一种基于 HBase 的数据持久性和可用性研究①

唐长城[1], 杨 峰[1,2], 代 栋[1], 孙明明[1], 周学海[1,2]

[1](中国科学技术大学 计算机科学与技术学院, 合肥 230027)
[2](中国科学技术大学 苏州研究院, 苏州 215123)

摘 要: HBase(Hadoop DataBase)是 Apache Hadoop 项目下的一款非关系型数据库, 它是一个基于列簇的开源数据存储系统, 关于 HBase 的研究和应用越来越受到关注. 由于 HBase 会在内存缓存数据后写文件系统, 所以缓存的大小成为影响系统性能的一个重要因素. 本文提出一种基于备份日志的持久性、可用性方案 Remote Log Process, 使得 HBase 能够在不同的缓存规模获得更好的写性能. 实验证明, 在保证数据的持久性和可用性前提下, RLP 能够在不同的缓存大小下获得稳定的性能, 并且在缓存不超过默认设置时明显提高写操作时间性能.

关键词: HBase; 持久性; 可用性; 预写日志; 写操作效率

Research of Data Durable and Available Base on HBase

TANG Chang-Cheng[1], YANG Feng[1,2], DAI Dong[1], SUN Ming-Ming[1], ZHOU Xue-Hai[1,2]

[1](College of Computer Science and Technology, University of Science and Technology of China, Hefei 230027, China)
[2](Suzhou Institute for Advanced Study, University of Scienceand Technology of China, Suzhou 215123, China)

Abstract: HBase, a NoSql database under Apache Hadoop, is an open source data storage system based on column family. Researches and applications based on HBase is more and more popular. But the size of memory buffer become a key factor to influence system performance as HBase will buffer data in memory before store them on file system. In this paper, we provide a new method based on copied log named Remote Log Process to make HBase perform better on write operation with different buffer size while keeping data durable and available. Experiments result indicates RLP can get a steady performance with different buffer size under the condition to guarantee durable and available of input data, while perform much better than pristine systems if the buffer isn't larger then default value.

Key words: HBase; durable; available; write ahead log; write performance

由于互联网上信息量的爆炸式增长, Web2.0 时代已经来临, 其所带来的海量数据的处理[1]也成为一个时代性的问题. 随着 Google 公开发表了三篇云计算领域的经典文章, 人们开始重新审视海量数据的存储和索引, 颇具颠覆性的非关系型数据存储系统开始发挥其巨大的作用. HBase[2]正是 Apache 参考 BigTable[3]所设立的一个开源项目, 它是 hadoop[4]的子项目, 一个非常典型的基于列族的非关系型数据库实现.

对于一个数据存储系统而言, 数据的持久性和可用性是其必须保证的基本原则. 在 HBase 的实现中, 系统有两个方法保证数据的持久性和可用性:

① 数据最终需要写入到磁盘上. 磁盘是非易失性存储介质, 通常认为磁盘上的数据是可以保障持久性和可用性;

② 引入了 WAL(Write Ahead Log)日志系统. 保证即使数据节点发生故障也能够提供有效的数据恢复能力, 通过日志也能保证尚未及时写入磁盘的数据的持久性和可用性.

当然, 这些复杂的保障性措施大大影响数据库的实际运行效率. 尤其针对大量数据的写入, 频繁写文件系统带来的磁盘 IO 操作必然会降低数据库的写操作效率. 因此, 本文旨在提出一种新的数据处理流程,

① 基金项目:江苏省产学研前瞻性联合研究(BY2009128);江苏省自然科学基金(BK2012194);国家自然科学基金(61272131)
收稿时间:2013-03-18;收到修改稿时间:2013-05-02

在写操作的过程中, 以保证数据持久性和可用性为基础, 通过增加在内存中存储的数据、减少写磁盘的频率, 提高数据写操作时间性能, 同时减小 HBase 写操作时间性能对 MemStore 大小的敏感度.

