面向 HBase 的大规模数据加载研究①

贺正红, 周 娅, 文缔尧, 吴清霞

(桂林电子科技大学 计算机科学与工程学院, 桂林 541004)

摘 要: 分布式数据库 HBase 在大规模数据加载中较传统关系型数据库有较大的优势但也存在很大的优化空间. 基于 Hadoop 分布式平台搭建 HBase 环境, 并优化自定义数据加载算法. 首先, 分析 HBase 底层数据存储, 实验得出 HBase 自带数据加载方式在效率和灵活性方面存在不足; 进而, 提出了自定义并行数据加载算法, 并针对集群进行优化. 实验结果表明, 优化后的自定义并行数据加载方式能充分发挥集群性能, 具有较好的加载效率和数据操作能力.

关键词: HBase; Hadoop; MapReduce; 数据加载; 性能优化

Research on Large Scale Data Loading Based on HBase

HE Zheng-Hong, ZHOU Ya, WEN Di-Yao, WU Qing-Xia

(Computer Science and Engineering, Guilin University of Electronic Technology, Guilin 541004, China)

Abstract: Distributed database HBase has the greater advantage than traditional relational database in large scale data loading but there is also a lot of optimization space. We build HBase environment based on the Hadoop distributed platform, and optimize self-defining data loading algorithm. Firstly, this paper analysis the HBase underlying data store, experiments work out that data loading methods of HBase are insufficient in efficiency and flexibility. Furthermore, it proposes self-defining parallel data loading algorithm, and optimizes the cluster. The experimental results show that the optimized self-defining parallel data loading method can give full play to the cluster performance, has good loading efficiency and data operational capacity.

Key words: HBase; Hadoop; MapReduce; data load; performance optimization

大数据时代的到来, 对数据的处理提出了更高的要求, 越来越多的企业需要存储 TB、PB 级的数据. 而传统的关系型数据库系统(RDBMS)只专注于一台机器, 然而单台机器无法满足大量数据的存储, 同时机器的 I/O 性能也成为海量数据存储中面对并发服务的瓶颈, 这样的体系结构也严重的影响了系统的扩展能力, 根本上限制了其存储能力和分析能力. 而分布式技术的发展有效的推动的海量数据处理的发展.

Apache 顶级开源项目 HBase 是分布式平台 Hadoop 中的开源数据库. HBase 不同于一般的关系数据库, 适合于非结构化数据存储, 其存储模型是基于列的. 该数据库架构在 Hadoop 之上, 与分布式文件系统 HDFS 和并行处理框架 MapReduce 结合的非常完善. 作为 Google 的 Bigtable 的开源, 实现了 Bigtable 系统的大部分功能.

本文考虑在使用 HBase 时首先需要将数据加载到 HBase 表格中. 在此 HBase 字身提供了两种加载数据的方法: 第一种通过工具直接将数据载入 HBase 表格; 第二种先生成 HFile 文件, 然后将 HFile 文件关联表格. 实验发现这两种方式存在灵活性和效率不足. 基于此问题, 本文深入研究 HBase 自带加载数据方式, 结合 HBase 对 MapReduce 的支持, 实现自定义的加载数据方法, 针对实验数据特点和集群的环境对加载方法进行了优化, 为海量数据的处理提供的基础保障.

① 基金项目:广西教育厅高校科技项目(2013YB095);广西信息实验科学中心重点项目(20130111);广西教育厅一般资助项目(2013YB051)
收稿时间:2015-10-19;收到修改稿时间:2015-11-25 [doi: 10.15888/j.cnki.csa.005194]

1　HBase介绍

1.1 HBase 架构

HBase 系统由一个主控节点(HMaster)和多个区域服务器(HRegionServer)组成, 它遵循主从式体系结构. HRegion 服务器由 HMaster 服务器调度和管理, 文件数据都存储在 HRegion 服务器上, 主服务器不存储实体数据. 利用 ZooKeeper 来协调和处理集群运行过程中出现的问题. 数据存储在分布式文件系统 HDFS 之上, 能够为用户提供可靠存储服务.

