基于 Flink 框架的 TopN 堆排序优化算法

关　沫,魏碧晴
(沈阳工业大学 信息科学与工程学院,辽宁 沈阳 110870)

摘　要:为了解决大数据 TopN 排序问题,将传统的堆排序进行优化,阐述了优化后的 HeapOptimize 方法的处理过程。HeapOptimize 方法基于 Flink 框架来完成 TopN 作业,可以实时地接收并处理大量的数据,根据单位时间需要处理的数据数量来调整算子的并行度,增加 Flink 框架的吞吐量,提高处理数据的速度。通过实验测量的数据结果佐证了 HeapOptimize 方法的优势。

关键词:大数据;TopN;Flink;吞吐量
中图分类号:TP311.13　　**文献标识码**:A　　DOI:10.19358/j.issn.2096-5133.2020.02.005
引用格式:关沫,魏碧晴.基于 Flink 框架的 TopN 堆排序优化算法[J].信息技术与网络安全,2020,39(2):23-26.

Flink-based heap ranking optimization algorithm for TopN problem

Guan Mo,Wei Biqing
(School of Information Science and Engineering,Shenyang University of Technology,Shenyang 110870,China)

Abstract:In order to solve the problem of TopN sorting for big data,the traditional heap sorting is optimized. The optimized method is named HeapOptimize,and the calculation process of HeapOptimize method is described. This method is based on Flink framework to complete TopN operations. It can receive and process large amounts of data in real time. It can adjust the parallelism of operators according to the number of data to be processed per unit time,increase the throughput of Flink framework,and improve the speed of data processing. The advantage of HeapOptimize method is confirmed by the data results of experimental measurement.

Key words: big data;TopN;Flink;throughput

0　引言

随着计算机技术和信息科技的快速发展,全球的数据量急剧增长,2015 年全球的数据总量达到 8.61 ZB,预估 2020 年全球的数据总量会超过 40 ZB。通过移动互联网、社交媒体等服务模式,大数据产业已渗透到人们生活的各个方面,并且数据价值的时效性越来越重要,集群必须以毫秒级的延迟从大规模的数据中提炼有价值的信息[1]。

TopN 问题就是从许多的数值选出前 N 个最大或者最小的数值有序排好,最常见的应用于微博热搜榜、歌曲人气榜、投票选举等。由此可见利用大数据技术和计算机技术能轻松解决传统排序问题。如微博热搜榜,需要实时更新点击量并按其从大到小的顺序排列。而使用流计算框架 Flink 来解决 TopN 问题可以满足其实时性和低延迟的要求。

1　Flink 框架

Flink 是一个开源的分布式流式处理框架,它针对数据流的分布式计算提供数据分布、数据通信和容错机制等功能[2]。

1.1　Flink 框架的优点

与 Storm 框架相比,Flink 能提供 Exactly-Once 精确性保证、吞吐量高、有背压处理机制[3]。与 Spark 框架相比,Flink 的延迟更短,是毫秒级别的,在流处理方面,Flink 的性能比 Spark 更好。

此外 Flink 框架使用 Chandy-Lamport 算法来做异步分布式快照,其本质是 checkpoint。checkpoint 操作是异步的,不会打断数据的处理,并且是非常轻量的,容错的代价相对很低。Flink 可以做到每条数据同步更新 watermark,watermark 的计算实时性高,输出延迟低[4]。

1.2　Flink 框架的缺点

由于 Flink 是以固定的缓存块为单位来进行的数据传输,缓存块的超时值可以人为规定,如果缓存块的超时值为无限大,则 Flink 的数据传输方式

就是批处理,此时系统可以获得最大吞吐量[5]。如果缓存块的超时值为0,Flink的数据传输方式是流处理,此时虽然系统可以获得最低的处理延迟,但是吞吐量也会降低[6]。

本文就基于Flink框架的TopN排序问题提高其吞吐量提出解决方案。

2 堆排序优化算法

2.1 传统的堆排序过程

先将待排序的数字构建成一个堆(堆是一棵完全二叉树),如果是要按从小到大的顺序排列,就先将这个堆调整成大顶堆,堆顶的元素就是最大值,然后将堆顶元素与末尾的元素交换位置,最大值"沉"到数组的末端。再将剩余的元素重新调整成大顶堆,把堆顶元素与倒数第二个元素交换位置,重复以上操作,最后就能得到一个从小到大排序的数组了。

