DOI:10.13471/j.cnki.acta.snus.2017.03.008

Spark DAG 优化 MapReduce 协同过滤算法*

廖彬[1]，张陶[2,3]，于炯[2]，国冰磊[2]，张旭光[1]，刘炎[4]
（1. 新疆财经大学统计与信息学院，新疆 乌鲁木齐 830012；
2. 新疆大学信息科学与工程学院，新疆 乌鲁木齐 830008；
3. 新疆医科大学医学工程技术学院，新疆 乌鲁木齐 830011；
4. 清华大学软件学院，北京 100084）

摘 要：大数据的规模效应给数据存储、管理以及数据分析带来了极大的挑战，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。为提高协同过滤算法的执行效率，对 MapReduce 架构下的算法执行步骤进行了分解，并对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，提出将算法从 MapReduce 平台移植 Spark 平台的改进思路。设计了算法在 Spark 中的实现流程，并通过参数调整、内存优化等方法进一步提高算法效率。实验结果表明：与 MapReduce 平台中的算法相比，基于 Spark DAG 调度的算法能够减少 65% 以上的 HDFS 重复 I/O 操作，执行效率与能耗效率分别提升近 200% 及 50%。

关键词：协同过滤；MapReduce；Spark；算法优化；能耗优化
中图分类号：TP393.09 **文献标志码**：A **文章编号**：0529－6579（2017）03－0046－11

Optimization of collaborative filtering algorithm based on DAG Spark scheduling

LIAO Bin[1], *ZHANG Tao*[2,3], *YU Jiong*[2], *GUO Binglei*[2], *ZHANG Xuguang*[1], *LIU Yan*[4]
（1. College of Statistics and Information, Xinjiang University of Finance and Economics, Urumqi 830012, China;
2. School of Information Science and Engineering, Xinjiang University, Urumqi 830008, China;
3. Department of Medical Engineering and Technology, Xinjiang Medical University, Urumqi 830011, China;
4. School of Software, Tsinghua University, Beijing 100084, China）

Abstract: The scale effect of big data has brought great challenges to data storage, management and analysis. And the high efficiency and low cost big data processing technology has become a hotspot research in academia and industry. In order to improve the efficiency of collaborative filtering algorithms, the implementation of the algorithm under the MapReduce architecture is decomposed in order to analysis the defects of the algorithm. For the Spark suitable for the iterative and interactive tasks, this paper presents the methods to improve the execution efficiency from the MapReduce platform to the Spark platform. The implementation flow of the algorithm in Spark is designed, and efficiency is improved by parameter adjustment and memory optimization. Experimental results show that: based on spark DAG scheduling,

* **收稿日期**：2016－04－26
基金项目：国家自然科学基金（61562078，61262088）；新疆维吾尔自治区自然科学基金（2016D01B014）；新疆财经大学博士启动基金（2015BS007）
作者简介：廖彬（1986 年生），男；**研究方向**：绿色计算、数据挖掘及大数据计算模型等；E-mail：liaobin665@163.com

the algorithm can reduce more than 65% HDFS I/O operations and enforce the efficiency and energy efficiency were increased by nearly 200% and 50%.

Key words: collaborative filtering; MapReduce; Spark; algorithm optimization; energy consumption optimization

