doi:10.11835/j.issn.1000-582X.2014.04.009

云计算环境下排序算法的性能分析

丁玉成[1],诸葛晴凤[2],沙行勉[2]
(1.湖南大学 信息科学与工程学院,长沙 410082;2.重庆大学 计算机学院,重庆 400044)

摘 要:随着云计算环境中数据量的激增,人们急需研究在云环境下如何对大量数据进行快速有效的分析与处理。在云环境下对大量数据进行高效地排序是其中一个重要问题。基于 Hadoop 平台研究并实现了几种高效的排序算法,包括:Radix sort,Quicksort 和 Sample sort 算法。对各个排序算法的执行效率、CPU 资源的消耗,内存的消耗,以及处理机间的通信量进行了研究和比较分析。通过大量运行在 Hadoop 上的实验,发现 Hadoop 平台上的 Sample sort 相较于 Radix sort 和 Quicksort 具有排序速度快,负载均衡度高,CPU 消耗低等优势。这一结果为云计算环境下设计更高效、节能的算法提供了有效的依据和基础。

关键词:云计算;hadoop;排序算法;MapReduce
中图分类号:TP302.7 **文献标志码**:A **文章编号**:1000-582X(2014)04-058-07

Performance analysis on sorting algorithms in cloud computing environment

DING Yucheng[1], *ZHUGE Qingfeng*[2], *SHA Xingmian*[2]
(1. College of Information Science and Engineering, Hunan University, Changsha 410082, China;
2. College of Computer Science, Chongqing University, Chongqing 400044, China)

Abstract: With the rapid increase of data amount in cloud computing environment, it is an urgent need to study how to analysis and process those data fast and effectively. How to sort large scale data efficiently in cloud computing environment is a significant problem. Whether the widely used sorting algorithms can achieve high-performance and how many cloud computing resources they consume are concerned problems. This paper studies and implements several efficient sorting algorithms, including Radix sort, Quicksort and Sample sort, based on Hadoop, analyzes and compares the efficiency, consumption of CPU resources, memory consumption and communication between machines. Through a large number of experiments, it's found that compared to Radix sort and Quicksort, Sample sort has the advantages of higher sorting speed, higher load balancing and lower CPU consumption. This result provides a valid basis and foundation for designing more efficient, energy-saving algorithms in cloud computing environment.

Key words: cloud computing; hadoop; sorting algorithm; MapReduce

收稿日期:2013-12-01
基金项目:国家自然科学基金资助项目(61173014)
作者简介:丁玉成(1986-),男,湖南大学研究生,主要从事云计算,分布式计算研究。
诸葛晴凤(联系人),女,重庆大学教授,博士生导师,(Tel) 13983096713;(E-mail)qfzhuge@cqu.edu.cn。

随着大型数据库的不断建立，智能手机、社交网络、微博等新兴媒介源源不断地产生海量数据，正走入大数据时代[1]。大批量数据具有高容量、高价值，多样化和持续性等特点，其非结构化的特性，使得数据的保存，管理，挖掘等成为了当前企业面临的挑战[2]，人们急需研究如何在新兴的云计算环境中快速有效地进行大量数据的分析与处理的问题。对数据的分析处理中，排序本身既是一种对大量数据普遍的处理方式，也是一种预先处理，能使得后续处理更加高效、快捷。由于云计算技术发展的突飞猛进[3]，在云计算环境下对大量数据进行高效排序是要研究的一个非常重要的问题。一些发展成熟并被广泛应用的排序算法能否在云计算系统中应用于大量数据的高效排序，是否可以高效的利用云计算资源是一个个令人非常关注的问题。

排序算法是计算机应用最基本的算法之一。在 1968 年，Batcher 最早提出了 Bitonic sort[4] 之后，更多关于并行排序的算法被研究，包括并行基数排序[5]和快速排序[6]，并行归并排序[7]并行插入排序[8]等。NANCY M. AMATO 等[9]在 1996 年对并行的 Bitonic sortSample sort 以及 Parallel radix sort 在 3 种并行架构下进行了比较研究。PETER M. G. APERS 提出了一种 Recursive Sample sort[10]。Owen O'Malley 等设计的排序算法能够在 Hadoop 上进行 1TB 数据的高效排序[11]。尽管现有的并行排序算法能够解决很多实际问题，但是却不能完全适用于大规模数据的排序。近年来出现的 MapReduc[12]并行编程框架定位于海量数据的处理，分布式文件系统容错性和扩展性良好，能够满足数据量迅速增长的需要，故可以用来解决大批量数据的排序问题。

