



传统的MapReduce框架慢在那里

architecture (6) (/categories.html#architecture-ref)

cloud ⁷ (/tags.html#cloud-ref)

spark ⁸ (/tags.html#spark-ref)

shark ¹ (/tags.html#shark-ref)

Hive ¹ (/tags.html#Hive-ref)

15 April 2013

本文就两个问题进行讨论：1. 相比于Shark (<http://shark.cs.berkeley.edu/>), 为什么像Hive (<http://hive.apache.org/>)之类的传统MapReduce框架比较慢? 2. 对于细粒度的任务模型 (fine-grained task model), 究竟有些什么优势?

background

本文翻译自 **Shark: SQL and Rich Analytics at Scale** (<http://shark.cs.berkeley.edu/>)的论文第七章节, 从理论上讨论了相比于Hive (<http://hive.apache.org/>), **Shark** (<http://shark.cs.berkeley.edu/>)的优势在哪里, 原文可见<http://www.eecs.berkeley.edu/Pubs/TechRpts/2012/EECS-2012-214.pdf> (<http://www.eecs.berkeley.edu/Pubs/TechRpts/2012/EECS-2012-214.pdf>).

为什么之前的MapReduce系统比较慢

常理上有几个理由使得MapReduce框架慢于MPP数据库:

1. 容错所引入的昂贵数据实体化 (data materialization) 开销。
2. 孱弱的数据布局 (data layout), 比如缺少索引。
3. 执行策略的开销[1 2]。

而我们对于Hive的实验也进一步证明了上述的理由, 但是通过对Hive“工程上”的改进, 如改变存储引擎(内存存储引擎)、改善执行架构 (partial DAG execution) 能够缩小此种差距。同时我们也发现一些MapReduce实现的细节会对性能有巨大的影响, 如任务调度的开销, 如果减小调度开销将极大地提高负载的均衡性。

中间结果输出: 类似于Hive这样的基于MapReduce的查询引擎, 往往会将中间结果实体化 (materialize) 到磁盘上:

- 在MapReduce任务内部, 为了防止Reduce任务的失败, Map通常会把结果存储在磁盘上。
- 通常一些查询在翻译到MapReduce任务的时候, 往往会产生多个stage, 而这些串联的stage则又依赖于底层文件系统(如HDFS)来存储每一个stage的输出结果。

对于第一种情况，Map的输出结果存储在磁盘上是为了确保能够有足够的空间来存储这些大数据批量任务的输出。而Map的输出并不会复制到不同的节点上去，因此如果执行Map任务的节点失效的话仍会造成数据丢失[3]。由此可以推出，如果将这部分输出数据缓存在内存中，而不是全部输出到磁盘上面也是合理的。Shark Shuffle的实现正是应用了此推论，将Map的输出结果存储在内存中，极大地提高Shuffle的吞吐量。通常对于聚合 (aggregation) 和过滤之类的查询，它们的输出结果往往远小于输入，这种设计是非常合理的。而SSD的流行，也会极大地提高随机读取的性能，对于大数据量的Shuffle，能够获得较大的吞吐量，同时也拥有比内存更大的空间。

对于第二种情况，一些执行引擎扩展了MapReduce的执行模型，将MapReduce的执行模型泛化成更为通用的执行计划图 (task DAG)，可以将多stage的任务串联执行而无需将stage中间结果输出到HDFS中去，这些引擎包括Dryad[4], Tenzing[5]和Spark[6]。

数据格式和布局 (layout)：由于MapReduce单纯的Schema-on-read的处理方式会引起较大的处理开销，许多系统在MapReduce模型内部设计和使用了更高效的存储结构来加速查询。Hive本身支持“分区表 (table partitions)” (一种基本的类索引系统，它将特定的键段存储在特定的文件中，可以避免对于整个表的扫描)，类似于磁盘数据的列式存储结构[7]。在Shark中我们更进一步地采用了基于内存的列式存储结构，Shark在实现此结构时并没有修改Spark的代码，而是简单地将一组列式元组存储为Spark内的一条记录，而对于列式元组内的结构则有Shark负责解析。

另一个Spark独有的特性是能够控制数据在不同节点上的分区，这为Shark带来了一种新的功能：对表进行联合分区 (co-partition)。

最后，对于RDD我们还未挖掘其随机读取的能力，虽然对于写入操作，RDD只能支持粗粒度的操作，但对于读取操作，RDD可以精确到每一条记录[6]，这使得RDD可以用来作为索引，Tenzing 可以用此来作为join操作的远程查询表 (remote-lookup)。

