基于二维划分的杰卡德相似系数批量计算效率优化

廖 彬[1] 张 陶[2] 于 炯[3] 国冰磊[3] 刘 继[1]
(新疆财经大学统计与信息学院 乌鲁木齐 830012)[1]
(新疆医科大学医学工程技术学院 乌鲁木齐 830011)[2] (新疆大学软件学院 乌鲁木齐 830008)[3]

摘 要 随着互联网用户及内容的指数级增长,大规模数据场景下的杰卡德相似系数计算对算法的效率提出了更高的要求。为提高算法的执行效率,对MapReduce架构下的算法执行缺陷进行了分析,结合Spark适用于迭代型及交互型任务的特点,基于二维划分算法将算法从MapReduce平台移植到Spark平台;并通过参数调整、内存优化等方法进一步提高了算法的执行效率。两组数据集分别在3组不同规模的集群上的实验结果表明,与MapReduce相比,Spark平台下的算法执行效率提高了4倍以上,能耗效率提升了3倍以上。

关键词 绿色计算,MapReduce,任务调度,温度感知
中图法分类号 TP393.09 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2017.01.042

Efficiency Optimization of Jaccard's Similarity Coefficient Based on Two Dimensional Partition

LIAO Bin[1] ZHANG Tao[2] YU Jiong[3] GUO Bing-lei[3] LIU Ji[1]
(College of Statistics and Information, Xinjiang University of Finance and Economics, Urumqi 830012, China)[1]
(Department of Medical Engineering and Technology, Xinjiang Medical University, Urumqi 830011, China)[2]
(School of Software, Xinjiang University, Urumqi 830008, China)[3]

Abstract With the exponential growth of Internet users and content, the efficiency of the Jaccard's similarity coefficient algorithm under big data scenario is more important than ever before. In order to improve the efficiency of Jaccard's similarity computing process, the implementation that the algorithm was analyzed under MapReduce architecture. Combining the characteristics of the Spark is more suitable for the iterative and interactive tasks, we transformed the algorithm from the MapReduce platform to Spark based on two dimensional partition algorithm. And we improved the efficiency of the algorithm by parameter adjustment, memory optimization and other. methods. With two data sets running on 3 clusters with different number of datanodes, the experimental results show that, compared with MapReduce, the algorithm execution efficiency under Spark platform improves more than 4 times, and energy efficiency improves more than 3 times.

Keywords Green computing, MapReduce, Task scheduling, Temperature-aware

1 引言

从Google提出MapReduce[1]分布式计算模式至今,Google的数据中心平均每天运行的MapReduce作业量超过100000[2]。由于MapReduce能够部署在通用平台上,并且具有可扩展性(scalable)、低成本(economical)、高效性(efficient)与可靠性(reliable)等优点,这使得它在分布式计算领域得到了广泛运用,并且已逐渐成为工业界与学术届事实上的海量数据并行处理标准[3]。基于此,Yahoo,Amazon,eBay,Facebook,Twitter,IBM,LinkedIn,RackSpace等公司在机器学习[4]、数据挖掘[5]、社会网络计算[6,7]、bioinformatics[8]、astronomy[9]和fingerprint discrimination[10]等领域大量使用MapReduce。

但是,MapReduce的优势在于处理批处理作业,对于具有复杂业务处理逻辑的互联网数据挖掘类作业(如相似度计算、社会网络数据挖掘等),MapReduce的计算效率并不理想。另外,在能耗利用率方面,文献[11,12]的研究结果表明:MapReduce集群内部服务器存在严重的高能耗、低利用率问题。针对MapReduce这两方面的缺陷,已有研究通常围绕存储、索引及迭代计算等方面对MapReduce进行效率改进[13,14],而文献[15-22]则针对MapReduce及HDFS存在的能耗问题进行改进。但是,由于MapReduce的设计灵感来源于函数式编程语言(如Lisp,Scheme,ML等),在进行架构设计时就没有考虑到系统的能耗问题,导致已有的能耗优化效果并不显著。

在此背景下,高效率、低成本的大数据处理技术成为学术

到稿日期:2015-11-05 返修日期:2016-04-17 本文受国家自然科学基金项目(61562078,61262088,71261025),新疆财经大学博士启动基金(2015BS007)资助。

廖 彬(1986-),男,博士,副教授,CCF会员,主要研究方向为绿色计算、数据库系统理论及数据挖掘,E-mail:liaobin665@163.com;张 陶(1988-),女,硕士,讲师,主要研究方向为分布式计算、网格计算;于 炯(1964-),男,博士,教授,博士生导师,主要研究方向为网络安全、网格与分布式计算;国冰磊 女,硕士生,主要研究方向为绿色计算、数据库系统等;刘 继 男,博士,教授,硕士生导师,主要研究方向为信息管理及数据挖掘。

界及工业界的研究热点。Spark 站在巨人的肩膀上，依靠 Scala 强有力的函数式编程、Actor 通信模式、闭包、容器、泛型，借助统一资源分配调度框架 Mesos，融合了 MapReduce 和 Dryad，研究开发了一种简洁、直观、灵活、高效的大数据分布式处理框架。与 Hadoop 不同，Spark 一开始就瞄准性能，将数据（包括部分中间数据）放在内存，在内存中对其进行计算。用户将重复利用的数据缓存到内存，提高了下次的计算效率，因此 Spark 尤其适合于迭代型和交互型任务。

