

比较Apache Hadoop 生态系统中不同的文件格式和存储引擎的性能

原创 2017-06-02 许江 whoami

主题

这篇文章提出了在Apache Hadoop 生态系统中对比一些当前流行的数据格式和可用的存储引擎的性能：Apache Avro, Apache Parquet, Apache HBase 和 Apache Kudu 空间效率，提取性能，分析扫描以及随机数据查找等领域。这有助于理解它们中的每一个如何(何时)改善你的大数据工作负载的处理能力。

引言

最初把hadoop文件格式和存储引擎做比较的想法是在初始系统修订版之一的驱动下完成的—这个系统是在CERN中大规模调节Hadoop—ATLAS EventIndex.

项目启动始于2012年。那时候用MapReduce处理CSV是最常见的处理大数据的方式。同期平台，像Apache Spark, Apache Impala (孵化), 或者文件格式像Avro 和Parquet 还没有成熟，也不像现在这么流行，甚至还没有出现。然而在现在看来选择基于HDFS MapFiles的设计概念就是陈旧和过时。

我们采用ATLAS EventIndex数据测试的最终目标是为了了解哪种存储数据方法是最佳应用方法以及这种应用在系统的主要使用案例方面预期的效益是什么。我们要比较的主要方面是数据的容量和性能。

- 数据提取。
- 少量数据查询。
- 全部数据扫描。

关于EVENTINDEX数据

ATLAS 是CERN中的粒子加速器，是构建 Large Hadron Collider 探测实验的七个粒子之一。

ATLAS EventIndex是所有碰撞（称作事件）中的一个元数据目录，在 ATLAS 实验中发现，后来被永久存储在CERN基础架构中（通常每秒钟会发生数以百计的事件）物理学家通过共同点和检查生产周期的一致性用这个系统区分和定位有用的事件和人口群组事件。

每个索引碰撞都是独立记录存储在ATLAS EventIndex，这种独立记录平均1.5KB长具有56种属性，其中有6个较为独特的标记为一个碰撞，大多数记录属性是文本类型，只有少部分是数字。在

给定时刻，HDFS可以存储6e10条记录占用上万兆兆字节内存（不包含复制数据）。

Hadoop的存储测试方法

将相同的数据集使用不同的存储技术和压缩算法存储在相同的Hadoop集群里（Snappy, GZip or BZip2）：

- Apache Avro是序列化数据，是小型的二进制格式的标准，广泛用于在HDFS中存储长久数据以及通讯协议。使用Avro特点之一是重量轻以及快速将数据序列化和反序列化，这样提取性能就会非常好。此外，即使它没有任何内部索引（例如在MapFiles情况下）当需要访问随机数据时，HDFS目录式分区技术可用于快速导航找到利益集合。

在测试中，把主键的前三列元组用作一个分区键。这就使得分区的数目（几千）和分区的平均大小（成百上千兆）保持了良好的平衡。

- Apache Parquet 是列导向序列化数据，是有效的数据分析的标准。其他优化包括编码 (RLE, Dictionary, 一些安装) 和压缩应用在同列的同系列值就能得到一个非常高的压缩比率。当在HDFS上用 Parquet 格式存储数据时，可使用类似Avro 案例中同样的分区策略。
- Apache HBase为了存储键值对在HDFS上可扩展的分布NoSQL数据库，键值作索引通常能非常快速的访问到记录。

当存储ATLAS EventIndex数据到HBase时每个事件的属性都存储在独立的单元格中并且行键值被组成串联事件标记为列属性。此外，不同的行键值 (FAST_DIFF)编码可以减少HBase块的大小（如果没有这一步每行有8KB长度）

- Apache Kudu是新开发可伸缩，分布式，基于表的存储方式。Kudu提供的索引和列式数据结构调和了提取速度和分析性能。像HBase案例中，Kudu APIs支持修改已经存储在系统中的数据。

在评估中，所有文本类型是字典编码式存储，数字类型是随机编码存储。此外，通过使用主键首列引入范围组合和hash分区（组成像HBase案例中组合同样的列）作为分区键。

