基于 Hadoop 平台的数据分析方案的设计应用

姜文[1,3]，辛阳[1,3]，陈林顺[2]
（1. 北京邮电大学网络与信息攻防技术教育部重点实验室，北京 100876;
2. 北京安码科技有限公司，北京 100876;
3. 灾备技术国家工程实验室，北京 100876）

摘要：面对互联网上的海量数据，单台主机已无法满足其存储和计算要求，分布式存储和分布式计算的应用成为必然的趋势。其中 Hadoop 是应用较多的分布式存储和计算框架之一。本文在该平台下，通过对国内某搜索引擎两个月内的上千万条用户搜索日志进行数据统计分析，给出相应 Map/Reduce 程序的设计思路和实例，并提出 Map/Reduce 分布式程序的部分设计和性能优化方法，实验结果表明，本文提出的这些方法能简化 Map/Reduce 程序设计、有效提高程序性能。

关键词：分布式计算；Map/Reduce；文本处理；数据分析

中图分类号：TP319

Designed Application of the Scheme of Data-Analysis based on Hadoop Platform

Jiang Wen[1,3], Xin Yang[1,3], Chen Linshun[2]
(1. Key Laboratory of network and information attack and defence technology of Ministry of Education, Beijing University of Posts and Telecommunications, Beijing 100876;
2. Beijing Safe-Code Technology Co.,Ltd, Beijing 100876;
3. National Engineering Laboratory for Disaster Backup and Recovery, Beijing 100876)

Abstract: To face of massive data on the Internet, a single host has been unable to meet the storage and computing requirements, distributed storage and distributed computing is an inevitable trend, Hadoop is the most popular of them. In this paper, we do data-statistic-analysis in Sogou search engine, which contains 10 million search logs with two moths, and shows the method and instance of programming design patterns with Map/Reduce framework, at last we propose some method of program performance Optimization in Map/Reduce.Experiment results show that these methods can simplify the design of Map/Reduce programming, and improve performance effectively.

Keywords:distributed computing; Map/Reduce; text-processing; data-analysis

0 引言

互联网高速发展，信息量不断膨胀，各种大型搜索引擎的访问量迅速增加，与此同时这些大型系统都记录下了海量的用户访问和查询日志，挖掘出日志中蕴藏的信息来改进搜索引擎性能、提升服务质量是非常有价值的，然而传统的日志分析和数据统计方法受到单机内外存、CPU 资源有限的限制，在进行海量日志数据的分析时运到了瓶颈。Hadoop 作为著名开源组织 Apache 旗下专注于分布式存储和计算的开源项目，越来越收到重视，现在广泛应用于网页搜索、日志分析、广告计算、数据挖掘等领域。

本文以国内著名搜索引擎的用户查询日志为数据源，介绍了 Hadoop 在文本处理及数据分析中的应用，使用的数据为搜索引擎用户查询日志。并根据实验结果提出 Hadoop 在数据

作者简介：姜文(1986-)，男，在读研究生，主要研究方向：分布式计算、网络安全. E-mail: jiangwen127@gmail.com

分析中的应用设计方法。

文献[1]提出利用 Map/Reduce 进行文本分析的方法，包括文本倒排索引和图论相关的应用；文献[3]提出对用户搜索日志进行分类的理论方法，但未给出具体的数据分析操作和效率；文献[4]提出了对用户查询进行相关性分析的理论方法，并对某搜索引擎的某 20 个热点查询词进行实际数据分析，但欠缺对大数据量的实验分析和验证；因此本文在前面的研究成果之上，在 Hadoop 分布式计算平台上对超过 G 级的搜索日志数据进行数据分析，并提出在该平台上进行分布式程序设计的部分具有普适性的方法。

1 Hadoop 分布式文件系统及分布式计算模型介绍

凭借其在海量数据处理领域的三大领先技术，Google 在激烈的行业竞争中占据极大优势,其中的两个技术就是 GFS 分布式存储和 Map/Reduce 分布式计算框架[5]。Hadoop 即是在参考 Google 相关技术而发展起来的开源分布式存储和计算系统。

