实用 | 不同文件格式和存储引擎在Apache Hadoop生态系统中的性能比较

2017-04-01 Cloudera中国

作者：Zbigniew Baranowski（Cern）
ZBigniew Baranowski是一位数据库系统专家，并且是提供和支持中央数据库和基于Hadoop服务的CERN（欧洲核子研究组织）的成员。此博客最初发表在CERN的“CERN数据库”博客上，其转发已获得CERN的许可。
网址：http://blog.cloudera.com/blog/2017/02/performance-comparing-of-different-file-formats-and-storage-engines-in-hadoop-file-system/

主题

本篇博文对Apache Hadoop生态系统中可用的几种流行数据格式和存储引擎（包括Apache Avro、Apache Parquet、Apache HBase和Apache Kudu）进行了性能比较，涉及空间效率、数据摄取性能、分析扫描和随机数据查询等。这些内容将有助于用户理解如何（以及何时）可以改善大数据工作负载的处理。

简介

比较Hadoop文件格式和存储引擎的最初想法是受第一个在CERN（ATNAS EventIndex）上大规模采用Hadoop系统版本启发的。

该项目于2012年开始启动，当时利用MapReduce处理CSV是处理大数据的常见方式。同时，Apache Spark、Apache Impala（正在孵化中）之类的平台或Avro、Parquet等文件格式不像现在这么成熟和流行，甚至都尚未启动。因此回顾过去，基于使用HDFS MapFiles选择的设计是一种“过时的”且较不受欢迎的概念。

使用ATLAS EventIndex数据进行测试的最终目标是了解可以最优的使用哪种存储数据方法；以及相对于系统的主要用例，此类应用程序的预期收益是什么。我们想要进行比较的主要方面是数据量和以下性能。

- 数据摄取；
- 随机数据查询；
- 全数据扫描。

EVENTINDEX数据概述

ATLAS是针对大型强子对撞机（CERN的粒子加速器）建造的七大粒子检测器实验之一。

ATLAS EventIndex是所有碰撞（称为“事件”）的元数据目录，这些碰撞在ATLAS实验中发生，后被永久存储在CERN存储基础设施中（通常每秒有几百个事件）。物理学家使用该系统来识别和定位感兴趣的事件，通过共性把事件群体进行分组，以及检查产生周期的一致性。

每个编入索引的碰撞均作为单独的记录存储在ATLAS EventIndex中，其平均长度为1.5KB，具有56个属性，其中6个属性唯一地标识了一个碰撞。大多数属性是文本类型，只有少数属性是数字类型。在某一给定时刻，包含占用几十T字节（不包括数据复制）的6e10个记录存储在HDFS中。

HADOOP上已经过检验的存储方法

已使用不同的存储技术和压缩算法（包括Snappy、GZip或BZip2）将相同的数据集存储在同一Hadoop集群中：

- Apache Avro是一种用于压缩二进制格式的数据序列化标准，其广泛应用于存储HDFS上的持久性数据以及通信协议。Avro的优点之一是轻量级及快速数据序列化和反序列化，这可以提供非常优异的数据撷取性能。此外，当需要实现快速随机数据访问时，即使没有任何内部索引（如在MapFiles的情况下），也可以应用HDFS基于目录的分区技术快速导航到感兴趣的集合。

在测试中，主键前3列的元组被用作分区键，允许在分区数（几千个）和平均分区大小（数百兆字节）之间获得良好的平衡

- Apache Parquet是一种用于高效数据分析的面向列数据的序列化标准。其优化包括应用于来自相同列的一系列值的编码（包括RLE、字典、位封包）和提供非常好的压缩率。以Parquet格式在HDFS上存储数据时，使用与Avro相同的分区策略。

- Apache HBase - HDFS上用于存储键值对的可扩展和分布式的NoSQL数据库。对键进行索引，通常可以提供对记录的快速访问。

当将ATLAS EventIndex数据存储到HBase中时，每个事件属性存储在单独的单元格中，并且行键由事件标识属性列的级联组成。另外，为减小HBase块的大小（否则每行长度会有8KB）启用了行键（DATA_BLOCK_ENCODING）的差分（FAST_DIFF）编码。

