# 起源

2006年，Google发表了一篇论文《BigTable：A Distributed Storage System for Structured Data》，介绍了一种数据模型——多维稀疏排序模型。

2007年，Powerset公司基于这篇文章研发了BigTable的开源版本，即HBase。

2008年，HBase成为Apache的顶级项目。

注：Google三驾马车：分布式文件系统GFS（开源实现：HDFS），分布式计算框架MapReduce（开源实现：Hadoop MapReduce），分布式结构化数据存储BigTable（开源实现：HBase）。

参考：<https://blog.csdn.net/qq_45786945/article/details/120248130>

# 概述

Apache HBase是一种Key/Value系统，它运行在HDFS之上。**和Hive不一样，Hbase的能够在它的数据库上实时运行，而不是运行MapReduce任务**。Hive被分区为表格，表格又被进一步分割为列簇。列簇必须使用schema定义，列簇将某一类型列集合起来(列不要求schema定义)。例如，"message"列簇可能包含："to", "from" "date", "subject"和"body"。每一个 key/value对在Hbase中被定义为一个cell，每一个key由row-key，列簇、列和时间戳。在Hbase中，行是key/value映射的集合，这个映射通过row-key来唯一标识。Hbase利用Hadoop的基础设施，可以利用通用的设备进行水平的扩展。

## 特点

HBase通过存储key/value来工作。

它支持四种主要的操作：增加或者更新行，查看一个范围内的cell，获取指定的行，删除指定的行、列或者是列的版本。

版本信息用来获取历史数据(每一行的历史数据可以被删除，然后通过Hbase compactions就可以释放出空间)。

虽然HBase包括表格，但是schema仅仅被表格和列簇所要求，列不需要schema。Hbase的表格包括增加/计数功能。

## 限制

HBase查询是通过特定的语言来编写的，这种语言需要重新学习。类SQL的功能可以通过Apache Phonenix实现，但这是以必须提供schema为代价的。另外，Hbase也并不是兼容所有的ACID特性，虽然它支持某些特性。最后但不是最重要的——为了运行Hbase，Zookeeper是必须的，zookeeper是一个用来进行分布式协调的服务，这些服务包括配置服务，维护元信息和命名空间服务。

## 联系

### GFS

BigTables的数据持久化到GFS。

### Chubby

## 区别

### HBase vs Redis

参考：<https://www.jianshu.com/p/74bd352d7493>

都是NoSQL数据库，存储的是KV键值，但是HBase是基于磁盘的K-V存储，而Redis是基于内存的K-V存储，二者有着本质区别。

### HBase vs LevelDB

都是基于磁盘的K-V存储，LevelDB可以认为是HBase的单机实现。

参考：

<https://zhuanlan.zhihu.com/p/91146725>

<https://blog.csdn.net/u011047968/article/details/125298135>

### HBase vs Hive

Hive和Hbase是两种基于Hadoop的不同技术，Hive是一种类SQL的引擎，并且运行MapReduce任务，Hbase是一种在Hadoop之上的NoSQL 的Key/vale数据库。当然，这两种工具是可以同时使用的。就像用Google来搜索，用FaceBook进行社交一样，Hive可以用来进行统计查询，HBase可以用来进行实时查询，数据也可以从Hive写到Hbase，设置再从Hbase写回Hive。

### HBase vs TiDB

HBase 和 TiDB 都是分布式数据库系统，但它们的设计目标、架构、使用场景和底层实现有显著区别。下面从多个维度对它们进行系统对比。

1、HBase 与 TiDB 简介



2、架构对比

1）HBase 架构（依赖 Hadoop）

[客户端]

↓

[HMaster + RegionServer]

↓

[HDFS]

存储层：HDFS（不可更新，追加写）

数据管理：RegionServer 管理区域（Region），每个 Region 存储表的一部分

写入流程：写 WAL（HLog）→ MemStore → Flush 到 HFile（HDFS）

2）TiDB 架构（存算分离）

[TiDB Server] ← SQL 接口（兼容 MySQL 协议）

↓

[Placement Driver (PD)] ← 分布式事务、调度元信息

↓

[TiKV（分布式 KV 存储）或 TiFlash（列式副本）]