Hadoop 分布式文件系统[8]HDFS(Hadoop Distributed FileSystem)被设计成适合运行在通用硬件上的分布式文件系统。它和现有的分布式文件系统有很多共同点。但同时，它和其他的分布式文件系统的区别也是很明显的。HDFS 是一个高度容错性的系统，适合部署在廉价的机器上。HDFS 能提供高吞吐量的数据访问，非常适合大规模数据集上的应用。HDFS 在本实验中的主要功能是提供海量数据的存储。

Hadoop Map/Reduce 是一个使用简易的软件框架，基于它写出来的应用程序能够运行在由上千台商用机器组成的大型集群上，并以一种可靠容错的方式并行处理上 T 级别的数据集。一个 Map/Reduce 作业通常会把输入的数据集切分为若干独立的数据块，由 Map 任务以完全并行的方式处理它们。框架会对 Map 的输出先进行排序，然后把结果输入给 Reduce 任务。通常作业的输入和输出都会被存储在文件系统中。整个框架负责任务的调度和监控，以及重新执行已经失败的任务。通常，Map/Reduce 框架是分布在分布式文件系统的节点上的，也就是说，计算节点和存储节点通常在一起。这种配置允许框架在那些已经存好数据的节点上高效地调度任务，这可以使整个集群的网络带宽被非常高效地利用。Map/Reduce 框架由一个单独的 Master 和集群节点 Slave 共同组成。Master 负责调度构成一个作业的所有任务，这些任务分布在不同的 Slave 上，Master 监控它们的执行，重新执行已经失败的任务。而 Slave 仅负责执行由 Master 指派的任务。

图1描述了 Map/Reduce 的计算流程，其中数据存取、系统容错以及任务调度都是由分布式存储和计算框架在系统内部完成的，用户无需对此进行操作，这在一定程度上大大提高了分布式应用的开发效率。

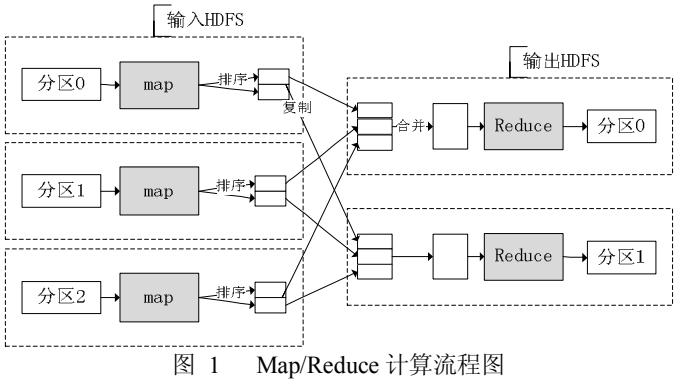

图 1 Map/Reduce 计算流程图

2 实验数据

本文使用的搜索引擎查询日志库为包括 2 个月内某搜索引擎所有网页查询需求及用户点击情况的网页查询日志数据集合，数据大小为 3.4G，含非空查询 205,235,203 个，非重复查询 20,748,341 个，Session 个数为 102,837,981 个。

表1 用户查询日志格式

名称	*记录内容*
ID	根据用户使用浏览器访问搜索引擎时的 Cookie 信息自动赋值
query	用户输入的查询词，多个查询词使用'+'连接
rank	用户此次点击的页面在所有返回页面中的 Rank 排名
order	这是用户在输入某查询词后的第几次点击
URL	点击页面的 URL

搜索引擎用户日志中的常用术语：

Term：不包含空格的一个字符串，这个字符串可由字母、数字、其他符号组成。

Query：一个查询包括一个或多个查询 Terms，很可能还包含一些逻辑操作符和修正符。这些符号是搜索引擎提供给用户，方便用户更精确的定位查询目标的工具。通常日志系统中的一行记录就是一个 Query。

Session：用户在几分钟甚至是几个小时之内的一系列 Queries 的集合，一个 Session 可以包含一个或多个 Queries。本实验中的数据使用用户 ID 来标识一个 Session，一般情况下，用户使用同一浏览器进行不同查询的情况不常发生，所以使用 ID 来划分 Session 的方式不失准确性。

