



一种基于改进的链式 MapReduce 的并行 ETL 应用^{*}

吴 斌, 刘心光

(北京邮电大学计算机学院通信软件工程中心 北京 100876)

摘 要: 介绍了并行 ETL 的相关工作和常见的处理多 MapReduce 作业流程的方法; 提出一种改进的链式 MapReduce 框架, 并将此框架应用于一个并行 ETL 工具, 同时提出一些针对 ETL 处理的流程级优化规则, 使 ETL 流程产生更少的 MapReduce 作业, 从而减少 I/O 以及网络传输的消耗; 利用某省份手机上网数据与 Hive 进行了大数据对比实验, 结果表明, 本 ETL 工具的性能平均比 Hive 快 10%~20%。

关键词: 改进的链式 MapReduce; ETL; 优化规则

doi: 10.3969/j.issn.1000-0801.2013.12.001

A Parallel ETL Tool Based on an Improved Chain-MapReduce Framework

Wu Bin, Liu Xinguang

(Telecommunication and Software Engineering Center, School of Computer Science,
Beijing University of Posts and Telecommunications, Beijing 100876, China)

Abstract: The related work in parallel ETL and common methods to deal with multiple MapReduce jobs were introduced. Then an improved chain-MapReduce framework was presented, based on this framework, a parallel ETL tool was designed. Several optimization rules on ETL which will make the ETL process generate less MapReduce jobs to avoid unnecessary I/O and network cost were presented. The ETL tool on real queries and real big datasets were evaluated. Compared with Hive, the tool reduces time on average by 10% to 20%.

Key words: improved chain-MapReduce, ETL, optimization rule

1 引言

大数据时代的信息爆炸,使得对大数据的处理变得非常重要。各个行业尤其是互联网行业,每天都会产生 TB 级的服务数据,如何对海量数据进行 ETL 处理有很重要的意义。MapReduce^[1]编程模型的提出,为大数据处理提供了新的思路,由于良好的易用性和可扩展性,得到了工业界和学术界的广泛支持和应用,基于 MapReduce 框架的 ETL 应用也越来越多。一个 ETL 处理流程,如果仅仅是顺序执

行而不考虑每一个处理操作间的关系,即便应用 MapReduce 编程模型,流程的整体性能同样也会受到影响。因此对于作业流程的合理拆分、整合,对于流程的处理性能有着很大的影响。国外已经出现了一些为了简化 MapReduce 程序实现细节,基于 MapReduce 处理大数据的查询语言的开源项目,这些项目对作业流程实现了各自的优化。其中包括基于传统 SQL 的 Hive^[2]、Cheetah^[3]、Dremel^[4]、Impala^[5]等;一些 ETL 流程可以用脚本表示成简单的操作流,远比使用 SQL 查询语句要简单、灵活,相对应地出现了

^{*} 国家自然科学基金资助项目(No.61074128)



针对半结构化的 ETL 工具 Pig^[6]和针对处理 JSON 格式的 Jaql^[7]等;FlumeJava^[8]是一个建立在 MapReduce 之上的 Java 库,适合使用在由多个 MapReduce 作业拼接在一起的复杂计算场景,Flumejava 首先优化执行计划,然后基于底层的原语来执行优化后的逻辑操作,与它类似的还有 Cascading^[9]。HortonWorks 开发的 Tez^[10]将多个有着依赖关系的 MapReduce 作业抽象成一个 DAG(有向图),将 map task 和 reduce task 进一步拆分成更细粒度的元操作,这些拆分后的元操作可以任意灵活组合,产生新的操作单元,这些可控制操作单元组合起来形成一个大的 DAG 作业。基于这个 DAG 对多个 MapReduce 进行合并,减少作业的数量以提高整体性能,但是目前 Tez 的发展还不成熟。上述的这些开源项目都有结合 ETL 处理流程的场景设计的相应优化规则,目的是尽量减少 ETL 流程产生的 MapReduce 作业的数量,避免不必要的 I/O 及网络传输的损耗。这些工具对底层 MapReduce 进行了封装,屏蔽了 MapReduce 作业的实现细节,用户不需要编写复杂的 MapReduce 作业,只需要使用较简单方便的 API 就可实现相应的功能,但是这些工具大多缺少前台使用界面,在用户的友好性上还有待提高。MapReduce 框架广泛应用于大数据的处理,但是在处理长流程的时候它也存在不足之处。本文在上述问题的基础上提出了一种改进的链式 MapReduce 框架,并将这一框架应用于一个基于拖拽组件的 ETL 处理工具上。同时针对 ETL 场景设计优化规则减少 ETL 长流程处理的总数据量,从而提高流程的整体性能。各个行业每天都会产生海量数据,通过数据挖掘算法对这些数据进行分析,挖掘出数据中的潜在信息,推动着企业的发展。而在数据挖掘前需要使用 ETL 算法将数据进行清洗,转换成适合进行数据挖掘的数据,可以说 ETL 是数据挖掘的前提,因此提升并行 ETL 流程的处理性能有很大的现实意义。本文对优化 ETL 流程处理性能上作出了以下贡献:结合链式 MapReduce 原理及其不足,重新设计了一个新的链式 MapReduce 的框架,并将其应用于一个基于拖拽组件的 ETL 处理工具上;在新的链式 MapReduce 框架的基础上,为减少链式作业流程处理的总数据量,结合 ETL 场景以及 MapReduce 的特点提出了几个优化规则;并对 group by 和 join 算法做了针对性的优化。