该数据库系统适合用来存储非结构化和半结构化的松散数据. HBase 满足海量数据的秒级简单查询, 为大数据的处理提供了基础. 由于HBase是Hadoop平台上的数据库, 可以通过不断增加廉价的商用服务器来扩展Hadoop平台的计算与存储能力, 这样HBase数据库也随之达到了横向扩展的目的. HBase 中表的特点:

(1) 存储海量数据: 一个表可以存储有上亿行、包含上百万列.

(2) 基于列族: 表中数据面向列族存储和权限控制, 按照列族检索.

(3) 存储数据稀疏: 包含的空单元格(cell)多, 但是空单元格并不会占用实际存储空间. 这样就可以将表设计的非常稀疏.

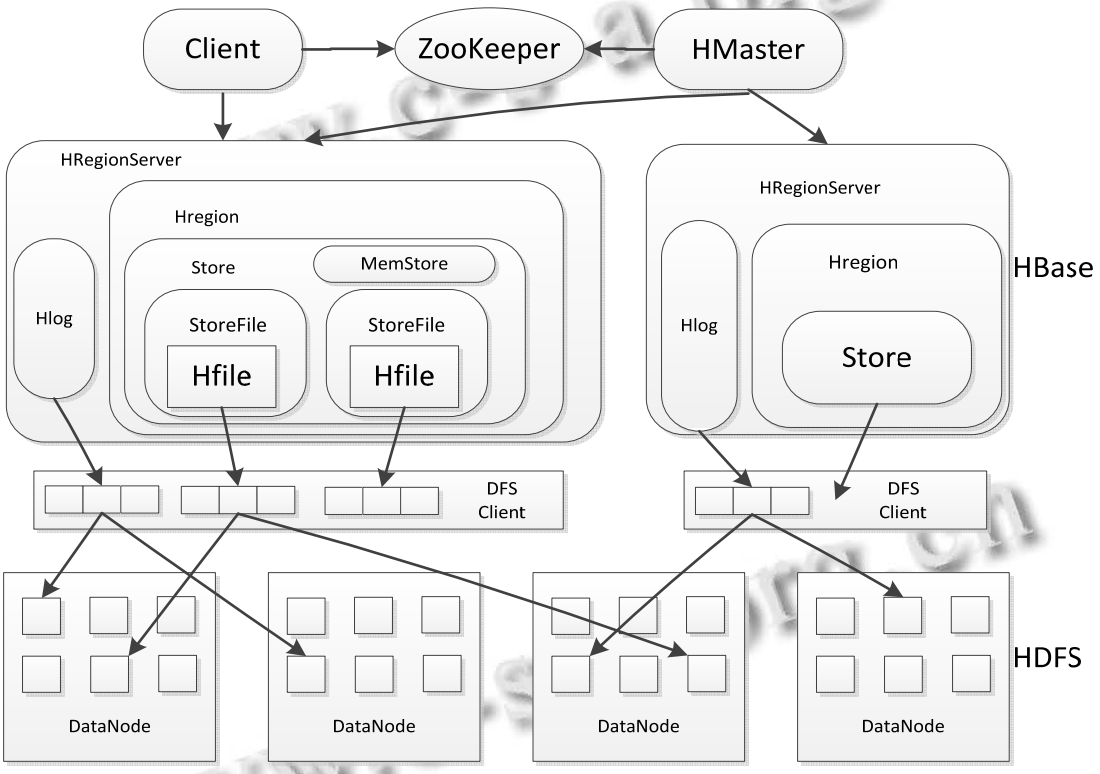

图 1　HBase 系统架构

HBase 是按照表格的形式来存储数据. 表由行键(rowkey)、时间戳(timestamp)和列(column)三部分组成, 其中整体列包含多个列族(column family). 每一个列族中又可以划分为多个具体的列(column), 存储具体的数据. HBase 存储的数据模型如表 1 所示.