存储层：TiKV 是 RocksDB+Raft 组成的 KV 存储；TiFlash 是列式 OLAP 存储引擎

分布式事务：实现 Percolator 模型，支持 ACID

SQL 层：完全兼容 MySQL 协议

3、关键技术对比



4、使用场景对比



5、优缺点总结

1）HBase

优点：

适合海量数据存储（PB 级）

写入吞吐高

灵活的列族模型，支持稀疏数据

缺点：

查询弱（无 SQL 支持）

最终一致性，开发难度大

依赖 Hadoop，部署复杂

2）TiDB

优点：

兼容 MySQL，SQL 体验好

分布式事务支持强一致

一套系统支持 OLTP + OLAP（HTAP）

运维简单，扩展性强

缺点：

写入放大较高（RocksDB）

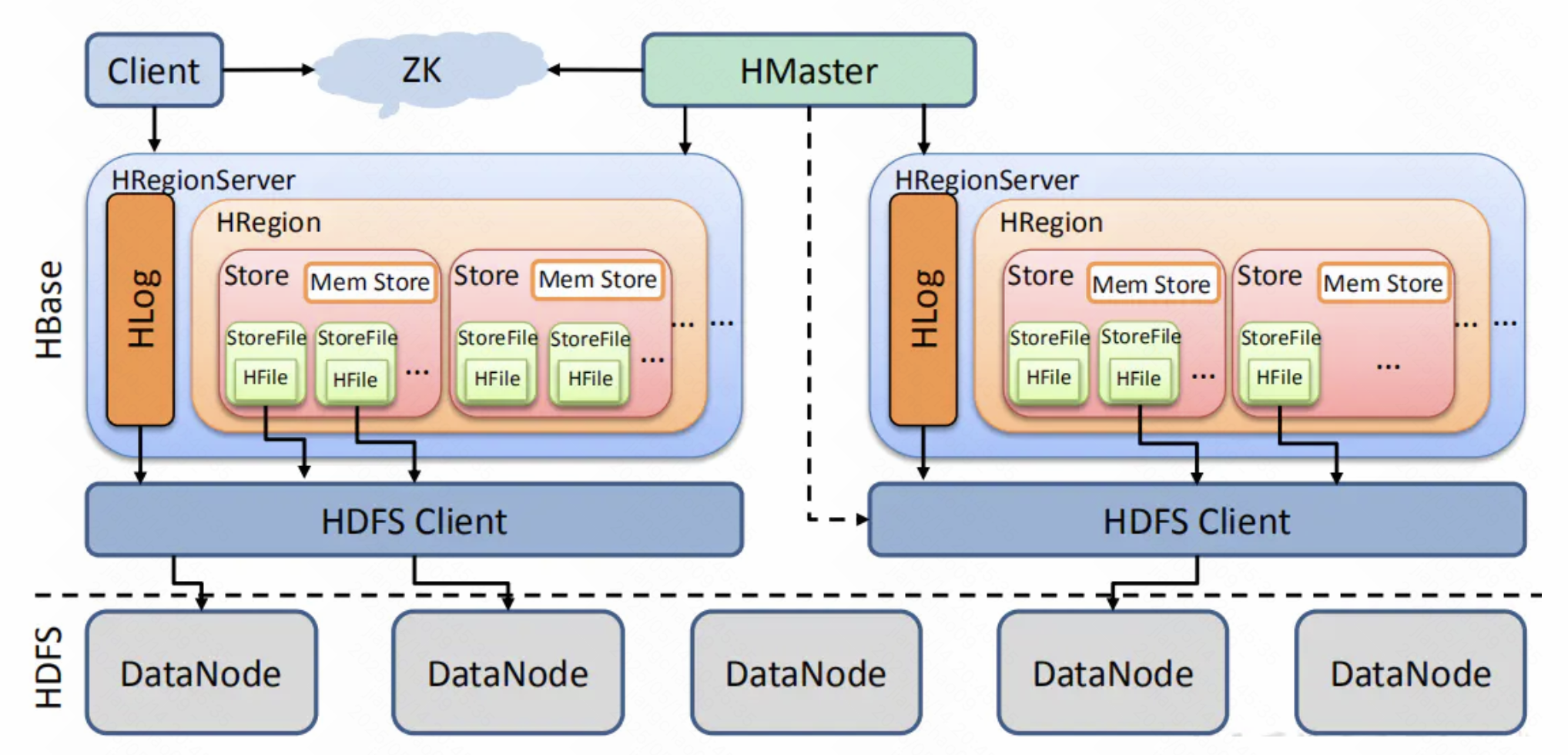
某些极端高写入场景不如 HBase

对大宽表或非结构化数据支持较弱

6、总结建议



# 架构



架构角色：

## Master

Master是所有Region Server的管理者，其实现为HRegionServer,主要作用有:

对于表的DDL操作：create，delete，alter；

对于RegionServer的操作：分配regions到每个RegionServer，监控每个RegionServer的状态，负载均衡和故障转移。

## Zookeeper

HBase通过Zookeeper来做Master的高可用、RegionServer的监控、元数据的入口以及集群配置的维护等工作。

## WAL

由于数据要经MemStore排序后才能刷写到HFile，但把数据保存在内存中会有很高的概率导致数据丢失，为了解决这个问题，数据会先写入Write-Ahead logfile的文件中，然后再写入到Memstore中。所以在系统出现故障的时候，数据可以通过这个日志文件重建。

## MemStore

写缓存，由于HFile中的数据要求是有序的，所以数据是先存储在MemStore中，排好序后，等到达刷写时机才会刷写到HFile，每次刷写都会形成一个新的HFile。

## StoreFile

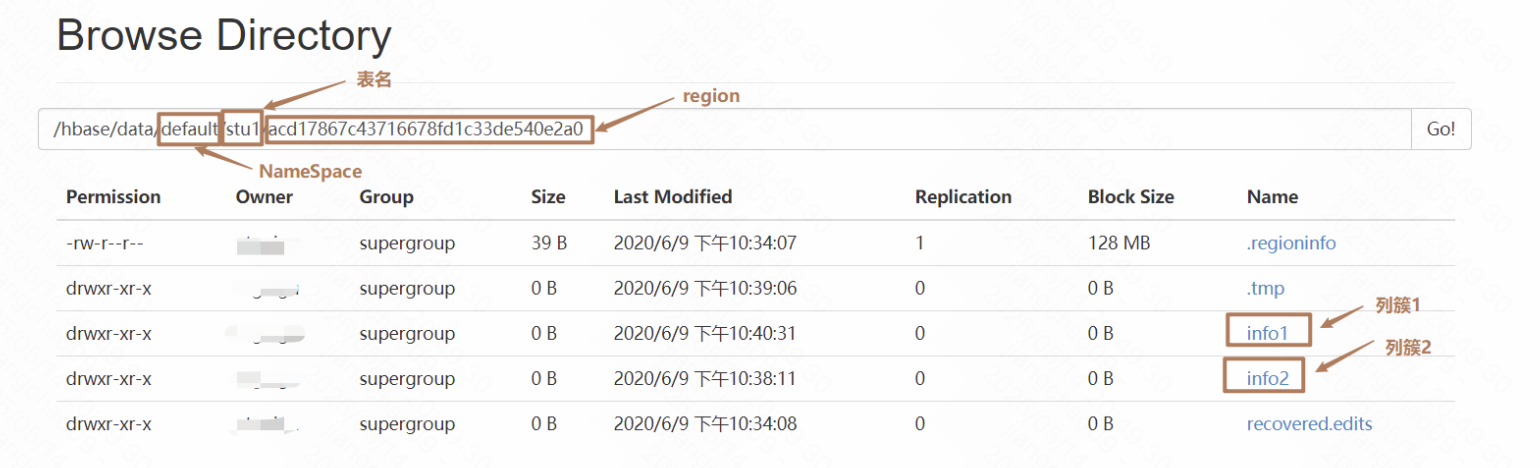
保存实际数据的物理文件，StoreFile以HFile的形式存储在HDFS上。每个Store会有一个或多个StoreFile(HFile)，数据在StoreFile上是有序的。

# 原理

参考：

<https://www.cnblogs.com/bbgs-xc/p/13070724.html>

## 数据模型



### Name Space

命名空间，类似于关系型数据库的DataBase概念，每个命名空间下有多个表。HBase有两个自带的命名空间，分别是hbase和default，hbase中存放的是HBase的内置表，default表示用户默认使用的命名空间。

### Region

类似于关系型数据库的表概念。不同的是，HBase定义表时只需要生命列簇即可，不需要声明具体的列。这意味着，往HBase写入数据时，字段可以动态、按需指定。

### Row

HBase表中的每行数据都由一个RowKey和多个Column(列)组成，数据是按照RowKey的字典顺序存储的，并且查询时智能根据RowKey进行检索，所以RowKey的书籍十分重要。

### Cloumn

HBase中的每个列都由Cloumn Family（列簇）和Cloumn Qualifier（列限定符）进行限定，例如info：name，info：age。建表时，只需指明列簇，而列限定符无需预先定义。