1 HBase存储结构

HBase 是根据 Google 的论文“BigTable: A Distributed Storage System for Structured Data”实现的. 它是基于列簇的分布式存储系统. HBase 在横向上把表切分成区域(Region), 每个区域包含的数据是表的一个子集. 对于任意一个表, 初始阶段只含有一个 Region, 但随着数据的增多, Region 的大小不断增长, 当到达一个阈值时, 就会以行为分界线, 分裂成两个类似大小的新 Region. 因此, 一个表的 Region 数是在动态增长的. 而一个表的多个 Region 可能分布在不同的 HRegionServer 上, 他们的分布由 HMaster 来负责, HMaster 会根据不同 HRegion Server 的负载情况进行调整[5].

1.1 HBase 存储模型

HBase 数据存储系统由功能不同的几个组件构成, 包括客户端, Zookeeper, HMaster, HRegionServer, HStore, HLog.

- 客户端: 使用 HBase 的 RPC 机制与 HMaster、HRegionServer 通信;
- Zookeeper: 存储-ROOT-表地址、HMaster 地址, 提供 HRegion 的信息;
- HMaster: 管理用户对表的操作; 管理 HRegion Server, 比如负载均衡, 调整 HRegion 分布; 负责失效的 HRegionServer 上的 HRegion 迁移;
- HRegionServer: HRegion 的载体, 响应客户端的数据操作请求等;
- HStore: HBase 的存储核心, 包括内存存储单元 MemStore 和文件系统存储单元 StoreFile;
- HLog: HBase 的日志系统.

HRegionServer 是 HBase 系统的核心模块, 它既是 HRegion 的载体, 也是负责数据操作的通信和处理单元. 如图 1 所示, 一个 HRegionServer 包含若干个 HRegion. 同一个 HRegionServer 上的所有 HRegion 共享一个 HLog(在启动一个 HRegion 时 HLog 作为参数传递过去), 即所有 HRegion 的日志信息都会写入到 HRegionServer 所带的 HLog 中. HRegion 中包含多个 Store, 每个 Store 保存一个列簇. 在 Store 中, 包括一个 MemStore 和若干个(包括 0 个)StoreFile. MemStore 是存储在内存中的经过排序的键值对, 是数据在内存中的存储形式. StoreFile 是存储在文件系统中的数据文件, 它是只读文件.

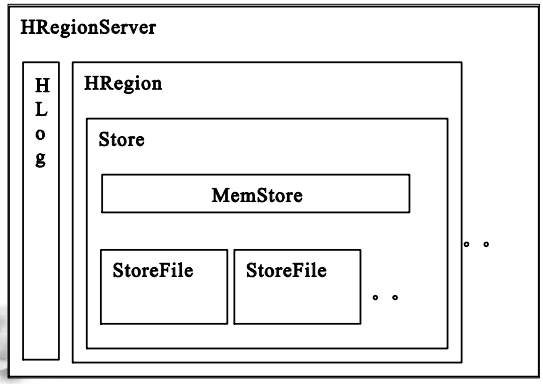


图 1 HRegionServer 结构模型

1.2 HBase 数据持久性和可用性

HBase 提供两种机制保证数据持久性和可用性: 把数据从易失性的内存中持久化到磁盘和日志系统 WAL(Write Ahead Log).

对于 HBase 的写操作, 在其执行过程中可以充分看出这两种手段的应用. HBase 的写操作处理流程如图 2(流程图突出强调的是持久化和写日志的过程), 描述如下:

<1> HRegion 接收到写操作请求后, 首先检查当前 HRegion, 此时会检查是否需要发送数据持久化请求(RequestFlush), 如果需要则先请求 Flush 当前 HRegion 的 MemStore, 否则进行下一步动作;

<2> 把数据写入 MemStore, MemStore 在内存中, 此过程不会有 Flush 请求, 继续进行第三步;