大数据时代，数据从简单的处理对象开始转变为一种基础性资源，如何更好地管理和利用大数据已经成为普遍关注的话题，大数据的规模效应给数据存储、管理以及数据分析带来了极大的挑战[1]。据文献［2］统计，2007 年全球数据量达到 281EB，而 2007 - 2011 年这 5 年时间内，全球数据量增长了 10 倍。数据量的高速增长伴随而来的是存储与处理系统规模不断的扩大，这使得运营成本不断的提高，其成本不仅包括硬件、机房、冷却设备等固定成本，还包括 IT 设备与冷却设备的电能消耗等其它开销。以至于文献［1］中指出：在能源价格上涨、数据中心存储规模不断扩大的今天，高能耗已逐渐成为制约云计算与大数据快速发展的一个主要瓶颈。一方面，虽然 MapReduce 逐渐成为工业与学术界事实上的海量数据并行处理标准，但 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 计算效率并不理想。另外，在能耗利用效率方面，文献［3 - 4］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。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[5-6]，而文献［7 - 18］则针对 MapReduce 或 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），并且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，最后产生了一个简洁、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）放在内存中进行计算。将重复利用的数据缓存到内存，以此提高计算效率，因此 Spark 尤其适合迭代型和交互型任务。但是随着 MapReduce 计算框架的广泛应用，已有的协同过滤算法大多基于 MapReduce 框架实现。与文献［3 - 4］研究结果一致，我们在前期研究[19-20]中发现 MapReduce 框架下的协同过滤算法执行效率较为低下。为了提高协同过滤算法的效率，本文做了如下工作：首先，对 MapReduce 框架下的协同过滤算法的实现步骤进行分解，在此基础上对算法的效率缺陷进行了分析；其次，在对 MapReduce 效率分析的基础上将算法从 MapReduce 移植到 Spark 平台，并对实现流程、移植方法及平台优化进行了研究；最后，通过实验证明了算法移植后在任务执行时间及能耗效率等方面的提升。

1　相关研究

协同过滤推荐算法的输入可以表示为 $m \times n$ 的用户项目评分矩阵（如表 1 所示）。其中 m 表示用户数量，n 表示物品数量，R_{ij} 表示第 i 个用户对第 j 个物品的评分（其中 $i \in [1,m]$，$j \in [1,n]$）。R_{ij} 值的范围可以根据实际应用系统的特点进行设定，连续或离散的情况都有，大多应用系统中采用数值为 1 ~ 5 的离散值来表示评分，而 0 则表示用户尚未对该项目进行评分。

表 1　用户项目评分矩阵

Table 1　User item score matrix

	Item_1	Item_2	…	Item_n
User_1	R_{11}	R_{12}	…	R_{1n}
User_2	R_{21}	R_{22}	…	R_{2n}
…	…	…	…	…
User_m	R_{m1}	R_{m2}	…	R_{mn}

传统的协同过滤系统主要分为基于用户、物品及混合协同过滤 3 种类型。其中基于用户的协同过滤的主要思想是：如果用户 i 没有对项目 j 进行评分，则找到与用户 i 最相似的 k 个邻居用户（通常采用 Pearson 相关系数计算用户之间的相似性）；然后利用这 k 个邻居用户对项目 j 的评分加权平均来预测用户对项目 j 的评分值 R_{ij}。采用 Pearson 相关系数计算用户 x 与 y 的相似度公式为：

$$\text{sim}(x,y) = \frac{\sum_{i \in Ixy}(R_{x,i} - \bar{r}_x)(R_{y,i} - \bar{R}_y)}{\sqrt{\sum_{i \in Ixy}(R_{x,i} - \bar{R}_x)^2 \sum_{i \in Ixy}(R_{y,i} - \bar{R}_y)^2}} \quad (1)$$

设用户 u 对物品 i 的评分值 $R_{u,i}$ 计算公式为：

$$R_{u,i} = R_u + \frac{1}{\sum_{u' \in U} \mathrm{sim}(u,u')} \cdot \sum_{u' \in U} \mathrm{sim}(u,u') \cdot (R_{u',i} - R_{u'}) \quad (2)$$

其中 R_u 表示用户 u 对所有商品评分的平均分，而 $\frac{1}{\sum_{u' \in U} \mathrm{sim}(u,u')}$ 为归一化因子。

基于物品过滤的协同过滤主要思想是：如果用户 i 对项目 j 没有评分，设 $R_{ij}=0$；找到与物品 j 最相似的 k 个近邻（通常采用余弦距离计算相似性）；然后利用 k 个邻居对项目 j 的评分的加权平均来预测用户 i 对项目 j 的评分。利用基于物品过滤的协同过滤方法计算用户 u 对商品 i 的评分公式为：

$$R_{u,i} = \frac{\sum_{i' \in \mathbf{N}} \mathrm{sim}(i,i') \cdot R_{u,i'}}{\sum_{i' \in \mathbf{N}} \mathrm{sim}(i,i')} \quad (3)$$