结果分析

数据访问和提取测试都是在由14个物理机器组成的集群上完成，每个物理机器配备：

- 2 x 8 cores @2.60GHz
- 64GB of RAM
- 2 x 24 SAS drives

Hadoop集群安装的是Cloudera Data Hub(CDH) 分布版本 5.7.0，包括：

- Hadoop core 2.6.0

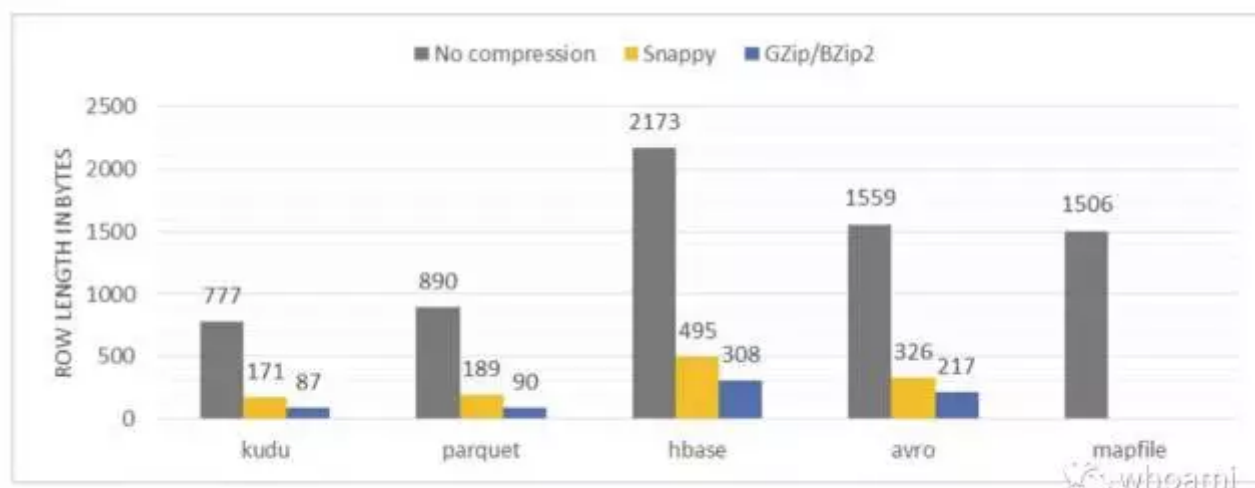
- Impala 2.5.0
- Hive 1.1.0
- HBase 1.2.0 (配置 JVM 堆 , 区域服务器大小 = 30GB)
- (不是 CDH) Kudu 1.0 (配置存储限制 = 30GB)

Apache Impala (孵化) 在所有测试中作为数据提取和数据访问框架最后呈现在这份报告中。

重点：尽管所有的努力是为了得到尽可能精确的测试结果，但是也不要把他们当做是普遍和基本的测试技术基准。有太多的影响测试的变量，所以要视情况而定。例如：

- 选择测试案例
- 使用数据模型
- 硬件规格和配置
- 用于数据处理和配置/调整的软件栈

空间利用率格式



The figure reports on the average row length in bytes for each tested format and compression type

图表显示是测试格式和压缩类型字节行的平均长度

测试描述：在使用不同的技术和压缩办法存储同样的数据集（数百万的记录）之后测量平均记录大小

评价：

- 根据测量结果，用 Kudu和Parquet 编码数据得到最高的压缩比率。用像Snappy 或者GZip 等压缩算法可以进一步显著的减少容量—通过factor10 比较用MapFiles编码的原始数据集
- 因为HBase的储存数据方法是一种空间用量更少高效的解决方案。尽管HBase块压缩得到非常不错不错的比率，然而，还是远远不如Kudu 和 Parquet。
- Apache Avro得到的空间占用方面的结果类似HDFS的行存储—MapFiles