2 复杂流程级 MapReduce 作业解决方法

一些复杂的任务难以用一个 MapReduce 作业处理完

成,需要多个 MapReduce 作业才能完成;针对 Hadoop 中这一场景,提供了 3 种解决方案:迭代式 MapReduce、依赖关系组合式 MapReduce、链式 MapReduce。

2.1 迭代式 MapReduce

Pagerank、K-means 等算法都需要多次迭代,由多个 MapReduce 作业才能完成算法。在 map/reduce 迭代过程中,采用 while 循环来管理迭代,直到某一判断条件满足后,结束迭代;迭代过程中前一个 MapReduce 的输出结果,作为下一个 MapReduce 的输入。迭代式的 MapReduce 有一个很大的缺点:每次迭代,所有 job(task)重复创建,代价将非常高;每次迭代都将数据写入本地和从本地读取,I/O 和网络传输的代价比较大。针对迭代式计算的特点,目前已经有一些开源项目在这方面做出了改进,修改传统 MapReduce 使之适应迭代计算,比如 Haloop^[11]和 Twister^[12]。

迭代计算的特点:输入数据可分为两类,即 static 和 variable 数据,而且大部分情况下 static 数据比 variable 数据量大很多,static 与 variable 数据完成计算之后,得到新的 variable 数据,然后新数据与 static 数据再次进行计算,如此循环,直到迭代结束条件得到满足之后结束任务。

针对其特点,Haloop、Twister 对 MapReduce 框架进行了优化:修改原有的 job 结构,使之能运行多个 map-reduce 对,这样就不用启动多个 job(迭代计算的循环体在 job 内发生,即 job 复用),job 复用可以大大提高迭代效率;map/reduce 静态数据缓存(常驻内存或者缓存进磁盘),让 map 任务可以快速进入计算。

但是 Haloop、Twister 模型抽象还不够高,支持计算模型有限,迭代式 MapReduce 还在发展当中。

2.2 依赖关系组合式 MapReduce

例如一个任务需要 3 个 MapReduce 作业:job1、job2、job3,其中 job1 和 job2 相互独立,job3 要在 job1 和 job2 完成之后才能执行。这种类型的作业就叫复杂数据依赖关系的组合式 MapReduce。Hadoop 为这种组合关系提供了一种执行和控制机制,在 job 初始化的时候对作业间的依赖关系进行配置,这种方式主要是对作业流程级别的控制。

传统 MapReduce 框架把一个作业分为两个阶段:map 阶段和 reduce 阶段,在 map 阶段,每个 map task 读取一个块数据(默认配置),并调用 map() 函数进行处理,然后将结果写到本地磁盘上;在 reduce 阶段,每个 reduce task 远程地从各个 map task 所在的节点上读取 reduce 端需要的数据,并调用 reduce() 函数进行数据处理,并将最终结果

写到 HDFS 上。在一个 MapReduce 作业中要读写磁盘两次,这虽然可以提高可靠性,但降低了系统性能,也正是这个原因,传统的 MapReduce 框架不太适合处理多个连续作业以及迭代式作业,这种类型的作业应用于 MapReduce 框架下会反复地进行磁盘的读写,作业的整体性能较差。Hadoop 在处理数据时会尽量保证数据的本地性,即数据存储在哪一个节点上,就由这个节点进行这部分的数据计算,这样可以减少数据在网络上的传输,降低对网络带宽的需求和加快处理速度;“本地计算”是最有效的一种节约网络带宽和缩短处理时间的手段。默认配置下每个 map task 默认配置下读取一个块,每个块对应的数据与 map task 在同一个节点下;如果修改默认配置使得每个 map task 可以读取多个块,这些块可以在集群的任意一个节点上,此时就需要将不在 map task 上的节点的数据复制到此节点,这会影 响作业的性能。reduce 端从其他节点复制需要用到数据所产生的网络传输是不可避免的。