随着 MapReduce 计算框架的广泛应用，已有的社会网络数据挖掘工作大多在 MapReduce 框架下完成，但是在 MapReduce框架下的相似度计算在算法运行过程中面临网络开销过大从而导致性能低下的问题。为了大幅度提高社交网络中共同好友数计算的效率，本文基于 Spark 对内存计算及 DAG 调度的支持，采用图划分的思想，对杰卡德相似系数计算过程进行优化，从而提高算法的执行效率。

2 相关背景

2.1 Spark 平台应用相关研究

Spark 诞生于伯克利大学 AMPLab 实验室，Spark 的核心 RDD（Resilient Distributed Datasets）以及流处理、SQL 智能分析、机器学习等功能都脱胎于学术研究论文。与本文工作类似，现有的部分研究工作则利用 Spark 的效率优势，将已有成熟算法移植到 Spark 平台，以此提高算法的计算效率。例如，王虹旭等人[23]提出一种基于 Spark 云计算平台的并行数据分析系统，该系统相对于以往的图数据挖掘系统可以更高效地完成计算任务，而且可以有效分析非图数据。邱荣财[24]在 Spark 平台上实现了对 CURE 算法的并行化，在提高聚类效率的同时改善了聚类效果。王诏远等人[25]借助 Spark 内存、分布式计算的特点，提出一种并行的基于 Spark 的蚁群优化算法，将算法效率提高了 10 倍以上。郑凤飞等人[26]提出了 Spark 框架下的矩阵分解并行化算法，提高了协同过滤推荐算法在大数据规模下的执行效率。严玉良等人[27]针对大规模图算法的效率在单机环境低下及 Hadoop 不适合运行迭代式算法的问题，提出了一种基于 Spark 的大规模单图频繁子图挖掘算法 FSMBUS，通过次优树构建并行计算的候选子图，在给定最小支持度时挖掘出所有的频繁子图，并利用非频繁检测和搜索顺序选择实现优化。实验证明，FSMBUS 的效率要比现有单图上最新的算法高一个数量级。本文与以上工作思路类似，即结合 Spark 的优势，基于二维划分算法将杰卡德相似系数算法从 MapReduce 平台移植到 Spark 平台，并通过参数调整、内存优化等方法进一步提高算法的执行效率。

2.2 杰卡德相似系数计算概述

杰卡德相似系数（Jaccard Coefficient）[28]是社会网络中计算任意两个节点之间影响强度的最基本方法。影响强度（Influence Strength）研究作为社交网络研究的一个重要方向，是指用户之间相互影响的能力。杰卡德相似系数依靠计算两个节点之间的共同好友（或邻居）的数目来度量它们之间的影响强度，设有节点 A 与 B，其共同好友的数目越多，说明影响强度越大，其数学表达式为：

$$JC(A,B)=\frac{|n_A \cap n_B|}{|n_A \cup n_B|} \tag{1}$$

其中，n_A 与 n_B 分别表示节点 A 与 B 的所有好友的集合。式(1)中，$n_A \cap n_B$ 表示 A 与 B 两个节点所有好友集合的交集，即计算共同好友数。$n_A \cup n_B$ 表示 A 与 B 两个节点所有好友集合的并集，即计算两个集合并集的元素个数。杰卡德相似系数的计算中最关键的步骤是计算两个好友之间共同好友的数目。

社会网络通常用图来表示，即 $G=\{V,E\}$，其中 $V=\{u_1,u_2,u_3,\cdots,u_n\}$ 表示所有节点的集合（设总共有 n 个节点），$E=\{\langle u_i \rightarrow u_j\rangle \mid u_i \in V, u_j \in V\}$ 表示任意两个节点间的好友关系的集合，例如：$u_i \rightarrow u_j$ 表示节点 u_j 是节点 u_i 的好友（而节点 u_i 则不一定是节点 u_j 的好友）。在实际的社会网络系统中，设表 User(uid)与表 Relation(uid, fid)分别存储节点的集合 V 与关系集合 E 的数据（与本文算法中的无关字段均省略）。在社会计算应用中，大多数采用 MapReduce 框架进行并行计算，在 MapReduce 框架上，利用 Hive 计算任意两个节点 A 与 B 的杰卡德相似系数可分解为以下 4 个基本步骤。

步骤 1 计算节点 A 的好友数量，设其值为 *CountA*。可用以下 SQL 进行实现：

```
SELECT count(uid) as CountA
FROM Relation r
WHERE r. uid=A;
```

步骤 2 计算节点 B 的好友数量，设其值为 *CountB*。可用以下 SQL 进行实现：

```
SELECT count(uid) as CountB
FROM Relation r
WHERE r. uid=B;
```

步骤 3 计算节点 A 与 B 的共同好友的数量，设其值为 *CountAB*。与步骤 1 和步骤 2 中计算 *CountA* 和 *CountB* 不同，计算 *CountAB* 的值并不能简单地通过 1 条 SQL 语句完成，特别在进行批量计算时，至少需要 1 次 Group（分组）及 2 次 Join（连接）操作。本文将在第 3 节对该计算过程进行详细阐述。

步骤 4 在步骤 1－步骤 3 的基础上，通过式(1)计算出节点 A 与节点 B 之间的杰卡德相似系数，其值为：

$$JC(A,B)=\frac{|CountAB|}{|CountA+CountB-CountAB|} \tag{2}$$

通过以上分析可知，杰卡德相似系数的批量计算瓶颈主要在于节点间共同好友数量的计算，本文将在下一节对其算法进行分析。

3 基于 MapReduce 的共同好友数算法分析

批量的共同好友数算法的计算过程如图 1 所示，其中图 1(a)所示为一个简单的社会网络，图 1(b)将其转化为 Relation(uid, fid)的关系型表存储形式，图 1(c)所示为最终的计算结果表 Mutual(uid, fid, mutual)，其中字段 mutual 为主键，表示 uid 与 fid 两个节点的共同好友数。

