

## 基于 MapReduce 面向列的数据库存储方案研究

李爱武<sup>1</sup>, 刘 宁<sup>2</sup>, 严升则<sup>2</sup>

(1 广东邮电职业技术学院 计算机系, 广东 广州 510630; 2 中山大学 软件学院, 广东 广州 510275)

**摘 要:** 将传统的并行 DBMS 技术应用到 Hadoop 框架, 在此基础上将面向列的数据库存储技术引入 Hadoop 的复制和调度机制, 进而获得高效的编程模型与编程 API. 实验表明, 提出的方案主要应用在 map phase 阶段, 同时能够使得 MapReduce 的性能提高两倍.

**关键词:** 列存储; 并行 DBMS 技术; Hadoop 框架; Map

**中图分类号:** TP274+.5

**文献标识码:** A

**文章编号:** 1000-7180(2013)06-0092-05

## Research on Column-Oriented Database Storage Scheme Based on MapReduce

LI Ai-wu<sup>1</sup>, LIU Ning<sup>2</sup>, YAN Sheng-ze<sup>2</sup>

(1 Department of Computer Science, Guangdong Posts and Telecom Vocational Technology College, Guangzhou 510630, China;  
2 School of Software, Sun Yat-sen University, Guangzhou 510275, China)

**Abstract:** To improve the performance of MapReduce, the paper introduces the parallel DBMS into the Hadoop framework and introduces the column-oriented storage technology into the copy and scheduling of Hadoop, to get efficient coding model. Experimental results show the scheme presented in the paper can improve the performance of MapReduce.

**Key words:** column storage; parallel DBMS technology; Hadoop Framework; MapReduce

### 1 引言

近年来,随着计算机处理的数据规模的迅速增长,对大规模数据处理系统的需求也在迅速增长.传统的处理方式是依赖于并行数据库管理系统.云计算的提出<sup>[4]</sup>,为大规模数据处理带来了新的思路<sup>[1-2]</sup>.作为一种对 MapReduce 开源的实施, Hadoop 具有使用方便,售价低,对硬件容错率地,编程灵活等特点而得到了业界的赞同,是目前应用非常广泛的云计算编程框架.

然而当企业利用 Hadoop 试图扩大他们的工作量的时候,却引发了极大的性能瓶颈.例如,一个大规模消费银行的用户想利用 Hadoop 处理网络应用上的日志.他们开始时得到的是未经处理的文本格

式的网络应用.由于来自其他网络应用的日志的增加并且保留日志的时间增加到 90 天,他们开始时使用的 20 个节点的 Hadoop 集群就不能在产生合理的时间记录.这些问题为 Hadoop 的进一步的应用提出了挑战.为了提高 Hadoop 处理大规模数据的性能,本文将传统的并行 DBMS 技术应用到 Hadoop 框架<sup>[3]</sup>,在此基础上将面向列的数据库存储技术引入 Hadoop 的复制和调度机制,进而获得高效的编程模型与编程 API.

本文首先设计并实现一种面向列的、二进制的存储格式,然后描述这种格式如何和 Hadoop 的分布式文件系统进行交互,以及列数据合并的重要的机制.最后通过实验证明 Hadoop 能改变改变存储格式而不引起大规模的重新构造数据记录.

收稿日期: 2012-08-10; 修回日期: 2012-10-17

基金项目: 国家自然科学基金项目(60180109)

## 2 相关技术与挑战

### 2.1 Hadoop 相关技术

首先,介绍 Hadoop 和本文会用到的用于实施面向列存储技术的一些相关理论。

图 1 是一个 MapReduce 的作业示例<sup>[5]</sup>,该作业用来分析抓取到的文件,并发现独特的“内容类型”。首先观察主程序中被配置成 InputFormat 和 OutputFormat 的地方。

```
class MyMapper{
void map (NullWritable key,Record rec) {
    String url=(String) rec.get("url")
    if(url.contains ("ibm. com/jp"))
        output. collect(null,
            rec.get("metadata"). get("content-type"));
}
}
class MyReducer{
Void reduce (NullWritable key, Iterator <Text> vals){
    HashSet <Text> distinctVals= new HashSet<Text>
();
    for (Text t: vals)
        distinctVals. add(t);
    for(Text t;distinctVals)
        output. collect (unll,key);
}
}
main(){
    Job job=new Job()
    job. setMapperClass(MyMapper);
    job. setReducerClass(My Reducer);
    job. setInputFormat(SequenceFileInputFormat. class);
    SequenceFileInputFoemat. addInputPath (job, "/data-
jan");
    job. set OutputFormat(TextOutputFormat. class);
    TextOutputFormat. setOutputPath ("/output/job1");
    Job Runner. submit(job);
}
```