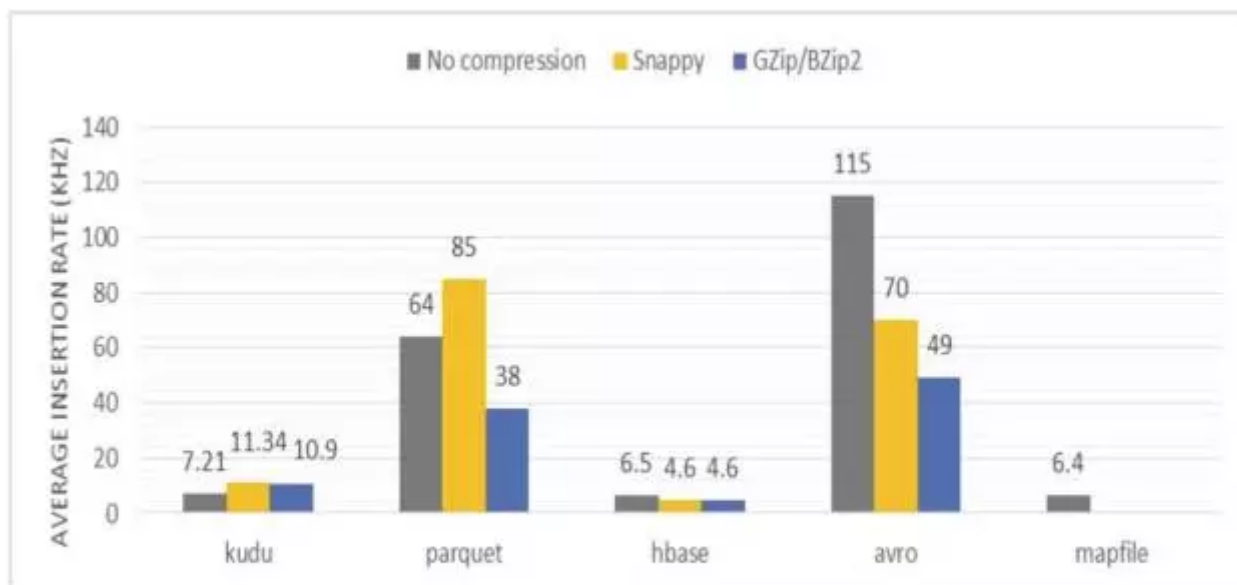


Figure reports on the average ingestion speed (10^3 records/s) per data partition for each tested format and compression type

测试描述：记录测量单个数据分区的提取速度

评价：

- 为了写入一系列的单个HDFS目录 (Hive 分区)Apache Impala执行了数据重组，得到的结果是HDFS 格式、 HBase 、 Kudu可以直接对比单个数据分区的提取效率。用Avro或者Parquet格式写HDFS文件编码 比用HBase 和 Kudu存储引擎得到的结果更好（至少是5倍）。
- 用Avro或者Parquet编码写HDFS文件比用HBase 和 Kudu存储引擎得到的结果更好（至少是5倍）因为Avro用的是重量最轻的编码器，达到了最佳提取性能。
- 在光谱的另一端，HBase在这个测试中很慢(比Kudu更慢)。导致这种情况最大的可能是行键值的长度(6个连接的列)，平均约为60字节。HBase不得不在单独的行中为每个列编码一个键，（有很多列）这对于长期记录并不是最好的选择。