表 1　HBase 数据模型

行键	时间戳	列族 "Contents"	列族 "anchor"	
			列 "col1"	列 "col2"
"rownum"	T1		"my.look"	
	T2			"CNN.com"
	T3	"<html>..."		
	T4	"<html>..."		
	T5	"<html>..."		

1.2 底层数据存储

HBase底层数据存储在分布式文件系统HDFS上, 存储格式为 HFile, 该文件通过 HFile 类来实现数据的高效存储, 实际上是对存储文件(Store File)做了轻量级包装, 即StoreFile底层就是HFile. 它是基于Hadoop的 TFile 类, 同时也模仿了 Google 的 Bigtable 架构中使用的 SSTable 格式. 由于 HBase 是以列为单位的数

据库, 其每个列族的数据存储在同一个 HFile 文件中. HFile 文件是变长的, 定长的块只有 File Info 和 Trailer 这两部分, Trailer 包含指向 File Info、Data Index 和 Meta Index 的指针, 被写入到文件的末尾. Data Index 和 Meta Index 分别记录了 Data 和 Meta 块的偏移, Data 块中包含了多个 KeyValue. 表格中的单元格由(Table,Key,Family,Qualifier,Timestamp)共同决定. HFile 抽象形式如表 2 所示.

表 2 HFile 文件结构

Data	Data	Data	...	Meta	Meta	...	File Info	Data Index	Meta Index	Trailer

HFile 数据部分中 KeyValue 对就是一个简单的 byte 数组, 其中包含有很多项信息, 并有固定的结构. 开始是两个固定的数值, 分别表示 Key 和 Value 的长度, 紧接的数值表示 Row 的长度, 跟着的是 Row, 然后是固定长度的数值, 表示 Family 的长度, 然后是 Family, 接着是 Qualifier, 然后是两个固定长度的数值, 表示 Time Stamp 和 Key Type(Put/Delete). 最后 Value 部分为纯粹的二进制数据. 具体格式如表 3 所示.

表 3 Key-Value 键值对详细信息

Key Length	Value Length	Row Length	Row	Column Family Length	Column Family	Column Qualifier	Time Stamp	Key Type	Value

2 加载数据方式研究

HBase 适用于海量数据的存储与查询, 使用 HBase 的第一步是将海量数据加载到 HBase 数据库中. HBase 本身提供了几种不同的数据加载方法, 同时也提供了自定义加载数据接口.

2.1 HBase 自带加载数据方式

本文使用的是 0.94.7 版本的 HBase, 该版本是官方确定的比较经典的版本. 其中提供的工具包括: 统计表中存贮单元(CellCounter);加载 HFile 文件到表格(completebulkload); 复制表格的数据(copytable); 表格数据导出到 HDFS(export);数据导入到表格(import); 导入 HDFS 中特定格式数据(importtsv); 统计表中数据行数(rowcounter); 比较集群间两表的数据(verifyrep).

由于 import 工具导入的是表格备份数据, 所以在实际加载数据中用处不大. 这里仅包含两种方式能够将特定格式导入到表格: 第一种是 importtsv 直接将数据导入到 HBase 表格; 第二种先用 importtsv 生成 HFile 格式数据, 然后再通过 completebulkload 将生成的 HFile 文件与表格关联.

2.1.1 通过 importtsv 载入数据

在 Hadoop 环境下执行 HBase 提供的 MapReduce 任务, 将 HDFS 上的固定格式的数据加载到 HBase 指定的表格中. 这里使用的是 importtsv 工具, 通过 MapReduce 任务的 Map 函数来完成, 根据给定的参数(表格名称、输入文件位置、数据分隔符、对应列、主键等)解析数据, 生成 Put 对象, 最终通过 TableMapReduceUtil 将 Map 数据输出到表格中. HBase 中包含了该方法的具体使用细则, 需要的相关参数等, 通过执行该命令查看.

其中必选参数包含有:

-Dimporttsv.columns=a,b,c: 对应表格中的列, 同一个列族使用(a:t1,a:t2,a:t3), 列中必须包含有行健 HBASE_ROW_KEY, 行健列的位置任意, 该列对应的文本数据作表格中的行健使用.