图 1 MapReduce 作业示例

一个输入格式(InputFormat)是 Hadoop 的一个重要的抽象和扩展点。主要负责两个功能:拆分数据,将有相同特点的数据放在同一个 map 组里面;将磁盘上的数据转换成键值对的形式,使其符合 map 的数据结构。

Hadoop 提供不同的 InputFormat 用来从文本文件,逗号分隔文件等中使用数据。在图 1 的示例

中,MapReduce 作业使用了 SequenceFileInputFormat 方法。一个 SequenceFile 用标准的序列化格式存储键值对。在 Hadoop 中与输入格式对应的是输出格式,使用 Mapreduce 技术将键值对转化存放在磁盘中。

Map 和 Reduce 的数据可以为任意的格式,本文假设记录的目标为值。像 AVRO, Thrift, Protobufs 等一些序列化方法可以用来抽象记录,可以用来将记录转化为未加工的字节,从磁盘上读取数据,或者将它们传递给 Map 和 Reduce 任务。使用 Java 的 get(name)方法获得记录的属性,将属性名作为参数。访问属性经常需要类型转换。

### 2.2 面临的挑战

在进行大规模数据分析时,使用诸如阵列,映射,嵌套存储等复杂数据类型会更加方便。最近研究表明<sup>[6]</sup>,当工作以读数据为主时,使用本地的嵌套表示的复杂类型效果会更好。

图 2 中显示了一个复杂类型的图表。这是一个实际的从企业内部网络搜索抓取文件,并且存储数据源,萃取注释和链接。因为注释和元数据在每一页上面都千差万别,所以存储使用 Map 结构。链接使用数组形式存储。复杂数据类型会引发一个严重的问题,即缺乏有效的面向列压缩技术,使得 Hadoop 的处理性能急剧降低。

```
URLInfo {
    Utf8 url,
    Utf8 srcUrl,
    time fetchTime,
    Utf8[] inlink,
    Map<String,String<metadata,
    Map<String, String>annotations,
    byte[]content
}
```

图 2 复杂数据示例

## 3 面向列存储技术

### 3.1 复制和主机代管

在 Hadoop 中实现面向列存储的一个简单方式是将每一列的数据存储在一个单独的文件中。这会引出两个问题:(1)我们怎么能产生大致相同大小的分割,使工作可以有效地在集群中并行执行?(2)如何确保在数据表中不同列的对应值能被协同地放在 Map 任务的相同节点上?

为了解决第一个问题,本文将数据表水平分区,

并且将每个水平分区中的数据存储在一个单独的子目录中. 每个子目录作为一个分割. 第二个问题比较难解决, HDFS 在商用服务器中使用 3 块级的复制来提供容错, 但默认的块不能保证主机代管.

假设一个数据表具有 3 列数据 C1, C2, C3 并且 3 列数据存储在一个文件之中, 如图 3 所示, 为了简化, 同样假设每个文件占有一个单独的 HDFS 块. 在实践中一个大文件可以跨越许多个 HDFS 块. 文件 C1-C3 需要被连接在一起, 作为一个分割. 但是按照 Hadoop 的默认安置规则, 他们会被随即的分配在不同的组中. 图 3 中表明 Hadoop 的默认安置规则的工作方式. 文件 C1-C3 被共同定位在节点 1, 但是没有被其他节点定位在相同位置. 假设一个 map 任务正在被 C1-C3 的分割调度, 但是节点 1 正在忙. 在这种情况下, Hadoop 会在其他节点上面调度 map 任务, 比如说节点 2, 但是性能会受到影响, 因此 C3 将不得不进行远程接入.

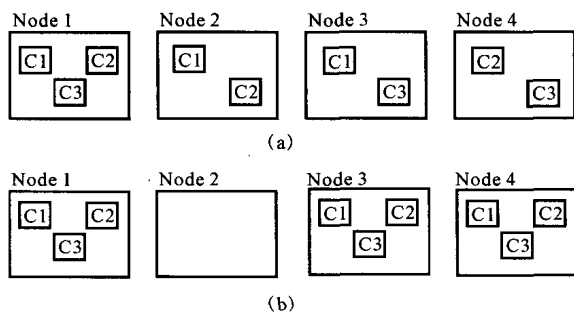


图 3 协同放置的列文件

近年来, 关于 RCFile 格式的工作取得了一定的进展, 它通过对一个 PAX 格式进行插入重排列面向列对象解决了这些问题. RCFile 通过将 HDFS 块排列成叫做行组的块. 每个行组在开始时包括一个同步发生器, 然后是元数据区, 再后面是数据如数据区以面向列方式布局. 元数据区包括数据区的列和它们的偏移量, 也包括数据区的行. 因此所有的列都被组装成单一的行组, 并且每一个行组能像一个分割一样单独在工作, 它避免了当存储列是面临的两方面挑战.