执行策略：Hive在数据Shuffle之前花费了大量的时间用来排序，同时将MapReduce结果输出到HDFS上面也占用了大量的时间，这些都是由于Hadoop自身基本的，单次迭代的MapReduce模型所限制的。对于Spark这样的更通用的执行引擎，则可减轻上述问题带来的开销。举例来说，Spark支持基于Hash的分布式聚合和更为通用任务执行计划图 (DAG)。

事实上，为了能够真正优化关系型查询的执行，我们发现在基于数据统计的基础上来选择执行计划是非常有必要的。但是由于UDF和复杂分析函数的存在，而Shark又将其视为一等公民 (first-class citizens)，这种统计将变得十分困难。为了能够解决这个问题，我们提出了partial DAG execution (PDE)，这使得Spark能够在基于数据统计的基础上改变后续执行计划图，PDE与其他系统(DryadLINQ)的运行时执行计划图重写的不同在于：它能够收集键值范围内的细粒度统计数据；能够完全重新选择join的执行策略，如broadcast join，而不仅仅是选择Reduce任务的个数。

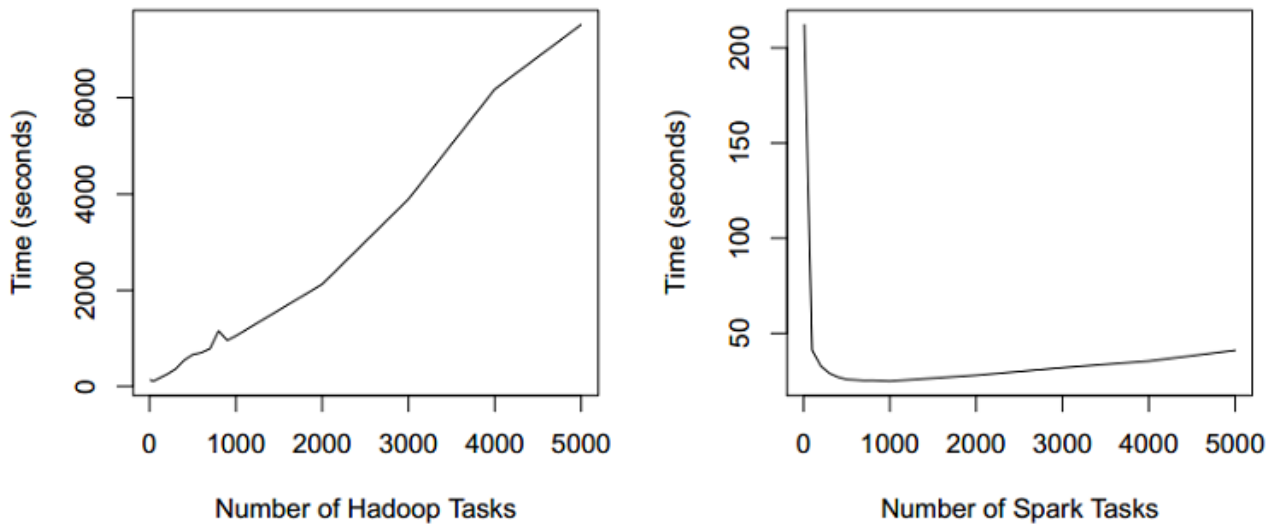
任务调度的开销：大概在诸多影响Shark的部分中，最令人感到意外的却只是一个纯粹工程上的问题：运行任务带来的开销。传统的MapReduce系统，就比如Hadoop，是为了运行长达数小时的批量作业而设计的，而组成作业的每个任务其运行时间则有数分钟之久，他们会在独立的系统进程中执行任务，在某些极端情况下提交一个任务的延迟非常之高。拿Hadoop打比方，它使用周期性的“心跳”消息来向工作节点分配任务，而这个周期是3秒钟，因此总共的任务启动延时就会高达5-10秒。这对于批处理的系统显然是可以忍受的，但是对于实时查询这显然是不够的。

为了避免上述问题，Spark采用了事件驱动的RPC类库来启动任务，通过复用工作进程来避免系统进程开销。它能够在一秒钟内启动上千个任务，任务之间的延时小于5毫秒，从而使得50-100毫秒的任务，500毫秒的作业变得可能。而这种改进对于查询性能的提升，甚至对于较长执行时间的查询性能的提升也令我们感到吃惊不已。