<3> 构建 WAL, 写日志, 随后同步日志, 尽快把日志记录写入文件系统;

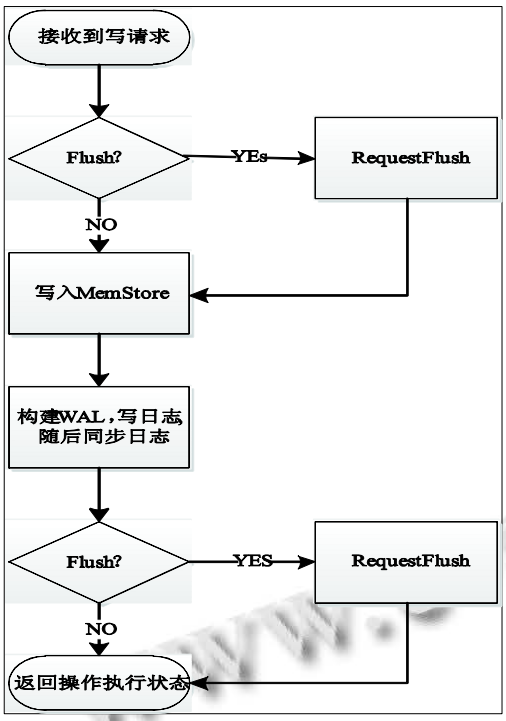
<4> 检查当前 HRegion 是否需要发起 Flush 请求, 如果需要发送 RequestFlush;

<5> 写动作结束, 返回执行状态.

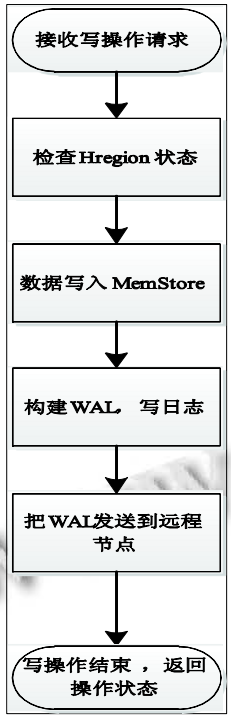
1.2.1 数据持久化

数据持久化是一种基本的保证数据持久性的手段, 它把数据和数据修改保存到非易失性存储介质中. 对于 HBase, 即把数据写入依托的文件系统如 HDFS.

当 HRegionServer 启动时, 它会立即启动一个线程(MemStoreFlusher), 用于检查当前 HRegionServer 上所有 MemStore 的状态, 并负责处理 HRegion 的 RequestFlush



(a) HBase 写操作执行过程



(b) RLP 过程中写操作执行过程

图 2 HBase 写操作的执行流程比较

图 2(a) HBase 对写操作的处理流程, 数据写入前后都可能引发写磁盘的操作;

图 2(b) 加入 RLP 过程后写操作的处理流程, 数据写入过程中没有显式的写磁盘操作.

请求, 把 MemStore 中的数据 Flush 到文件系统. 对于任意 HRegion, 当需要把 MemStore 中数据写入文件系统时, 就发送 Flush 请求, 这个请求会把相关信息加入队列中去, 并在稍后予以处理. 触发 Flush 动作的情形有两个:

① 当某个 HRegion 的 MemStore 中的数据大于设定的阈值时, 发出 Flush 请求.

② 当 HRegionServer 中所有 HRegion 包含的 MemStore 数据总和大于给定的阈值时, 持久化进程会选取若干个合适的 HRegion, 把其数据 Flush 到文件系统, 直至 MemStore 数据总和不超过设定的阈值为止.

1.2.2 日志系统 WAL

WAL(Write Ahead Log)是 HRegionServer 在处理数据操作请求(增、删、改)的过程中记录的一种日志. 在每一次数据操作过程中, HBase 都会封装一个 WAL, 然后以追加的形式写入当前 HRegionServer 的 HLog 中, 并及时的把 HLog 写入文件系统, 保证日志的持久性和可用性. HLog 由 HRegionServer 所承载的所有 HRegion 共享, 它完整的记录用户对数据库的修改, 可以在节点甚至系统宕机的情况下恢复数据, 也能够保证尚在 MemStore 中的数据不会轻易丢失.