尽管 RCFile 非常简单并且很适合 Hadoop 的限制, 但 RCFile 格式中, 列都被插入单独的 HDFS 块中, 因此, 由于 HDFS 被重取和放置, 提高 I/O 效率变的很困难. 调整行组的大小和 IO 传输大小正确也变得至关重要. 像 1MB 那样大的 I/O 传输, 包含多于 4 列的记录的传输使用默认设置的 RCFile 传输的性能极差. 最后, 每一组数据都需要额外的元数据, 会占用额外的空间.

本文提出 CIF 存储格式来弥补 RCFile 格式的不足.

### 3.2 CIF 存储格式

本文实现一种新的 HDFS 块的方式来解决主机代管问题. 通过对 HDFS 的“dfs. block. replicator. classname”的配置, 可以改变 HDFS 的放置策略, 指定它指向适当的级别. 这一特点已经在 Hadoop0.21.0 版本开始使用, 并且不需要重新编译 Hadoop 和 HDFS.

列放置策略(CPP)是我们面向列块安置的类的名称. 为了描述上的方便, 我们假设每一个列文件占有一个单独的 HDFS 块并且像文件的层次一样描述 CPP. CPP 能有效的保证同一个分割的不同列的文件能出现在不同副本中的同一位置. 正如图 3b 所示, C1-C3 会通过 CPP 被放在不同副本的同一个位置. 存储分割的子目录需要为 CPP 命名一个特定的规则. 不遵循这个命名规则的文件就会重复使用 HDFS 默认的放置规则.

我们用两个阶段来部署我们的面向列存储格式的逻辑: 列输入格式(CIF)和列输出格式(COF). 数据可以以任意的格式进入到 Hadoop, 一旦数据在 HDFS 中, 就会使用一个并行装载机用 COF 装载数据.

假设在一个场景中出现图 2 中所描述的数据. 爬取数据按一定的时间间隔进行, 并且一天的数据已经到达并且需要被存储在“/data/2011-01-01”中. 当有一组资料被使用 COF 的方式被装载到子目录的时候, 它将数据资料组分割成小的水平的分割. 每个分区, 被称作一个分裂目录, 是一组文件的子目录, 每列一个数据集. 每个子目录还有一个单独的文件用来描述整体概要. 图 4 中用来显示子分区 s0 和 s1 使用 COF 的数据布局.

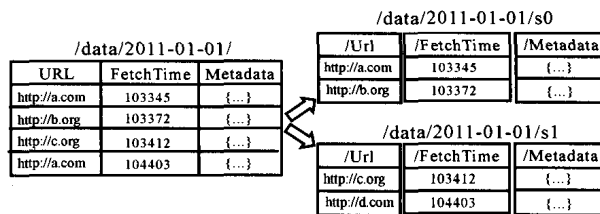


图 4 COF 的数据布局

当读一个数据集的时候, CIF 能实际地将一个或多个分裂目录分配给一个单独的分割. 一个分割目录的列文件被反复浏览, 并且记录重新以位置为依据被重新组装. 可以通过提供列清单而将预测加入到 CIF 中. 这些工作可以在配置一个 MapReduce

作业的时候按如下方法做: ColumnInputFormat.setColumns(job,"url,metadata");被 CIF 创建的对象只能由被选中的字段填充. 剩余的字段不会被浏览.

#### 4 实验

本文通过实验来展示如何将面向列存储技术应用在 Hadoop 中,以提高 Hadoop 的性能,并将 CIF 与其他流行的格式,如文本格式,查询文件和 RCFile 进行比较,来验证本文提出的方法的有效性.

实验运行在一个包含 42 个节点 1 GB 数据量的以太网簇中. 两个节点被保留用来运行 Hadoop 的作业追踪和命名节点. 剩下的 40 个节点用在 HDFS 和 MapReduce 中. 每个节点有 8 个核(通过两个 2.4 GB 的四核), 32 GB 的主存,和 5 个本地的 500 GB 的硬盘. 数据节点将他们的数据分布在 4 个硬盘上,使用 Hadoop 0.21.0 版本,并且每个节点被配置成 6 个 Mapper 和一个 Reducer.