<tablename>: HBase 中目标表格的名称

<inputdir>: HDFS 中源数据的路径

其中还包含有一些可选参数. 本文在数据导入中使用到的可选参数是数据分隔符-Dimporttsv.separator, 默认的分隔符是制表符'\t', 由于源数据中的分隔符是'|', 这里将此参数设置为'|'. 在导入数据时使用的命令如下:

hadoop jar /usr/local/hbase/hbase-0.94.7-security.jar importtsv -Dimporttsv.separator='|' -Dimporttsv.columns=HBASE_ROW_KEY,a:t1,a:t2,a:t3, a:t4,a:t5,a:t6, a:t7,a:t8,a:t9 test /put/data.dat

在 hadoop 集群环境下执行. 其中使用 HBase 提供的 HBase-0.94.7-security.jar, 设置分隔符 separator 为'|', 源数据中列对应在 HBase 表格中的存储列, 其中第一列是作为表格的行健, 剩余的 9 列存储在列族 a 中的 t1 到 t9 列里面, 对应的 HBase 表格是 test, 源文件在 HDFS 中的路径是/put/data.dat.(具体执行结果见实验一中方法一)

2.1.2 先生成 HBase 内部存储文件 HFile, 然后与表格关联

HBase 提供的 importtsv 工具, 其中包含一个可选参数-Dimporttsv.bulk.output. 设置该参数后, importtsv 工具并不会将数据直接加载到 HBase 指定的表格中, 而是通过 MapReduce 任务, 按照设置的表格列的信息生成HBase内部存储格式文件HFile, 存储在HDFS中. 然后利用 completebulkload 工具将 HFile 文件与指定的表格关联起来.

创建 HFile 对应的 MapReduce 任务, 在 Map 阶段会根据给定的输入路径、分隔符解析数据, 并生成对应的 put 对象. 然后再 Reduce 阶段根据输出文件的路径, 判定生成的为 HFile, 这时就直接将生成的文件写入到 HDFS 上对应的输出目录. 关联阶段是通过 API 方式将表格保存数据目录指定为 HFile 文件所在路径.

生成 HFile 文件命令如下:

hadoop jar /usr/local/hbase/hbase-0.94.7-security.jar importtsv -Dimporttsv.separator='|' -Dimporttsv.columns=HBASE_ROW_KEY,a:t1,a:t2,a:t3,a:t4,a:t5,a:t6,a:t7,a:t8,a:t9 -Dimporttsv.bulk.output=/out test /put/data.dat

生成的 HFile 文件在 HDFS 的/out 目录下, 且该 HFile 文件满足 test 表格的格式. 然后利用 completebulkload 工具将生成的 HFile 文件加载到 test 表格中. HBase 中包含了 completebulkload 命令的具体使用说明, 包含的参数等, 执行该命令可以查看.

其中包含的参数有:

/path/to/hfileoutputformat-output: HFile 文件在 HDFS 中的路径

tablename: HBase 中目标表格的名称

加载 HFile 文件的完整命令如下:

hadoop jar /usr/local/hbase/hbase-0.94.7-security.jar completebulkload /out test(结果见实验一中方法二)

2.2 自定义加载数据方式

由于 HBase 本身提供的方法在数据加载时, 其中的行健必须是其中的某一列数据, 这样在加载数据时存在格式不够灵活. 在实际中, 通常会遇到对行健的设计要求比较高的情况. 同时面对大量数据时, 难免会出现一些垃圾数据, 这就需要我们对源数据进行清洗整理. 而提供的方法不能这样的需求, 我们就需要自定义来满足.

在 HBase 设计中, 其也提供了接口供用户自己来实现自定义数据加载. 提供的接口可以分为两类: 第一类直接使用 Java API 接口完成; 第二类是通过启动 MapReduce 任务来完成数据加载.