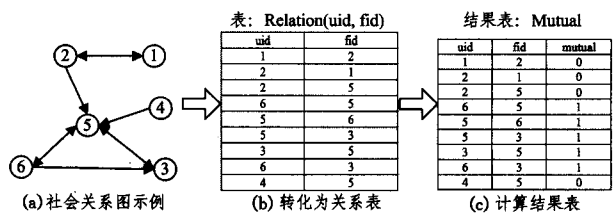

图 1 共同好友计算示意图

在 MapReduce 编程模式下，利用 Hive 处理图 1(b)向图 1(c)的详细计算过程，如图 2 所示，从计算流程图可以看出，计算共同好友需要以下 4 个步骤。

步骤 1　将输入数据表 Relation(uid，fid)进行 Group by(即分组)操作，得到分组表(uid，fids)。

步骤 2　输入数据表 Relation(uid，fid)与步骤 1 中分组得到的分组表(uid，fids)进行 Join by(即连接)操作，得到临时表：Temp 表 1 (uid，fid，uid-fids)。

步骤 3　将步骤 1 得到的分组表(uid，fids)与步骤 2 中得到的 Temp 表 1 (uid，fid，uid-fids)进行第二次 Join by 操作得到第二张临时表：Temp 表 2 (uid，fid，uid-fids，fid-fids)。

步骤 4　对步骤 3 得到的 Temp 表 2 (uid，fid，uid-fids，fid-fids)中的 uid-fids 与 fid-fids 字段进行交操作。最终得到结果表：Mutual(uid，fid，mutual)。

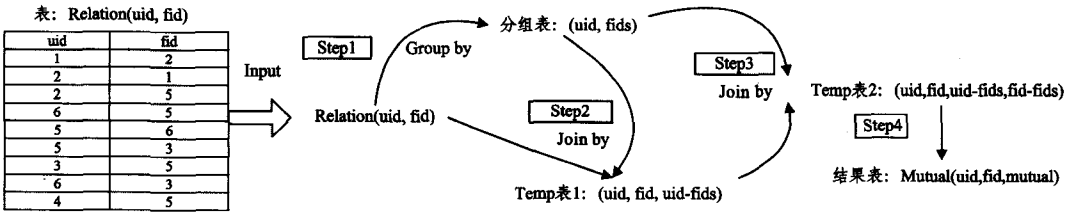

图 2　共同好友数计算流程图

通过以上对 SQL 语句的执行流程的分解可知，当计算的数据量上亿(即表 Relation(uid，fid)的数据条数过亿)时，Step1 中的分组操作以及 Step2 与 Step3 两个步骤中的 Join 操作成为性能的主要瓶颈。当进行数据量上亿的实验时，发现 Join 操作带来了大量的网络数据传输及磁盘的频繁读写操作，整个网络及磁盘数据的读写负载压力较大。正因为如此，当集群中的网络或磁盘发生故障时，容易造成任务的失败，并且作业的运行时间过长会造成用户等待时间过长。

4　基于 Spark 的相似度计算优化算法

通过 Hive 执行 SQL 来进行相似度计算，当数据量上亿时，在中等规模的 MapReduce 集群(节点数 10～50)中计算的时间往往需要数小时，并且面临任务容易失败的问题(特别在集群不稳定的情况下)。所以，需要对 MapReduce 架构下的算法进行优化。本文提出基于节点关系表分区的优化与基于二维划分的两种优化算法。

基于节点关系表分区的优化算法是将节点关系表根据一定规则拆分成多个子表，分别独立地针对子表进行计算，最后将子表中的计算结果汇总，得到最终的结果。通过实验发现，虽然基于节点关系表分区的优化算法提升了执行效率，但是作业的总执行时间仍然较长，优化效果并不十分明显。所以，下文主要对基于二维划分的优化算法进行介绍。

4.1　基于二维划分的优化算法

通过图 2 中对算法执行流程的分析以及对实验过程中资源使用情况的监控，发现数据的网络及磁盘 I/O 主要发生在 Group 及 Join 操作期间，并且包含大量的重复数据复制操作(由任务的切分造成)。为了解决以上问题，本文提出基于二维划分的优化算法，算法主要步骤分解如下。

步骤 1　关系表 Relation(uid，fid)的矩阵转化。

将关系表 Relation(uid，fid)转化为一个大的矩阵 $M(n\times n$，其中 n 为关系网络图中节点个数)，矩阵的两个维度都用于表示具体的对象，矩阵的元素 m_{ij} 表示节点 i 和节点 j 是否在关系表中存在关联，如果有关联，则 m_{ij} 的值为 1；否则，m_{ij} 的值为 0。图 3 所示为将图 1 中的示例数据转化为矩阵 M 的示例，示例中的对象(节点)个数为 6，即转化后的矩阵维度为 6×6。

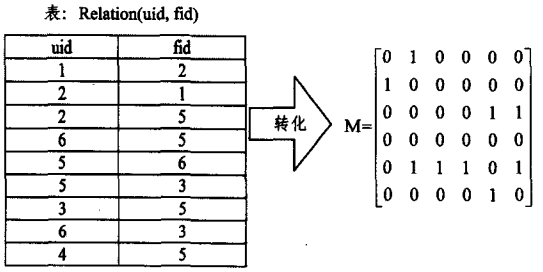

图 3　关系表的矩阵转化示例

步骤 2　矩阵分区。

将步骤 1 中得到的矩阵 M 进行二维划分。假设将示例中的 M 矩阵划分为 4 个区，划分方法及每个元组所对应的分区如图 4 所示。

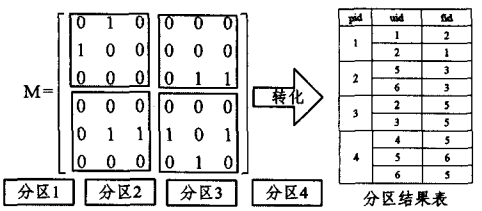

图 4　分区划分方法及转化示例

步骤 3　得到分区结果表。

在步骤 2 分区的基础上，对关系表 Relation(uid，fid)进行处理，得到分区结果表。即在关系表的最前面加入列 pid 表示分区 id。其分区结果表如图 4 所示。