当需要处理多个连续操作时,希望通过减少生成 MapReduce 作业的数量,进而减少读写磁盘的次数,以提升作业流程的整体性能;Hadoop 提供的链式 MapReduce 框架以及随后提出的改进的链式 MapReduce 框架都是基于这个目的而设计的。

2.3 链式 MapReduce

链式 MapReduce 就是在核心的 MapReduce 之外,增加一个辅助的 map 过程,然后将这个辅助的 map 过程和核心的 MapReduce 过程合并为一个链式的 MapReduce,从而完成整个作业。Hadoop 提供了专门的 ChainMapper 和 ChainReducer 来处理链式任务,ChainMapper 允许一个 map 任务中添加多个 map 的子任务,ChainReducer 可以在 reducer 执行之后,再加入多个 map 的子任务。前一个 map 的输出作为后一个 map 的输入,最后一个 map 输出作为 reduce 的输入。最终的结果由 reduce 输出到 HDFS 上。但是在链式 MapReduce 中,在 mapper 中做的逻辑操作还是顺序执行的,没有考虑到各个作业间的关系,对性能还是有一定影响的。

对于一个复杂的 ETL 处理流程,它需要由多个 MapReduce 作业完成其业务逻辑。在 ETL 的场景中迭代式的流程比较少,ETL 流程中很多流程符合链式 MapReduce 的场景。本文在链式 MapReduce 的基础上,提出了一种改进的链式 MapReduce 框架,并将这一框架应用于一个基于拖拽组件的 ETL 处理工具上。第 3 节将对这一新框架的设

计进行介绍。

3 新链式 MapReduce

新的链式 MapReduce 框架将原有的 mapper、reducer 抽象成一个逻辑操作(operator,OP),每一个 OP 需要实现 3 个主要的方法。

- init:在 Driver 类中调用,对 MapReduce 作业中需要用到的变量进行赋值等操作,相当于原 MapReduce 作业中的 configure 方法。
- process/reduceprocess:用于实现操作的业务逻辑,相当于实现普通 MapReduce 函数中 map、reduce 函数中的业务逻辑。
- forward:将该 OP 的处理结果传递到下一个 OP。

新链式中定义一个 Driver 类来管理 MapReduce 作业参数的配置,记录需要分配给 map、reduce 阶段的 OP,并设置 OP 间的父子关系;定义 DMap 类和 DReduce 类分别管理作业的 map 部分和 reduce 部分。此外在这个框架中还设计了 3 个特殊的 OP,MapSink 负责反序列化数据以及记录 map 阶段逻辑操作,并作为 map 阶段的开始;ReduceSink 负责输出 map 阶段的处理结果;FileSink 负责将 reduce 的处理结果写到 HDFS 上。

图 1 描述了一个基于上述新链式 MapReduce 架构流程的执行过程,具体介绍如下。

首先通过 Driver 类设置需要在一个作业中完成的多个 OP,将其传递给 map、reduce 端,并设置好 OP 间的父子关系。在 DMap 的 map 方法中对每一条数据进行处理,首先将执行 MapSink 的 process 方法,MapSink 记录了 map 端的第一个 OP,通过其 forward 方法将反序列化后的数据传递到第一个 OP,调用其 process 方法对数据进行处理,并将处理结果通过其 forward 方法传递到下一个 OP 并调用 process 方法,反复进行上述的流程直到遇到 ReduceSink,调用其 process 方法将处理结果输出到 reduce 端。在 reduce 端只能有一个 reduce 类的 OP,在它的后面可以有多个类似 mapper 的 OP。在 DReduce 的 reduce 方法完成 reduce 端的操作,首先调用其 reduce 类 OP 的 process 方法对 map 端传递过来的数据进行处理,通过 forward 方法将处理结果传递到下一个 OP,反复进行上述的流程直到遇到 FileSink,调用其 process 方法将处理结果输出到 HDFS 上。