2 基于远程日志的数据持久性、可用性实现

2.1 远程日志 RLP

RLP(Remote Log Process)的核心要义是采用集群节点间数据备份的方式保证数据的可用性和持久性[6], 即在写操作过程中, 并不急于把日志持久化到文件系统, 而是把本地 WAL 同时发往到预先指定的远程目标节点(如图 3), 这个过程称之为 RLP.

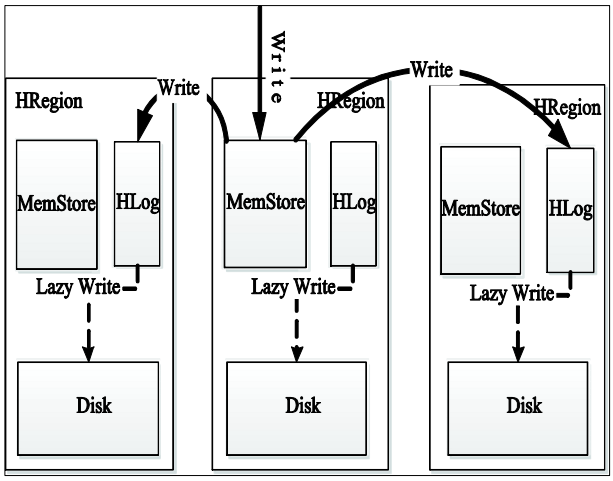


图 3 远程日志备份

在一个 HBase 集群中, 通常会存在一个 HMaster 集群和若干个 HRegionServer 节点. HMaster 负责管理 HRegionServer 节点, 为 HRegionServer 分配备份日志的目标节点. HRegionServer 则是 RLP 的处理单元. 集群中任何一个 HRegionServer 节点既是接受 RLP 的服务端, 也是发起 RLP 过程的客户端. 即当前 HRegionServer 如果存在某个 HRegion 执行写操作进程时, 会把 WAL 写入到预先分配的目标节点; 同时也会接受其他节点转发过来的 WAL. 其具体操作过程如下:

Procedure of Remote Log Process

Roles in System:

HMaster: 维护系统, 管理 HRegionServer 节点;

Zookeeper: 存储系统元数据, 比如各个节点远程日志使用的目标节点 IP;

HRegionServer: 处理来自客户端的请求, 负责数据和日志的实际读写, 为 HRegion 提供服务, 提供 WAL 日志服务.

Begin

① 系统启动时, HMaster 为每个 HRegionServer 分配其备份日志的目标节点, 并把 IP 信息写入 Zookeeper[7];

② HRegionServer 启动, 初始化 RLP 进程: 启动 RLP 服务, 作为集群中某个节点备份日志的目标节点(RLP 服务端); 同时从 Zookeeper 处读取当前节点备份的目标节点 IP, 作为整个 HRegionServer 所有 HRegion 发起 RLP 共享的变量使用;

③ 当 HRegion 调用写操作时, 把用户数据写入内存中的 MemStore, 然后构造 WAL, 写入内存, 等待调度进程写文件系统, 同时启用 RLP 过程把 WAL 发往远程目标节点, 等待目标节点发来的写入确认(RLP 客户端);

④ 当 HRegionServer 接受到集群中某个节点发来的 WAL 时, 将它写入本地日志文件, 返回确认;

⑤ 如果写进程在设定时间内收到超过一半的目标节点返回的写确认就认为写成功, 返回写成功; 否则返回写失败.

End

系统在处理写请求的过程中, 不再频繁的判断当前 HRegion 是否需要把数据写入磁盘, 也就可以大大减少磁盘 IO 的频率.