步骤 4　得到分区路由表。

根据步骤 3 中得到的分区结果表中的 uid 列表，计算出每个 uid 所在的分区列表，即得到分区路由表，该表记录了每个 id 所在的分区信息，其转化过程如图 5 所示。

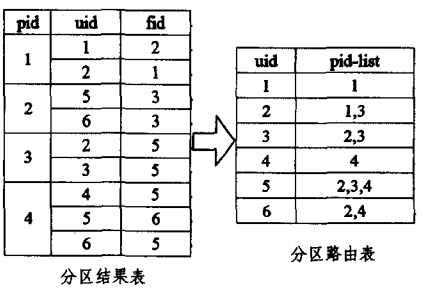

图 5　分区结果表转化分区路由表

步骤 5 将关系表 Relation(id,fid)进行分组操作,得到分组表(uid,fid-list)。其分组过程如图 6 所示。

表: Relation(uid, fid)

uid	fid
1	2
2	1
2	5
6	5
5	6
5	3
3	5
6	3
4	5

group by uid

分组表: (uid, fid-list)

uid	fid-list
1	2
2	1,5
3	5
4	5
5	3,6
6	5,3

图 6 关系表分组

步骤 6 将步骤 4 与步骤 5 中的分区路由表与分组表进行链接(Join)操作,得到临时表:(uid,pid-list,fid-list)。其计算 Join 过程如图 7 所示。

分区路由表

uid	pid-list
1	1
2	1,3
3	2,3
4	4
5	2,3,4
6	2,4

分组表: (uid, fid-list)

uid	fid-list
1	2
2	1,5
3	5
4	5
5	3,6
6	5,3

Join by uid

uid	pid-list	fid-list
1	1	2
2	1,3	1,5
3	2,3	5
4	4	5
5	2,3,4	3,6
6	2,4	5,3

图 7 分区路由表与分组表的 Join 操作

步骤 7 将步骤 6 中得到的(uid,pid-list,fid-list)根据 pid 进行分组操作,可得到临时表:(pid,uid,fid-list),即记录了每个分区中每个用户的所有好友的列表。其计算过程如图 8 所示。

临时表: (uid, pid-list, fid-list)

uid	pid-list	fid-list
1	1	2
2	1,3	1,5
3	2,3	5
4	4	5
5	2,3,4	3,6
6	2,4	5,3

临时表: (pid, uid, fid-list)

pid	uid	fid-list
1	1	2
1	2	1,5
2	3	5
2	5	3,6
2	6	5,3
3	2	1,5
3	3	5
3	5	3,6
4	4	5
4	5	3,6
4	6	5,3

图 8 得到的临时表(pid,uid,fid-list)

步骤 8 将步骤 7 中计算得到的临时表(pid,uid,fid-list)与步骤 3 中得到的分区结果表(pid,uid,fid)进行 Join 操作,最终得到最终结果 mutual 表。其过程如图 9 所示。

通过以上基于二维划分的优化算法,对在 Group 及 Join 操作期间的网络及磁盘 I/O 进行了优化,并且减少了因任务的切分而造成的重复数据复制操作。假设关系表 Relation(uid,fid)中的某条数据在分区 1 中出现了 n 次,基于二维划分的算法只会将数据发送 1 份到分区 1 中,而基于 Hive 的 SQL 计算方法会发送 n 次,从而产生冗余的网络传输。4.2 节将通过实验证明算法的有效性。

pid	uid	fid
1	1	2
	2	1
2	5	3
	6	3
3	2	5
	3	5
4	4	5
	5	6
	6	5

pid	uid	fid-list
1	1	2
1	2	1,5
2	3	5
2	5	3,6
2	6	5,3
3	2	1,5
3	3	5
3	5	3,6
4	4	5
4	5	3,6
4	6	5,3

Join by pid

uid	fid	mutual
1	2	0
2	1	0
2	5	0
6	5	1
5	6	1
5	3	1
3	5	1
6	3	1
4	5	0

图 9 Join 得到 mutual 表

4.2 相似度计算算法移植到 Spark 环境

为了最大程度提高算法的计算效率,特将本文算法从 Hadoop 移植到 Spark,移植的原因主要有以下两点:

(1)Spark 的中间数据直接缓存到内存中,相对 Hadoop 其迭代运算效率更高。Spark 相比 Hadoop 更适合做迭代运算。因为 Spark 里的 RDD(Resilient Distributed Dataset)直接 cache 到内存中。每次对 RDD 数据集操作之后的结果都存放到内存中,下一个操作可以直接从内存中输入,相比 MapReduce 省去了大量的磁盘 IO 操作。