新链式 MapReduce 中,数据的序列化、反序列化以及字段类型转换的过程与链式有很大不同。首先在 MapSink

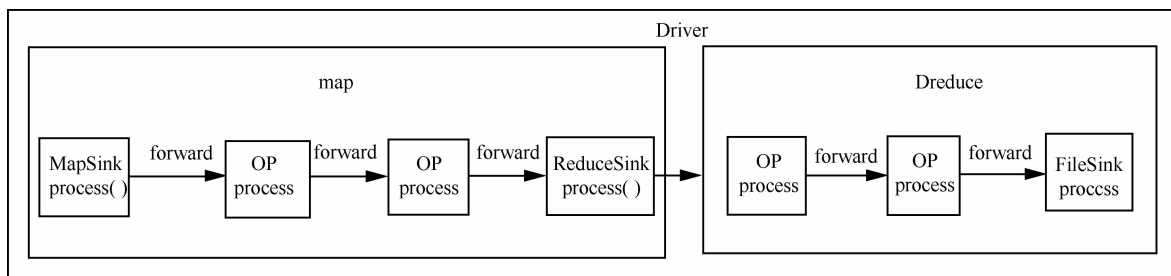


图1 新链式 MapReduce 架构

中对输入数据进行反序列化,将 Writeable 类型的数据转换成 ROW 对象,该对象保存着 byte 类型的数据。之后在 OP 之间传输数据都是使用 ROW 对象进行传递,减少了 OP 之间不必要的序列化、反序列化过程。由于在每个 OP 的 init 方法中会记录用到输入输出数据的数据类型以及 OP 要用到的参数。当在每个 OP 中调用 process 时,只有当真正用到某一个字段的时候,根据记录的该字段的数据类型,才会将这个字段的 byte 数据转换成相应的类型(数据的每个字段是由分隔符分割的)。在新的链式 MapReduce 框架中,OP 间传递数据使用的是 object 的 ROW 对象,数据类型细节封装在底层,这样设计不用像链式 MapReduce 中必须保证每一个 mapper 的输入 key、value 的数据类型要与其上一个 mapper 的输出 key、value 的数据类型保持一致,旧版链式 MapReduce 的这种方式对输入输出的设计有很大的限制。

新链式 MapReduce 和普通链式 MapReduce 相比,减少了逻辑操作(mapper)间的序列化与反序列化过程,整个作业只需要一次序列化与反序列化的过程;同时新框架不需要保证上一个逻辑操作的输出数据类型是下一个操作的输入数据类型,新框架统一采用 ROW 对象传递数据,封装了底层数据类型的转换,使得代码设计更加简单。

4 优化规则及 ETL 算法优化

4.1 MapReduce 流程级优化规则

抛开 MapReduce 程序算法实现的好坏,MapReduce 作业间的 I/O 和网络传输的消耗是影响 MapReduce 作业性能的主要瓶颈。为了提供细粒度的容错,每个作业的结果都会写回到分布式文件系统(HDFS)上,每一个作业会读取上一个作业的输出进行处理,这里的 I/O 消耗很大,输出结果默认也存在多个副本来保证它的容错性。不论是链式还是新链式 MapReduce,都是为了减少 I/O 和网络传输的代价而设计的。它们将多个逻辑操作在一个 map/reduce

函数中处理,但是如果仅仅是顺序执行这些逻辑操作而不考虑这些逻辑操作的特点,还是不能使作业的性能达到最佳。因此一个合理的执行计划,减少作业的数目可以避免不必要的中间结果的输出,从而提高流程的整体性能。下文结合 ETL 的具体应用,设计了一些针对 ETL 应用的流程级优化规则。

新链式 MapReduce 框架的设计目的就是减少 I/O 和网络传输的代价。此框架中,在 map/reduce 阶段实现了多个逻辑操作,每个逻辑操作的处理结果,传递到下一个逻辑操作。设计流程级的优化规则,调整逻辑操作的顺序,主要目的是要减少上一个逻辑操作向下一个逻辑操作传递的数据量,因此尽量将可以明显减少数据量的操作提前,例如过滤操作。以下总结了 4 个基于 map、reduce 结构的优化规则,主要目的是减少逻辑操作间传递的数据量。

(1) 优化规则 A

如图 2 所示,两个都可以在 map 端完成的作业通过规则合并成一个 MapReduce 作业,在 map 端完成两个作业的内容。



图2 优化规则 A

(2) 优化规则 B

如图 3 所示,前一个作业是一个包含 map 和 reduce 的作业,后面一个是只要有 map 就可以完成的作业。通过规则合并成一个 MapReduce 作业,在 map 端完成第一个作业 map 端的内容,reduce 端完成第一个作业 reduce 端的内容和第二个作业 map 端的内容。

其中优化规则 A、B 也就是链式 MapReduce 的优化规则。

(3) 优化规则 C

如图 4 所示,前一个作业是一个包含 map 和 reduce 的作业,后面一个是只要有 map 就可以完成的作业,通过

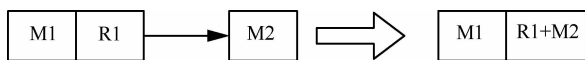


图3 优化规则 B

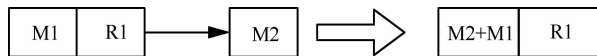


图4 优化规则 C

规则合并成一个 MapReduce 作业,在 map 端完成第二个作业和第一个作业 map 端的内容,reduce 端完成第一个作业 reduce 端的内容。例如:包含 map 和 reduce 作业(M1,R1)+过滤(M2),主要目的就是减少逻辑操作间数据的传递。