2.2 流程描述

引入 RLP 进程后, 系统处理写操作的流程如图 2(b)所示, HRegion 写磁盘的频率大大减少, 所有的写文件系统操作都是由其他进程在后续某时刻进行的, 描述如下:

<1> 检查 HRegion, 但不检查是否需要持久化内存 MemStore 中的数据;

<2> 把数据写入 MemStore, MemStore 是内存存储结构, 不会请求写文件系统;

<3> 构建 WAL, 把日志内容写入内存;

<4> 把本地的日志记录发往远程备份节点;

<5> 写动作结束, 返回执行状态.

2.3 持久性、可用性论证

引入 RLP 进程后, HRegionServer 发起数据持久化进程的频率大大减少, 当且仅当 HRegionServer 中所有 MemStore 数据总和超过阈值时会启动持久化进程, 把选中的若干 MemStore 写入磁盘, 能够实现最终的数据持久化.

对于尚未能写入磁盘的数据, 系统通过本地以及远程备份的日志保证其持久性和可用性.

① 如果节点宕机, 会导致暂时存储在 MemStore 中的数据丢失. 此时当节点重启的时候, 就可以读取本地的 Log 文件, 恢复 MemStore 中的数据;

② 如果节点彻底失效, 或者磁盘损坏, 或者本地日志已经难以完全恢复当前节点数据, 可以通过查找并重组远程备份的日志文件, 实现 Region 的迁移或者重建.

3 实验结果和分析

为了论证 RLP 方案的性能, 本文分别针对单机模式和完全分布式模式下的 HBase 数据库进行写操作的时间测试. 在两种的模式下均选取如表 1 所示的测试用例分别对原生 HBase 系统以及改用 RLP 进程的系统进行测试.

表 1 测试用例

MemStore Size	数据写入数量（行）
1M	10W/20W/50W/100W
8M	10W/20W/50W/100W
32M	10W/20W/50W/100W
128M	10W/20W/50W/100W

3.1 单机模式

单机模式下, Zookeeper、HMaster、HRegionServer 都在本地物理机上. Zookeeper 采用 HBase 托管模式. 持久化文件直接保存在 ext3 文件系统中.

3.1.1 写操作时间性能稳定性

从图 4 可以看出, 原生 HBase 的写操作的时间性能对 MemStore 的大小非常敏感, 随着 MemStore 的增大, 写操作的时间消耗明显的下降, 换言之, 当 MemStore 设置的比较小时, 写操作的时间性能会有非常明显的下滑(图 4 的 HBase 部分). 而加入了 RLP 进程的 HBase 在 MemStore 变化的情况下, 虽然在某些时候出现一些变化, 但是系统整体表现比较平稳. 可以认为引入 RLP 进程后 HBase 的写性能对于 MemStore 大小不再敏感.

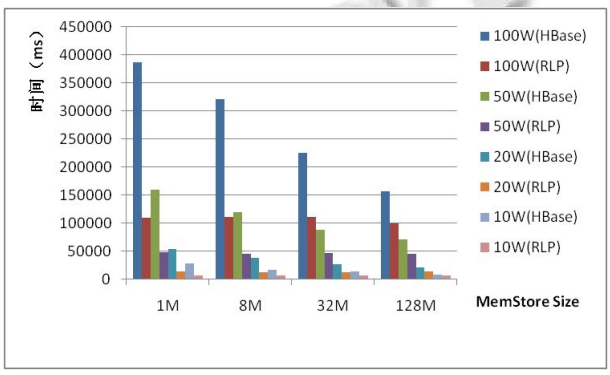


图 4 单机模式下不同 MemStore 时间性能对比

3.1.2 写操作速率

从写操作的时间性能来看, 为 HBase 引入 RLP 过程后, 系统性能有比较明显的提高. 从图 5 可以看出, MemStore 小于等于 128M 时, 任何输入情况下, 引入 RLP 后的系统写操作时间消耗都小于原生的 HBase. 尤其是当 MemStore 的大小设置的比较小的时候, 使用 RLP 进程可以明显提高写入速度. 例如在 MemStore 为 1M 时. 写入 10 万行数据, 原生的 HBase 所需时间是使用 RLP 过程的 4 倍, 写入 20 万行数据为 4 倍, 写入 50 万行和 100 万行分别为 3.31 倍和 3.54 倍.