堆排序的难点在于将无序堆调整成大顶堆,调整步骤如下:定义一个数组arr,设父节点是arr[i],那么这个父节点的左孩子节点就是arr[$2i+1$],它的右孩子结点就是arr[$2i+2$]。首先比较左孩子节点和右孩子节点的大小,然后将其中较大的孩子节点与父节点进行比较,如果父节点小于孩子节点,就需要交换它们的位置,否则就不用交换。从最后一个非叶子节点开始,从左至右,从下至上进行调整。

而TopN问题只需要选出前N个最大数即可,所以不需要把后面那些很小的数值也排出来。为了降低程序时间复杂度和空间复杂度,可以在原有的堆排序基础上加以改进。定义初始化一个长度为N,元素都为0的数组arr,然后用迭代器的it.hasNext()方法判断是否有待排序的元素,如果有就用it.next()方法接收待排序的元素x。把接收的元素x与arr数组的第一个元素进行比较,如果接收的元素x大于arr[0],就用x替换arr[0],然后把arr数组里的数字用大顶堆按从小到大的顺序排列。利用迭代器依次接收待排序的元素,如此循环往复,最终就能选出最大的N个数值。

2.2 HeapOptimize优化算法

上文所描述的TopN问题的解决方法只能一个一个地接收数据,数据的吞吐量较小,不具备处理大数据的能力,所以还需要继续改进这个算法。把图1中的TopN的解决步骤取名为Process,改进后的算法取名为HeapOptimize,该算法的步骤如图1所示。

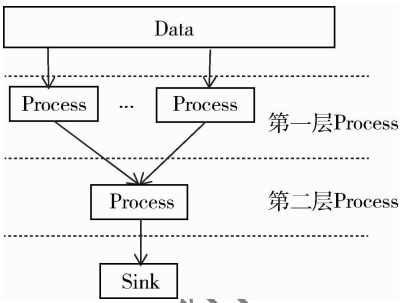

图1 HeapOptimize结构图

箭头表示数据流Stream的走向,最开始的Process算子接收并处理数据流,每一个Process从接收到的数据挑选出前N个最大的数,然后把这N个数传给下一层的Process去处理。第二层的Process把第一层所有Process选出的较大值再进行挑选排序,把最终的结果传给Sink。图中以两层Process为例展示其算法过程,也可以扩展为更多层的Process来处理数据,增加数据的吞吐量。可以根据需求,选择合适的Process层数,快速处理大量的数据。

本文目前实现两层的Process解决TopN问题,TopN作业在Flink框架上运行成功的算子拓扑图如图2所示。先将Source数据源里的数据通过FlatMap把String类型格式转化成int类型,便于后面的计算。

下一步要做窗口计算,在定义窗口之前,要指定流是否需要keyed,使用keyBy("word")将无界流分成逻辑的keyed stream。拥有keyed stream将允许窗口计算由多个任务并行执行,因为每个逻辑key流都可以独立于其余任务进行处理,key相同的元素将被发送到同一个任务。这里采用的是滚动窗口,没有时间重叠,窗口周期是1 s。滚动窗口分配器将每个元素分配给固定窗口大小的窗口。例如,如果指定大小为1 s的滚动窗口,则将执行当前窗

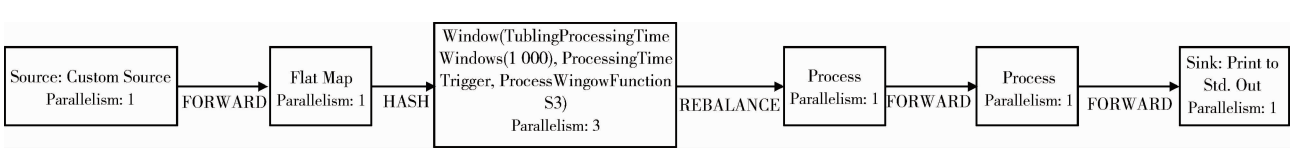

图2 算子拓扑图

口，并且每 1 s 钟将启动一个新窗口，如图 3 所示。

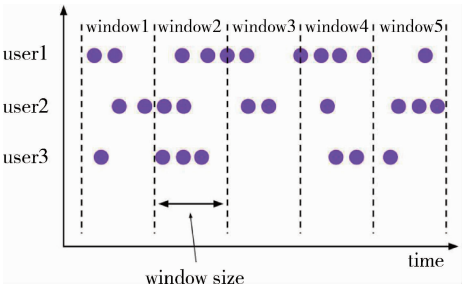

图 3　滚动窗口示意图

在这里可以设置 Process 的并行度（即一层 Process 的个数）。如果需要多层 Process 计算，可以重复从 keyBy 分流到 Process 这些步骤。最后需要把数据都集中到一个 Process 算子作业来排序，再把计算结果输出到 Sink 里打印最终结果。算子拓扑图的第三个算子是窗口计算和第一次 Process 排序合在一起的。