在 Hadoop 平台上实现了 Radix sort，Quicksort 及 Sample sort 算法，并通过大量运行在 Hadoop 平台上的实验，对各个算法的排序效率，CPU 资源的消耗，内存消耗及处理机间通信量等性能方面进行了深入地比较和分析。对 Radix sort，Quicksort 及 Sample sort 在串行和并行平台上的适用情形做对比分析。最后得出结论，在 Hadoop 平台上，Sample sort 相比于 Radix sort 和 Quicksort 具有排序速度快，负载均衡度高，CPU 消耗少的优势。

1 Hadoop 平台及相关工作

由 Apache 基金会开发的 Hadoop，使得用户可以在不了解分布式底层细节的情况下，开发分布式程序来充分利用集群的计算能力进行高速运算和高效存储。Hadoop 实现了一个分布式文件系统，简称 HDFS。HDFS 具有高容错性，适合部署于较低成本的硬件上且能为应用程序提供了较高的数据吞吐率，适合于大批量数据的应用。MapReduce 是在 Hadoop 中基于 HDFS 之上的引擎，由 JobTracker 和 TaskTracker 组成。

1.1 分布式文件系统(HDFS)

HDFS 是 Hadoop 分布式文件系统，架构如图 1。

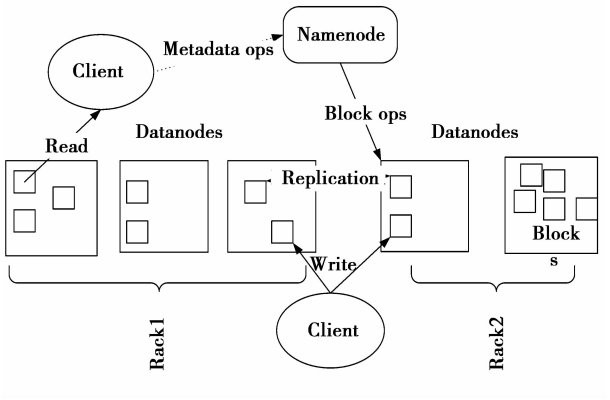

图 1 HDFS 架构

HDFS 的架构包括有 NameNode 和 DataNode。NameNode 主要提供内部元数据服务，DataNode 提供具体数据存储。NameNode 可以控制排序过程中的文件操作，资源分配以及任务运行状态监控。

1.2 MapReduce 框架

MapReduce 是一种支持大规模数据集并行运算编程模型，支持在大数量网络联接的处理机上对数据并行处理。由 JobTracker 和 TaskTracker 组成。

1.2.1 JobTracker 和 TaskTracker

MapReduce 框架是一个主从架构，它有一个主服务器(JobTracker)和若干个从服务器(TaskTracker)组成。JobTracker 是用户与系统交互的关键。用户将自定义的排序任务提交给 JobTracker，JobTracker 将 map 和 reduce 操作分发给 TaskTracker。JobTracker 监控 TaskTracker 的操作。

1.2.2 MapReduce 的 shuffle 和排序

在 Hadoop 中，map 和 reduce 是 2 个基本的数据搜集和处理步骤。在 map 阶段，map 产生的输出将根据最终被传送到的 reduce 进行分区，排序后

写入本地磁盘。在 reduce 阶段，reduce 复制来自若干个 map 任务的输出文件，进行合并排序。各个任务为其在内存和磁盘中的文件建立最小堆，通过不断移动指向堆根节点的迭代器，将 key 相同的数据顺序交给 reduce() 函数处理，将输出写入 HDFS。

MapReduce 保证每个 reduce 进程的输入都已按 key 排序。在系统运行的过程，map 的输出传送至不同 reduce 作为输入的过程为 shuffle 阶段。Shuffle 是 MapReduce 最为关键的部分[12]。整个 shuffle 和排序的过程如图 2 所示。