每种格式查询随机数据的延迟：

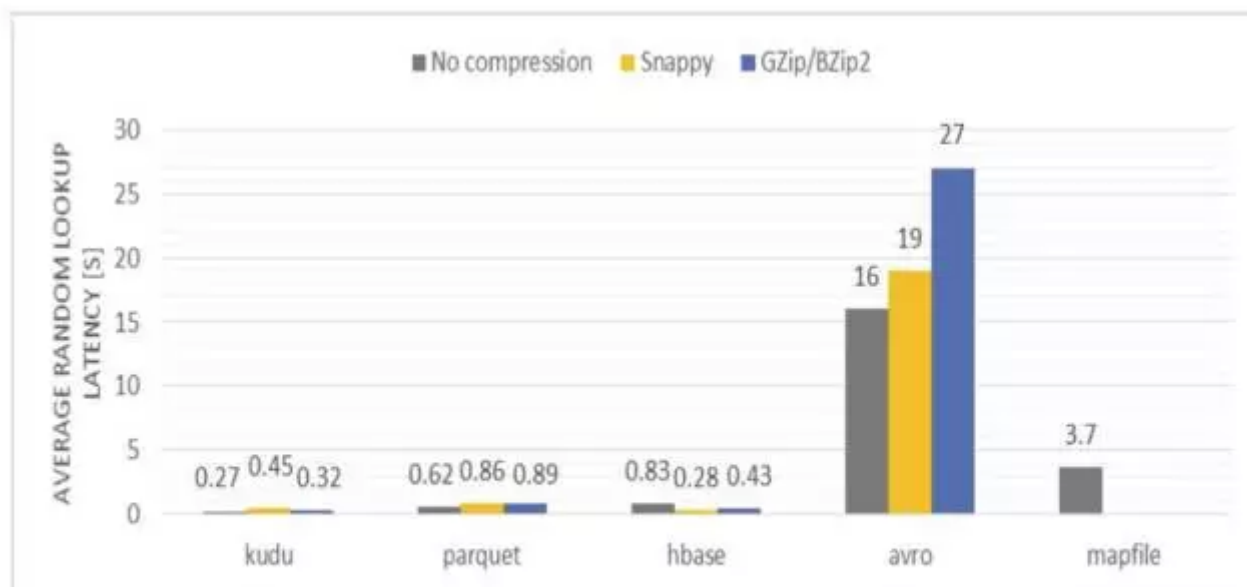


Figure reports on the average random record lookup latency [in seconds] for each tested format and compression type

表格显示每种测试格式和压缩类型平均的随机查询延迟记录【以秒计算】

测试描述：通过提供的一个记录标识符号从记录中检索一个非键属性（一个混合的键）

评价：

- 当通过记录键访问数据时，Kudu 和 HBase是最快的，因为他们使用的是内置索引。布局上的值是用冷缓存测量的。
- Apache Impala的随机查询结果仅次于Kudu和 HBase，一个显著原因是在真正执行查询之前，设置查询（计划，代码生成等）用了大量的时间——通常约为200Ms。因此对于低延迟数据访问建议跳过Impala使用专用的APIs（我们曾尝试将这种方法用于Kudu 和 HBase，结果差不多——用冷缓存小于200ms,用热缓存小于80ms）。
- 和Kudu HBase相反，从单独的记录中检索数据存储为Avro格式只能在整个数据分区暴力的扫描才能完成（提示数据是由记录键的一部分分区的，因此精简分区仅用于这种情况下）通常分区的大小为GB，因此要获取想要的记录需要花几秒钟（取决于IO 吞吐量）使用了大量重要的集群资源。而且必须在集群上全速执行最终才能降低并行查询的数量，。
- 同样的问题用于Parquet，然而，列式格式的本来的属性就允许执行分区扫描相对较快。由于投影列和列断言下推，一组扫描输入最终从GBs减少到少量MBs(实际上只有3列扫描56)。

