整个文件系统里面写多份；这里是多份副本全部写完成才会返回。读数据的话，在底层HDFS的处理是读主要的这份数据就可

以，因为多份数据都是一样的（正常情况）。

C在建表的时候会指定表的副本数（常见3副本），一次数据写入，会基于当前表的副本数以及节点的snitch策略来找到需要写的数据节点，发出多份请求（3份），然后基于传递的写入级别等待对应的响应数即可，比如A,B,C三个副本，QUORUM级别写入，可能存在的情况是，A,B成功C写入失败(无论是网络还是节点跪了)，那么这个时候3个副本实际上是不一致的。但是这里对于 C 来说，她能忍。而且如果你读的级别是QUORUM可以读到最新的数据，但是如果你用ONE级别去读数据，那么存在读到老数据的风险，但是 C 存在这种情况（当然这里先不考虑C 内部的机制自己修回副本数据）。

C *由于不能保证副本一致性，自己提出了几种方式来做弥补：1.读修复；2.Hinted-Handoff；3.Anti-entropy repair ;但是各种机

制的引入也不是很完美的解决问题，此外还相应的会引入一些问题，比如使用1的话，那些读不到的数据存在一直数据不一致的风险；使用3的方式去进行全量/增量数据对比，会消耗很多物理资源，影响在线服务的请求，这是在线服务不能忍受的。

这种情况下来看，HBase的读写以及数据一致性模型是比较简单的。简单就是美，这对具体的业务场景进行适配也是很nice的一面，如果一个数据库的模型过于复杂，业务方拿来用也需要很高的门槛。

2.2.2 shadring 方式：

HBase 的各个regionserver 在最初负责的region，是可以在最初的建表时候，可以做预分配，也可以让hbase自己做这件事情，那么每个regionserver就会负责相应的region的数据的读写等。对于出现热点region的情

况的话，hbase自己支持region的split操作，将热点region一分为二。

C* 不支持所谓的热点数据spli region的功能，那么对于这种情况的话，她做了一个预先设置，输入的数据做hash打散，也就是我们知道的一致性hash，她内部支持4种hash策略，Murmur3，Random，OrderPreserving等，其中，前2种是做了随机的hash，OrderPreserving 是类似字典序的方式，最初无论是使用vnode的方式还是initial_tokne的方式人为设置节点token，来一个请求，计算随机hash可以把key比较随机打散到集群中的某个节点，通过snitch 和keyspace 副本方式找到落得节点信息，因为前面的随机hash可以人为是比较随机的，那么这实际上可以理解为一种负载均衡。但是如果是OrderPreserving 这种方式，实际上就会有问题；出现热点也没办法；

而且我们知道随机hash如果要做scan，实际上是很蛋疼的，基本上所有节点都要操作。字典序可以避免操作所有的数据节点，从这点看，HBase还是占点优势。

2.2.3 用户使用方式

HBase现在提供给用户主要是JAVA/THrift/REST的接口,大概的操作也就是CRUD操作；这些使用还是不是具有亲和性。比如我现在要往表“table”里面写一条数据，rowkey是“test” 列是“cn1” "cn2"，value是“value1”, "value2"那么我需要进行下面的操作：

```
Configuration conf = HBaseConfi

HTable htable = new HTable( con

Put put = new Put(Bytes.toBytes
```

```
        put.add(Bytes.toBytes("cn1"), B

            Bytes.toBytes("value1"));

        put.add(Bytes.toBytes("cn2"), B

            Bytes.toBytes("value2"));

        table.put(put);
```

但是对于C 而言，支持2种方式,1.THrift；2.CQL；thrift我们就不讨论了，主要介绍CQL，这是一种类sql的语言，主要用于操作C,那么通样写一条数据cql是：

```
 INSERT INTO xxx.table (cn1, cn2) VA
```

此外C* 支持多种索引，特别是有一种新的索引，sasi index支持prefix，contains等操作；

不过HBase也有支持sql的组件且也在往这边发展也不是太大问题；

2.2.4 数据复制

一般数据复制，有全量数据复制和增量数据复制2个情况，主要是为了集群的高可用做铺垫，全量复制的时候，有的使用copytble的方式，这边比较简单，但是因为是mr做批量读取写入（我们之前就是用这种方式导Cassandra的数据），但是会比较耗时，因为每次请求一个网络来回；也有基于snapshot做复制的方式，这种方式会好点；增量复制的话，使用的是复制Hlog的方式，这是一种异步的复制方式，在zk记录checkpoint，复制完log以后修改checkpoint的问位置。

C 的数据复制，也与2种方案，但是都是需要时在Network的拓扑配置下，同cluster的不同dc环境下的操作，全量复制是一种叫做

rebuild的方式，直接拖数据的stream，类似C 做节点启动bootstrap的过程，C* 的增量复制 提供了2种方式：EACH_QUORUM,LOCAL_QUORUM 前者是同步写主备，后者是写本地，返回client，远端是否成功不care；对于这个而言的话，异步存在丢数据的风险，同步在跨region以及并发量大的情况下，请求失败率会很高。但是，上面的方式是需要部署在同clsuter下面的不同dc，不同cluster没有解决方案。此外对于同步和异步方案之间没有一种折中的方式，毕竟有的场景对性能要求高，同步复制影响线上写成功的概率，异步复制会丢数据。但是阿里HBase在这方面去做了一个同步异步并存的方式，同步异步共存复制。

四.各自优势

通过上述 的描述以及各自的功能特性的对比，我们可以得到HBase 和 C 对比，2种产品各有千秋，其中HBase对数据一致性要求

更高，C相应的强调可用性。且现阶段C 的接口比较有亲和性，但是HBase 和 Hadoop 系统天然的无缝对接，这是C 还有点欠缺的。C *也使用了各自方式去弥补欠缺，但是实际上数据一致性上做的还是和这种强一致的系统差把火，而HBase在可用性上所做的工作实际上个人觉得可以做的更好，亲和性以及易用性这些事情实际上HBase也在做工作。

由于篇幅限制，还有一些别的内容没有写进来，接下来会一步步补齐的。接下来会就Compaction策略的对比，索引实现的对比进行分析。

数据存储与数据库 分布式 大数据

hbase 数据库 配置 集群

负载均衡 同步 HASH Cassandra

数据节点

作者

玄陵

TA的文章

手把手教你通过Thrift 访问ApsaraDB for HBase

HBase的备份以及恢复方案

HBase read replicas 功能介绍系列

相关文章

9月6日云栖精选夜读：DMS前后端技术揭秘及最佳实践

浅谈开源大数据平台的演变

浅谈开源大数据平台的演变

图数据库浅谈

Http协议中Get和Post的浅谈