- Apache Kudu是一种基于表的可扩展和分布式的新存储方式。Kudu提供了索引和列数据组织，在获取速度和分析性能之间实现了良好的折衷。与HBase的情况一样，Kudu API允许修改已经存储在系统中的数据。

在评估中，所有文字类型都以字典编码存储，数字类型则以位随机编码存储。此外，通过使用主键的第一列（由与HBase案例中相同的列组成）作为分区键，引入了范围和散列分区的组合。

得出的结果

数据访问和撷取测试在由14台实体机器组成的集群上进行，每台机器配备有：

- 2 块8核@ 2.60GHz；
- 64GB内存；

- 2块24 SAS驱动器。

从Cloudera Data Hub（CDH）发行版本5.7.0安装的Hadoop集群包括以下几个方面：

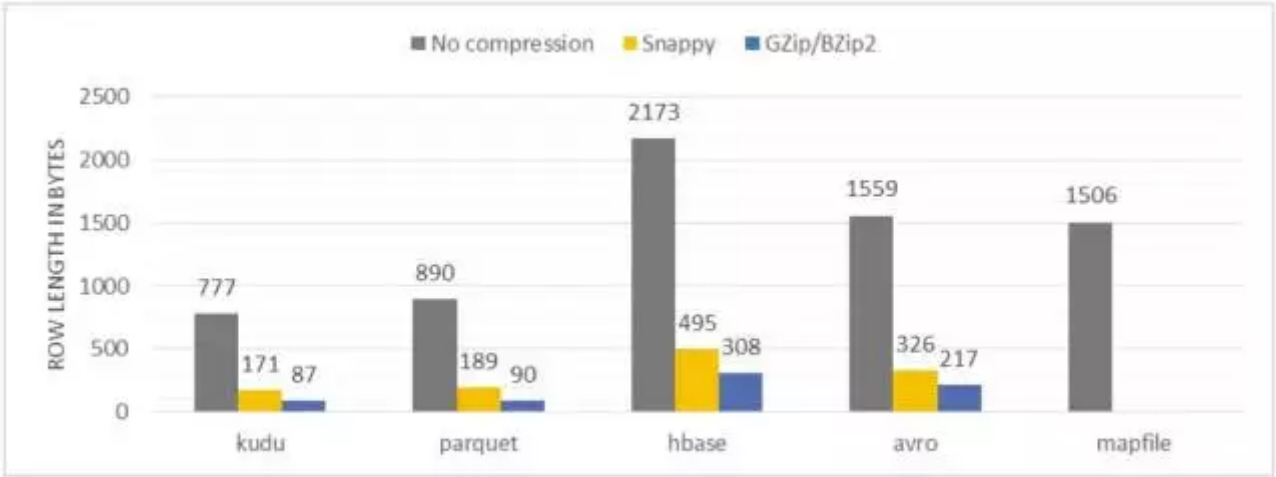
- Hadoop内核2.6.0；
- Impala 2.5.0；
- Hive 1.1.0；
- HBase 1.2.0（为区域服务器配置的JVM堆大小= 30GB）；
- （不是来自CDH）Kudu 1.0（配置内存限制 = 30GB）。

在本报告后面提到的所有测试中，使用Apache Impala（正在孵化中）作为数据摄取和数据访问框架。

重要提示：尽管本次测试为获得尽可能精确的结果付出了一些努力，但这不应被视为测试技术的通用和基本基准。因为存在太多可能影响测试的变量，所以具体情况应该具体分析，例如：

- 选择的测试用例；
- 使用的数据模型；
- 硬件规格和配置；
- 用于数据处理及其配置/调优的软件堆栈。

每种格式的空间利用



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

ROW LENGTH INBYTES
行长度字节
No compression
无压缩
Snappy
GZip/BZip2

The figure reports on the average row length in bytes for each tested format and compression type
该图显示了各种测试格式和压缩类型的平均行长度（以字节为单位）