3 方案设计

本节给出两个 Hadoop Map/Reduce 在搜索引擎用户搜索日志分析中的应用设计：点击 URL 序号-频度关系、查询词相关性分析。在这两个 Map/Reduce 程序设计方法中，本文尽可能的利用 Map/Reduce 的框架特性，简化程序设计，减少数据 IO，提高数据分析效率。其他数据分析方法可以从这两个设计实例进行类比。

3.1 点击 URL 序号-频度关系

在搜索引擎中，用户每输入一个 Query，系统将返回几百甚至上千个结果 URL 列表，它们按照与查询词的相关度进行排序，每个结果 URL 有一个序号，一般每 10 个 URL 及其摘要组成一页的结果信息。对点击的 URL，按所在的序号进行点击频度统计，得到序号-频度关系，可以直观的反映出用户的点击习惯以及相关性排序是否符合用户需求。

根据 Keys 将<Key, Value>进行分类、排序，然后对同一类 Keys 下的 Values 列表进行数据统计分析，这是 Map/Reduce 提供的最基本也是最有效的运算方式。Map/Reduce 程序中只需要设置<Rank, 1>为 Map 任务的输入，在 Reduce 任务中对同一 Rank 下的所有 Values 值进行累加，即可得到<Rank, Count>的输出。

当数据量大于系统可用内存时，很多高效的内存内排序、计算算法都无法使用，而内存外的算法和数据管理较复杂。Map/Reduce 框架解决了这些问题，对大数据量的处理，具有较高的可扩展性，计算框架内部实现了数据的分类、排序和数据存取、数据网络传输。

3.2 Query 相关性分析

在搜索引擎的使用过程中，用户可以尝试不同的查询以期得到更加准确的查询结果。为

方便用户进行查询修正或查询扩展，一些搜索引擎系统已经在用户第一次提交查询请求后，在系统的返回结果页面中为用户提供了一些相关查询词列表，供用户再次查询时参考，这为用户更精确地表达自己的信息需求提供了便利。

文献[8]中给出了计算查询词相关性的公式：

$$Similar(q_i,q_1)=\sum_{j=1,\ldots,|U|}\frac{\left|\min(m(q_{ij}),m(q_{1j}))\right|}{\left|m(q_1)\right|}*p_j \quad \text{(公式1)}$$

其中，设查询集合 W 为$\{q_1,\ldots,q_m\}$，点击 URL 集合 U 为$\{URL_1,\ldots,URL_m\}$，查询-点击矩阵为M_{m*n}。m_{ij}表示第 i 个查询q_i点击第 j 个URL_j的次数。记集合 U 为$|U|$。其中，$m(q_1)$表示查询词为q_1的所有点击 *URL* 总数，即矩阵 M 第一行的各元素总和。p_j表示查询词为q_1时点击第 j 个 *URL* 的概率，可以近似的用频度替代。

这里涉及到较多的计算，需要使用多次 Map/Reduce 计算来得出结果，本文的 Query 相关性分析需要使用三次 Map/Reduce 计算流程，表 2 中列出了三个步骤的伪代码，，前一次输出作为后一次的输入:

1) 对每一条查询日志，Map 同时输出<(Query, URL), 1>和<Query, 1>两个键值对，在 Reduce 阶段同时统计输出<(Query, URL), Q-Ucnt>和<Query, Qcnt>(分别记为 Q-Ucnt 和 Qcnt)，这两个值对应公式(1)中的$m(q_{ij})$和$m(q_i)$。

2) 将点击相同 URL 的 Query 归类到一个 Query 队列中。输出<URL, Q-List[1]>形式的统计信息，其中 Q-List 中的每个元素包含信息集合(Query, Q-Ucnt, Qcnt)。

3) 根据相关性的定义，步骤 2 中的 Q-List 中的元素两两相关，对每一个<URL, Q-List>输入，Map 阶段两两取出 Q-List 中的数据节点，根据公式(1)计算两者的部分相关性，并在 Reduce 阶段统计得出所有 Query 之间的相关性。

