Facebook 的实时 Hadoop 系统

Posted: 21 Jul 2011 11:38 AM PDT

Facebook 在今年六月 SIGMOD 2011 上发表了一篇名为“Apache Hadoop Goes Realtime at Facebook”的会议论文 (pdf)，介绍了 Facebook 为了打造一个实时的 HBase 系统使用到的独门秘技。由于该论文提到的应用场景与小弟负责的系统要解决的问题域有相似之处，因而抽时间仔细阅读了这篇论文。下面便是结合论文的内容，谈一谈我的一些看法和感想，如有谬误，敬请指正。

这篇 10 页的长文主要的内容是 Facebook 在 Hadoop 系统上的工程实践，这些工程实践的目标则是题目所点出的——实时。虽然缺乏 Hadoop 系统的开发或使用经验，但是我觉得并没有妨碍我对这篇论文的理解。在我的脑子里，HDFS 就是 GFS，HBase 就是 BigTable。它们实现上可能有差异之处，但主要的思想应该是相通的。如果熟悉 GFS 和 BigTable 那两篇文章，这篇文章就可以视为 GFS 和 BigTable “进阶”。

1. 应用场景和需求

文章的最初是一些背景介绍，主要给出了三类应用场景：Facebook Messaging、Facebook Insight 和 Facebook Metrics System(ODS)。Messaging 就是 Facebook 的新型消息服务，Insight 是提供给开发者和网站主的数据分析工具，ODS 则是 Facebook 内部的软硬件状态统计系统。这三个应用场景都有各自的特色，但简单地来说，面临的问题是同样的：单机或者拆分的关系型数据库无法满足需求。

基于应用场景的数据特征，Facebook 抽象出了几个对存储系统的需求。由于描述起来有些复杂，例如 Efficient and low-latency strong consistency semantics within a data center，这些需求就不一一列举了。相比需求，更让人感兴趣的是它的那些“非需求”，总共有三条：

1. 容忍单数据中心内部的网络分化，Facebook 认为这个问题应该从网络硬件层面（做冗余设计）而不是软件层面去解决；
2. 单个数据中心宕机不影响服务，Facebook 认为这种灾难很难发生，因而愿意接受这种风险；
3. 跨数据中心的数据热备服务能力，Facebook 假设用户数据是分配到固定的数据中心的，可能带来的响应延迟问题应该通过缓存来解决。

从这些“非需求”上可以看出，Facebook 考虑的是更实际的情况，而不是一个理想中的分布式系统，在这点上有一定的借鉴意义。

根据以上的需求和非需求，Facebook 自然而然地给出选择 Apache Hadoop 这套系统的理由，其中有社区的成熟度、Hadoop 在一致性、扩展性、可用性、故障容忍、读写效率等等的各项优点，这些方面的优点也是有目共睹的。