2.2.1 Java API 数据加载

首先需要与 HBase 集群建立连接, 必须要指定 HBase 在 HDFS 中的存储位置和 HBase 使用的 zookeeper 的位置, 然后就可以获取到 HBase 表格对应的描述器类 HTable, 这样我们就可以使用表格了. 对需要加载的数据以行为单位进行处理, 根据给定的分隔符将一行的数据分解为不同的字段, 按照要求取第一、二、四列组合作为行健. 以行健为参数创建加载数据的类 put, 通过 put 就可以操作表格中一行的数据, 将解析的行数据与列关联通过 put 类的 add 方法加入到 put 对象中. 一列数据处理完成后调用 HTable 类的 put 方法将创建的 put 对象加入到表格中, 这样一列数据就成功加载到 HBase 表格中. (实验得到该方法大数据量时处理很慢)总结为如下算法:

算法 1 Java API 加载数据

输入: 数据文件的完整路径

输出: 加载数据是否成功

1.获取 HBase 的配置类 Configuration
2.设置 HBase 在 HDFS 上的存储路径
3.设置 HBase 的 zookeeper 所在节点的位置
4.创建表格对应的描述器类 HTable
5.while 读取文本中的一行数据
6. 按照'|'将一行文本拆分成字符串数组
7. 取字符串数组的第一、二、四元素组成行健 ROWKEY
8. 创建 Put 类的集合 List<Put>
9. 按照列名和拆分字段数据创建 Put 对象, 同时将 Put 加入到集合中
10. 通过表格的描述对象 HTable 的 put 方法将 Put 集合提交
11.End-while

2.2.2 启动 MapReduce 加载数据

Java API 方式加载数据是串行执行的, 在数据量不是很大的情况下能够很好的完成数据加载任务, 但是在面对海量数据入库需求时则显得效率太低, 不能有效的处理加载任务. 此时我们可以借助于Hadoop平台提供的 MapReduce 并行处理框架, 利用 HBase 提供的 MapReduce 接口来实现海量数据加载.

通过 MapReduce 程序来实现, 通常我们都需要实

现自己的 Map 类和 Reduce 类. 这里的数据加载可以直接在 Map 函数中完成, 无需 Reduce 过程来处理, 这样就省去中间结果写入本地磁盘过程. 实现该方式的核心部分是实现 Mapper 类, 需要实现该类的 setup、map 和 cleanup 方法. 用 Map 函数操作 HBase 中的表格前都需要先获取表格的描述符对象, 因为 setup 方法正好能够在所有执行 map 方法前执行完成, 这样我们就在 setup 方法中完成连接表格和获取表格描述符的操作, 得到 HTable 对象. 对数据的处理在 map 方法中, 输入数据的格式为 TextInputFormat, 将文本数据按照行为单位转换为行偏移量和一行的内容, 作为 map 函数的输入参数. 这样在 map 中就可以按照'|'作为分隔符将一行的数据拆分成字符串数组, 取数组的第一、二和四个元素组合作为行健 ROWKEY,根据行健创建一行的 Put 对象, 然后遍历数组中的元素按照给定的列信息将列值添加到 Put 对象中, 这样一行的数据处理完成后就需要通过表格对象 HTable 提交一行的 Put 对象, 将数据存储到指定的表格里面. 所有的 Map 函数处理完成数据集加载后, 需要关闭打开的表格资源, 正好 cleanup 方法是在所有的 Map 执行完成后执行, 这里我们就在 cleanup 中将缓存的 Put 数据持久化到数据库文件中, 同时关闭表格描述符对象. 总结为如下算法:

算法 2 MapReduce 任务加载数据

输入: 源文件在 HDFS 上的路径
输出: 数据存储为 HBase 表格文件

1.自定义类实现 Hadoop 中的 Mapper 类
2.重写其中的 setup, map 和 cleanup 方法
3. 在初始化方法 setup 中, 配置 HBase 的存储路径和 zookeeper 所在节点位置
4. 根据配置信息和表格名称, 创建表格描述类
5.编写自己的 map 方法来处理一行数据
6. 根据'|'分割一行数据
7. 组合第一, 二、四元素为 ROWKEY
8. 创建行健对应的 Put 对象
9. 按照列名添加数据进入 Put 对象
10. 利用 HTable 提交一行数据
11.cleanup 方法完成资源回收
12. 提交缓存中未持久化的 put 数据
13. 关闭 HTable 对象
14.配置 job 信息, 并提交 MapReduce 作业