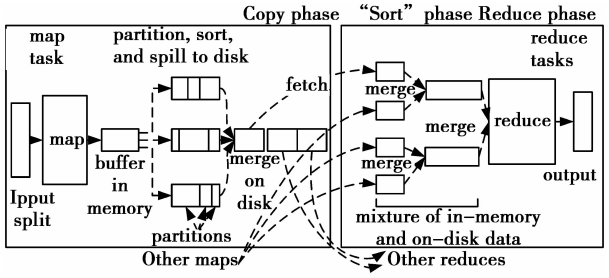

图 2 MapReduce 排序和 shuffle

1.2.3 Hadoop 中的 Partition 分区和比较器

在 Hadoop 中，MapReduce 支持对中间 key 分区，从而使得 map 的输出发送到特定 reduce 上。本文通过自定义分区函数来实现数据的整体有序。Hadoop 中 key 与 key 的比较在排序阶段完成。Hadoop 中提供的原生比较器接口，用于序列化字节间的比较，可加以优化。通过自定义比较器制定不同排序算法键值比较规则。

2 基于云计算平台 Hadoop 的排序算法

在 Hadoop 的处理过程中，数据采用＜key，value＞形式。在研究的算法举例中，只对 key 排序，用 V 表示 value。在算法的实现中，map() 函数和 reduce() 函数分别采用 IdentityMapper、IdentityReduce 默认类中的函数。在算法分析时，由于主机间通过高速千兆网络连接，网络传输的时间相比于算法排序的时间较小，所以不考虑网络的传输时间。

2.1 Radix sort

Radix sort 对 key 中由高到低的数位进行排序，在完成所有数位的排序之后，key 的顺序就已决定。文献[14] 中提出了一种划分后再进行局部基数排序的算法，其基本思想是先将整体数据划分为不同部分，然后每一部分由一个处理器来进行处理。其中 i 部分的数据都小于 $i+1$ 部分的数据。每个处理器只对划分到自己的那部分数据进行基数排序。最后将各个处理器上的数据按序收集则能实现全局有序。此算法要应用在 Hadoop 平台上，需要考虑数据在 Hadoop 上数据划分及 Radix sort 的排序方法的实现。Hadoop 平台上的 Radix sort 算法描述如下：

map() 函数的输出经自定义的分区函数分区后进行 Radix 排序，之后数据传送至分区对应的 reduce 进行处理，最后将各个 reduce 的数据按序收集。

图 3 表示的是基于 Hadoop 平台的 Radix sort 算法处理数据的过程。笔者用 5 个步骤对基于 Hadoop 平台的 Radix sort 算法进行简要说明。

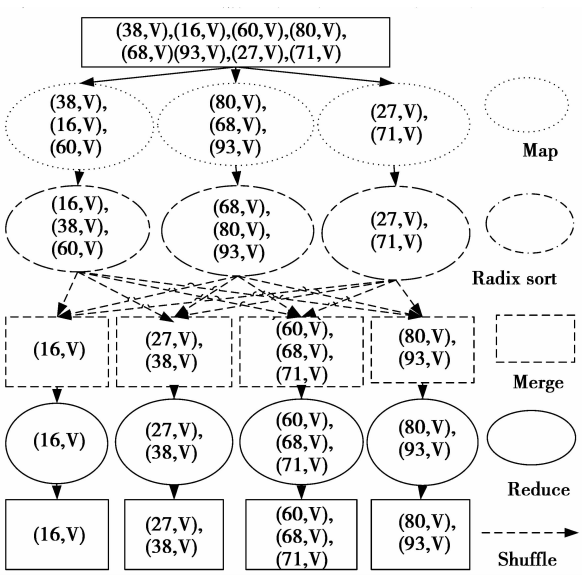

图 3 Radix sort 算法

步骤 1：所有数据分块后，每个块由单个 map 处理。例如，将 8 对数据分割成 3 块后由 3 个 map 进行单独处理。

步骤 2：map 的输出按等距区间分区后进行基数排序。例如，划分等距区间，整个数据范围介于 0－100 之间，有 4 个 reduce，可设定 4 个等距区间：依次为 0－25，25－50，50－75，75－100。将 map 的输出按照 key 所对应的区间进行分区。例如第一个 map 输出的(16，V)划分至 0－25 区间，(38，V)划分至 25－50 区间，(60，V)划分至 50－75 区间。每个 Map 对划分到同一分区的数据进行 Radix 排序。