其中 N 表示商品 i 的近邻集合，相似性计算公式可采用余弦距离。混合协同过滤则是在基于用户的协同过滤基础上，在计算两个用户之间的相似度时嵌入了 item-based CF 思想，对相似性的度量进行了改进。

除了以上 3 种常用协同过滤算法之外，还有基于内容的协同过滤[21-22]、基于规则的协同过滤[23-24]、基于人口统计信息的协同过滤[25-26]、基于网络结构的协同过滤[27-28]及混合过滤[20, 29-34]等。以上这些协同过滤研究工作的目的都是以提高算法在特定应用领域的推荐质量为首要目标，很少针对算法的运行平台、算法平台之间的移植及算法的执行效率进行研究。在此背景下，本文以基于 Item的协同过滤算法为例。首先对 MapReduce 架构下的协同过滤算法运行方式的缺陷进行了分析；利用 Spark 相比于 MapReduce 的优点，将算法从 MapReduce平台移植到 Spark 平台，从而较大程度上的提高算法的效率（包括时间、资源利用、能耗等效率）。

2 基于 MapReduce 的协同过滤算法及缺陷分析

本文以基于 Item 的协同过滤算法为例，分析在 MapReduce 平台下实现协同过滤算法存在的问题。之所以选择基于 Item 的协同过滤算法为例，这是由于 Item 与 Item 之间的相似度比较稳定，并且可以进行离线计算，可实现定期更新功能，这使得基于 Item的协同过滤算法在电子商务领域运用更为广泛。

首先，基于 Item 的协同过滤算法认为相似的 Item获得同一用户的评价也应该相似。所以，算法的基本思路是：首先计算用户（User）对物品（Item）的喜好程度；然后根据用户的喜好，计算 Item之间的相似度；最后找出与每个 Item 最相似的前 N 个 Item 推荐给用户。在进行算法实现时，可将算法切分为以下 7 个 MapReduce 任务，具体的切分如表 2 所示。

表 2 基于 Item 的系统过滤算法 MapReduce 实现步骤切分

Table 2 Item based system filtering algorithm MapReduce implementation steps

步骤编号	MapReduce 作业编号	功能	说明
Step1	MR JOB1	计算用户喜好	由于不同用户对 Item 的评分值可能具有较大的差异性，可将评分做二元化处理
Step2		计算 Item 的相似性，可进一步分解为以下 3 个小的步骤（Sub2_ 1 – Sub2_ 3）。	可采用 Jaccard 系数作为计算两个 Item 的相似性方法，也可以采用其他的相似性计算算法
Sub2_ 1	MR JOB2	Item 好评数统计	计算出 Item 的好评用户数
Sub2_ 2	MR JOB3	Item 好评的键值对 < key，value > 统计	计算任意两个有关联 Item 的相同好评用户数
Sub2_ 3	MR JOB4	Item 相似性计算	计算任意两个有关联物品 Item 之间的相似度
Step3		找出最相似的前 N 个 Item，可进一步分解为以下 3 个步骤	首先对 Item 的相似度归一化后整合，然后求出每个 Item 最相似的前 N 个 Item
Sub3_ 1	MR JOB5	Item 相似性归一化	进行 Item 的相似度归一化后整合计算
Sub3_ 2	MR JOB6	Item 相似性评分整合	整合计算
Sub3_ 3	MR JOB7	TOP – N 计算	计算每个 Item 相似性最高的前 N 个 Item