也需要指出的是, 按照实验揭示的趋势来看, 如果 MemStore 设置的非常大, 在写操作的时间性能上, 引入 RLP 也许不会在对原生 HBase 保持较大优势. 但在 MemStore 为默认设置时, RLP 在写操作的时间性能上仍然有着极大的优势.

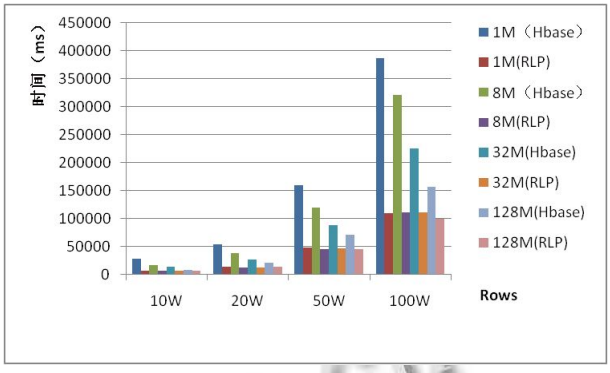


图 5 单机模式不同输入的时间性能对比

3.2 完全分布式模式

完全分布式模式下, 采用的测试环境是一个包括单 HMaster、单 Zookeeper, 5 个 HRegionServer 的集群. 所有节点均处于局域网中, 底层使用 HDFS 作为存储依托.

3.2.1 写操作时间性能稳定性

原生的 HBase 在设置不同的 MemStore 大小时, 写操作的时间性能依然会有比较明显的差别. 尤其是在进行较大数据量的写入时表现的更加突出, 如图 6 所示, 写入 100 万行数据时, 写操作的时间随着 MemStore 的变大明显下降, 其他数据量也可以看见相同的趋势. 引入 RLP 后, 系统对 MemStore 的敏感性下降, 对于不同的 MemStore 值, 写入相同数据量时时间消耗基本相同. 同时无论写入的数据量为多少, 写操作的时间性能基本只和写入的数据量有关系, 并没有随着 MemStore 大小的变化产生较大的起伏. 在 HBase 中加入 RLP 后, 系统写操作的时间性能与 MemStore 大小已经几乎没有关系.

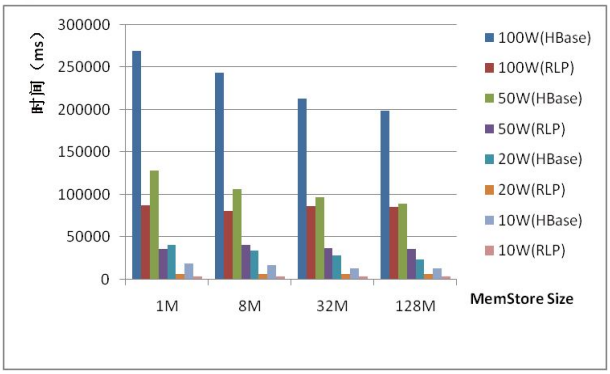


图 6 分布式模式下不同 MemStore 时间性能对比

3.2.2 写操作速率

完全分布式部署时, 引入 RLP 对于写操作时间性

能仍有明显的提升. 结合图 6 和图 7 可以看出, 对于所有的数据写入量, 引入 RLP 的时间曲线明显低于原生 HBase 写操作时间曲线. 实验取得的数据中, 大部分情况下, 原生 HBase 的写操作时间消耗是引入 RLP 后的 2-5 倍. 尤其是在 MemStore 设置比较小的时候, 性能的对比尤其突出, 时间差通常能达到 4 倍以上.