亚秒级的任务使得引擎能够更好地在工作节点之间平衡任务的分配，甚至在某些节点遇到了不可预知的延迟 (网络延迟或是JVM垃圾回收)的情况下也能较好地平衡。同时对于数据倾斜也有巨大的帮助，考虑到在100个核上做哈希聚合 (hash aggregation)，对于每一个任务所处理的键范围需要精心选定，任何的数据倾斜的部分都会拖慢整个作业。但是如果将作业分发到1000个核上面，那么最慢的任务只会比平均任务慢10倍，这

就大大提高了可接受程度。而我们在PDE中应用倾斜感知的选择策略后，令我们感到失望的是相比于增大Reduce任务个数带来的提升，这种策略所带来的提升却比较小。但不可否认的是，引擎对于异常数据倾斜有了更高的稳定性。

在Hadoop/Hive中，错误的选择任务数量往往会比优化好的执行策略慢上10倍，因此大量的工作集中在如何自动的选择Reduce任务的数量[8 9]，下图可以看到Hadoop/Hive和Spark Reduce任务数量对于作业执行时间的影响。因为Spark作业能够以较小的开销运行数千个Reduce任务，数据倾斜的影响可以通过运行较多任务来减小。



事实上，对于更大规模集群(数万个节点)上亚秒级任务的可行性我们还未探究。但是对于Dremel[10]这样的周期性地数千个节点上运行亚秒级作业的系统，实际情况下当单个主节点无法满足任务调度的速度时，调度策略可以将任务委派给子集群的“副”主节点。同时细粒度的任务执行策略相比于粗粒度的设计不仅仅带来了负载均衡的好处，而且还包括快速恢复 (fast recovery) (通过将失败任务分发到更多的节点上去)、查询的弹性 (query elasticity)。

细粒度任务模型 (Fine-Grained Task Model) 带来的其他好处

虽然这篇文章主要关注的是细粒度任务模型带来的容错性优势，这个模型同样也提供了许多诱人的特性，接下来将会介绍在MapReduce系统中已被证明的两个特性。

伸缩性 (Elasticity): 在传统的MPP数据库中，一旦分布式执行计划被选中，系统就必须以此并行度执行整个的查询。但是在细粒度任务系统中，在执行查询的过程中节点可以增删节点，系统会自动地把阻塞的作业分发到其他节点上去，这使得整个系统变得非常具有伸缩性。如果数据库管理者需要在这个系统中移除某些节点，系统可以简单地将这些节点视为失效节点，或者更好的处理方法是将这些节点上的数据复制到其他节点上去。与删除节点相对应的是，当执行查询变得更慢时，数据库系统可以动态地申请更多的资源来提升计算能力。亚马逊的Elastic MapReduce[11]已经支持运行时调整集群规模。

多租户架构 (Multitenancy): 多租户架构如同上面提到伸缩性一样，目的是为了在不同用户之间动态地共享资源。在传统的MPP数据库中，当一个重要的查询提交的时候已经有一个较大的查询占据了大多数的集群资源，这时能做的选择不外乎就是取消先前的查询等有限的操作。而在基于细粒度任务模型的系统中，查询作业可以等待几秒到当前作业完成，然后提交新的查询作业。Facebook和Microsoft已经为Hadoop和Dryad开发了公平调度器，使得大型的、计算密集型的历史记录查询与实时的小型查询可以共享集群资源而不会产生饥饿现象[12 13]。