步骤 3：进行 shuffle，将各分区的数据传送至对应的 reduce。例如，(16，V) 送至第 1 个 reduce，

(27,V),(38,V)送至第 2 个 reduce,(60,V),(68,V),(71,V)送至第 3 个 reduce,(80,V),(93,V)送至第 4 个 reduce。各个 reduce 对各自数据合并排序。

步骤 4：各个 reduce 进行 reduce()函数处理后将结果写入 HDFS。

步骤 5：按 reduce 序收集写入 HDFS 的结果。

Radix sort 的输入为 n 个 d 位数，每位 k 种可能性。在串行系统中的时间复杂度为 $O(d(n+k))$。在 Hadoop 下的 Radix sort 算法中，m 为 hadoop 中 map 的个数，r 为 reduce 的个数。数据分割成 m 块的时间复杂度为 $O(m)$，每个 map 进行基数排序的时间复杂度为 $O(d(n/m+k))$。在 reduce 端进行的归并操作，最坏情况即所有 map 输出数据都由一个 reduce 来处理，其时间复杂度为 $O(n\log(n))$。整个 Radix sort 的时间复杂度为 $O(d(n/m+k))+O(n*\log(n))$。可知，当排序任务负载均衡度不高时，加之集群运行时间消耗和网络传输，则 Hadoop 下 Radix sort 效率可能低于串行运行时的效率。

2.2 Quicksort

快速排序基本思想是：通过一趟排序将要排序的数据分割成独立的 2 部分，其中一部分的所有数据都小于另外一部分所有数据，再按照此方法对这 2 部分数据进行递归快速排序。

采用在 Hadoop 中的默认的经过优化的快速排序算法。通过自定义分区函数以保证数据整体有序。数据经过 map 函数操作后，通过分区函数进行数据等距划分后进行快速排序，不同范围内的数据划分到不同分区后由对应的 reduce 处理，最后按序收集各个 reduce 的数据。

图 4 表示的是基于 Hadoop 平台的 Quicksort 算法处理数据的过程。用 5 个步骤对基于 Hadoop 平台的 Quicksort 算法进行简要说明。

步骤 1：所有数据按块大小分割后交由 map 处理。如图 4 中，8 对数据分块，交由 3 个 map 处理。

步骤 2：map 的输出按固定区间进行分区后进行快速排序。区间划分同 Radix sort 步骤 2。之后对每个 Map 中的数据进行快速排序。

步骤 3：进行 shuffle 后将各个分区的数据送到对应的 reduce。数据传送过程同 Radix sort 步骤 3。

步骤 4：reduce 进行 reduce()函数处理后将结果写入 HDFS 中。

步骤 5：按 reduce 序收集写入 HDFS 的结果。

Quicksort 的输入为 n。在串行系统中，使用

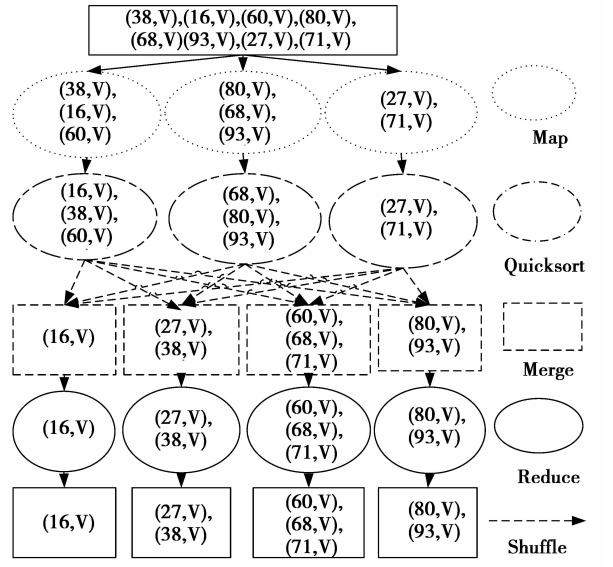

图 4 Quicksort 算法

Quicksort 排序的时间复杂度为 $O(n\log(n))$。Hadoop 下的 Quicksort 算法，m 为 hadoop 中 map 的个数，r 为 reduce 的个数。数据分割成 m 块的时间复杂度为 $O(m)$，每个 map 进行 Quicksort 时间复杂度为 $O(n/m\log(n/m))$。在 reduce 端进行的归并操作，最坏情况即所有的 map 数据都由一个 reduce 来处理，其时间复杂度为 $O(n\log(n))$。整个 Quicksort 的时间复杂度为 $O(n\log(n))$。可知，当排序任务负载均衡度不高，加之集群运行时间消耗及网络传输，则 Hadoop 下 Quicksort 效率可能低于串行时的效率。