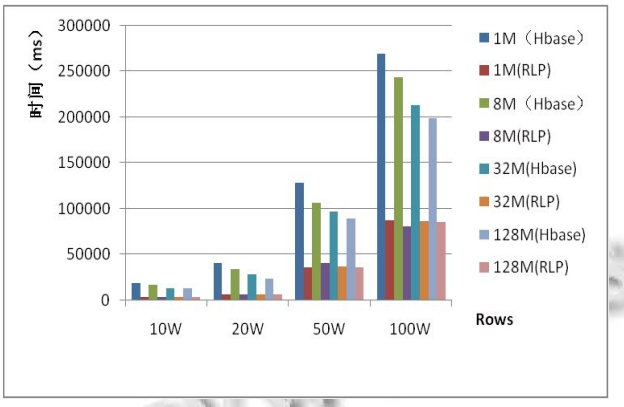


图 7 分布式模式下不同输入的时间性能对比

图 7 从另一方面也可以佐证, 引入 RLP 之后 HBase 写操作的时间消耗在不同的 MemStore 大小时几乎没有明显变化. 使用 RLP 的四条时间曲线几乎重合, 再一次明显的看出, 引入 RLP 的 HBase 对 MemStore 大小不再敏感.

综合单机模式和完全分布式模式的实验结果, 可以看出, 为 HBase 引入 RLP 之后, 系统写操作对 MemStore 大小的敏感度明显降低, 即写操作的时间性能不再依赖 MemStore 的大小设置. 同时, 在 MemStore 不超过默认大小时, RLP 能够明显提高系统的写操作时间性能. 另一方面, RLP 的引入会增加系统运行过程中的网络负载, 也会增加集群整体的日志数量.

4 结语

HBase 依靠把数据写入文件系统保证数据的持久性和可用性, 所以会有大量的写文件系统操作. 本文提出利用集群中节点间数据备份来保证数据的持久性和可用性, 减少本地节点写文件系统操作, 并以此为基础提高 HBase 写操作的时间性能. 通过实验证明:

① 在 MemStore 设置不超过默认设置的场景下, 远程日志方法能够有效提高写操作的时间性能;

② 采用远程日志方法能够大幅度降低 HBase 写操作时间性能对 MemStore 大小的敏感度, 使得系统写操作的执行时间不再和 MemStore 大小息息相关.

参考文献

1 Kubiatowicz J, Bindel D, Chen Y, Czerwinski S, Eaton P, Geels D, Gummadi R, Rhea S, Weatherspoon H, Wells C, Zhao B. OceanStore: an architecture for global-scale persistent storage. SIGARCH Comput. Archit. News 28,5 (Dec.2000), 190–201.

2 HBase. http://hbase.apache.org/.

3 Chang F, Dean J, Ghemawat S, Hsieh WC. Bigtable: A distributed storage system for structured data. ACM Trans. on Computer Systems(TOCS) 2008, 26(2).

4 Tom White,曾大聃,周傲英,周敏译.Hadoop 权威指南.北京:清华大学出版社,2010:366–429.

5 Lars George.HBase:The Definitive Guide(影印版).南京:东南大学出版社,2012:315–384.

6 Ousterhout J, Agrawal P, Erickson D, Kozyrakis C, Leverich J, Mazières D, Mitra S, Narayanan A, Parulkar G, Rosenblum M, Rumble SM, Stratmann E, Stutsman R. The case for RAMClouds: Scalable high-performance storage entirely in DRAM. SIGOPS Operating Systems Review, December 2009, 43(4): 92–105.

7 Dai D, Li X, Wang C, Sun MM, Zhou XH. Sedna: A memory based key-value storage system for real time processing in cloud. The 2012 International Conference on Cluster Computing Workshops(IASDS 2012) in Conjunction with IEEE Cluster'12, September 2012: 24–28.