本文采用如下的一个合成的数据集: 每个记录由 6 个字符串, 6 个整数和一个 Map 组成. 整型数据是随即生成的 1 到 10 000 之间的数. 随即的字符串的长度是 20 到 40 之间的随机产生的可读 ASCII 字符. 每个 Map 的列由 10 组标签组成, 每个标签都是随即生成的长度为 4 的字符串, 值是随机生成的整型. 数据被使用每种格式写. 对于 SEQ, NullWritable 被作为键使用. 一个包含上述的方面记录被作为值类. 在 SEQ 格式下数据总的大小为 57 GB. I/O 转换大小被设置为 128K. 分别用 4 kB 和 1 MB 重复实验产生相似的结果. 浏览各种不同工程的数据集所用的时间在图 5 中显示. 文件系统的 cache 在每次实验前都被刷新. 在 TXT 和 SEQ 中, 浏览所用的时间大致相同, 因此只重复一个数据.

与 TXT 格式的比较: 正如图 5 所示, 使用 SEQ 浏览所用的时间比 TXT 快 3 倍. 分析文本中的每一列数据导致 TXT 迅速成为绑定 CPU. 然而, 这种分析的消耗在 SEQ 中是没有的.

与查询文件的比较: 当使用 CIF 时, 浏览一个单独的整型, 字符串或 Map 的速度比 SEQ 快 2.5 到 95 倍. 在每一个例子中, 提速的原因都归功于 CIF 比 SEQ 读更少的数据. 当读全部的数据集时, CIF 用的时间比 SEQ 多 25%. 这是因为当从列中增加数据并存储到不同的文件中的时候 CIF 额外的查询会增加时间消耗. 在其他方面, CIF 比 TXT 和 SEQ 更优越.

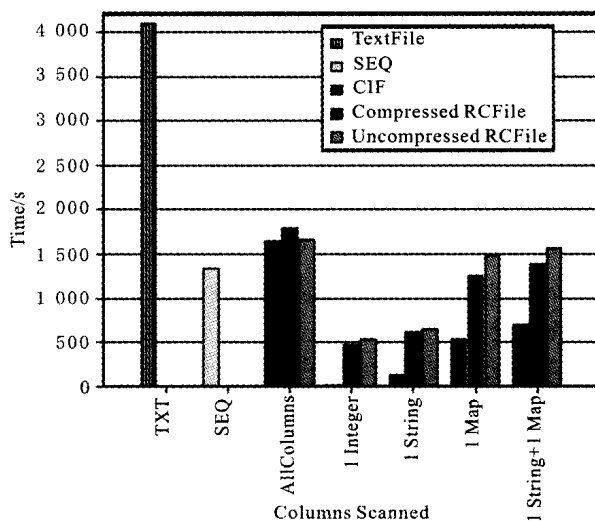


图 5 TEXT, SEQ, CIF 和 RCFile 实验结果对比

与 RCFile 比较: 未压缩的 RCFile 有 69 GB 大小, 而压缩的 RCFile 有 43 GB. RCFile 的行组被设置为 4 MB 大小. 当数据集中有一小部分列被选择时, CIF 比 RCFile 在消除不必要的 I/O 消耗方面更加高效. 对于单一整型数据的情况, RCFile 表现不好, CIF 比未压缩的 RCFile 快 3 倍. 通过测量 io 消耗显示, RCFile 比 CIF 多读 20 多个字节, 虽然只浏览了不到一列的数据. 这些消耗一部分的原因是在 RCFile 中使用了不高效的序列化. 此外 RCFile 因为要在每 4 MB 就中断一次元数据块, 所以有更多的 CPU 消耗. 不同存储格式存储速率比较如表 1 所示.

表 1 不同存储格式存储速率比

| Layout       | Data Read/GB | Map Time/s | Map Time Ratio | Total Time/s | Total Time Ratio |
|--------------|--------------|------------|----------------|--------------|------------------|
| SEQ-uncorncp | 6 400        | 1 416      | —              | 1 482        | —                |
| SEQ-record   | 3 008        | 820        | —              | 880          | —                |
| SEQ-block    | 2 848        | 806        | —              | 886          | —                |
| SEQ-custom   | 3 040        | 754        | 1.0x           | 806          | 1.0x             |
| RCFile       | 1 113        | 702        | 1.1x           | 761          | 1.1x             |
| RCFile-comp  | 102          | 202        | 3.7x           | 291          | 2.8x             |
| CIF-ZLIB     | 36           | 12.8       | 59.1x          | 77           | 10.4x            |
| CIF          | 96           | 12.4       | 60.8x          | 78           | 10.3x            |
| CIF-LZO      | 54           | 12.4       | 61.0x          | 79           | 10.2x            |
| CIF-SL       | 75           | 9.2        | 81.9x          | 70           | 11.5x            |
| CIF-DCSL     | 61           | 7.0        | 107.8x         | 63           | 12.8x            |