2.3 加载数据性能优化

在 2.2 中的 MapReduce 方法已经完成海量数据的并行加载. 但是通过对实验 2 结果以及加载过程的分析, 得出影响 MapReduce 加载数据的因素包括有: 磁盘 I/O 量、网络 I/O 量、集群的负载均衡和入库应用程序等方面. 本节对这些因素进行分析, 从而做出了相应的方法优化调整.

2.3.1 使用 Snappy 压缩算法

由于 Snappy 旨在提供高速压缩速度和合理的压缩率, 以其独特的优势在 Google 内部从 BigTable 到 MapReduce 以及内部的 RPC 系统被广泛的使用. 针对在数据入库中磁盘 I/O 和网络 I/O 量比较高, 导致集群的整体 I/O 吞吐量大. 这里选择使用该压缩算法对入库的数据进行压缩, 这里以一组实验数据为例说明. 入库前文件的记录数是 3000 万条, 总大小是 1.98G, 加入到表格后数据存储为 2 份, 在 HDFS 上的文件总大小为 1.12G, 压缩率约为 0.57. 使用 Snappy 后系统的 I/O 量下降, 入库的性能也有了较大的提高.

2.3.2 写表格操作优化

本文针对日志数据的特点和实验集群环境, 对 HBase 中的有关参数做了调整, 使集群能够有更好的入库性能. 主要对以下几个参数做了调整:

(1) 自动刷写

在默认情况下 HBase 每执行一次数据 Put 操作就向 Region 服务器发送一次请求. 通过调用 HTable 的 setAutoFlush 方法将客户端的自动刷写关闭, 这样就不用每次 put 都会发送请求, 等 put 请求填满缓存后才会发起请求, 可以节省每次刷新消耗的时间.

(2) 预写日志

在 HBase 中客服端向集群提交数据时, 会首先将数据操作写入到日志文件, 当日志文件写成功后才会实际的将数据写入到 MenStore; 当日志文件写失败, 客服端就会被通知提交操作失败. 这样的设计是为了保证数据的安全性, 当出现机器故障后能够通过日志文件来恢复数据. 由于本文中使用的数据不太重要, 同时也为了追求数据的入库效率, 通过调用 put 的 setWriteToWAL 将预写日志关闭, 这样就不会写日志而是直接写入内存, 省去了写日志的时间.

(3) 缓存大小

在 HBase 中 regionserver 会为每个 region 提供一个 memstore, 当 memstore 满 64MB 以后, 才会启动 flush 将内存中的数据刷新到磁盘. 通过调用 HTable 的 setWriteBufferSize 方法来调整客户端的写缓存大小,

writeBufferSize 的大小单位是 byte, 针对实验环境以及加载的数据特点来调整缓存大小, 尽可能多的将数据加载到缓存中.

(4) 批量写

在 HBase 的 put 方法中可以将一个指定的 put 记录加载到表格中, 同时还提供了另一种 put 方式, 该方式的参数是 List<Put>, 用来批量写入多行记录. 这样做的好处是多个 RowKey 记录批量执行, 只需要一次网络 I/O 请求, 在面对数据高实时性和高网络传输的情况有明显的性能提升(优化后的加载效率见实验三) .

3 实验结果与分析

3.1 实验环境与数据

实验集群包含五个节点, 其中一个主控节点, 四个计算节点. 每个节点都配有双核 CPU, 主频 2.53GHz; 4G 内存; 500G 本地存储磁盘; 节点间通过局域网互联; 操作系统为 Red Hat Linux6.4. Hadoop 版本为 1.1.2, 包含一个 namenode 节点, 四个 datanode 节点. HBase 版本为 0.94.7, 包含一个 master 节点, 四个 region 节点.

实验数据来源于第二届中国大数据技术创新大赛中基于互联网大数据的日志类应用处理提供的数据. 数据结构下表 4.