(2)Spark 比 Hadoop 更具有灵活性。与 Hadoop 只提供 Map 和 Reduce 两种基本操作不同,Spark 提供的数据集操作类型则更为丰富,比如 map, filter, flatMap, sample, groupByKey, reduceByKey, union, join, cogroup, mapValues, sort, partionBy 等,而这些操作可统称为 Transformations。并且 Spark 同时还提供 Count, collect, reduce, lookup, save 等多种 action。Spark 这些多种多样的数据集操作类型给上层应用编程提供了更多的自由度。各处理节点之间的通信模型不再像 Hadoop 那样只是唯一的 Data Shuffle 模式。用户可以命名、物化、控制中间结果的分区等,从而 Spark 的编程模型比 Hadoop 更灵活。

5 评价与比较

5.1 实验环境

为了对算法性能效果进行对比,实验选取了两组数据集分别在 3 组不同规模的集群上进行测试,实验的详细配置环境如表 1 所列。

表 1 总体实验环境描述

项目	描述
操作系统	Debian 7.0
JVM 版本	1.6 for Linux
Hadoop 版本	1.0.4
Spark 版本	1.2
能耗数据测量	北电电力监测仪(USB 智能版),标准为 GB/T17215-2003,功率误差值±0.01～0.1W,采样频率为 1.5～3s 之间,单位为 kWh
能耗数据采集	电力监测仪用电监测管理系统 V1.0.1
能耗相关单位	功率:瓦特(W),能耗:焦耳(J)
数据采样频率	1 秒采集数据 1 次
节点 CUP	Intel core2 duo E8400 3.00GHz
节点内存	2GB-DDR2-800MHZ (1GB*2)
节点硬盘	320GB 7200 转/s
网卡信息	Realtek RTL8168/8111 PCI-E Gigabit Ethernet NIC-100Mbps

其中，两组数据集的规模分别为 1 亿与 2 亿左右（关系表中的数据量）；3 组集群的机器数分别为 5，10，20。对能耗数据进行测量，实验采用北电电力监测仪（USB 智能版），数据采样频率设置为 1 秒/次，各节点能耗数据（包括瞬时功率、电流值、电压值、能耗累加值等）可通过 USB 接口实时地传输到能耗数据监测机中。在软件版本选择方面，Hadoop 版本为 1.0.4，Spark 版本为 1.2。

下文将从算法效率、算法移植后的平台优化以及算法能耗优化 3 个方面展开。

5.2　算法完成效率对比

为了减小实验的误差，每组实验执行 10 次，最后取均值作为结果。对比实验一共有 6 组，变化的参数是数据规模与集群规模，分别在优化前的 Hadoop 集群中（基于 Hive 的 SQL 相似度计算算法）与优化后的 Spark 中（基于二维划分的算法）进行实验对比。得到的最终完成时间对比如表 2 所列。

表 2　算法完成时间对比

编号	数据量（亿）	实验规模	Hadoop 执行时间（s）	Spark 执行时间（s）
1	1	5	10108	1912
2	1	10	4680	777
3	1	20	2935	613
4	2	5	19330	6407
5	2	10	9744	2393
6	2	20	5096	1810

如表 2 中算法完成时间对比所列，由于 Spark 在内存及迭代计算中相比 MapReduce 在架构模式上更具优势，对应实验编号，Spark 比 Hadoop 在任务完成时间效率方面分别提高约 5.29，6.02，4.79，3.02，4.07，2.82 倍；平均效率提升 4.33 倍。Hadoop 集群中，当数据规模为 1 亿时，随着集群规模的递增，将集群规模为 5 时作为参考，算法效率随着集群规模的增加而提高。Spark 平台中，当数据规模为 1 亿时，随着集群规模的递增，将集群规模为 5 时作为参考，算法效率也随着集群规模的增加而提高。由此可见，无论是 Hadoop 还是 Spark，它们都可通过增加集群节点数目达到减少算法执行时间的目的。

图 10 所示为 Spark 优化后的网络传输及磁盘 I/O 情况，由于采用基于二维划分的优化算法，对在 Group 及 Join 操作期间的网络及磁盘 I/O 进行了优化，减少了因任务的切分而造成的重复数据复制操作。当数据规模为 2 亿时，Hadoop 平台中任务运行时产生大量的网络数据传输及磁盘的频繁读写操作，整个网络及磁盘数据的读写总量超过了 100GB。而在 Spark 中，采用二维划分的优化算法，网络及磁盘数据的读写总量控制到了 30GB 以下，从而减小了 300%以上的数据传输量。

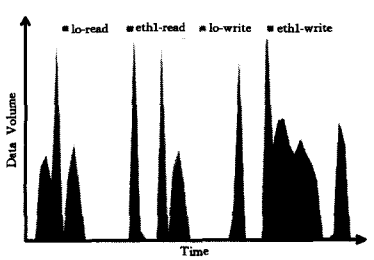

图 10　优化后的网络传输及磁盘 I/O 情况

5.3　算法移植后的平台优化

从实验过程中发现，相似度算法在 Spark 平台上具有较大的优化空间，优化的方法主要有分区参数优化、内存优化、集群规模优化及系统参数优化 4 个方面，优化方法所对应的说明如表 3 所列。