每种格式的数据扫描率：

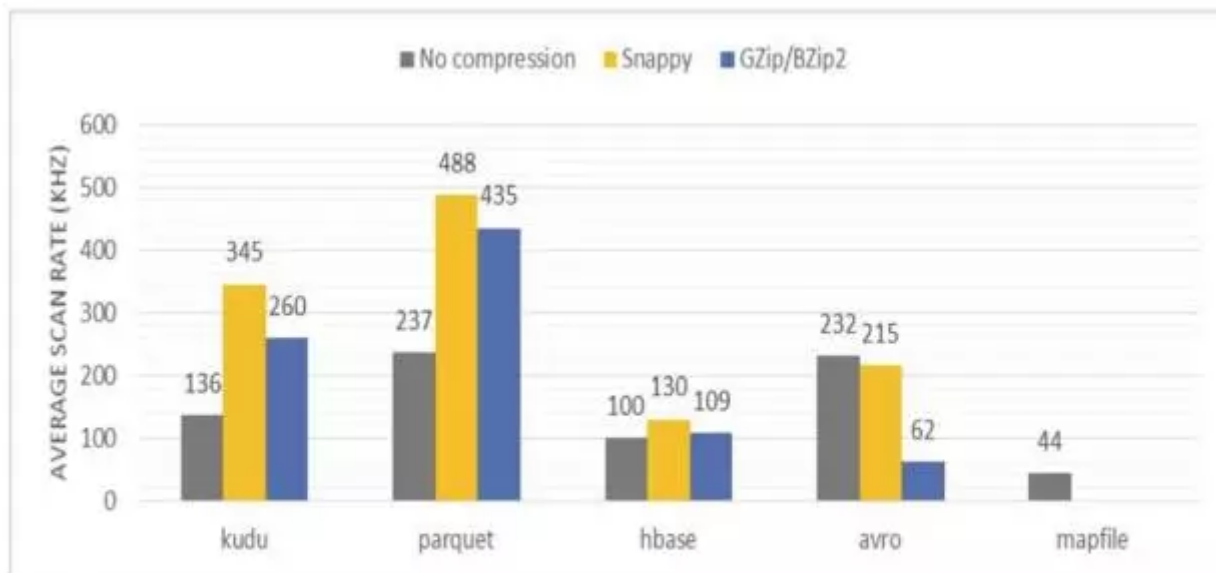



Figure reports on the average scans speed with the same predicate per core [in k records/s] for each tested format and compression type 

图表显示了每种测试格式和压缩类型在每个核中同样的断言的平均扫描速度

测试描述：在整个记录集合中计算在非键值列之一中的固定子字符串的记录数目。

评价：

- 由于采用投影列减少输入集，在此次测试中Parquet落在 Avro后面。它不仅是单核处理率方面最有效率而且最快结束处理。
- 平均扫描速度 (KHZ)
- 在Parquet 和 Avro是HDFS文件块的情况下数据单元可以并行化访问，由于所有的可用资源在一个Hadoop集群上所以要均匀分布处理非常简单。
- 得益于列投影，在扫描效率方面Kudu (具有Snappy压缩) 和Parquet差不多。
- 用Kudu和 HBase存储扫描数据可能会平衡因为并行单元在两种情况下都是分区表。因此参与扫描的资源量取决于给定分区表的数量，以及对其在集群中的分布。
- 在这个测试案例中，使用Kudu本地断言下推功能是不可能的，因为Kudu不支持使用断言。在其他测试中证实当Kudu支持使用断言时它的扫描速度就比 Parquet更快。
- 在用HBase执行测试之前扫描列被分离在一个专门的HBase列家族中—通过factor5提高了扫描效率。但是还是远远不如Parquet或者Kudu。

从测试中习得的经验：

在这段我们想分享关于使用的数据格式其他的想法，优缺点，脱离测试和我们的工作负载参考：

- **存储效率**— 对比未压缩的简单的序列化格式 用Parquet 或者 Kudu 和Snappy压缩可以通过 factor10减少全部的数据容量
- **数据提取速度**— 基于解决方案的所有的测试文件提供的提取率（在2倍和10倍之间）比专门存储在引擎或者 MapFiles(按顺序存储) 中快。

- **随机访问数据时间**— 使用HBase或者Kudu，一般随机查询数据的速度低于500ms。使用智能HDFS命名Parquet分区空间可以提供一个第二水平的随机查询但是会消耗更多的资源。
- **数据分析**— 使用Parquet 或者Kudu 可以执行快速、可扩展的(一般每个CPU核心每秒超过300k以上记录)的数据聚合、过滤和报告。
- **支持数据原地突变**— HBase和Kudu可以原地修改记录，如果将数据直接存储在HDFS文件中是不可能修改的。

值得注意的是，压缩算法扮演了一个非常重要的角色不仅是减少了数据容量还提高了数据提取和数据访问的性能。比起未压缩的普通编码，编解码器在所有的领域为所有的测试技术提供了最好的研究结果(除了 Avro 案例)。