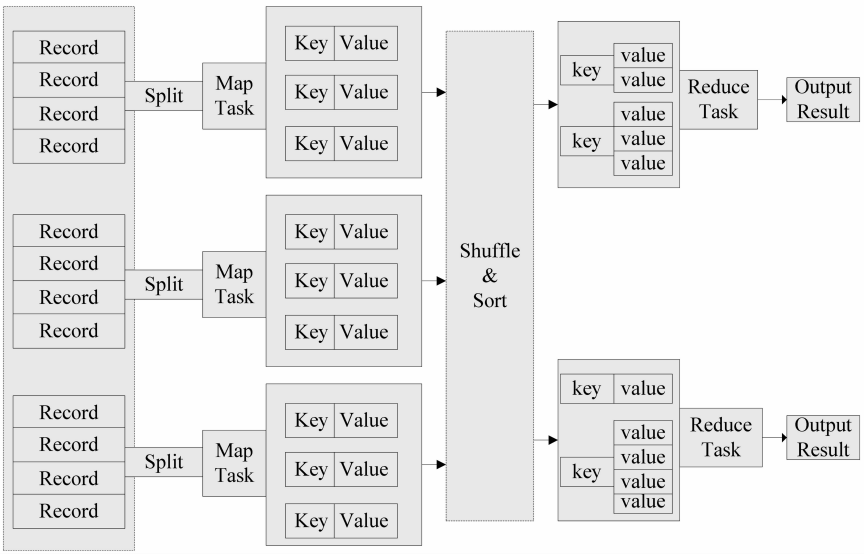

图 1　MapReduce 作业的任务分解

Fig. 1　MapReduce job decomposed into tasks

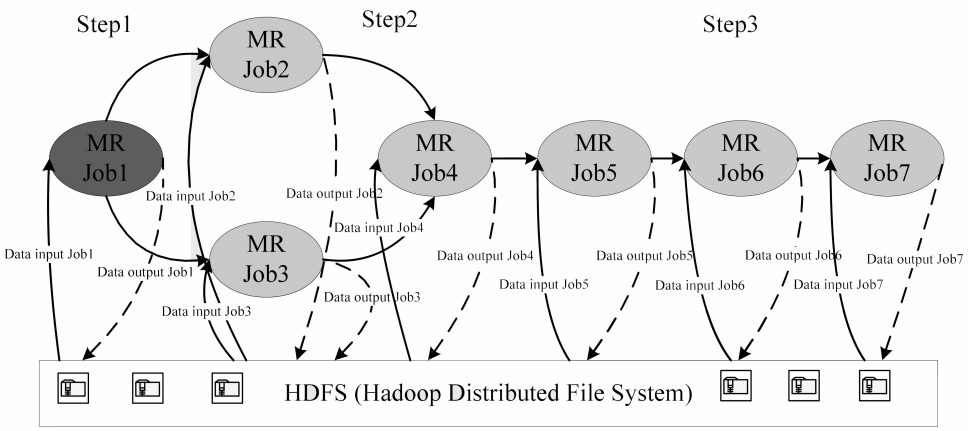

图 2　算法 MapReduce 作业关系图

Fig. 2　MapReduce Job relationship diagram

如表 2 所示，在 MapReduce 平台中实现基于 Item的协同过滤算法一共需要执行 7 个 MapReduce 作业（Job）。如图 1 所示为 MapReduce 的作业切分模型，每个 MapReduce 作业都会被切分为多个 Map 与 Reduce 任务（Task）。Map 任务执行过程中需要将数据从 HDFS 中读取进来，计算完毕后再通过 Shuffle 与 Sort 操作将 < key，value > 键值对发送给 Reduce 任务。而 Reduce 阶段则以 < key，Iterator < value > > 作为数据输入，计算完毕后将处理结果写入到 HDFS 中。

如图 2 所示为算法 7 个作业之间的联系图，由于基于 Item 的协同过滤算法一共有 7 个 MapReduce 作业，意味着算法需要进行 7 次的 HDFS 读取及写入操作，这些重复的数据读写操作耗费集群资源的同时降低了算法执行的效率。并且，MapReduce 作业在集群中进行调度，当集群资源出现“竞争”，或任务之间出现“等待”现象时，将对算法的执行效率产生影响。为了解决这些 MapReduce 计算效率低效的问题，本文将在下一章对基于 Spark DAG 调度的优化算法进行详细介绍。