表 3

编号	优化方法	优化说明
1	分区参数优化	在相似度算法的计算过程中，分区数越多，会导致节点的复制份数增加，从而增大网络数据传输量。因此可基于中间结果的统计信息来确定分区个数，使得在充分利用节点内存与 CPU 资源的前提条件下，最小化分区数。
2	内存优化	由于计算数据量大，内存中对象数据庞大，从而导致内存使用量较高，GC 的时间周期较长。实践中可使用列存储格式来对内存数据进行压缩，以此减少内存中数据与对象的数量。在实践中，需要注意 Parallel GC 和 CMS GC 两种策略的选择，Parallel GC 注重更高的吞吐量，CMS GC 注重更低的延迟。对应于不同的策略，对 UseAdaptiveSizePolicy，SurvivorRatio，InitialSurvivorRatio，ParallelGCThreads，ConcGCThreads 等参数进行合理配置。
3	优化集群规模	随着集群规模的不断增大，集群网络连接复杂度也会成倍增加。当网络出现拥挤现象时容易带来连接超时，从而导致 shuffle 数据的拉取失败。更糟糕的情况是，网络超时会让 Master 误认为 Executor 已经丢失，故会使得整个 Executor 上已经完成的任务全部重做。因此在 shuffle 时增加网络超时重试机制，同时控制每次发送的请求连接数，避免 shuffle 读取数据超时，从而减少任务失败的次数，防止 Executor 丢失的情况。
4	系统参数优化	由于每个 Executor 进程还会使用堆外内存，因此Executor进程占用的内存往往会大于 JVM 设定的最大值，为了保证 Gaia 不会将超过 JVM 内存的 Executor 进程杀死，可通过配置参数 spark.yarn.executor.memoryOverhead 以避免这个问题。由于 Executor 在 Full GC 时需要较长时间，需要配置参数 spark.storage.blockManagerSlaveTimeoutMs 来延长 blockManager 的超时时间。在实验过程中，发现 Spark job 大多数运行失败问题都是由于系统资源不够用（和集群资源数量相关），通过监控日志等判断是由于内存资源占用过高导致的问题，可尝试通过调整配置参数的方法来解决。如：调整 spark.akka.frameSize，设置 spark.shuffle.manager = SORT 及增加 spark.yarn.executor.memoryOverhead 等。

图 11 所示为平台优化前后的任务完成时间对比，其中对比实验一共有 6 组，数据规模与集群规模参数如表 3 所列。

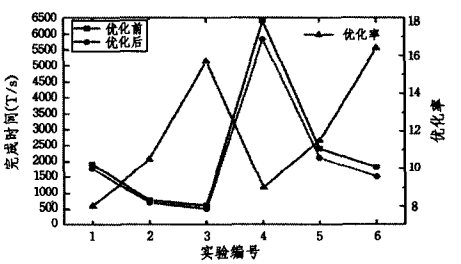

图 11　优化前后的任务完成时间对比

如图 11 所示，实验组 1－6 通过优化，将平均任务时间由原来的 1912s，777s，613s，6407s，2393s，1810s，优化到了 1759s，695s，517s，5830s，2118s，1531s；任务完成时间效率提升分别为：8%，10.5%，15.7%，9%，11.5%，16.5%，平均优化率为 11.85%。从图中优化率结果可以看出集群节点数越大，优化效果越明显。

5.4　算法能耗优化

无论算法的执行环境是 Hadoop 还是 Spark，任务（作业）执行过程中消耗的资源包括 CPU、磁盘、内存、网络等，而任务的执行能耗则是由这些资源所产生的能耗累加而成。

设任意任务执行成功后的能耗为 E，由于任务执行过程中主要消耗了 CPU、磁盘、内存及网络资源，因此在任务执行

完毕的时间周期 $T=[t_s,t_e]$内，其能耗可由式(3)得到：

$$E=E_{cpu}+E_{mem}+E_{disk}+E_{net}$$
$$=\int_{ts}^{te}(a_{cup}\cdot u_{cpu}+\lambda_{cpu})+\int_{ts}^{te}(a_{mem}\cdot N+\lambda_{mem})+$$
$$\int_{ts}^{te}(a_r\cdot r+a_w\cdot w+\lambda_{disk})+\int_{ts}^{te}(a_s\cdot s+$$
$$a_v\cdot v+\lambda_{net}) \quad (3)$$

通过分析式(3)可知，可通过减小同一任务的执行时间及优化任务执行过程中的资源使用情况来达到优化任务能耗利用率的目的。本文通过电力监测仪动态地收集相似度计算算法执行过程中的能耗数据，从而得到算法优化前后的能耗效率对比。图 12 所示为 6 组实验在 3 种不同的情况(Hadoop 平台、Spark 平台及优化的 Spark 平台)下的能耗对比。

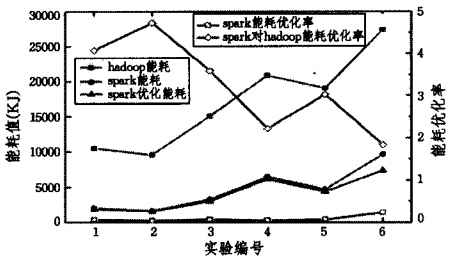

图 12 6 组实验能耗对比(KJ)

图 12 所示为每组实验能耗值及能耗优化率，结合式(3)的分析，虽然 Hadoop 平台上的平均功耗为 98～102W，相比 Spark 平台的平均功耗 118～125W 要小得多，但是由于在 Spark 平台上的执行时间比在 Hadoop 平台上的执行时间大大缩短，因此 Spark 平台的算法能耗效率比 Hadoop 高很多，并且由于系统的优化，能耗效率进一步提升。总体而言，将算法移植到 Spark 以后，与 Hadoop 相比，能耗利用效率提高了 3.25 倍以上；对 Spark 进一步优化后，能够再次提高能耗效率 5%以上。

值得注意的是，Hadoop 平台的平均功耗相比 Spark 平台的平均功耗要小得多，这是由于平台对资源的使用方式造成的，因为 Spark 使用 RDD 直接将数据 cache 到内存中，资源利用率(如 CPU、内存等)比 Hadoop 高；而 Hadoop 在执行任务的过程中不少节点之间存在任务等待的情况，资源利用效率不如 Spark。