在进行以上程序设计时，在一个 Map/Reduce 过程中通常会进行多种数据结构的统计分析，这是由 Map/Reduce 的计算方法所决定的，我们必须尽量减少 Map/Reduce 的执行次数，以减少磁盘 I/O，到达提高系统效率的目的，这是 Map/Reduce 在处理海量数据时需要遵循的一个最基本的准则。

表2 查询词相关性分析的伪代码

Step1	Step2	Step3
1. Method Map(Each Query Line) 2. Emit(tuple(Q,U), 1) 3. Emit(Q, 1) 4. Method Reduce(key, [c_1,c_2,…]) 5. sum ← 0 6. for all count c∈[c_1,c_2,...] do 7. sum ← sum + c 8. if IsTuple(key) then 9. Emit(tuple(Q,U), sum) 10. else IsQ**(key)** then 11. Emit(Q, sum)	1. Method Map(Lines) 2. Every two lines do 3. Emit(U, tuple(Q, sum_{tuple}, sum_O)) 4. Method Reduce(U, [$tuple_1$, $tuple_2$,…]) 5. QList ← null 6. for all tuples t∈[$tuple_1$, $tuple_2$, …] do 7. QList ← QList + t 8. Emit(Q, QList)	1. Method Map(Q, QList) 2. for each tuple pair p(t_1,t_2)∈QList do 3. q_1 ← Query in tuple t_1 4. q_2 ← Query in tuple t_2 5. c ← correlations between q_1, q_2 6. Emit(tuple(q_1,q_2), c) 7. Method Reduce(tuple(q_1,q_2), [c_1,c_2,…]) 8. sum ← 0 9. for all the same tuples do 10. sum ← sum + c 11. Emit(tuple(q_1,q_2), $sum_{correlation}$)

注：伪代码中的tuple(a,b)表示a和b的组合；Q表示查询词；U表示URL

[1] Q-List 表示这是一个由 Query 相关信息组成的数组或链表

3.3 实验结果

表格 2 列出本节设计的两个应用实例的性能测试结果，单机表示在单个 PC 上设计的具有相应功能的非分布式程序(使用 Linux Shell 作为程序设计工具)，分布式表示在多个 PC 上运行的 Map/Reduce 分布式程序。同时，对分布式程序，通过改变其分布式运算平台的节点数量来进行性能比较。

表3　处理 3.4G 日志数据所消耗的时间

	单机1PC	分布式2PC	分布式5PC	分布式8PC
URL-频度关系(秒)	82	88	43	30
Query相关性(秒)	562	586	365	239

注:单机指采用分布式方法在单个PC上进行数据操作。PC配置：CPU2.6GHz，内存2G

可以看到，在节点较多的时候，分布式计算框架较单机具有更好的计算效率。并且随着节点的增多，分布式计算框架的性能能得到相应的提高，这就是分布式计算较单机具有较强扩展性的体现。

4 Map/Reduce 程序设计和性能优化

4.1 本地聚集

在向网络中发送结果之前，Map/Reduce 中间结果先写本地磁盘。相对于内存操作来说，网络和磁盘 IO 的延迟具有更高的时间代价。在 Map/Reduce 中，中间结果的本地聚集是提高程序性能的一个有效手段。

Map/Reduce 框架的 Combiner 组件提供了对 Map 输出的分块数据进行合并的操作，通常这个操作时在 Reduce 阶段进行的。本地聚集的优化方法将 Combiner 移到 Map 阶段来执行，因此把这种优化操作称作“In-Map-Combiner”，它减少了用于 Reduce 阶段混洗的中间键值对的数据量，大大减轻中间值的网络传输负载，并且网络传输时需要对数据进行序列化和反序列化，减小传输的数据量同时也减少了这部分操作所需要的时间。