2.3 Sample sort

Sample sort 类似于快速排序，基本思想是用 $p-1$ 个分隔值(spliter)将序列分割到 p 个桶(bucket)中，然后每个处理器再按照分割点将数据送到合适的桶中，各桶内排序之后进行整体归并操作。

在 Hadoop 平台上实现 Sample sort，通过抽样来获得数据区间的划分。抽样采用 hadoop 抽样器。具体的实现过程如下：假设有 n 条数据，m 个 map，r 个 reduce。数据在进行分块时，Hadoop 抽样器在各块中各选取个元素。将个合并排序后等距抽取 $r-1$ 个数据作为最终分割元素。最终分割元素放入分布式缓存。在 map 阶段将数据按照分布式缓存中的分割元素进行分区后并快速排序。不同范围内的数据传送至对应的 reduce 处理，最后将各个 reduce 的数据按序收集。

图 5 表示的是基于 Hadoop 平台的 Sample sort 算法处理数据的过程。用 5 个步骤对 Hadoop 平台上的 Sample sort 算法进行简要说明。

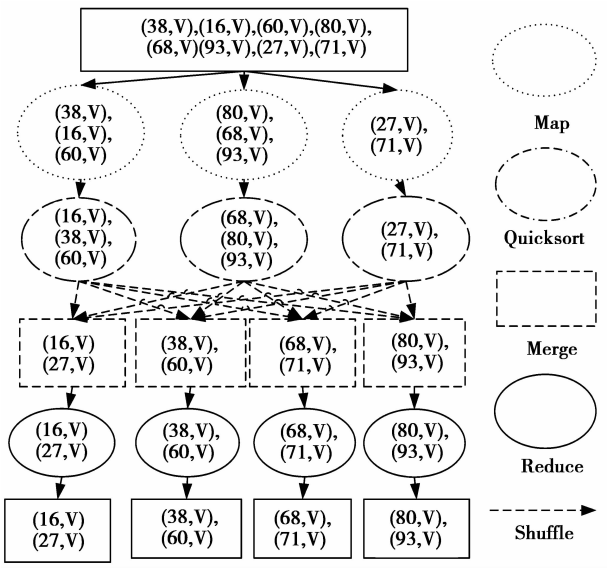

图5 Sample sort算法

步骤1:所有的数据按照块的大小进行分割,然后由3个map进行处理。例如图5,在分割块中随机选出样本,排序后找到3个最终分割点38,68和80。

步骤2:数据块中数据按行进入map()函数进行处理后输出。

步骤3:数据分区并排序。每个TaskTracker根据分布式缓存中的分割点将map()输出的数据分区后排序。如图5,数据整体介于0－100之间,按4个reduce,划分4个区间:0－38,38－68,68－80,80－100。在将map()函数输出按区间分区。例如第2个map的输出结果中(68,V)划分到第3分区,(80,V),(93,V)划分到第4分区。每个map中对数据进行快速排序后经过shuffle阶段将数据发送至对应的reduce上。

步骤4:reduce进行数据合并排序等处理后将结果写入HDFS。

步骤5:按reduce序收集写入HDFS的结果。

n为Sample sort数据输入条目,在串行系统中,k为分割成的块数,Sample sort运行时间复杂度为$O(n\log(n/k))+O(\min(kk\log k),n\log n)$。在Hadoop平台上的Sample sort,$m$个map,$r$个reduce。在分割过程中,数据分割成$m$块的时间复杂度为$O(m)$,每个map输入对应的块中抽样数为$s$,时间复杂度为$O(s)$,$m$个块$ms$个样本集合后排序的时间复杂度为$O(ms\log(ms))$,选取第$s,2s,\cdots,(r-1)s$个数据作为最终分割点的时间复杂度为$O(r-1)$。各个map内部排序的时间复杂度为$O(n/m\log(n/m))$,各个map根据最终分割点将数据分割到相应的reduce,每个reduce对各自的n/r数据排序的时间复杂度为$O(n/r\log(n/r))$。所以最终的时间复杂度为$O(ms\log(ms))+O(n/m\log(n/m))+O(n/r\log(n/r))$。由此可知,排序数据量很大时,Hadoop上Sample sort排序效率高于串行时的效率。