测试描述：在使用不同技术和压缩方法存储相同的数据集（百万条记录）后，测量记录的平均大小。

注释：

- 根据测量结果，利用Kudu和Parquet编码的数据提供了最佳的压缩率。与使用MapFiles的原始数据集编码相比，使用类似Snappy或GZip之类的压缩算法可以进一步显着减少数据量达10倍。
- 由于HBase存储数据的方式是一个空间效率较低的解决方案，虽然HBase块的压缩给出相当好的比率，但是与Kudu和Parquet相比差距仍然较大。
- 就像其他HDFS行存储方式（例如MapFiles）一样，Apache Avro在空间占用方面提供了类似的效果。

各种格式的摄取速度

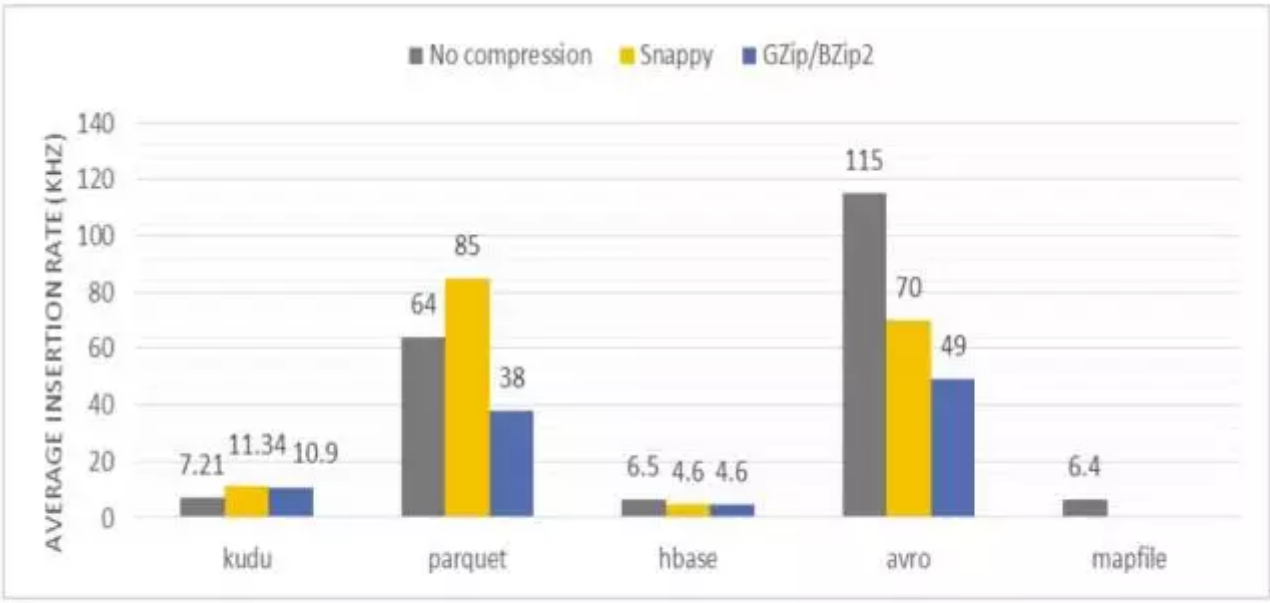


Figure reports on the average ingestion speed (10^3 records/s) per data partition for each tested format and compression type

AVERGE INSERTION RATE(KHZ)

平均插入速率（KHZ）

Figure reports on the average ingestion speed (103 record/s) per data partition for each tested format and compression type

该图显示了各种测试格式和压缩类型的每个数据分区的平均摄取速度（103个记录/秒）

测试描述：测量单个数据分区中的记录摄取速度。

注释：

- 由于Apache Impala执行数据重构以串行写入单个HDFS目录（Hive分区），因此对于HDFS格式和HBase或Kudu的格式，可以直接比较单个数据分区摄取效率。使用Avro或Parquet编码写入的HDFS文件比存储引擎（如HBase和Kudu）提供了更好的结果（至少5倍）。
- 使用Avro或Parquet编码写入的HDFS文件比存储引擎（例如HBase和Kudu）提供了更好的结果（至少5倍）。由于Avro具有最轻量的编码器，因此其实现了最好的摄取性能。
- 另一方面，在这个测试中HBase非常慢（性能比Kudu差）。这很可能是由于行键的长度（6个并置列）引起的，其平均约为60个字节。HBase必须为一行中的每一列分别编码一个键，这对于长记录（包含许多列）可能不是最佳的方法。