总结：

在Hadoop生态系统中流行的存储技术，从很多方面证明了他们中每一个的优点和缺点，像减少整体数据容量，简化了数据提取，提高了数据访问的性能等。

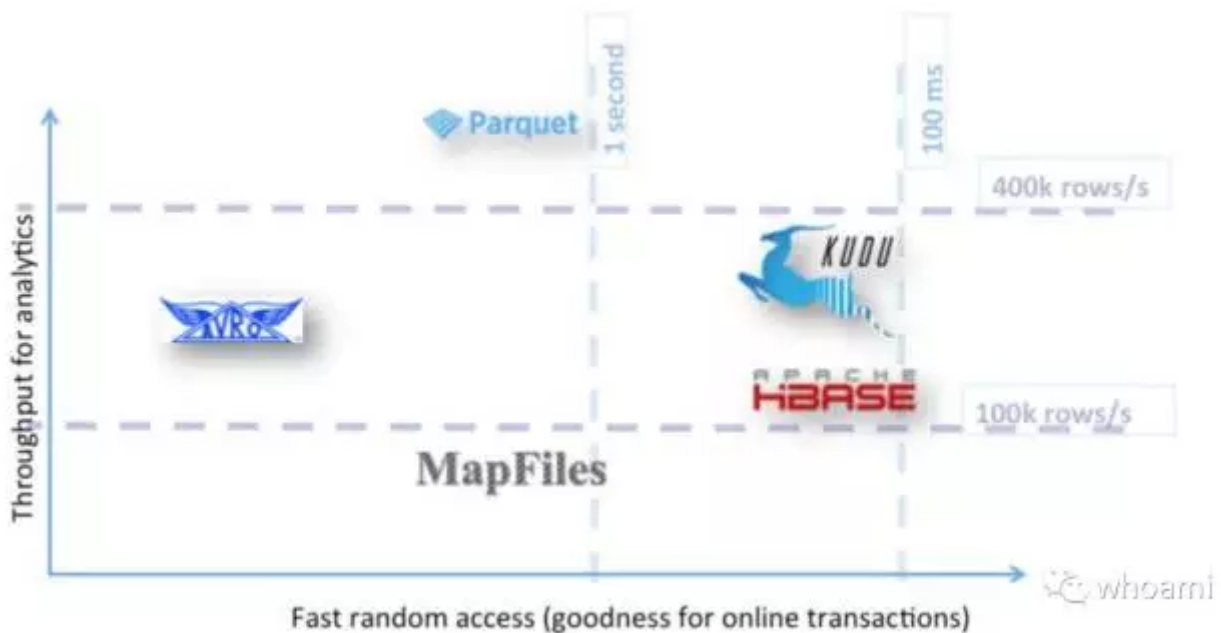
Apache Avro 已被证明是结构化数据快速通用的编码器。由于非常有效的序列化和反序列化，此格式可以确保良好的性能，同时支持随时访问记录的所有属性、数据传输、分级区等。

另一方面，Apache HBase提供了良好的随机的数据访问性能以及最大的灵活性在呈现数据存储时（无模式表）HBase数据的批量处理的性能很大程度上取决于数据模型的选择，并且其他测试技术无法在这一领域竞争。因此用Hbase的数据能执行任何分析技术都很罕见。

根据测试中的列式存储像Apache Parquet 和Apache Kudu在快速提取数据，快速随机查询数据，可扩展的数据分析，同时确保系统的简化—仅用一种数据存储技术，表现了非常好的灵活性。

Kudu擅长快速随机查询而Parquet擅长快速扫描及提取数据。

不同于单一存储技术的实施，混合系统被认为是由批量处理的原始存储技术以及随机访问的索引层技术组成的。这完全得益于特定访问路径提供的最佳性能对技术专业化/优化。值得注意的是，这种方法是数据重复，整体系统的复杂性和更贵的维护成本为代价。因此如果简化的系统是非常重要的因素那么Apache Kudu显然是一个不错的选择。



快速随机访问（在线交易的优势）

译文原文：<https://blog.cloudera.com/blog/2017/02/performance-comparing-of-different-file-formats-and-storage-engines-in-hadoop-file-system/>

欢迎关注微信公众号，第一时间，阅读更多有关云计算、大数据文章。



长按指纹
识别图中二维码



whoami

原创文章，转载请注明：转载自Itweet的博客

本博客的文章集合：<http://www.itweet.cn/blog/archive/>

阅读原文