**Cassandra V.S. HBase**

**背景**

“这是最好的时代，也是最坏的时代。”

每个时代的人都在这么形容自己所处的时代。在一次次IT浪潮下面，有人觉得当下乏味无聊，有人却能锐意进取，找到突破。数据存储这个话题自从有了计算机之后，就一直是一个有趣或者无聊的主题。上世纪七十年代，关系数据库理论的出现，造就了一批又一批传奇，并推动整个世界信息化到了一个新的高度。而进入新千年以来，随着SNS等应用的出现，传统的SQL数据库已经越来越不适应海量数据的处理了。于是，这几年NoSQL数据库的呼声也越来越高。

在NoSQL数据库当中，呼声最高的是HBase和Cassandra两个。虽然严格意义上来说，两者服务的目的有所不同，侧重点也不尽相同，但是作为当前开源NoSQL数据库的佼佼者，两者经常被用来做各种比较。

去年十月，Facebook推出了他的新的Message系统。Facebook宣布他们采用HBase作为后台存储系统。这引起了一片喧哗声。因为Cassandra恰恰是Facebook开发，并且于2008年开源。这让很多人惊呼，是否是Cassandra已经被Facebook放弃了？HBase在这场NoSQL数据库的角力当中取得了决定性的胜利？本文打算主要从技术角度分析，HBase和Cassandra的异同，并非要给出任何结论，只是共享自己研究的一些结果。

**选手简介**

* **HBase**

HBase是一个开源的分布式存储系统。他可以看作是Google的Bigtable的开源实现。如同Google的Bigtable使用Google File System一样，HBase构建于和Google File System类似的Hadoop HDFS之上。

* **Cassandra**

Cassandra可以看作是Amazon Dynamo的开源实现。和Dynamo不同之处在于，Cassandra结合了Google Bigtable的ColumnFamily的数据模型。可以简单地认为，Cassandra是一个P2P的，高可靠性并具有丰富的数据模型的分布式文件系统。

**分布式文件系统的指标**

根据UC Berkeley的教授Eric Brewer于2000年提出猜测- CAP定理，一个分布式计算机系统，不可能同时满足以下三个指标：

* Consistency 所有节点在同一时刻保持同一状态
* Availability 某个节点失败，不会影响系统的正常运行
* Partition tolerance 系统可以因为网络故障等原因被分裂成小的子系统，而不影响系统的运行

Brewer教授推测，任何一个系统，同时只能满足以上两个指标。

在2002年，MIT的Seth Gilbert和Nancy Lynch发表正式论文论证了CAP定理。

而HBase和Cassandra两者都属于分布式计算机系统。但是其设计的侧重点则有所不同。HBase继承于Bigtable的设计，侧重于CA。而Cassandra则继承于Dynamo的设计，侧重于AP。

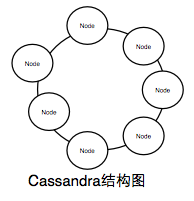
**架构比较**

HBase和Cassandra是基于两篇不同著名论文的开源实现。

* HBase：“Bigtable: A Distributed Storage System for Structured Data”
* Cassandra：“Dynamo: amazon's highly available key-value store”

简而言之，HBase是解决如何在HDFS上面构建分布式的基于表和列的存储系统。而Cassandra则是一个P2P的没有单点故障的存储系统，并且其数据模型是采用类似Bigtable的ColumnFamily。按照Cassandra的介绍，Cassandra是Bigtable和Dynamo的混合体。但是要搞清除的是，混合体是混合Bigtable的数据模型和Dynamo的存储模型。

下面是HBase和Cassandra的架构图：

[](http://photo.blog.sina.com.cn/showpic.html#blogid=633f4ab20100r9nm&url=http://s10.sinaimg.cn/orignal/633f4ab2ta35270a33ef9)[](http://photo.blog.sina.com.cn/showpic.html#blogid=633f4ab20100r9nm&url=http://s7.sinaimg.cn/orignal/633f4ab2ta35270db2356)

上面两个一个是HBase的架构层次图，一个是Cassandra的结构图。其实两者并不对等。但是HBase内部的结构图更加复杂，Hadoop，HDFS和HBase每个单独部分都有若干个角色。而Cassandra与之不同，它不依赖任何一个已有的框架，且每个节点都是对等的。从架构上面来看，Cassandra显然比HBase更加简单。

造成这种设计层次的区别的原因，其实很显而易见。HBase是一个Google Bigtable的复制者。Google Bigtable要解决的问题是如何基于Google File System构建一个分布式数据库存储系统。对应的，HDFS是Google File System，因此HBase的任务事实如何基于HDFS构建一个分布式数据库存储系统。