2.3　Process 的层数

假如一分钟需要处理 v 条数据，一个 Process 算子一分钟最多能处理 m 条数据，数据通过一个 Process 算子处理后得到 N 条数据传输到下一层，如果内存和 CPU 足够大，设 x 层需要执行 Process 算子任务的个数（Process 的并行度）是 p_x，最后一层的并行度一定是 1（因为最后需要统计出 N 条最大的数据），前几层的 Process 的并行度不固定，但有一个上限，假设前 n-1 层的 Process 的并行度取最大值，则可以得出层数与数据量的关系。

如果只有一层 Process，该 Process 的并行度也为 1，需要满足的条件是 $v \leqslant m$；

如果有两层 Process，需要满足的条件是 $\frac{v}{p_1} = m$ 且 $N \times P_1 \leqslant m$，即 $v \leqslant \frac{m^2}{N}$；

如果有三层 Process，需要满足的条件是 $\frac{v}{p_1} = m, \frac{N \times p_1}{p_2} = m, N \times p_2 \leqslant m$，即 $v \leqslant \frac{m^3}{N^2}$；

根据数学的不完全归纳法推导出 n 层 Process，v 与 n 的关系：$v \leqslant \frac{m^n}{N^{n-1}}$。

3　实验结果与分析

3.1　实验环境

实验搭建的集群是由两台 PC 组成，由分布式队列 Kafka 作为数据源点生产数据，Flink 提供 FlinkKafkaConsumer 接口从 Kafka 取数据，交给后面的 operator 来处理。用 TaskManager 节点构建整个算子拓扑，将计算结果保存在 HDFS 中，以 Zookeeper 作为集群的同步协调节点负责分布式节点间的信息同步。集群分布情况如表 1 所示。

表 1　集群节点分布信息

节点名称	软件版本	节点数
Zookeeper	Zookeeper 3.4.13	1
JobManager	Flink 1.6	1
TaskManager	Flink 1.6	1
Kafka	Kafka 2.11-2.2.0	1
HDFS	Hadoop 2.5.2	1

由于实验条件有限，TaskManager 节点只创建了一个，当然 TaskManager 节点数越多越好。如果条件允许也可再创建一个 JobManager 节点作备用。集群节点的配置参数如表 2 所示。

表 2　节点配置参数

配置项	配置参数
CPU	Intel(R) Core(TM) i5-8400 CPU @ 2.80GHz
内存	16.0 GB
操作系统	Ubuntu 18.04.1
Java 版本	jdk-8u201-linux-x64

3.2　对比实验与分析

在一开始执行作业的时候，Kafka 开始生产数据，Process 算子的吞吐量从 0 开始慢慢变大到 2 000。当 Kafka 生产的数据量特别大，一个 Process 算子接收数据的速率高于它在一个瞬时脉冲内处理的数据，Flink 系统就会出现背压，把数据量压到底层算子 Source 上，数据流降速，并行度为 1 的 Process 算子的吞吐量也会变小，由于数据源的多样性和输入速率的变化以及集群的不稳定性，使得 Process 算子的吞吐量在某个范围波动，如图 4 所示，该算子的吞吐量大概稳定在 1 500 左右，处理数据的速度较为缓慢。

此时若把该算子的并行度改为 3，让三个 Process 算子同时处理从数据源传来的数据流，然后把这三个算子处理后的数据合流到下一个 Process 集中处理，也可以完成 TopN 作业。这三个 Process