结束语 高效率低成本的大数据处理技术成为学术界及工业界的研究热点。虽然 MapReduce 已逐渐成为工业界与学术届事实上的海量数据并行处理标准，但一方面由于 MapReduce的优势在于处理批处理作业，对于具有复杂业务处理逻辑的互联网数据挖掘类作业，MapReduce 的计算效率并不理想；另一方面，由于进行架构设计时没有考虑到系统的能耗因素，导致存在严重的高能耗低利用率问题。本文对大规模数据场景下的杰卡德相似系数计算进行研究，为提高算法的执行效率，首先对 MapReduce 架构下的算法执行缺陷进行了分析，结合 Spark 适于迭代型及交互型任务的特点，基于二维划分算法将算法从 MapReduce 平台移植到 Spark 平台，并通过参数调整、内存优化等方法进一步提高算法的执行效率。两组数据集分别在 3 组不同规模的集群上的实验结果表明，与 MapReduce 相比，Spark 平台上的算法的执行效率提高了 4 倍以上，能耗效率提升了 3 倍以上。

下一步工作则是在本文研究的基础上，将基于内容、基于规则及基于网络结构等推荐算法移植到 Spark 平台上，以此提高大数据环境下推荐算法的效率。

参 考 文 献

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

[2] DEAN J,GHEMAWAT S. MapReduce:a flexible data processing tool[J]. Communications of the ACM,2010,53(1):72-77.

[3] WANG P,MENG D,ZHAN J F,et al. Review of Programming models for data-Intensive computing[J]. Journal of Computer Research and Development,2010:47(11),1993-2002. (in Chinese)
王鹏，孟丹，詹剑锋，等. 数据密集型计算编程模型研究进展[J]. 计算机研究与发展，2010，47(11)：1993-2002.

[4] CHEN S,SCHLOSSER S W. Map-Reduce Meets Wider Varieties of Applications[R]. Intel Research Pittsburgh, Technical Report IRPTR-08-05,2008.

[5] WHITE B,YEN T,LIN J,et al. Web-Scale Computer Vision using MapReduce for Multimedia Data Mining[C]//Proceedings of the International Workshop on Multimedia Data Mining. 2010:1-10.

[6] X-RIME[OL]. [2015-10-15]. http://xrime. sourceforge. net.

[7] SHI J,XUE W,WANG W,et al. Scalable Community Detection in Massive Social Networks using MapReduce[J]. IBM Journal of Research and Development,2013,57(3/4):1-12.

[8] MATSUNAGE A,TSUGAWA M,FORTES J. CloudBLAST: Combining MapReduce and Virtualization on Distributed Resources for Bioinformatics Applications[C]// Proceedings of the IEEE International Conference on e-Science. IEEE, 2008: 222-229.

[9] WILEY K,CONNOLLY A,Gardner J,et al. Astronomy in the Cloud:Using Mapreduce for Image Co-addition[J]. Astronomy, 2011,123(901):366-380.

[10] LU W,HUANG J,HONG L. Massive Data MapReduce Fingerprint Discriminant Algorithm Based on Hadoop[J]. Applied Mechanics and Materials,2012,263:2655-2660.

[11] TIMES N Y. Power,Pollution and the Internet [EB/OL]. [2016-1-2]. http://www. nytimes. com/2012/09/23/technology/ data-ceneters-waste-vast-amounts-of-energy-belying-industry-image. html.

[12] BARROSO L A,HLZLE U. The datacenter as a computer:An introduction to the design of warehouse-scale machines [R]. Morgan:Synthesis Lectures on Computer Architecture,Morgan & Claypool Publishers,2009

[13] HUANG S,WANG B T,WANG G R,et al. A Survey on MapReduce Optimization Technologies[J]. Journal of Frontiers of Computer Science and Technology,2013,7(10):865-885. (in Chinese)
黄山，王波涛，王国仁，等. MapReduce 优化技术综述[J]. 计算机科学与探索，2013，7(10)：865-885.

[14] LIU Y,JING N,CHEN L,et al. Algorithm for processing k-nearest join based on R-tree in MapReduce[J]. Journal of Software, 2013,24(8):1836-1851. (in Chinese)
刘义，景宁，陈荦，等. MapReduce 框架下基于 R-树的 k-近邻连接算法[J]. 软件学报，2013，24(8)：1836-1851.

[15] GOIRI Í,LE K,NGUYEN T D,et al. GreenHadoop:leveraging

green energy in data-processing frameworks[C]// Proceedings of the 7th ACM European Conference on Computer Systems. ACM, 2012: 57-70.

[16] CARDOS M, SINGH A, PUCHA H, et al. Exploiting spatio-temporal tradeoffs for energy efficient MapReduce in the cloud [R]. Department of Computer Science and Engineering, University of Minnesota, 2010.

[17] CHE Y, GANAPATHI A, KATZ R H. To compress or not to compress-compute vs. io tradeoffs for mapreduce energy efficiency[C]// Proceedings of the First ACM SIGCOMM Workshop on Green Networking. ACM, 2010: 23-28

[18] SONG J, LI T T, ZHU Z L, et al. Benchmarking and analyzing the energy consumption of cloud data management system [J]. Chinese Journal of Computers, 2013, 36(7): 1485-1499. (in Chinese)
宋杰，李甜甜，朱志良，等. 云数据管理系统能耗基准测试与分析[J]. 计算机学报，2013，36(7)：1485-1499.

[19] LIAO B, YU J, SUN H, et al. Energy-efficient algorithms for distributed storage system based on data storage structure reconfiguration[J]. Journal of Computer Research and Development, 2013: 50(1): 3-18. (in Chinese)
廖彬，于炯，孙华，等. 基于存储结构重配置的分布式存储系统节能算法[J]. 计算机研究与发展，2013，50(1)：3-18.