3　基于 Spark DAG 调度的优化算法

本章一共分为 3 个小节，首先将协同过滤算法从 MapReduce 移植 Spark 平台的优势进行了分析；其次介绍了在 Spark 平台中算法的执行流程；最后介绍通过分区参数优化、内存优化及集群规模优化等方法提高算法执行效率。

3.1 算法移植 Spark 平台原因

本文将推荐算法从 Hadoop 平台移植到了 Spark 平台，移植的原因主要有以下 3 点：

（i）Spark 的中间数据直接缓存到内存中，对于 Hadoop 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 数据集的操作之后的结果，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 省去了大量的重复磁盘 IO 操作。

（ii）Spark 实现业务逻辑比 Hadoop 更灵活。

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，partitionBy 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。各个处理节点之间的通信模型不再像 Hadoop 那样就是唯一的 Data Shuffle 一种模式。用户可以命名，物化，控制中间结果的分区等，使得 Spark 的编程模型比 Hadoop 更灵活。

（iii）Spark 编程代码更为简洁。

由于 Spark 在设计时模仿了 Scala 的函数式编程风格及 API。而 MapReduce 的设计灵感来自于函数式编程语言 LISP。所以，在进行 MapReduce 向 Spark 移植的过程其本质是两种编程风格的移植。尽管 Spark 的主要抽象是 RDD，实现了 map，reduce 等操作，但这些都不是 Hadoop 的 Mapper 或 Reducer API 的直接模仿。这些转变也往往成为将 Mapper 和 Reducer 类平行迁移到 Spark 的障碍。与 Spark 中经典函数语言实现的 map 和 reduce 函数相比，原有 Hadoop 提供的 Mapper 和 Reducer API 更灵活也更复杂。如下代码所示：在实际的移植过程中，Spark 省略了 Hadoop 如下繁琐的 Map 与 Reduce 函数代码，只需直接编写业务实现代码。

```
public class MyMapper extends
Mapper < LongWritable, Text, IntWritable, IntWritable > {
protected void map (LongWritable lineNumber, Text line, Context context)
throws IOException, InterruptedException {
//Map 函数业务代码
}
} public class MyReducer extends Reducer < IntWritable, IntWritable, IntWritable, IntWritable > {
protected void reduce (IntWritable length, Iterable < IntWritable > counts,
Context context) throws IOException, InterruptedException {
//Reduce 函数业务代码
}
}
```

Spark 中，输入只是 String 构成的 RDD，而不是 key-value 键值对。Spark 中对 key-value 键值对的表示是一个 Scala 的元组，用（A，B）这样的语法来创建。Spark 的 RDD API 有 reduce 方法，但它会将所有 key-value 键值对 reduce 为单个 value。这并不是 MapReduce 编程模型中的行为，Spark 中与之对应的是 reduceByKey。

3.2 基于 DAG 调度的算法实现

经过上节的分析，可知相比与 MapReduce，Spark 提供了更加灵活的 DAG（Directed Acyclic Graph）编程接口，包含 map、reduce 接口的同时，还增加了 filter、flatMap、union 等操作接口，使得编写 Spark 程序更加灵活方便。在 Spark 中实现基于 Item的协同过滤算法如图 3 所示。

如图 3 所示为基于 Item 的协同过滤算法在 Spark 中的实现流程，整个流程分为 8 个阶段，18 个 RDD 数据。将算法从 MapReduce 平台移植到 Spark 平台，并将算法流程从图 2 变换为图 3；至少在以下 3 个方面优化了算法执行时间及资源的利用。

（i）Spark 中的 RDD（Resilient Distributed Dataset）数据模型相对于存储在 HDFS 中的数据在读写方面更为高效。算法在 Spark 平台执行过程中的中间数据都以 RDD 的形式存储（实质是 RDD 分布存储于各 slave 节点的内存中），相对于 MapReduce 大大的减少了计算过程中读写磁盘的次数。另外，RDD 还提供了 Cache 机制，比如 RDD3 进行 Cache 后，RDD4 和 RDD7 都可以同时访问 RDD3 的数据。相对于 MapReduce 平台减少了 MR JOB2 和 MR JOB3（如图 3 所示）重复从 HDFS 中读取相同数据的问题。