关于GFS和Bigtable，这里要多啰唆几句。GFS是google构建的，一个非POSIX的分布式文件存储系统。之所以GFS没有严格遵循POSIX的标准，因为GFS构建的目的是如何在造价低廉的服务器上存储超大文件这样一个系统。而POSIX对于小文件的查询等要求则被GFS给忽略了。因此，GFS适合做的事情是在在计算机集群里面有效地存取超大规模的文件。但是真实的世界里面，基于表结构的存储系统更加有用。传统的SQL数据库无法使用GFS因为GFS根本不提供POSIX接口，并且基于B+树的存储结构也让做超大规模索引而引发的树的分裂和重排成为一个巨大的瓶颈。Bigtable就是为了解决这个问题，提供一个基于表和列存储的一个系统，以方便Google为他们的搜索引擎提供存储网页和倒排索引的数据库。

而Cassandra的被模仿者Dynamo在Amazon被使用的，是一个无单点故障的P2P的Key-Value的数据库存储系统。为了让Cassandra更方便使用，Cassandra的开发团队也加入了ColumnFamily, Column, SuperColumn的概念。可以说，Dynamo所使用P2P的概念和相关技术（Bloom Filter，DHT等）不是第一次被软件开发者使用，但是是严肃的大型软件第一次成功应用。Cassandra的设计者正是Facebook从Amazon挖来的Dynamo的设计者和开发者。我们相信Cassandra和Dynamo的设计实现是非常相似的。

**特性比较**

由于HBase和Cassandra的数据模型比较接近，所以这里就不再比较两者之间数据模型的异同了。接下来主要比较双方在数据一致性、多拷贝复制的特性。

* **HBase**
  1. HBase保证写入的一致性。当一份数据被要求复制N份的时候，只有N份数据都被真正复制到N台服务器上之后，客户端才会成功返回。如果在复制过程中出现失败，所有的复制都将失败。连接上任何一台服务器的客户端都无法看到被复制的数据。
  2. HBase提供行锁，但是不提供多行锁和事务。
  3. HBase基于HDFS，因此数据的多份复制功能和可靠性将由HDFS提供。
  4. HBase和MapReduce天然集成。
* **Cassandra**
  1. 写入的时候，有多种模式可以选择。当一份数据模式被要求复制N份的时候，可以立即返回，可以成功复制到一个服务器之后返回，可以等到全部复制到N份服务器之后返回，还可以设定一个复制到quorum份服务器之后返回。Quorum后面会有具体解释。复制不会失败。最终所有节点数据都将被写入。而在未被完全写入的时间间隙，连接到不同服务器的客户端有可能读到不同的数据。
  2. 在集群里面，所有的服务器都是等价的。不存在任何一个单点故障。节点和节点之间通过Gossip协议互相通信。
  3. 写入顺序按照timestamp排序，不提供行锁。
  4. 新版本的Cassandra已经集成了MapReduce了。

从以上的比较，我们可以看到，HBase相对Cassandra来说，模块更多。一个HBase的Instance，必须同时还是HDFS的某一个角色。通常，HBase的Master Server，也同时是HDFS的NameNode。而HBase的RegionServer，同时也是HDFS的DataNode。因为HBase是基于HDFS构建的，模块化显得更加清楚。

而Cassandra每个节点并无特殊之处，不依赖其他任何一个模块，每个启动的节点都是一个java进程。所有的功能都集成在这一个进程里面了。

相对于配置Cassandra，配置HBase是一个艰辛、复杂充满陷阱的工作。Facebook关于为何采取HBase，里面有一句，大意是，Facebook长期以来一直关注HBase的开发并且有一只专门的经验丰富的HBase维护的team来负责HBase的安装和维护。可以想象，Facebook内部关于使用HBase和Cassandra有过激烈的斗争，最终人数更多的HBase team占据了上风。对于大公司来说，养一只相对庞大的类似DBA的team来维护HBase不算什么大的开销，但是对于小公司，这实在不是一个可以负担的起的开销。

另外HBase在高可靠性上有一个很大的缺陷，就是HBase依赖HDFS。HDFS是Google File System的复制品，NameNode是HDFS的单点故障点。而到目前为止，HDFS还没有加入NameNode的自我恢复功能。不过我相信，Facebook在内部一定有恢复NameNode的手段，只是没有开源出来而已。

相反，Cassandra的P2P和去中心化设计，没有可能出现单点故障。从设计上来看，Cassandra比HBase更加可靠。

关于数据一致性，实际上，Cassandra也可以以牺牲响应时间的代价来获得和HBase一样的一致性。而且，通过对Quorum的合适的设置，可以在响应时间和数据一致性得到一个很好的折衷值。