[20] LIAO B, YU J, ZHANG T, et al. Energy-efficient algorithms for distributed file system HDFS[J]. Chinese Journal of Computers, 2013, 36(5): 1047-1064. (in Chinese)
廖彬，于炯，张陶，等. 基于分布式文件系统 HDFS 的节能算法[J]. 计算机学报，2013，36(5)：1047-1064.

[21] LIN J C, LEU F Y, CHEN Y. Impact of MapReduce Policies on Job Completion Reliability and Job Energy Consumption[J]. IEEE Transactions on Parallel and Distributed Systems, 2015, 26(5): 1364-1378.

[22] LIAO B, YU J, ZHANG Tao, et al. Energy-Efficient Algorithms for Distributed Storage System Based on Block Storage Structure Reconfiguration [J]. Journal of Network and Computer Applications, 2015, 48(2): 71-86.

[23] WANG H X, WU B, LIU Y. Parallel graph data analysis system based on Spark[J]. Journal of Frontiers of Computer Science and Technology, 2015, 9(9): 1066-1074. (in Chinese)
王虹旭，吴斌，刘旸. 基于 Spark 的并行图数据分析系统[J]. 计算机科学与探索，2015，9(9)：1066-1074.

[24] QIU R C. Parallel design and application of CURE algorithm based on Spark platform[D]. South China University of Technology, 2014. (in Chinese)
邱荣财. 基于 Spark 平台的 CURE 算法并行化设计与应用[D]. 广州：华南理工大学，2014.

[25] WANG Z Y, WANG H J, XING H L, et al. Ant colony optimization algorithm based on Spark[J]. Journal of Computer Applications, 2015, 35(10): 2777-2780. (in Chinese)
王诏远，王宏杰，邢焕来，等. 基于 Spark 的蚁群优化算法[J]. 计算机应用，2015，35(10)：2777-2780.

[26] ZHENG F F, HUANG W P, JIA M Z. Matrix factorization recommendation algorithm based on Spark[J]. Journal of Computer Applications, 2015, 35(10): 2781-2783. (in Chinese)
郑凤飞，黄文培，贾明正. 基于 Spark 的矩阵分解推荐算法[J]. 计算机应用，2015，35(10)：2781-2783.

[27] YAN Y L, DONG Y H, HE X M, et al. FSMBUS: A frequent subgraph mining algorithm in single large-scale graph using spark[J]. Journal of Computer Research and Development, 2015 (8): 1768-1783. (in Chinese)
严玉良，董一鸿，何贤芒，等. FSMBUS：一种基于 Spark 的大规模频繁子图挖掘算法[J]. 计算机研究与发展，2015(8)：1768-1783.

[28] SAMANTHULA B K, JIANG W. Secure Multiset Intersection Cardinality and its Application to Jaccard Coefficient[J]. IEEE Transactions on Dependable & Secure Computing, 2016, 13(5): 1.

(上接第 207 页)

[6] HINZ S, SCHMIDT K, STAHL C. Transforming BPEL to Petri Nets[C]// International Conference on Business Process Management. 2005: 220-235.

[7] STAHL C. A Petri Net Semantics for BPEL: Technical Repert 188[R]. Humboldt University Zu Berlin, Institut for Informatik, 2005.

[8] HOLZMANN G J. The Spin Model Checker: Primer and Reference Manual[D]. Addison-Wesley, Boston, MA, USA: 2004.

[9] SCHMIDT K. LoL A-a low level analyser[C]// Proceedings of the 21st International Conference on Application and Theory of Petri Nets, Volume1 825 of Lecture Notes in Computer Science. Aarhus, Demnark: Springer, Verlag, 2000: 465-474.

[10] LOHMANN N. A Feature-Complete Petri Net Semantics or WS-BPEL2. 0[C]// Web Services and Formal Methods International Workshop(WSFM07). 2007: 77-91.

[11] SUN Xi-long. Research on Web Service Composition Testing Based on BPEL[D]. Beijing: Beijing University of Technology, 2009. (in Chinese)
孙喜龙. 基于 BPEL 的 Web 服务组合测试研究[D]. 北京：北京工业大学，2009.

[12] YU Bo. Application of Petri Net to Improve the Correctness of BPEL Program[J]. Application Research of Computers, 2011 (28): 3348-3350. (in Chinese)
余波. 应用 Petri 网改进 BPEL 程序的正确性[J]. 计算机应用研究，2011(28)：3348-3350.

[13] LUO Xiang-yu, TAN Zheng, SU Kai-le, et al. A Web Service Composition Verification Method Based on Cognitive Model Detection[J]. Chinese Journal of Computers, 2011 (34): 1041-1061. (in Chinese)
骆翔宇，谭征，苏开乐，等. 一种基于认知模型检测的 Web 服务组合验证方法[J]. 计算机学报，2011(34)：1041-1061.

[14] SUN Lin, LIU Jiu-fu, YANG Zhen-xing. Software Test Case Generation Method Based on Petri Net[J]. Computer Measurement & Control, 2010(18): 2019-2022. (in Chinese)
孙琳，刘久富，杨振兴. 基于 Petri 网的软件测试用例生成方法[J]. 计算机测量与控制，2010(18)：2019-2022.

[15] MOU Xiao-ling. Research on Service Composition Testing Based on Extended Colored Petri Nets[D]. Chongqing: Southwest University, 2012. (in Chinese)
牟小玲. 基于扩展着色 Petri 网的服务组合测试研究[D]. 重庆：西南大学，2012.