（ii）Spark 作业启动后会立即申请所需的 Executor 资源，并且所有 Stage 的 Tasks 以线程的方式运行，共用 Executors 资源。这相对于 MapReduce 以心跳的方式管理 Slot 资源，Spark 申请资源的次数大大的减少，导致资源管理效率高于 MapReduce。

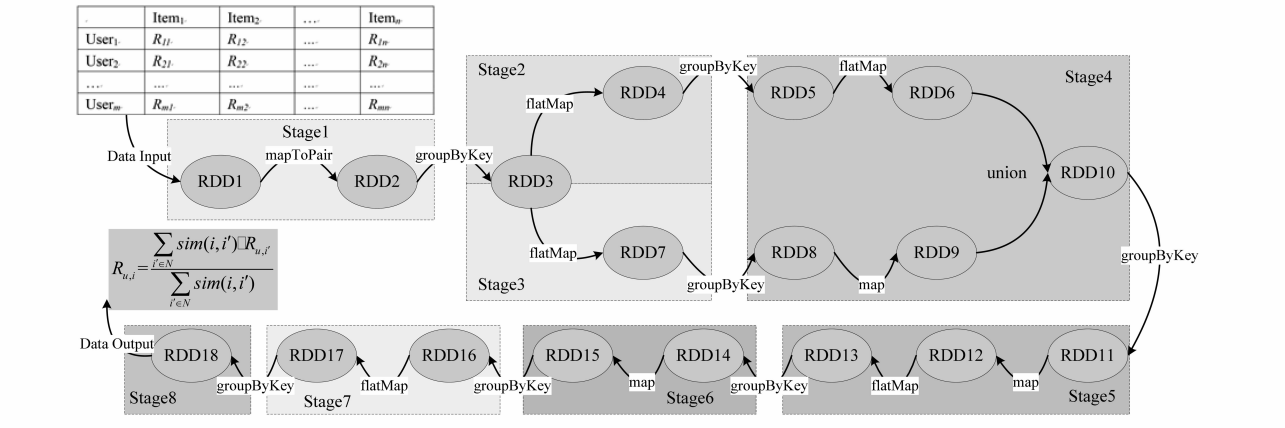

图 3　基于 Item 的协同过滤算法在 Spark 中的实现流程

Fig. 3　The implementation flow of item based collaborative filtering algorithm in Spark

（iii）Spark 实现方法最重要的改进是，MapReduce 中一共有 7 个 Job，而通过 Spark 中的 DAG 编程模型，可以实现将 7 个 MapReduce 作业简化为一个 Spark DAG 作业。如图所示，Spark 会将 DAG 作业分解为 8 个 Stage，每个 Stage 包含多个可并行执行的 Tasks。Stage 之间的数据通过 Shuffle 传递，而在算法的数据输入与输出方面，Spark 也只需要读取和写入 HDFS 一次，从而减少了 6 次 HDFS 数据的读写操作。

3.3　系统层级优化方法

实验过程中我们发现，算法在 Spark 平台执行过程可通过分区参数优化、内存优化、集群规模优化及系统参数优化 4 个方面对算法执行效率进行改进。优化方法所对应的说明如表 3 所示。

表 3　优化方法及说明

Table 3　Optimization method and description

编号	优化方法	优化说明
1	分区参数优化	任务计算过程中，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。需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。
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 进程还会使用到堆外内存，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。

4　实验评价与比较

4.1　实验环境

为了对算法改性效果进行对比，实验的测试数据量为 5 亿，其中数据的格式为（uid，itemId，ratingScore），含义为用户 uid 对 itemId 的评分结果为 ratingScore。实验在 2 组不同规模集群上进行测试，第 1 组节点规模为 10，第 2 组节点规模为 20。为了保证测试的有效性，每组测试实验重复进行 20 次。实验的配置参数如表 4 所示。