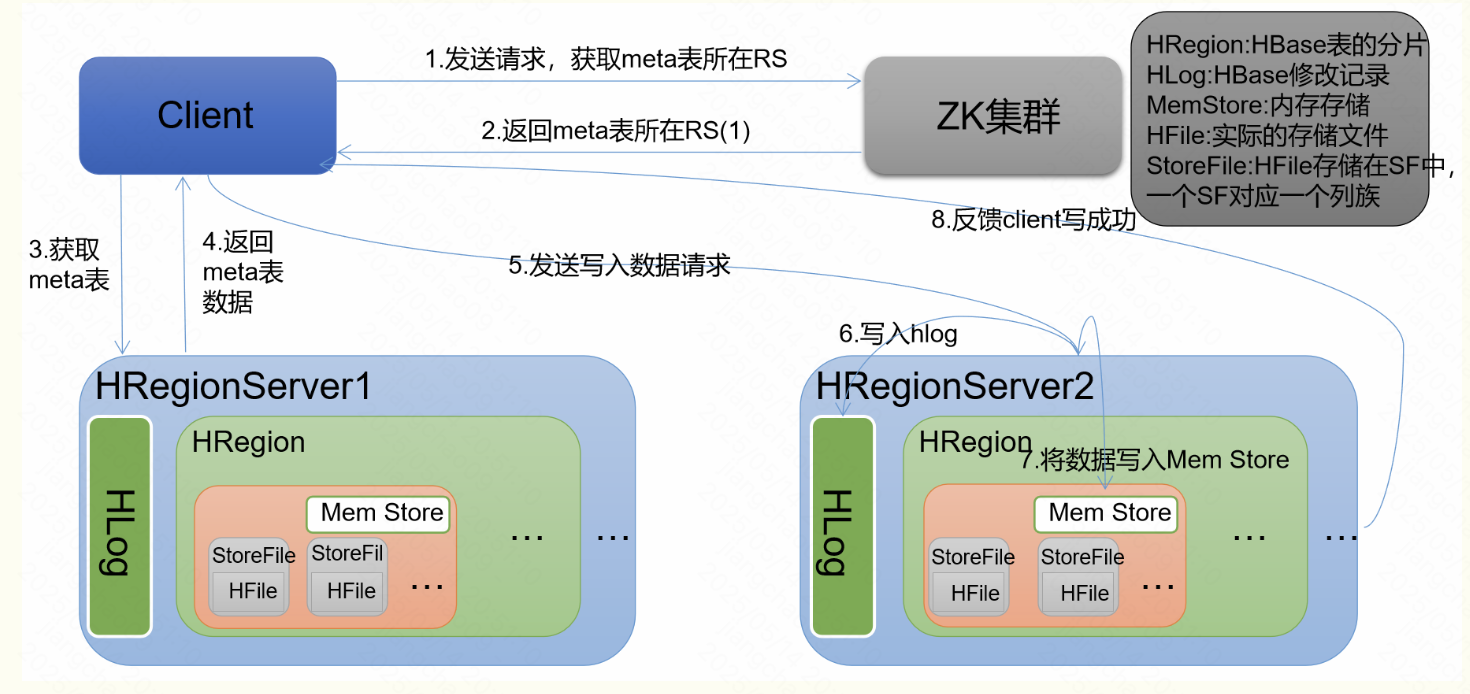
### Time Stamp

用于标识数据的不同版本（version），每条数据写入时，如果不指定时间戳，系统会自动为其加上该字段，其值为写入HBase的时间。

### Cell

由{rowkey, column Family：column Qualifier, time Stamp} 唯一确定的单元。cell 中的数据是没有类型的，全部是字节码形式存贮。

## 写流程



1、Client 先访问 zookeeper，获取 hbase:meta 表位于哪个 Region Server。 #zk get /hbase/meta-region-server

2、访问对应的Region Server，获取hbase:meta表，根据读请求的 namespace:table/rowkey，查询出目标数据位于哪个Region Server中的哪个Region中。并将该 table 的 region 信息以及 meta 表的位置信息缓存在客户端的 meta cache，方便下次访问。