虽然本地聚集是一种简单有效的性能优化方式，但是它需要足够的内存来存储中间结果，直到单个 Map 全部处理完当前分块数据，这是其存在的性能瓶颈的问题。

针对 3 节查询相关性分析设计的程序，表 4 列出了使用本地聚集和未使用本地聚集的性能比较。可以看出，对于这个应用实例，使用本地聚集相对于未使用本地聚集的设计有 10%左右的性能优化。

表 4 使用本地聚集优化的查询词相关性分析程序性能

	分布式 2PC	分布式 5PC	分布式 8PC
未使用本地聚集(秒)	586	365	239
使用本地聚集(秒)	529	330	215

4.2 二次排序

Map/Reducer 框架在记录到达 Reducer 之前对这些记录按键排序。然而对于任何特定的键，其对应的值没有排序。每次运行时值的顺序甚至都不固定，因为它们来自不同的 Map 任务，每次运行时完成时间都不同。一般而言，Map/Reducer 的程序都编写为不依赖于值到达 Reducer 函数的顺序。Map/Reduce 在 Shuffle 和 Sort 阶段，通过 Keys 对 Key-Value 对进行排序。这种排序机制在 Reduce 阶段常常能简化运算。然而，如果想在 Keys 排序的基础上，

进一步要求 values 也是有序的，就需要使用到二次排序的方法。

考虑下面格式的一组数据：(K，V_1，V_2)。输出目标是先按 K 的值进行排序，相同的 K 值内部按 V_1 的值进行排序。最简单的方法是将一个 Map 的数据全部存放在内存里，在内存中实现二次排序。然而，在内存中缓存数据很容易导致性能瓶颈。更常用的方法是，首先通过 K 值进行第一次排序分组操作，然后再每一个分组里按 V_1 值进行排序，这种方法被称为“Value-To-Key Conversion”设计模式。基本思想是将 Value 的部分值(在这个示例中即为 V_1)转移到 Key 中，和 K 值组合成一个复合键(K，V_1)，让 Map/Reduce 框架来执行内部的排序。为了达到这一目的，必须自定义复合键的排序方式，先按复合键的左值 K 排序，在相同的 K 值内部按复合键的右值 V_1 排序。同时须自定义分组函数 Partitioner，以 K 值进行分组，保证同样的 K 值的复合键值对<(K，V_1)，V_2>被混洗输出到同一个 Reducer。

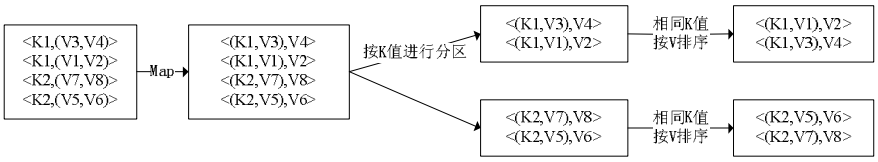

图2　二次排序示例

以上提到的内存内排序和“Value-To-Key Conversion”两种方法，前者效率更高但是在对大数据量进行扩展时容易遇到瓶颈，而后者将排序任务交给 Map/Reduce 框架来完成，在处理 T 级数据时具有更好的并行扩展性。这是 Map/Reduce 计算方法中的一种重要的编程模型。

4.3 Reduce 端联接

Map/Reduce 能够在大型数据集之间进行联接(Join)。联接方法有 Map 端联接和 Reduce 端联接，后者相对于前者更普遍，因为输入数据集不需要有特定的结构，但是所有数据集必须经过 Map/Reduce 洗牌，导致性能稍有下降。Reduce 端联接的基本思路是 Mapper 给每个记录打上它的源标签，并且用联接键作为 Map 的输出键，使有相同键的记录可以聚集在同一个 Reducer。假设需要联接的两个日志数据集具有如下格式：<ID，(Query，Rank)>和<ID，URL>。需要用 ID 做 Key 将这两个数据集联接为如下的格式：<ID，(Query，Rank，URL)>，可以用以下的方法来实现：

1）多样输入。一般来说，该数据集的输入源有不同的格式，如数据集 1 中的 Value 值为(Query，Rank)数据对(其中 Query 为字符串，Rank 为整型)，数据集 2 中的 Value 为 URL 字符串，因此可以很方便地使用 MultipleInputs 类[2]来分隔用于解析和标记每个源的逻辑。