3 实验评估

3.1 实验环境

由于实验条件的限制,学术界往往用有限的计算节点模拟云计算环境[15-17]。笔者以4个节点模拟云计算环境并对模拟实验环境下的各种排序算法的性能差异进行对比分析。4个配置都为Intel Core i5 3.10 GHz CPU,4 G内存,操作系统为Ubuntu11.10,主机间通过高速千兆网络连接。其中1台master,3台slave。所用Hadoop版本为0.20.2,JDK版本为1.6。输入数据由Hadoop程序产生,每条记录都为<key,value>形式,只对key进行排序。为了有效的比较算法性能,在算法验证的运行过程中,出现数据块丢失等情况,则视此次运行无效,故修改Hadoop文件块默认的3个副本为1个副本。实验在Eclipse上编写、调试代码和运行任务。通过Ganglia监控集群,收集信息并显示。

对3种排序算法所研究的性能指标主要有:1)算法运行时间;2)CPU资源的消耗;3)内存的消耗;4)处理机之间的通信量。

3.2 实验结果与分析

3.2.1 Radix sort,Quicksort及Sample sort排序时间

图6是在相同输入量的情况下3种排序算法在Hadoop平台上效率的比较。比较得出,随着输入量增大,3种排序算法所用的时间差距在增大。输入量较大时,Sample sort排序时间最少,主要得益于Sample sort较高的负载均衡及排序算法效率。由于负载均衡度较低,Quicksort排序时间多于Sample sort,Radix sort排序效率和负载均衡度都较低。

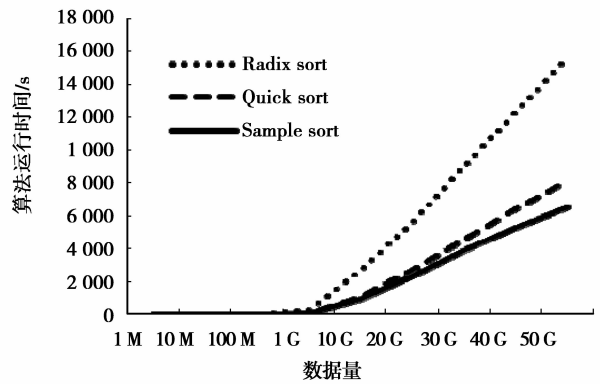

图6 相同输入量下3种排序算法运行的时间

3.2.2 相同输入量时运行 Radix sort，Quicksort 及 Sample sort 时集群 CPU 负载情况

图 7 表示输入 13 G 数据后 3 种算法运行时集群 CPU 消耗情况，其中 cpu_user 和 cpu_system 分别代表用户空间和内核空间消耗 CPU 资源的百分比。比较得出，Sample sort 运行时集群 CPU 消耗量最小，Radix sort 运行时最多，Quicksort 运行时居中。