#hbase scan 'hbase:meta' 查询到具体哪张表由哪个Region Server维护

3、与目标 Region Server 进行通讯；

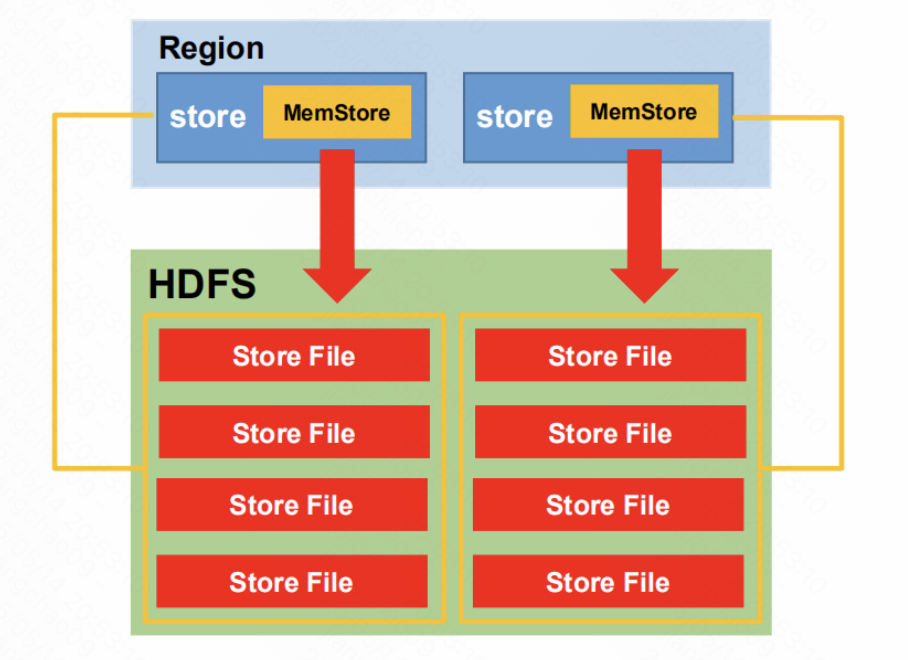
4、将数据顺序写入（追加）到 WAL；

5、将数据写入对应的 MemStore，数据会在 MemStore 进行排序；

6、向客户端发送 ACK；

7、等达到 MemStore 的刷写时机后，将数据刷写到 HFile。

### MemStore Flush刷写



1、当某个MemStore的大小达到了hbase.hregion.memstore.flush.size（默认值 128M），其所在 region 的所有 memstore (对应的列簇)都会刷写。

当达到128M的时候会触发flush memstore，当达到128M \* n还没法触发flush时候会抛异常来拒绝写入。两个相关参数的默认值如下：

hbase.hregion.memstore.flush.size=128M(默认)

hbase.hregion.memstore.block.multiplier=4(默认)

2、当 region server 中 memstore 的总大小达到

java\_heapsize(应用的堆内存)

hbase.regionserver.global.memstore.size（默认值 0.4）

hbase.regionserver.global.memstore.size.lower.limit（默认值 0.95），

region 会按照其所有 memstore 的大小顺序（由大到小）依次进行刷写。直到 region server中所有 memstore 的总大小减小到上述值以下。当 region server 中 memstore 的总大小达到java\_heapsize\*hbase.regionserver.global.memstore.size（默认值 0.4）时，会阻止继续往所有的 memstore 写数据。

3、到达自动刷写的时间，也会触发 memstore flush。自动刷新的时间间隔由该属性进行配置 hbase.regionserver.optionalcacheflushinterval（默认 1 小时）。

4、当 WAL 文件的数量超过 hbase.regionserver.maxlogs，region 会按照时间顺序依次进行刷写，直到 WAL 文件数量减小到 hbase.regionserver.maxlogs 以下（该属性名已经废弃，现无需手动设置，最大值为 32）。

## 读流程

1、Client 先访问 zookeeper，获取 hbase:meta 表位于哪个 Region Server。

2、访问对应的 Region Server，获取 hbase:meta 表，根据读请求的 namespace:table/rowkey，查询出目标数据位于哪个 Region Server 中的哪个 Region 中。并将该 table 的 region 信息以及 meta 表的位置信息缓存在客户端的 meta cache，方便下次访问。

）与目标 Region Server 进行通讯；

）分别在 Block Cache（读缓存），MemStore 和 Store File（HFile）中查询目标数据，并将查到的所有数据进行合并。此处所有数据是指同一条数据的不同版本（time stamp）或者不同的类型（Put/Delete）。

）将从文件中查询到的数据块（Block，HFile 数据存储单元，默认大小为 64KB）缓存到Block Cache。

）将合并后的最终结果返回给客户端。

## 事务

弱一致性。

# 应用

Hbase非常适合用来进行**大数据**的**实时查询**。Facebook用Hbase进行消息和实时的分析。它也可以用来统计Facebook的连接数。