2）二次排序。Reduce 会看到来自含有相同键的源的记录，但并不保证它们有任何特定的顺序。然而，为了执行联接，有顺序地接收数据是很重要的。对于上例，每个 ID 对应的(Query，Rank)数据对必须是第一个值，然后是 URL 字符串(如果将接收到的数据在内存中缓存，那么无序的接收数据时可行的，但应该避免这种扩展性不佳的设计方法)。为了实现顺序输出的目的，使用(ID，NUM)的符合键值对做 Key，对第一个数据集，输出键值对<(ID，0)，(Query，Rank)>，对第二个数据集，输出键值对<(ID，1)，URL>，然后利用二次排序的方法，对数据对(ID，Num)进行二次排序，这样对于相同 ID 的数据记录，必然先输出 Key

[2] MultipleInputs 类用来读入不同类型的输入数据，具体请参看 http://hadoop.apache.org

数据对(ID，0)所对应的值：(Query，Rank)数据对，然后输出(ID,1)所对应的值：URL 字符串，只要按顺序将它们联接输出即可。

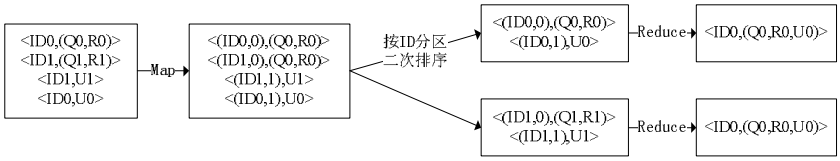

图2 Reduce 端 Join 示例

5 总结

本文介绍了 Hadoop 中的最重要也是应用最广泛的组件：分布式文件系统 HDFS 以及分布式计算框架 Map/Reduce。在 Hadoop 平台下对搜索引擎用户查询日志进行数据统计和分析，给出了点击 URL-频度关系、Query 相关性分析、Query 分类的实现方法和思路，其他相似的数据分析方法可以进行类比。最后在实例分析的基础之上，给出了本地聚集、二次排序、Reduce 端联接三个常用的 Map/Reduce 程序设计和性能优化方法。相对于文献[2,3,8]中提到的数据分析和分类方法，Hadoop 的 Map/Reduce 计算框架具有更好的扩展性，能够快速处理 TB 级的数据，这是该计算框架最大的优势。

本文后续将继续深入研究 Hadoop 在海量文本统计分析、数据挖掘中的应用，对文本处理的效率做进一步的研究和比较；研究 Map/Reduce 分布式计算框架的其他程序设计方法和性能优化措施。

[参考文献] (References)

[1] JIMMY L, CHRIS D. Data-Intensive Text Processing with Map/Reduce[Z]. University of Maryland: College park, April 11, 2010.
[2] THORSTEN Joachims. Optimizing Search Engines using Clickthrough Data[Z]. New York: ACM, 2002.
[3] UICHIN Lee, LIU Zhenyu, JUNGHOO Cho. Automatic Identification of User Goals in Web Search[Z]. New York: ACM, 2005.
[4] 王继明, 彭波. 搜索引擎用户点击行为分析[J]. 情报学报, 2006, 25(2): 12-21
[5] J Daniel E.Rose, Danny Levinson. Understanding User Goals in Web Search[Z]. New York: ACM. 2004.
[6] RALF Lammel. Google's MapReduce Programming Model-Revisited[J]. Science of Computer Programming, 2008, Volume 70: Page 1-30
[7] TOM Wbite. Hadoop: The Definitive Guide[M]. US: O'Reilly. 2005.
[8] DHRUBA Borthakur. The Hadoop Distributed File System: Architecture and Design[OL]. [2007] http://74.125.155.132/scholar?q=cache:SMfTUjMXFUQJ:scholar.google.com/&hl=zh-CN&as_sdt=2000
[9] 余慧佳, 刘奕群. 基于大规模日志分析的网络搜索引擎用户行为研究[J]. 中文信息学报, 2007, 10(1): 103-129