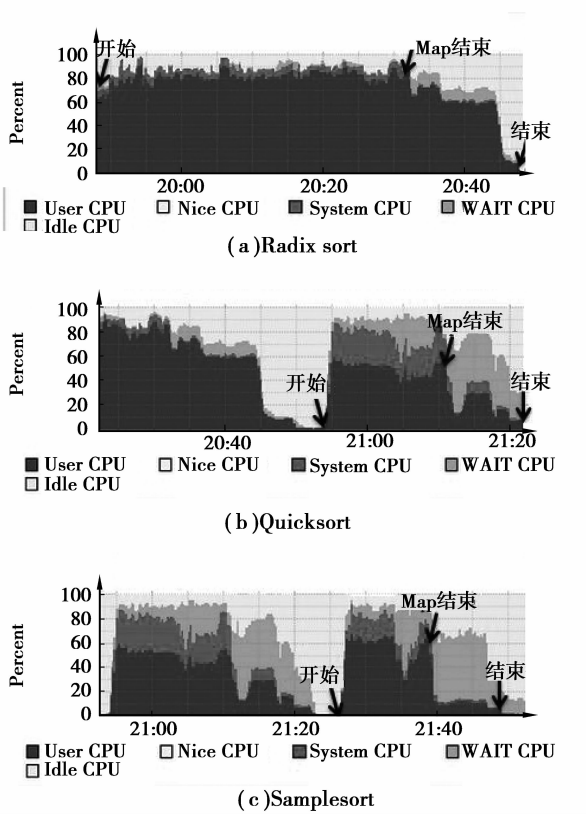

图 7 具有相同输入量下集群排序时 CPU 负载情况

3.2.3 相同输入量下集群运行 Radix sort，Quicksort 及 Sample sort 时内存资源消耗

图 8 表示在 13 G 数据输入的情况下 3 种算法运行时集群内存资源消耗情况的比较，其中 memory used 表示内存使用量。比较得出 Sample sort 运行时，内存资源消耗最多，Radix Sort 和 Quicksort 消耗量相近。在 Sample sort 运行中，需要对数据进行抽样，合并后排序以及在分布式缓存中存取数据，这些操作都占用内存，导致 Sample sort 的消耗较多内存。

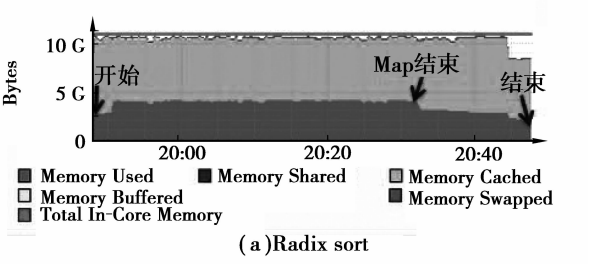

图 8 相同输入量下集群排序时 Memory 使用情况

3.2.4 相同输入量下集群运行 Radix sort，Quicksort 及 Sample sort 时集群通信量

图 9 表示在 13 G 数据输入的情况下 3 种算法运行时集群通信量的比较，其中 In 和 Out 分别表示每秒进和出的包数。可以看出在 map 阶段，Sample sort 有较高的通信量，在 reduce 阶段，Sample sort 和 Radix

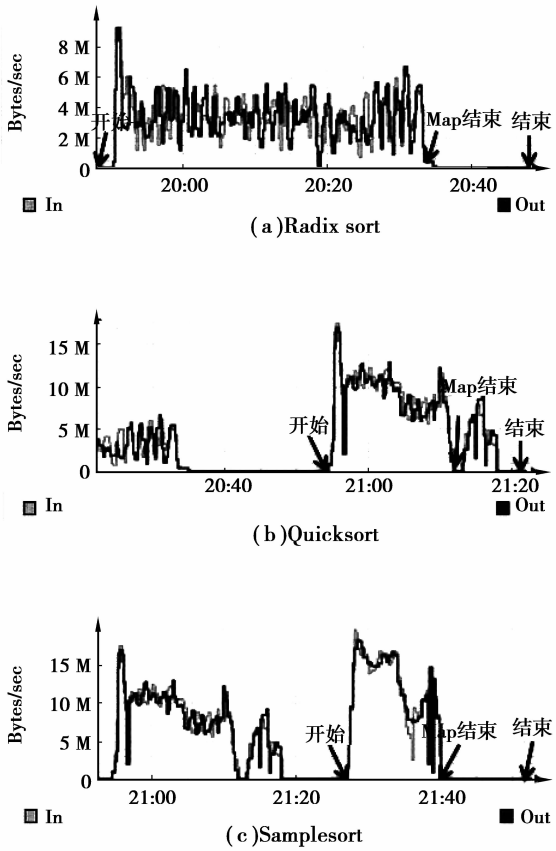

图 9 相同输入量下集群排序时通信情况

sort的通信量几乎为零,而Quicksort仍有一定的通信量。整体比较下,运行Radix sort时集群的通信量最少,Sample sort运行时的通信量最多。

4 结 论

针对在云计算平台Hadoop上如何实现大数据的排序进行了深入的研究。首先,在Hadoop平台上实现了Radix sort,Quicksort和Sample sort算法。对基于Hadoop的3种排序算法在具有相同输入量的情况下进行排序时间,集群CPU资源消耗,集群内存资源消耗以及集群整体通信量等方面进行了性能的比较分析。实验结果表明,在对相同较大数据量进行排序时,Radix sort具有较少通信量和消耗较少集群内存,Quicksort占用较多集群内存和通信量,排序效率高于Radix sort。Sample sort排序效率和负载均衡度较高,具有通信量较多,对集群CPU资源和内存资源占用量较少的特点。由此可知,Sample sort更适合于具有较大数据量处理能力的集群来完成大量数据的排序任务。

参考文献:

[1] 覃雄派,王会举,杜小勇,等. 大数据分析——RDBMS与MapReduce的竞争与共生[J]. 软件学报,2012,23(1):32-45.
QIN Xiongpai,WANG Huiju,DU Xiaoyong,et al. Big data analysis-competition and symbiosis of RDBMS and map reduce [J]. Journal of Software,2012,23(1):32-45.