(4) 优化规则 D

如图 5 所示,前一个作业是只要有 map 就可以完成的作业,后面一个是包含 map 和 reduce 的作业。通过规则合并成一个 MapReduce 作业,在 map 端完成第二个作业 map 端的内容,reduce 端完成第一个作业 map 端的内容和第二个作业 reduce 端的内容。例如:计算生成列(通过表达式生成一个新的字段)+包含 map 和 reduce 作业的操作,如果第二个作业的 map 端没有用到计算生成列生成的数据就可以应用这一规则,目的也是为了减少向下传递的数据量。

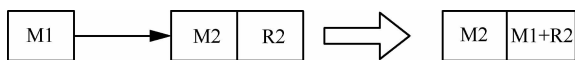


图5 优化规则 D

4.2 ETL 算法优化

第 4.1 节设计的优化规则是针对 MapReduce 架构而设计的,这些规则只要满足条件则可以同时使用。在 ETL 处理的场景中 group by 和 join 是常用的操作,在很多场景中 group by 和 join 同时出现,针对这两个操作,也设计了一些优化策略。

4.2.1 join 与 group by 组合场景

两张表先进行 join 之后再行 group by 操作和一张表先进行 group by 再和另一张表进行 join,计算时所用到的数据量是不一样的,当上述两种操作所得到的结果一样时,后一种操作会更优。例如:有一张大表 A(a1,a2,a3,a4,a5)和它的维度表 B(b1,b2),A、B 表通过 a1 和 b1 进行 left join 后,处理结果相当于在 A 表后面加了几个维度信息。如果对 join 的结果进行 group by 操作,当 group by 的 key 为 a1、a2、a3 时同 group by 的 key 为 a1、a2、a3、b2 时的结果是一样的,也就是说 a1、a2、a3 的值一样时,b2 的值就确定了。当遇到这种情况时,先对 A 表进行 group by 再和 B 表进行 join 操作就会对性能有很大提升。主要是由于 group

by 的操作减少了数据量。与第 4.1 节设计的基于 MapReduce 结构的优化规则的目的一样,是减少逻辑操作间的数据传递量。这是针对一些特殊场景才可以应用的优化规则。对于一般场景,group by 和 join 算法的好坏对性能起着决定性的作用。

4.2.2 join 算法设计

设计 join 算法时利用了 MapReduce 的二次排序。在 Hadoop 中,默认情况下是按照 key 值进行排序,即在 reduce 中对于相同的 key,其对应的 value 是无序的。但是在 join 中,往往希望相同的 key 中小表对应的 value 排在前面,这样就可以将它们放入内存中操作。因此设计 key 时将原有 key 后面加上一个的 tag (标识是哪一个表的数据),这样通过原 key 值进行分割,key 值相同的数据就分配到同一个 reduce 中,在 reduce 中 key 值相同的数据根据 tag 进行降序排列(大表的 tag 为 0),只要实现相应的 Partitioner 和 GroupingComparator 就可以实现二次排序。在 join 的 reduce 阶段,将多个小表的数据放到内存中,大表的数据再和这些小表进行 join 操作就可以很高效地完成。以下是 join 算法在 MapReduce 中的设计。

(1) map 阶段

输入为需要 join 的表;输出为(key,value)对:(joinKey,tag joinValue,tag)。

(2) reduce 阶段

while(tag!=0)将小表数据放入内存中(利用二次排序,小表数据排在大表前面);

while (value.hasNext())将大表与小表进行关联并输出

4.2.3 group by 算法设计

设计 group by 算法时采用了基于 map 端的 group by,即利用内存首先在 map 端计算部分聚集操作,最后再在 reduce 端进行汇总,完成最终的聚集操作。相比于仅在 reduce 端进行聚集操作的 group by 算法,本方法更充分地利用系统的内存资源,并且减少了 reduce 端发生数据倾斜的可能,算法更优。以下是 group by 算法在 MapReduce 中的设计。

(1) map 阶段

读入一条数据,将它放入 hashmap 中,其中 key 为进行汇总的字段,value 为需要聚集的字段(即将需要处理的数据放入到内存中),每读入 1 000 条数据对内存进行检查;

if(内存使用>设置的内存使用上限),将 hashmap 中的部分数据(10%~20%)进行聚集计算并输出;



else 继续将数据放入内存中;

当所有数据都已经读入,最终将 hashmap 中剩余数据进行聚集计算。

(2)reduce 阶段

对相同 key 值的数据进行聚集计算(sum,count,max 等)。

当 group by 操作并没有明显减少记录条数时就不适合使用 map 端的 group by,这时需要在 map 阶段判断是否需要 map 端的 group by,如果不需要,直接将数据输出到 reduce 阶段,在 reduce 阶段完成 group by 操作。