表 4　测试数据格式

字段名	数据格式	含义和取值范围
SrcIP	数字	源 IP 地址
DestIP	数字	目的 IP 地址
SrcPort	数字	源端口
DestPort	数字	目的端口
CaptureTime	数字	截取时间
Flag	数字	标记位
Protocol	数字	协议代码
ISP	字符串	运营商
Area	字符串	地区
QueryType	数字	请求类型代码

样例数据:

7276164|16307329|2892|61106|1388506321|33704|10|VP|AO|44

8229582|6726568|7241|2615|1388507975|33705|16|VP|RC|47

19160976|12068839|36004|44646|1388508685|33707|16|LC|RG|15

测试数据量:

表 5　测试数据

数据	1.dat	2.dat	3.dat	4.dat	5.dat	6.dat	7.dat
行数	5000	50,000	500,000	1,000,000	5,000,000	10,000,000	30,000,000
大小	283kb	2.82mb	28.21mb	56.42mb	330.07mb	660.14mb	1,980.43mb

3.2 实验内容与结果分析

本节在实验数据基础上测试了 HBase 自带加载数据方法, 自定义加载数据方式以及自定义加载数据性能调优后的入库性能, 并做了对比.

实验 1: HBase 自带加载数据方式.

以下对加载数据方法一和方法二进行性能测试, 将数据集分别用这两种方法处理.

图中结果显示, 在数据量比较小的情况下方法一和方法二的效率基本持平; 当数据量在增加时, 方法二比方法一要快. 对于方法二而言, 由于其在运行过程中对集群硬件环境要求较高, 如果数据量增大到一定量, 该方法则不能够正常的运行. 在实验中, 数据集为 data6.dat 与 data7.dat 时, 在实验集群下方法二不能够运行完成. 得出, 在数据量不是很大的情况下, 用方法一进行数据加载; 在数据量很大, 同时在集群硬件环境也比较好的情况下用方法二加载会有更好的时间效率.

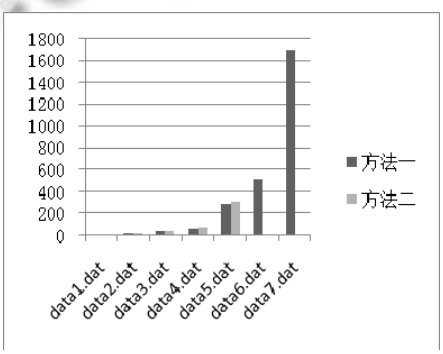

图 2　HBase 自带加载数据性能测试

实验 2: HBase 自定义加载数据方式.

以下对 Java API 方式和 MapReduce 方式加载数据在测试数据集上进行性能测试. 由于 Java API 方式是串行的加载数据, 当数据量比较大的后, 加载的时间

会特别长, 不能满足海量数据处理需求. 以下对 MapReduce 加载方式和 HBase 自带加载方式对比, 入库性能结果如图 3.

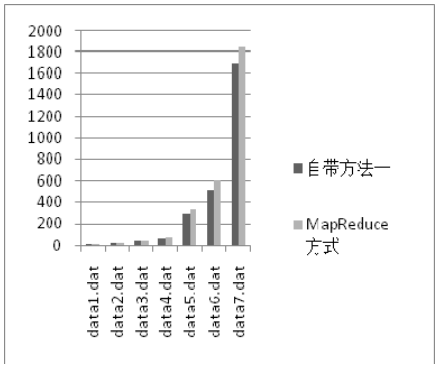

图 3 HBase 自定义加载数据性能测试

通过图中的结果可以看出, 自定义 MapReduce 方式和自带加载数据方法一相比, 在数据量小时基本持平, 当数据量增大后自定义 MapReduce 方式的效率比 HBase 自带方法的效率要低. 虽然这两种方式实质上都是通过启动 MapReduce 任务来实现数据加载, 但是因为自定义 MapReduce 在加载数据时没有考虑到数据预分区, 自动刷写, 预写日志, 缓存设置, 批量写等情况. 这样就导致数据加载的时间比自带的方式加载效率低一些.