到目前为止的分析，我暂时得出的结论，Cassandra没有被Facebook选择成为Message系统的存储数据库，似乎大部分是公司内部斗争，而非技术选择的结果。

**数据一致性问题**

* HBase

对于HBase来说，最大的卖点在于对于数据一致性的保证。这里不做更多描述了。

* Cassandra

Cassandra，按照Facebook官方博客所说，其数据模型的不一致性导致他们最终没有选择Cassandra。让我们仔细看一下Cassandra对于数据一致性的几种不同配置。

写操作：

|  |  |
| --- | --- |
| **Level** | **Behavior** |
| ANY | 确保数据被写入任何一个节点，其中包括写入HintedHandoff的节点。 |
| ONE | 确保数据被写入至少一个节点的commit log和memory table，再成功返回。这一个节点不包括HintedHandoff节点。 |
| QUORUM | 确保数据被写入N/2+1个节点，再成功返回。 |
| LOCAL\_QUORUM | 确保数据被写入N/2+1个节点，再成功返回,这N/2+1个节点都位于同一个数据中心。这时候需要采用NetworkTopologyStrategy |
| EACH\_QUORUM | 确保数据被写入N/2+1个节点，再成功返回,这N/2+1个节点位分布于每个数据中心。这时候需要采用NetworkTopologyStrategy |
| ALL | 确保数据被写入所有节点之后，再成功返回。任何一个节点不响应，写入操作都将失败。 |

读操作：

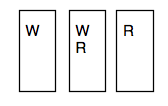
|  |  |
| --- | --- |
| **Level** | **Behavior** |
| ONE | 客户端返回第一个响应的节点。当CL被设置成ONE的时候，客户端会有后台线程做一致性检查。即使这次读到了更老的数据，下次也能得到正确的数据。这称之为ReadRepair。 |
| QUORUM | 查询所有节点，直到满足(N/2+1)个节点都拥有同样的最新的时间戳的时候再返回。同样，剩下的不满足要求的节点还是会由后台线程进行检查。 |
| LOCAL\_QUORUM | 和QUORUM一样，只不过必须满足(N/2+1)个节点必须位于同一数据中心。 |
| EACH\_QUORUM | 和QUORUM一样，只不过必须满足(N/2+1)个节点必须分布于所有数据中心。 |
| ALL | 查询所有节点，当所有节点都同样拥有最新的时间戳的时候再返回。任何一个不响应的节点都将导致此次读操作失败。 |

从上面两个表，我们发现，如果Cassandra要达到HBase的一致性，其实有很多办法。最简单，最粗暴的设置，可以是W(ONE)/R(ALL), W(ALL)/R(ONE)。

不过Cassandra可以有更加高效的办法达到一致性。这就是Cassandra的默认的读写模式W(QUORUM)/R(QUORUM)。

这里我们详细分析一下QUORUM模式。

Quorum顾名思义，就是获得多数投票赞成的一个委员会。在Cassandra当中，满足Quorum的条件是(N/2+1)。如果N是3，则满足Quorum的个数是2。那么在真正情况会是怎么样呢。这就叫Write/Read Overlap。下图显示了N=3，读写操作下面的情况：

[](http://photo.blog.sina.com.cn/showpic.html#blogid=633f4ab20100r9nm&url=http://s16.sinaimg.cn/orignal/633f4ab2ta3527690318f)

从上面的图示我们可以发现，即使是在最坏情况下面，读到了数据还未被复制到的节点，client也会等到该节点数据被复制过来才返回。

到这里，我们的结论就是，Cassandra的确也能提供和HBase一样的数据一致性功能。而且Quorum模式，能让Cassandra以更小的代价获得数据高一致性。

**实际问题的解决**

在真实世界里面，有的实际问题往往不是简单的上述的一致性问题就能解决的。让我们设想一个在真实世界广泛存在的一个问题：

某银行帐户，在T时刻，有余额A元。这时候，两个好心人B1和B2在T时刻都决定向该帐户捐赠一笔钱，分别是M1和M2元。那么最终该帐户余额在数据库当中是多少钱呢？

这是一个小学生都会做的计算题，当然A‘=A+M1+M2。

可是在计算机世界当中这就不一样了。

在T时刻，两个线程（或者两个进程），同时取得了该帐户的余额为A元。双方分别存入M1和M2元的时候，在自己的程序上下文中，该帐户的余额变为了A+M1和A+M2元。这时两个线程同时更新数据库。无论谁最后更新，最终余额都将是错误的。