各种格式的随机数据查找延迟

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

AVERGE RANDOM LOOKUP LATENCY[S]

平均随机查找延迟 [单位：S]

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

该图显示了每种测试格式和压缩类型的平均随机记录查找延迟 [以秒为单位]

测试描述：通过提供记录标识符（复合键）从记录中检索非键属性。

注释：

- 当通过记录键访问数据时，因为使用了内置索引，Kudu和HBase的访问速度是最快的。图上的值都是基于冷缓存（cold cache）进行测量。
- 使用Apache Impala进行随机查找测试对于Kudu和HBase来说是次优选择，因为在真正执行查询（计划、代码生成等）之前耗费了大量的时间 - 通常大约是200ms。因此，对于低延迟数据访问，建议跳过Impala并使用专用API（我们也尝试过这种方法，Kudu和HBase的结果类似 - 冷缓存 < 200ms，预热缓存 < 80ms）。
- 与Kudu和HBase相反，检索以Avro格式存储的单个记录中的数据只能在对整个数据分区的强力扫描中完成（需要注意的是 - 数据由记录键的一部分进行分区，因此针对这种情况应用分区修剪技术）。平均分区的大小为GB级，因此获取所需的记录需要耗费几秒钟的时间（取决于IO吞吐量），并使用大量的集群资源。这最终减少了必须在集群上全速执行的并发查询的数量。
- 同样的问题也适用于Parquet，然而，Parquet格式的柱状特性允许相对快速地执行分区扫描。由于列投影和列谓词的下推，扫描输入集的大小最终从数GB减少到只有几MB（非常高效，56列经过扫描后只剩下3列）。

各种格式的数据扫描速率

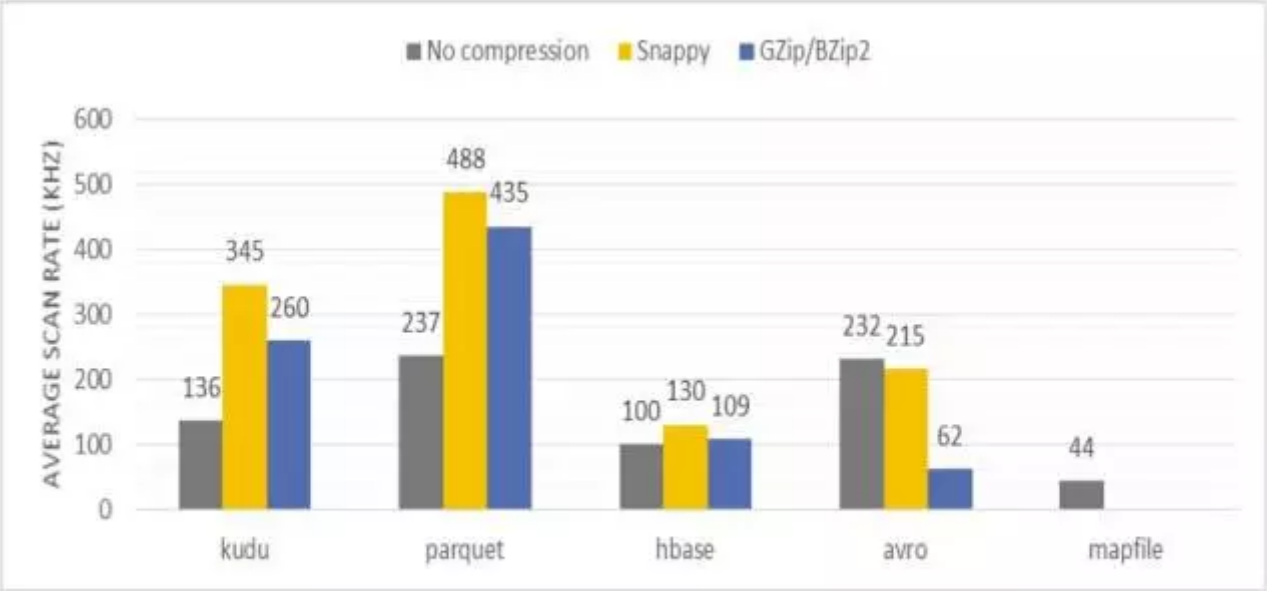


Figure reports on the average scans speed with the same predicate per core [in k records/s] for each tested format and compression type