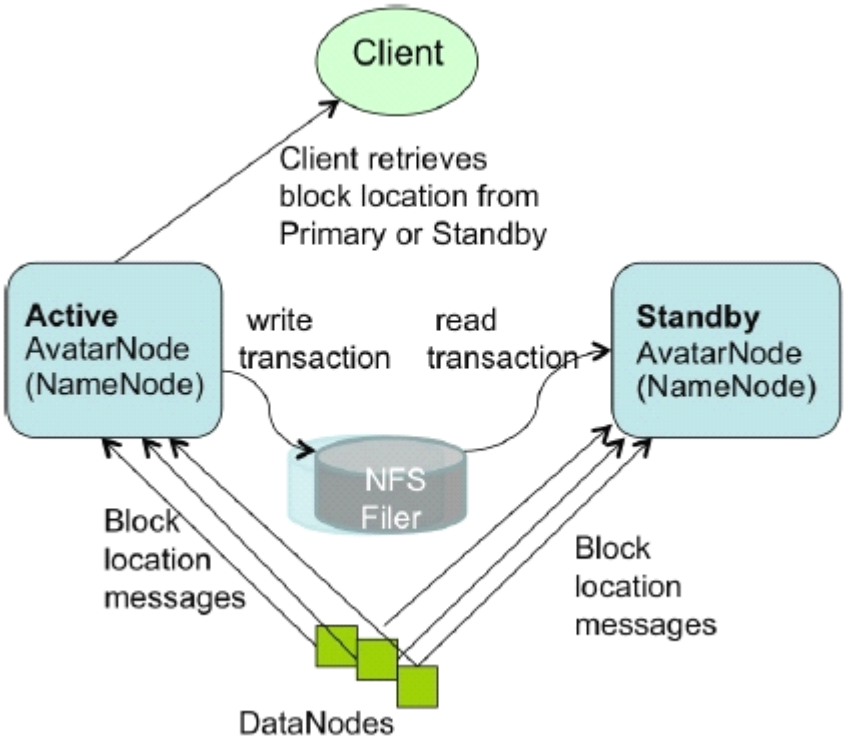
2. 打造实时的 HDFS

HDFS 本身设计来支持离线 MapReduce 计算的分布式文件系统，虽然在扩展性和吞吐上有很好的表现，但在实时性方面表现并不好。如果想让基于 HDFS 的 HBase 有更好的性能，HDFS 层的优化是不可避免

的。为了把 HDFS 打造成一个通用的低时延文件系统，Facebook 主要做了以下一些优化。

2.1 实现 NameNode 的高可用——AvatarNode

HDFS 的 NameNode 是系统单点，就意味着 NameNode 挂掉会导致系统的不可用。NameNode 重启时加载内存快照、应用 log 和收集 DataNode 的数据块信息报告大概需要 45 分钟。即便使用了 BackupNode，仍然需要收集数据块信息报告，切换的时间仍然可能大于 20 分钟。但有实时性需求的系统一般都会要求系统 24x7 的可用性，因而 Facebook 对单点的 NameNode 进行了改进，实现了 NameNode 的双节点热备，称为 AvatarNode，如下图所示：



AvatarNode

简单地来说，备份 AvatarNode 通过 NFS 读取并回放主 AvatarNode 的事务日志来保持数据的同步，并同时接收 DataNode 的数据块信息报告，这保证了主备 AvatarNode 的数据差距尽可能地小，使得备份 AvatarNode 能够很快地切换为主节点的角色。主备 AvatarNode 的角色是注册到 ZooKeeper 中的，DataNode 可以根据 ZooKeeper 中信息判断需要服从哪个 AvatarNode 节点的指令。

为了实现热备 AvatarNode 的数据同步和易用性，Facebook 还改进了 NameNode 事务日志，并部署了 DAFS (Distributed Avatar File System) 屏蔽了 AvatarNode 的故障切换，使得这些改变对客户端透明。文中并没有提到 AvatarNode 的切换是手工还是自动进行的，但是考虑到 ZooKeeper 的 lease 机制，自动切换应该不难实现。

2.2 Hadoop RPC 兼容性和数据块可用性

在之前的系统需求中，有提到一点是 Fault Isolation，并且 Facebook 的 Hadoop 系统是在单机房部署的，因而同一个服务必然会使用多套 Hadoop 系统。为了系统升级独立方便，使客户端兼容不同版本的 Hadoop RPC 是自然而然的事情。

HDFS 在分配副本数据块位置时，虽然会考虑到机架位，但整体来说仍然是相当随机的。其实我以前也曾经与同事讨论过类似的问题，到底是选择随机分配副本位置，还是使用一定的组策略去分配。随机分配的好处是简单均衡，坏处是一旦发生多台宕机，由于副本随机分布，导致某块数据副本全部丢失概率很大；用一定的组策略去分配的好处是多台宕机如果不发生在同一组里，不会丢数据，但是一旦多台宕机发生在同一组，会丢很多数据。看来 Facebook 是选用了组策略分配的方法，认为多台宕机发生在同一组的概率不大。

但这样做是否正确，我是有疑问的。同一个机架或相邻机架上的服务器一般上架时间、硬件型号等都相同，那么同时发生故障的事件不是完全独立的，其概率是要大于理想故障分布情况下概率的。我想这也是为什么 Facebook 最终方案中一组机器是 (2, 5)，2 个机架，5 台服务器。这两个机架的选择，如果很谨慎的话，能够尽量避免我说的这种情况。不过，凡事还得看执行力，如果不了解部署情况去选择机架的话，不一定能够达到预期效果。