这看起来是一个很古老的题目不是么？在RMDB时代，你可以用事物，存储过程，锁来进行操作。可是移植到NoSQL数据库，比如HBase或者Cassandra呢？对不起，无论是谁，都不提供和RMDB一样的事务机制。为了达成这个任务，都需要额外的模块。

所幸的是，Hadoop下面的另外一个项目，Zookeeper能够帮助提供全局锁。更进一步的，基于Zookeeper的项目Cage是一个专门为Cassandra提供锁的项目。可以帮助我们构建对重要数据操作时候的各种锁，就像SQL时代一样，功能又回来了。

你可以说，这么做系统变复杂了，我也可以辩解说，这么做事情变得更加模块化了。Cassandra或者HBase只管存储，其他模块来提供全局锁。

**最后要提的一些技术细节和结论**

文章写到这里，实际上我的倾向性很明显。我的确更主张，如果仅仅是存储大量的增长的消息，Cassandra的确更适应这个数据量爆炸的SNS时代。但是我也不敢轻易做结论，毕竟每个应用的场景是不一样的，每个公司的情况也是不同的。在最后结文之前，我还是要记录一些技术细节，供大家参考。

* Pipeline：这个技术是HDFS参考GFS实现所采用的复制数据的办法。顾名思义，当数据从客户端到达第一台Server的时候，第一台Server先将数据放入自己的缓存区。在缓存区满了的时候，再打开和下一台Server之间的通道，将数据发送给下一台Server。直到形成一个管道。数据能够通过Streaming的方式从客户端传送到所有的Server。这么做的好处是复制数据所带来的网络传输压力从客户端分担到了各个服务器。最大限度地利用了网络的吞吐量。不过带来的弊病也很明显，如果管道中任意一台Server速度降低，将直接影响本次拷贝的速度，甚至失败。
* Gossip：这是Cassandra节点互相通报状态的一个协议。通过不停地说悄悄话，Cassandra的节点之间能够清楚地知道谁还活着，谁已经死了。从而能够最快地检测到死去的节点，将节点上的数据分担给其他节点。同时，新节点加入也不需要任何配置，通过Gossip就能自动加入存储集群，分担存储任务。但是有研究表明，当节点过多的时候，Gossip可能会多到拖慢整个网络。

总结，前面虽然我提到，我认为Cassandra作为一个NoSQL用来存储海量消息，相比HBase有很大的优势。其优势主要表现在：

* 配置简单，不需要多模块协同操作。
* 功能灵活性强，数据一致性和性能之间，可以根据应用不同而做不同的设置。
* 可靠性更强，没有单点故障。

尽管如此，Cassandra就没有弱点吗？当然不是，Cassandra有一个致命的弱点。

这就是存储大文件。虽然说，Cassandra的设计初衷就不是存储大文件，但是Amazon的S3实际上就是基于Dynamo构建的，总是会让人想入非非地让Cassandra去存储超大文件。而和Cassandra不同，HBase基于HDFS，HDFS的设计初衷就是存储超大规模文件并且提供最大吞吐量和最可靠的可访问性。因此，从这一点来说，Cassandra由于背后不是一个类似HDFS的超大文件存储的文件系统，对于存储那种巨大的（几百T甚至P）的超大文件目前是无能为力的。而且就算由Client手工去分割，这实际上是非常不明智和消耗Client CPU的工作的。

因此，如果我们要构建一个类似Google的搜索引擎，最少，HDFS是我们所必不可少的。虽然目前HDFS的NameNode还是一个单点故障点，但是相应的Hack可以让NameNode变得更皮实。基于HDFS的HBase相应地，也更适合做搜索引擎的背后倒排索引数据库。事实上，Lucene和HBase的结合，远比Lucene结合Cassandra的项目Lucandra要顺畅和高效的多。（Lucandra要求Cassandra使用OrderPreservingPartitioner,这将可能导致Key的分布不均匀，而无法做负载均衡，产生访问热点机器）。

所以我的结论是，在这个需求多样化的年代，没有赢者通吃的事情。而且我也越来越不相信在工程界存在一劳永逸和一成不变的解决方案。当你仅仅是存储海量增长的消息数据，存储海量增长的图片，小视频的时候，你要求数据不能丢失，你要求人工维护尽可能少，你要求能迅速通过添加机器扩充存储，那么毫无疑问，Cassandra现在是占据上风的。

但是如果你希望构建一个超大规模的搜索引擎，产生超大规模的倒排索引文件（当然是逻辑上的文件，真实文件实际上被切分存储于不同的节点上），那么目前HDFS+HBase是你的首选。

就让这个看起来永远正确的结论结尾吧，上帝的归上帝，凯撒的归凯撒。大家都有自己的地盘，野百合也会有春天的！