实验 3: 自定义 MapReduce 加载数据性能优化.

以下为自定义 MapReduce 方式进行优化后和 HBase 自带加载数据方法一. 在实验数据上进行加载测试, 入库性能对比图:

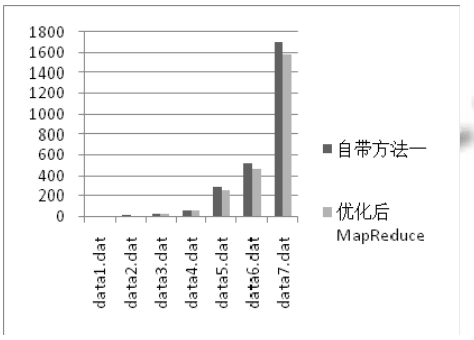

图 4 HBase 自定义加载数据优化性能测试

图中结果显示, 自定义并行加载数据 MapReduce 方式在进行相应的调优设置后, 性能与自带的方法一相比要更加优秀. 此外自定义加载方法还可以指定加载数据中特定的列, 对行健的设置可以根据需求设置更加完善, 同时对数据的处理操作空间更大. 相对于 HBase 自带的加载数据方法, 自定义 MapReduce 方式加载数据具有很大的灵活性, 同时在性能方面也具有一定的优势.

4 结语

本文通过对 HBase 底层存储方案讨论, 并针对大数据时代, 海量数据存储的要求, 对 HBase 自身提供的两种数据加载方法进行研究, 分析发现其中的优点和不足. 最终基于 HBase 提供的相关接口, 借助于 MapReduce 并行处理框架实现自定义更加完善的数据加载方法. 针对集群环境以及数据特点进行分析, 通过对算法与集群的配置来优化 MapReduce 加载数据方法的性能.

实验结果表明, 优化后的自定义加载数据方法的性能优于 HBase 自带的加载数据方法, 并且对数据的处理更加灵活, 有利于开发人员基于 HBase 进行开发.

参考文献

1 Apache Hadoop. http://hadoop.apache.org.
2 Chang F, Dean J, Ghemawat S, et al. Bigtable: A distributed storage system for structured data. ACM Trans. on Computer Systems, 2008, 26(2): 205−218.
3 Shvachko K, Kuang H, Radia S, et al. The Hadoop Distributed File System. 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST). IEEE Computer Society. 2010. 1−10.
4 Borthakur D, Gray J, Sarma JS, et al. Apache hadoop goes realtime at Facebook. ACM SIGMOD International Conference on Management of Data, SIGMOD 2011. Athens, Greece. June. 2011. 1071−1080.
5 George L. HBase: the definitive guide. Sebastopol. USA: O'Reilly Media, 2011.
6 Stonebraker M. SQL databases vs NoSQL databases. Communications of the ACM, 2010, 53(4): 10−11.
7 Ghemawat S, Gobioff H, Leung ST. The Google file system. Proc. of the 19th ACM Symp. on Operating Systems Principles. New York. ACM Press. 2003. 29−43
8 覃熊派,王会举,杜小勇,王珊.大数据分析 RDBMS 与 MapReduce 的竞争与共存.软件学报,2012,23(1):32-45.
9 Lars George 著,代志远,刘佳,蒋杰译.HBase 权威指南.北京: 人民邮电出版社, 2013,10.
10 Lam C,韩翼中译.Hadoop 实战.北京:北京人民邮电出版社,2011.
11 姚林,张永库.NnSQL 的分布式存储与扩展解决方案.计算机工程,2012,38(6):40−42.
12 刘星.HBase 性能深度分析.程序员,2011,(7):102−104.
13 田胜利,徐锡山,杨树强,华中杰.针对 HBase 的 MapReduce 访问接口的优化.第九届中国通信学会学术年会论文集.2012.
14 王培建.云计算环境下大规模数据存储技术研究[学位论文].南京:南京邮电大学,2013.
15 刘鹏.云计算.第二版.北京:电子工业出版社,2011.