Reference

- [1] “A. Pavlo et al. A comparison of approaches to large-scale data analysis. In SIGMOD, 2009.” (<http://www.cse.nd.edu/~dthain/courses/cse598z/spring2010/benchmarks-sigmod09.pdf>)
- [2] “M. Stonebraker et al. Mapreduce and parallel dbmss: friends or foes? Commun. ACM.” (<http://cs.brown.edu/~pavlo/presentations/mapreduce-pavlo09.pdf>)
- [3] <http://www.eecs.berkeley.edu/Pubs/TechRpts/2012/EECS-2012-214.pdf> (<http://www.eecs.berkeley.edu/Pubs/TechRpts/2012/EECS-2012-214.pdf>).
- [4] “M. Isard et al. Dryad: distributed data-parallel programs from sequential building blocks. SIGOPS, 2007.” (<https://www.cs.cmu.edu/afs/cs.cmu.edu/Web/People/15712/papers/isard07.pdf>)
- [5] “B. Chattopadhyay, , et al. Tenzing a sql implementation on the mapreduce framework. PVLDB, 4(12):1318–1327, 2011.” (http://static.googleusercontent.com/external_content/untrusted_dlcp/research.google.com/zh-CN/pubs/archive/37200.pdf)
- [6] “M. Zaharia et al. Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing. NSDI, 2012.” (http://www.cs.berkeley.edu/~matei/papers/2011/tr_spark.pdf)
- [7] “A. Thusoo et al. Hive-a petabyte scale data warehouse using hadoop. In ICDE, 2010.” (http://www.facebook.com/note.php?note_id=89508453919)
- [8] “Y. Kwon et al. Skewtune: mitigating skew in mapreduce applications. In SIGMOD ’12, 2012.” (<http://nuage.cs.washington.edu/pubs/sigmod2012-kwon-correct.pdf>)
- [9] “B. Guffler et al. Handling data skew in mapreduce. In CLOSER, 2011.” (<http://www-db.in.tum.de/research/publications/conferences/closer2011-100.pdf>)
- [10] “S. Melnik et al. Dremel: interactive analysis of web-scale datasets. Proc. VLDB Endow., 3:330–339, Sept 2010.” (http://static.googleusercontent.com/external_content/untrusted_dlcp/research.google.com/zh-CN/pubs/archive/36632.pdf)
- [11] <http://aws.amazon.com/about-aws/whats-new/2010/10/20/amazon-elastic-mapreduce-introduces-resizing-running-job-flows> (<http://aws.amazon.com/about-aws/whats-new/2010/10/20/amazon-elastic-mapreduce-introduces-resizing-running-job-flows>).
- [12] “M. Zaharia et al. Delay scheduling: A simple technique for achieving locality and fairness in cluster scheduling. In EuroSys 10, 2010.” (http://www.cs.berkeley.edu/~matei/papers/2010/eurosys_delay_scheduling.pdf)
- [13] “M. Isard et al. Quincy: Fair scheduling for distributed computing clusters. In SOSP ’09, 2009.” (<http://www.sigops.org/sosp/sosp09/papers/isard-sosp09.pdf>)

[← Previous \(/architecture/2013/04/02/spark-streaming-introduction\)](/architecture/2013/04/02/spark-streaming-introduction)

[Archive \(/archive.html\)](/archive.html)

Next →

[\(/architecture/2013/04/21/Spark%E6%BA%90%E7%A0%81%E5%88%86%E6%9E%90%E4%B9%8B-scheduler%E6%A8%A1%E5%9D%97\)](/architecture/2013/04/21/Spark%E6%BA%90%E7%A0%81%E5%88%86%E6%9E%90%E4%B9%8B-scheduler%E6%A8%A1%E5%9D%97)



Start the discussion...

Best ▾ Community

Share ↗ Login ▾


Be the first to comment.

ALSO ON JERRY SHAO'S HOMEPAGE

WHAT'S THIS?

Spark源码分析之-Storage模块

1 comment • 2 months ago

 Wangda Tan — 楼主太强大了，这些正是最近想学的，谢谢楼主的分享！！

Spark源码分析之-scheduler模块

20 comments • 7 months ago

 liangliang — 每个Action都会生成一个job

Spark源码分析之-deploy模块

3 comments • 8 months ago

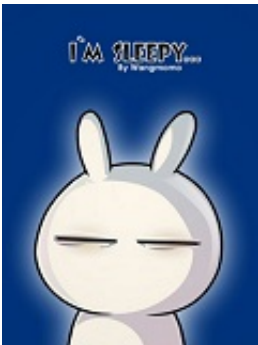
 Huangdong Meng — 明白～ 期待大神的更多大作哈～ 比如shark～ RDD的部分复杂的operator的实现 等data processing层的讲解

Spark Overview

3 comments • 7 months ago

 vincent_hv —

 Subscribe  Add Disqus to your site



CATEGORIES

- lessons (1) (/categories.html#lessons-ref)
- test (1) (/categories.html#test-ref)
- architecture (6) (/categories.html#architecture-ref)
- functional programming (1) (/categories.html#functional programming-ref)
- arhitecture (1) (/categories.html#arhitecture-ref)

LINKS

阮一峰的网络日志 (<http://www.ruanyifeng.com/blog/>)

刘未鹏 (<http://mindhacks.cn/>)

酷壳 (<http://coolshell.cn/>)

BeiYuu.com (<http://beiyuu.com/>)

MY FAVORITES

© 2013 Jerry Shao with help from Jekyll Bootstrap (<http://jekyllbootstrap.com>) and Twitter Bootstrap (<http://twitter.github.com/bootstrap/>)