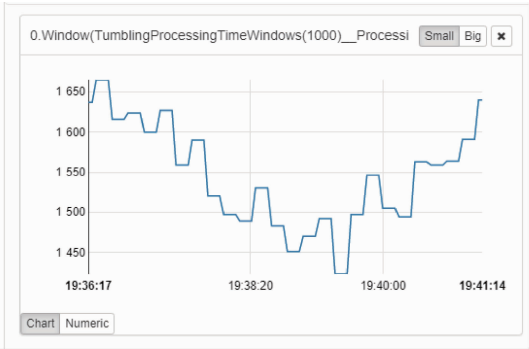

图 4　Process 的吞吐量 v_0

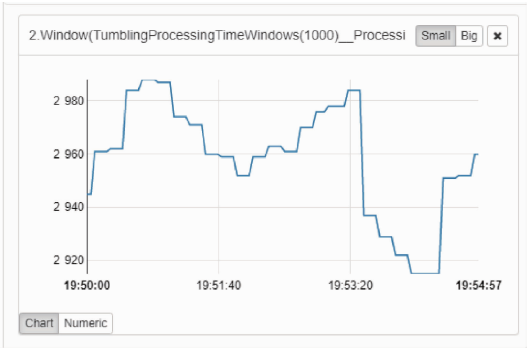

图 7　Process 的吞吐量 v_3

的吞吐量如图 5、图 6、图 7 所示，一个算子的吞吐量能达到 2 000，$v_1 + v_2 + v_3 > v_0$，把 Process 算子的并行度提高到 3 之后，整个 TopN 作业的吞吐量提高了约 7 000 条，TopN 作业的吞吐量提高了。由此可见，当算子出现性能瓶颈时，增加 Process 的并行度可以增加整个排序作业的数据处理速度，解决 Flink 的背压问题，同时也增加了排序作业占用的内存和 CPU。

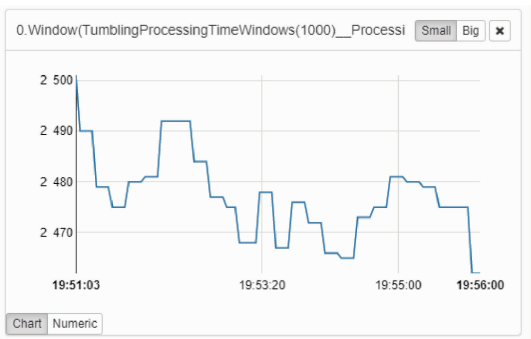

图 5　Process 的吞吐量 v_1

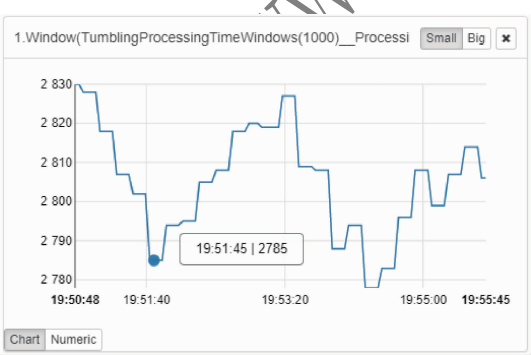

图 6　Process 的吞吐量 v_2

4　结论

本文是将传统的堆排序进行优化，用以处理 TopN 问题，把 HeapOptimize 算法放在 Flink 框架去执行，把待处理的海量数据分割成小块的数据来处理，用化整为零的思想处理数据，把第一层的 Process 算子的处理结果传到下一层 Process 进行处理，这种方法比传统方法的计算速度快，吞吐量增大。Process 的层数根据数据量来设定，由数据量大小计算出理想的 Process 层数。把 Flink 资源合理地分配利用，最大限度地提高 Flink 吞吐量。

参考文献

[1] 蔡鲲鹏. 基于 Flink 平台的应用研究[J]. 现代工业经济和信息化，2017，7(2)：99-103.

[2] 李梓杨，于炯，卞琛，等. 基于流网络的流式计算动态任务调度策略[J]. 计算机应用，2018(9)：2560-2567.

[3] 张洲，黄国锐，金培权. 基于 Storm 的任务调度：现状与研究展望[J]. 计算机科学，2019，46(9)：28-35.

[4] 李梓杨，于炯，卞琛，等. 基于流网络的 Flink 平台弹性资源调度策略[J]. 通信学报，2019，40(8)：85-101.

[5] 徐刘根. 大数据平台加速处理技术的研究与实现[D]. 成都：电子科技大学，2019.

[6] 崔星灿，禹晓辉，刘洋，等. 分布式流处理技术综述[J]. 计算机研究与发展，2015，52(2)：318-332.

（收稿日期：2019-12-30）

作者简介：

关沫（1976 - ）女，博士，副教授，主要研究方向：智能信息处理。

魏碧晴（1995 - ），女，硕士，主要研究方向：大数据技术。

版权声明