5 测试实验

本文将新的链式 MapReduce 框架应用于一个基于拖拽组件的 ETL 工具上,前台提供算法参数配置,方便用户使用,点击提交执行,在生成 MapReduce 作业之前,根据 ETL 流程特点以及优化规则进行流程级优化,最终生成 MapReduce 作业。实验选定 4 个对手机上网数据进行统计分析的 ETL 流程作为测试用例,对 ETL 工具与 Hive 0.11.0 进行了对比测试。

5.1 测试环境

本次测试使用的硬件环境中共有 10 台服务器均部署了 Hadoop,版本为 1.0.3。其中 1 台为 NameNode,9 台作为 DataNode。测试使用的 Hive 版本为 0.11.0。

单台服务器的具体配置为:硬盘 3 TB,内存 48 GB,16 核 CPU[Intel(R) Xeon(R) CPU E5530 @ 2.40 GHz]。

5.2 测试用例

本次测试所用数据为某省份手机上网日志数据,分为 2 个表。大表为用户上网记录表,有 2 610 636 757 条数据,容量为 151 GB,主要包含的字段有:TIME_ID(时间)、MSISDN(移动标识号码,唯一标识用户)、SP_DOMAIN(网址域名)、USER_AGENT(手机终端类型)、UPSTREAM_VOL(上行流量)、DOWNSTREAM_VOL(下行流量)、URL_CNT(访问次数)。小表为地市维表,有 150 662 916 条数据,容量为 2 GB,主要包含的字段有:USR_NBR(移动标识号码)、CITY_CD(地市编码)。设计了如下 4 个测试用例(分别是 Hive 脚本以及基于新链式框架的并行 ETL 组件流)。

(1)case 1:单表汇总 A

· 流程目的:根据手机终端类别汇总出总流量及访问网页次数。

· Hive 脚本:INSERT OVERWRITE TABLE TEST_1

```
SELECT USER_AGENT, SUM (UPSTREAM_VOL +
DOWNSTREAM_VOL) AS TOT_FLU, SUM
(URL_CNT) AS URL_CNT FROM WAPURL GROUP
BY USER_AGENT。
```

· 新链式 ETL 工具组件流:case 1 流程如图 6 所示。

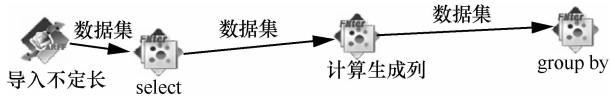


图 6 case 1 流程

(2)case 2:单表汇总 B

· 流程目的:根据用户和终端类别汇总出时间 12:00~16:00 的总流量及访问网页次数。

· Hive 脚本:INSERT OVERWRITE TABLE TEST_2
SELECT MSISDN, USER_AGENT, SUM (UPSTREAM_VOL + DOWNSTREAM_VOL) AS TOT_FLUX, SUM (URL_CNT) AS URL_CNT FROM WAPURL WHERE SUBSTR (TIME_ID, 9, 2) IN (12, 13, 14, 15, 16) GROUP BY MSISDN, USER_AGENT。

· 新链式 ETL 工具组件流:case 2 流程如图 7 所示。

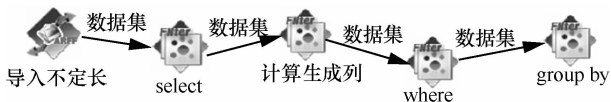


图 7 case 2 流程

(3)case 3:关联汇总 A

· 流程目的:根据用户和终端类别及日期汇总出总流量及访问网页次数,并同地市维表进行关联。

· Hive 脚本:INSERT OVERWRITE TABLE TEST_3
SELECT W.RPT_DT, W.MSISDN, W.USER_AGENT, W.URL_CNT, W.TOT_FLUX, C.CITY_CD FROM (SELECT SUBSTR (TIME_ID, 1, 8) AS RPT_DT, MSISDN, USER_AGENT, SUM (URL_CNT) AS URL_CNT, SUM (UPSTREAM_VOL + DOWNSTREAM_VOL) AS TOT_FLUX FROM WAPURL GROUP BY SUBSTR (TIME_ID, 1, 8), USER_AGENT, MSISDN) W LEFT OUTER JOIN CITY C ON W.MSISDN = C.USR_NBR。