AVERGE SCAN RATE(KHZ)

平均扫描速率（KHZ）

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

该图显示了各种测试格式和压缩类型对每个核心具有相同的谓词[单位：k 条记录/秒]的平均扫描速度

测试描述：计算在整个记录集合中的非键列之一中具有某个子串的记录数。

注释：

- 由于通过应用列投影输入集数量减少，Parquet在此测试中胜过了Avro。Parquet不仅在每内核处理速率方面保持了最高效率，同时也在完成处理方面保持最快速度。
- 平均扫描速度（KHZ）。
- 在Parquet和Avro的情况下，数据访问并行化的单位是HDFS文件块 - 其很容易在Hadoop集群上的所有可用资源上均匀分布处理。
- 在扫描效率方面，Kudu（采用Snappy压缩）与Parquet相差不大。因为列投影，其受益匪浅。
- 由于数据访问并行化的单位是表分区，扫描存储在Kudu和HBase中的数据可能不平衡。因此，扫描中涉及的资源量取决于给定表分区的数量及其在集群中的分布。
- 在这个测试案例中，因为Kudu不支持使用的谓词，所以不可能使用Kudu的本地谓词下推功能。附加测试结果证明，当使用支持的谓词时，Kudu扫描速度比Parquet更快。
- 在使用HBase进行测试之前，扫描的列在专用HBase列族中被分离 - 这就提高了5倍的扫描效率。但仍然与Parquet或Kudu存在较大差距。

测试经验教训

在本节中，我们想分享关于数据格式使用的其它注意事项及其优点和缺点，因为这些是从我们的参考工作负载测试中得出的：

- **存储效率** – 采用Parquet或Kudu和Snappy压缩，与未压缩的简单序列化格式相比，总的数据量可以减少10倍。
- **数据撷取速度** - 所有基于文件的解决方案提供了比专用存储引擎或MapFiles（排序后的序列）更快的数据撷取速度（在2倍-10倍之间）。
- **随机数据访问时间** - 使用HBase或Kudu，典型的随机数据查找速度低于500ms。使用智能HDFS名字空间分区Parquet可以提供一秒级的随机查询速度，但是会消耗更多的资源。
- **数据分析** – 利用Parquet或Kudu可以执行快速和可扩展（通常每个CPU内核每秒超过300k条记录）的数据聚合、过滤和报告。
- **支持就地数据突变** - HBase和Kudu可以就地修改记录（模式和值）；与之对比，不可能就地修改直接存储在HDFS文件中的数据。

值得注意的是，压缩算法不仅在减少数据量方面发挥了重要作用，在增强数据撷取和数据访问的性能方面也扮演着重要角色。在所有这些领域中，Snappy编解码器为所有测试技术提供了最佳的结果，比没有压缩的纯编码（Avro除外）更好。