[2] 王珊,王会举,覃雄派,等. 架构大数据:挑战、现状与展望[J]. 计算机学报,2011,34(10):1742-1743.
WANG Shan, WANG Huiju, QIN Xiongpai, et al. Architecting big data:challenges,studies and forecasts [J]. Chinese Journal of Computers,2011,34(10):1742-1743.

[3] 陈康,郑纬民. 云计算:系统实例与研究现状[J]. 软件学报,2009,20(5):1337-1348.
CHEN Kang, ZHENG Weimin. Cloud computing: system instances and current research [J]. Journal of Software,2009,20(5):1338-1345.

[4] Kenneth E. Sorting networks and their applications [C]// Proceedings of Spring Joint Computer Conference,April 30-May 2,1968,Atlantic City,NJ. New York: ACM, 1968:307-314.

[5] Xi N,Fang Z Y,Miao Z,et al. Parallel radix sort and its application in ray tracing [C]// Proceedings of 2010 the Second International Conference on Information Science and Engineering, December 4-6, 2010, Hangzhou, China. Piscataway:IEEE Press,2010:1366-1369.

[6] Korthikanti V A,Gul A. Analysis of parallel algorithms for energy conservation in scalable multicore architectures [C]// Proceedings of 2009 International Conference on Parallel Processing, September 22-25, 2009,Vienna. Piscataway:IEEE Press,2009:212-219.

[7] Davidson A,Tarjan D,Garland M,et al. Efficient parallel merge sort for fixed and variable length keys [C]// Proceedings of Innovative Parallel Computing,May 13-14, 2012,San Jose,CA. Piscataway:IEEE Press,2012:1-9.

[8] Biernacki C,Jacques J. A generative model for rank data based on insertion sort algorithm [J]. Computational Statistics & Data Analysis,2013,58:162-176.

[9] Amato N M,Iyer R,Sundaresan S,et al. A comparison of parallel sorting algorithms on different architectures [R]. Texas:Texas A&M University,1996.

[10] Shimizu S,Sugisaki K,Ohmori H. Recursive sample-entropy method and its application for complexity observation of earth current [C]// Proceedings of 2008 International Conference on Control, Automation and Systems,October 14-17,2008,Seoul. Piscataway:IEEE Press,20008:1250-1253.

[11] O'malley O,Murthy A C. Winning a 60 second dash with a yellow elephant[EB/OL]. (2009-04)[2013-05-24]. http://sortbenchmark.org/Yahoo2009.pdf.

[12] Dean J, Ghemawat S. MapReduce: simplified data processing on large clusters [J]. Communications of the ACM,2008,51(1):107-113.

[13] White T. Hadoop: the definitive guide [M]. Santa Clara:O'Reilly Media/ Yahoo Press,2012.

[14] Lee S J,Jeon M,Kim D,et al. Partitioned parallel radix sort[J]. Journal of Parallel and Distributed Computing, 2002,62(4):656-668.

[15] Jiang J,Lu J,Zhang G Q,et al. Scaling-up item-based collaborative filtering recommendation algorithm based on hadoop [C]// Proceeding of 2011 IEEE Congress on Services,July 4-9,2011,Washington,DC. Piscataway: IEEE Press,2011:490-497.

[16] Yang L,Shi Z Z. An efficient data mining framework on hadoop using java persistence API [C]// Proceedings of 2010 IEEE 10th International Conference on Computer and Information Technology, June 29-July 1, 2010, Bradford. Piscataway:IEEE Press,2010:203-209.

[17] Zhang C,De S H. CloudBATCH:a batch job queuing system on clouds with Hadoop and HBase [C]// Proceedings of 2010 IEEE Second International Conference on Cloud Computing Technology and Science,November 30-December 3,2010,Indianapolis, IN. Piscataway:IEEE Press,2010:368-375.

(编辑 侯 湘)