· 新链式 ETL 工具组件流:case 3 流程如图 8 所示。

(4)case 4:关联汇总 B

· 流程目的:根据用户和网址域名汇总出总流量及访问网页次数,并同地市维表进行关联。

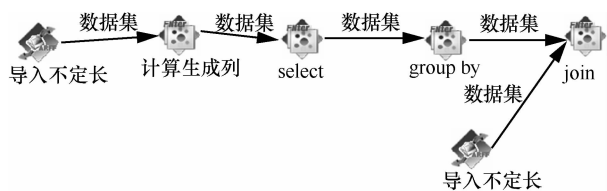


图 8 case 3 流程

- Hive 脚本 :INSERT OVERWRITE TABLE TEST_4 SELECT W.MSISDN, W.SP_DOMAIN, C.CITY_CD, W.URL_CNT, W.TOT_FLUX FROM (SELECT MSISDN, SP_DOMAIN, SUM (URL_CNT) AS URL_CNT, SUM (UPSTREAM_VOL + DOWNSTREAM_VOL) AS TOT_FLUX FROM WAPURL GROUP BY MSISDN, SP_DOMAIN) W LEFT OUTER JOIN CITY C ON W.MSISDN = C.USR_NBR。
- 新链式 ETL 工具组件流 :case 4 流程如图 9 所示。

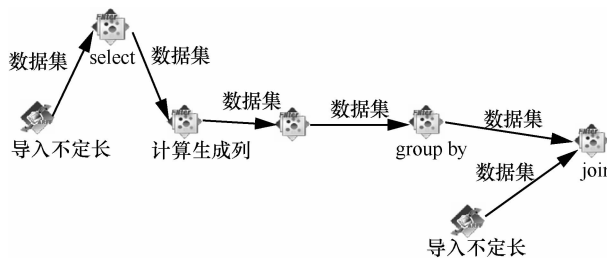


图 9 case 4 流程

5.3 测试结果及分析

图 10 为测试结果时间对比,测试结果见表 1。

由于 ETL 的场景较多,本文没有把所有场景一一列出,只挑选了 4 个比较典型的测试用例。从测试结果可以看出,基于新链式 MapReduce 框架的 ETL 工具在大数据处理的性能上比 Hive 平均快 10%~20%,最好情况会达到 30%,所选用的测试用例在跟 Hive 的对比测试中优势明显。在 4 个流程中,由于是基于新链式 MapReduce 框架实现的 ETL 工具,多个操作合并到一个作业中执行,缩短了流程执行的时间。在 case 3 和 case 4 中,启动了两个 MapReduce 作业,第二个作业是 join 操作,join 在 ETL 工具中是利用 MapReduce 的二次排序实现的,在 case 3 中缩短

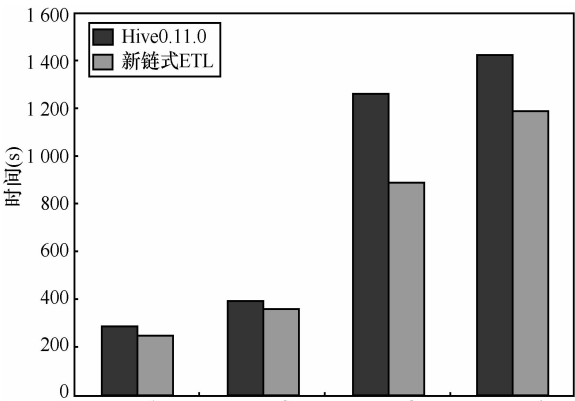


图 10 测试结果时间对比

了一倍的时间,性能提升明显;交换 join 和 group by 的顺序可提升性能 15%左右(case 3 为交换后的流程)。在 case 2 中应用优化规则 C,where 组件(过滤操作)处在靠后的位置,根据优化规则,应当尽可能地将减少数据量明显的逻辑操作前提,以提高处理性能,根据流程特点最终将其调整为第一个逻辑操作,减少了之后逻辑操作处理的数据量。在之前的实验中,map 端的 group by 操作比有 combiner 的 group by 快 30%左右,没有 combiner 的 group by 容易发生数据倾斜,导致作业性能很差,基于 map 端的 group by 就很好地避免这一情况。在 4 个 case 中都涉及 group by 操作,在实验结果中显示出不错的性能。本 ETL 工具基于拖拽组件的形式,使流程更加清晰,提供配置面板进行的参数配置,方便了用户的使用;同时工具底层基于新链式 MapReduce 框架,保证了工具的处理性能。通过性能对比试验,表明基于新链式 MapReduce 的 ETL 工具有较高的处理性能,链式 MapReduce 框架结合优化规则对于 ETL 流程处理性能的提高有明显作用。

6 结束语

本文首先介绍了并行 ETL 的相关开源项目以及常见的处理多个 MapReduce 作业的方法;随后提出了一种改进的链式 MapReduce 框架并设计了一些针对 ETL 处理的流程级的优化规则,将这一框架以及优化规则应用于一个并