结论

对Hadoop生态系统上流行存储技术的评估已经在许多方面展示了每种技术的利弊，这些方面例如减少总体数据量、简化数据撷取及提高数据访问的性能。

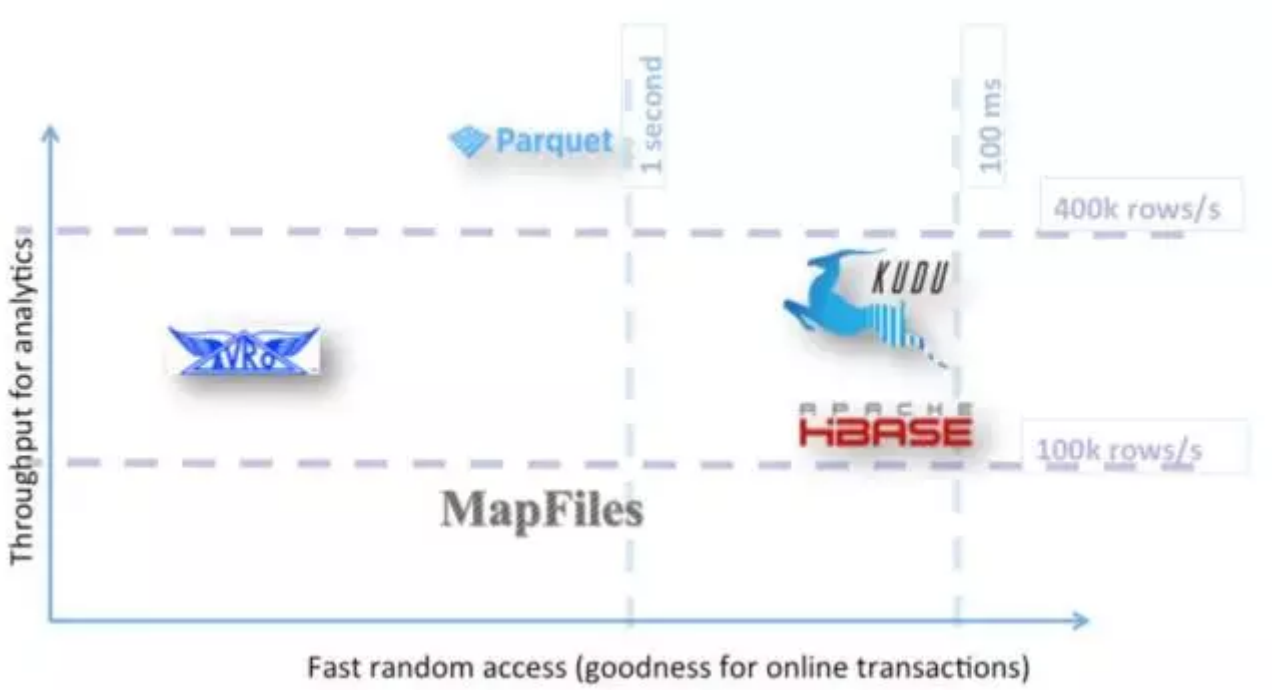
Apache Avro已被证明是一种用于结构化数据的快速通用编码器。由于具备非常高效的序列化和反序列化性能，当需要同时访问记录的所有属性时，此格式可以保证非常好的性能 - 数据传输、分段区域等。

另一方面，Apache HBase提供了非常优异的随机数据访问性能，以及如何存储数据（无模式表）的最大灵活性。HBase数据的批处理性能在很大程度上取决于所选择的数据模型，并且通常不能在该领域与其他测试技术竞争。因此，任何使用HBase数据的分析都应该很少执行。

同时列存储方式，例如Apache Parquet和Apache Kudu，在快速数据采集、快速随机数据查找和可扩展数据分析之间提供了非常好的灵活性，同时确保了系统简单性 - 只需要利用一种存储数据的技术。

Parquet在更快的数据扫描和撷取方面具有优势，而Kudu擅长于更快的随机查找。

替代单一存储技术实现可以考虑由用于批处理（如Parquet）的原始存储和用于随机存取的索引层（例如HBase）组成的混合系统。这允许在某些访问路径上充分利用技术专业化/优化，并提供最佳性能。值得注意的是，这种方法存在数据重复和系统架构总体复杂性的问题，并且需要以更高的维护成本为代价。因此，如果系统的简单性是重要因素之一，Apache Kudu似乎是一个很好的折衷方式。



Throughput for Analytics
分析吞吐量
Map Files
地图文件
Fast random access (goodness for online transactions)
快速随机访问（在线交易的优点）

分析负载和随机查找负载综览的性能（在单个数据单元/分区上）。

◆◆◆

请点击 **“阅读全文”** 进入微站获取源文件
（更多技术干货、行业动态，请关注【微站】，不定时更新）

阅读原文