当使用压缩的 RCFile 时, 运行时间有所提高. 然而, CIF 仍然在所有的情况下都更快. 比如, 当一个单独的整型被规划, CIF 比压缩的 RCFile 快 33

倍. 在一个数据集中所有的数据都被检查的情况, CIF, 压缩的 RCFile, 未压缩的 RCFile 有相似表现. SEQ 比其他的方法快 1.2 倍.

## 5 结束语

本文描述了一种不仅兼容 Hadoop 的编程接口而且对 Hadoop 的核心没有要求改变的新的二进制存储格式的面向列存储技术. 它包括轻量级压缩、懒惰记录建设等特点来避免反序列化不必要的记录. 一个在企业内部网络爬行的实验用来证明, 面向列存储技术可以在 Hadoop 的 map 阶段两个数量级的表现, 并且能提高整个 MapReduce 作业一个数量级的表现.

### 参考文献:

- [1] Dean J, Ghemawat S. MapReduce: A flexible data processing tool[J]. CACM, 2010, (53): 72-77.
- [2] Pavlo A, Paulson E, Rasin A, et al. A comparison of approaches to large-scale data analysis[C]// proceedings of the 2009 ACM SIGMOD International Conference on Management of data. NY, USA: ACM,

2009; 165-178.

- [3] Abouzeid A, Bajda-Pawlikowski K, Abadi D J, et al. HadoopDB: an architectural hybrid of map reduce and DBMS technologies for analytical workloads[C]// PV-LDB Lyon, France. ACM, 2009; 24-28.
- [4] Feng Zhao. Sensors meet the Cloud: planetary-scale distributed sensing and decision making[C]// proceedings of the 9th IEEE International Conference, Cognitive Informatics. Beijing, China: IEEE, 2010; 998-998. 2010.
- [5] Dean J, Ghemawat S. MapReduce: simplified data processing on large clusters[C]// Proceedings of Operating Systems Design and Implementation. San Francisco, CA: IEEE, 2004; 137-150.
- [6] Melnik S. Dremel: interactive analysis of web-scale datasets[J]. PVLDB, 2010, 3(1): 330-339.

### 作者简介:

李爱武 男, (1969—), 硕士, 讲师. 研究方向为数据库与数据挖掘.

刘宁 男, (1974—), 博士, 副教授. 研究方向为数据库与数据挖掘.

(上接第 91 页)

## 5 结束语

本文设计了一个兼容 AHB 总线的 Nor Flash 控制器 IP. 为满足医学系统芯片(SOC)的需要, 采用了一种改进的设计方案, 该设计方案有效地控制了 IP 的面积和成本, 减少了 Flash 对总线的占用时间, 提高了系统的工作效率. FPGA 的验证也表明, 该 IP 满足设计要求.

### 参考文献:

- [1] Teng Xiao-Fei, Zhang Yuan-Ting, Carmen C Y Poon et al. Wearable medical systems for p-Health biomedical engineering[J]. IEEE Reviews, 2008(1): 62-74.
- [2] 李严, 张元亭, 王磊. 低功耗、低频率、低噪声医学芯片的设计及应用[J]. 中国医疗器械信息, 2009(1): 17-29.

- [3] 王力伟. 多端口存储器控制器 IP 核的设计与实现[J]. 武汉大学学报: 理学版, 2007(5): 617-621.

- [4] ARM Company, AMBA Specification (Rev 2.0)[R]. Technical Reference Manual.

- [5] Technical Datasheet. S29AL008J[R]. United States: Spansion Inc, 2012.

- [6] Technical Datasheet. SST39VF1601 [R]. United States: Silicon Storage Technology, Inc, 2012.

- [7] Technical Datasheet. MX29LV800C [R]. Taiwan: Macronix International Co, Ltd, 2012.

### 作者简介:

田鹏 男, (1987—), 硕士研究生. 研究方向为控制器和接口设计.

张正平 男, (1964—), 教授, 博士生导师. 研究方向为通信新技术.