表 1 测试结果

流程名	结果条数	Hive 0.11.0	新链式 ETL	性能提升(减少时间百分比)
case 1	120 079	4 min 49 s	4 min 16 s	11%
case 2	47 533 740	6 min 39 s	6 min 6 s	8%
case 3	170 440 935	21 min 7 s(14 min 13 s+6 min 54 s)	14 min 52 s(11 min 25 s+3 min 27 s)	30%
case 4	886 981 116	23 min 50 s(15 min 11 s+8 min 39 s)	19 min 54 s(13 min 13 s+6 min 41 s)	17%



行 ETL 工具,通过这些优化使 ETL 流程产生更少的 MapReduce 作业,减少流程处理的总数据量,从而减少 I/O 以及网络传输的消耗;最后利用某省份手机上网数据,对 ETL 工具和 Hive 做了大数据的对比实验,实验结果表明,基于新链式 MapReduce 的 ETL 工具的性能平均比 Hive 快 10%~20%,优化方案的优化效果明显。下一步工作首先会丰富优化规则,之后会对 MapReduce 进行更细维度的研究,如 shuffle、sort 过程,减少 MapReduce 作业中无用过程的调用。

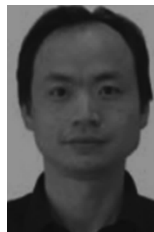
参考文献

- 1 Dean J, Ghemawat S. MapReduce:simplified data processing on large clusters. Proceedings of the 6th Conference on Symposium on Operating Systems Design & Implementation, San Francisco, USA, 2004
- 2 Apache. Hive. <http://hive.apache.org/>, 2013
- 3 Chen S T. Cheetah:a high performance, custom data warehouse on top of MapReduce. Proceedings of the VLDB Endowment, 2010,3(1):1459~1468
- 4 Melnik S, Gubarev A, Long J J, *et al.* Dremel: interactive analysis of web-scale datasets. Communications of the ACM, 2011,54(6):114~123
- 5 Cloudera. Impala. <https://ccp.cloudera.com/display/IMPALA10/BETADOC>, 2013
- 6 Olston C, Reed B, Srivastava U, *et al.* Pig Latin: a not-so-foreign language for data processing. Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data,

2008:1099~1110

- 7 IBM. JAQL. <http://www-01.ibm.com/software/data/infosphere/hadoop/jaql/>, 2013
- 8 Chambers C, Raniwala A, Perry F, *et al.* FlumeJava: easy, efficient data-parallel pipelines.ACM SIGPLAN Notices, 2010,45(6): 363~375
- 9 Cascading. <http://www.cascading.org/>, 2012
- 10 Horton Works. Tez. <http://hortonworks.com/blog/introducing-tez-faster-hadoop-processing/>, 2013
- 11 Bu Y Y, Howe B, Balazinska M, *et al.* HaLoop: efficient iterative data processing on large clusters.Proceedings of the VLDB Endowment, 2010, 3(1/2): 285~296
- 12 Ekanayake J, Li H, Zhang B J, *et al.* Twister: a runtime for iterative MapReduce. Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing, Chicago, Illinois, 2010:810~818

[作者简介]



吴斌,男,北京邮电大学教授、博士生导师,主要研究方向为数据挖掘、云计算。

刘心光,男,北京邮电大学硕士研究生,主要研究方向为数据库、数据挖掘、复杂网络与复杂系统。

(收稿日期:2013-09-11)

· 简讯 ·

2013(第三届)中国未来网络 发展与创新论坛开幕

2013年12月17日,2013(第三届)中国未来网络发展与创新论坛在中国(南京)未来网络谷隆重举行,本次论坛由中国工程院、南京市人民政府联合主办,江苏省未来网络创新研究院承办,国际著名专家及各互联网龙头企业、高校、科研院所 300 多位代表出席论坛。本次论坛围绕“未来网络试验设施及创新应用”主题,针对当前网络面临的挑战与对策、未来网络的技术发展趋势以及未来网络国家重大基础设施的建设与试验应用前景等重大议题展开深入讨论。

2011年,在江苏省、南京市的大力支持下,中国科学院计算所、北京邮电大学、清华大学 3 支团队落户南京,成立江苏省未来网络创新研究院(以下简称研究院),开展未来网络体系架构、关键技术等研发。目前,研究院研究团队已由 3 个扩展到 45 个,并已研发出可编程虚拟路由器平台、网络感知测量系统、智能网络资源控制调度分发系统、试验网综合管控系统等未来网络关键设备。基于上述成果,研究院于 2013 年 8 月 8 日开通了我国首个可编程未来网络小规模试验设施,该设施覆盖南京、北京、西安、重庆 4 个城市 7 个节点,网络管控中心设在中国(南京)未来网络谷,并通过联邦 SFA 机制与美国 GENI、欧盟 OneLab 互联,实现了跨实验床的资源调配与大规模实验部署。