2.3 实时负载的性能优化

除了上面的改动之外，Facebook 还对客户端的 RPC 过程进行了优化。为 RPC 添加超时机制，加快文件 lease 的撤销速度（由于对 HDFS 文件操作不了解，我没明白为什么要加快 lease 撤销）。

此外，还提到了最重要的一点：局部性！Facebook 增加了一个检查文件块是否在本机的功能，如果在本机就直接读取。不知道它具体实现方式是怎样的，但我觉得这个做法其实是“很黄很暴力”的，不知道会不会破坏数据一致性。

2.4 HDFS sync 优化和并发读

为了提高写性能，Facebook 允许不等待 sync 结束就继续写，这一点看起来也很暴力，不知道会不会影响数据正确性。

为了能够读到最新数据，Facebook 允许客户端读一个还未写完的数据文件。如果读到正在写入的最后一个块，就重新计算 checksum。

3. 打造实时生产坏境的 HBase

3.1 行级别原子性和一致性

虽然 HBase 已经保证了行级别的原子性，但节点宕机可能导致最后一条更新日志不完整。Facebook 不够满意，引入了 WALEdit，一个日志事务概念来保证每条更新日志的完整性。

一致性方面，看来 HBase 能够满足需求。不过对于 3 个副本同时校验失败导致数据块不可用的情况，Facebook 增加了事后分析的机制，而不是简单丢弃。

3.2 可用性

为了提高 HBase 的可用性，Facebook 对其进行了完善的测试，并解决了以下几个问题：

4 重写 HBase Master，将 ragion 分配信息存储到 ZooKeeper 中以保证宕机切换正确完成。
5 使得 compaction 可以中断以加速 RegionServer 的正常退出速度，并实现 rolling restarts（就是逐台升级），降低程序升级对服务的影响。
6 将宕机 RegionServer 的日志拆分功能从 Master 中拆离，由多个 RegionServer 进行拆分，以提高 RegionServer 故障恢复效率。

这几个问题的解决倒是有通用的用途，我想不久以后很有可能会合并到 Hadoop 的代码中。

3.3 性能优化

性能优化主要从两点进行，一个是 compaction 性能，另一个是读性能。

读过 BigTable 论文的应该对其 memtable 和 compaction 的特性比较熟悉。这里主要讨论了让 minor compaction 也删除数据的好处，以及如何做 major compaction 能够提高合并的性能。

在数据读性能方面，文章里主要讨论了减少 IO 操作的方法，其中包括 bloom filter 和特定类型 meta 信息（时间戳）的使用。还有很重要的一点，在部署上保持 RegionServer 和物理文件的局部性！

文章后面还给出了 Facebook 在部署和运维方面的一些经验，其中有一些有趣的点，我后续可能会写篇文章专门讨论，这里就不详细说明了。

4. 总结

以前我们也曾经讨论过如何在分布式文件系统的基础上搭建一套实时数据分析系统，当时认为如果有成熟的 GFS 可用的话，这个工作会比较简单。现在读到 Facebook 的这篇文章，才发现当初想法的幼稚。仅仅从这篇文章中的技术点体现出的工作量来看，文中说这个系统是多年持续工作的结晶是令人信服的。当然，这也意味着想复制一套这样的系统并不是件轻松容易的事。

从系统设计的成果来看，这个系统应该能达到文章开头制定的需求目标，并也能够满足大部分应用场景的需要。不过有一点，我存在疑问，即是为 Insights 提供的 Realtime Analytics 功能。Realtime 没问题，但使用 HBase， Analytics 究竟能支持多好呢？可能还需要再去了解 HBase 的功能才能有答案。

从这个系统的很多细节可以发现，有不少折中和 trick。我想这就是现实世界，凡事很难做到尽善尽美，工程也一样。在设计系统时追求完美没有错，但是需要考虑代价和可行性，不要忘记满足需求才是最重要的目标。除此之外，也不妨再列出一些“非需求”，排除这些限制可能会降低不少的系统复杂度