表 4 总体实验环境描述

Table 4 Description of experimental environment

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 ±0.01～0.1 W，采样频率为 1.5～3 s 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：W，能耗：J
数据采样频率	1s 采集数据 1 次
节点 CUP	Intel core2 duo E8400 3.00 GHz
节点内存	4 GB－ DDR2－800 MHz（2 GB＊2）
节点硬盘	Hitachi HDP725032GLA380（320 G 7 200 转/s）
网卡信息	Realtek RTL8168/8111 PCI－E Gigabit Ethernet NIC－ 100 Mbps

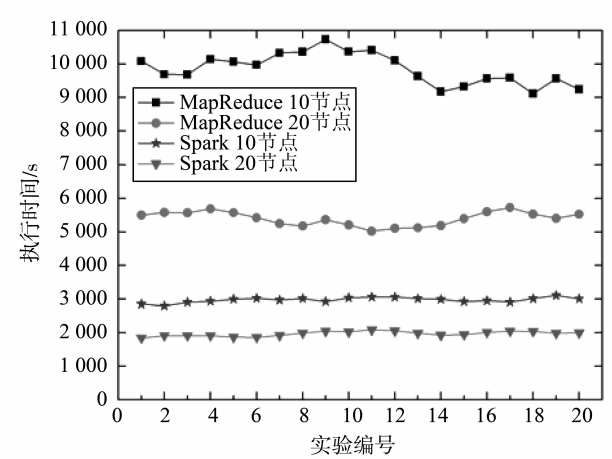

图 4 算法执行时间对比

Fig. 4 The algorithm execution time comparison

本文将从算法执行效率、算法移植后的平台优化以及算法能耗优化 3 个方面展开。

4.2 算法完成效率对比

实验一共分为 4 组，分别为 MapReduce 集群 10 节点、MapReduce 集群 20 节点、Spark 集群 10 节点及 Spark 集群 20 节点。为了减小实验误差，将每组实验至少成功执行 20 次。不同实验组的任务执行时间如图 4 所示。

如图 4 所示为算法执行时间对比，分别将 20 组实验执行时间求平均，得到 MapReduce 集群中 10 节点与 20 节点执行时间分别为 9 854、5 400 s；Spark 集群中 10 节点与 20 节点执行时间分别为 2 976、1 970 s。可以看出，集群节点数量的增加，分别提高 MapReduce 与 Spark 集群 82.5%、51.1% 的算法执行时间效率提升。MapReduce 与 Spark 算法效率比较，集群规模为 10 时，Spark 较 MapReduce 效率提升了近 231.1%；当集群规模为 20 时，效率提升了近 174.1%。由此可得出结论，借助于 Spark 的迭代计算和内存计算优势，将协同过滤算法从 MapReduce 移植到 Spark 环境，能够大幅度提升算法的执行效率。另外，我们在实验过程中发现，Spark 集群中的算法执行失败率小于 MapReduce 集群，这与集群的稳定性及任务执行时间密切相关。

算法资源利用方面，由于通过 Spark 中的 DAG 编程模型，将原始的 7 个 MapReduce 作业简化为一个 Spark DAG 作业。Spark 将 DAG 作业分解为了 8 个 Stage，每个 Stage 包含多个可并行执行的 Tasks。而 Stage 之间的数据通过 Shuffle 传递，而在算法的数据输入与输出方面，Spark 也只需要读取和写入 HDFS 一次，从而减少了 6 次 HDFS 数据的读写操作。在 MapReduce 集群中，任务运行时伴随大量的磁盘及网络 I/O 操作；而 Spark 中磁盘及网络 I/O 操作相比 MapReduce 得到了较好的控制，密集的 I/O 操作都发生在算法执行的前后。总体上，Spark 中的 DAG 模型减少了 65% 以上的 HDFS 重复读写 I/O 操作，提高资源利用效率的同时缩短了算法的执行时间。

4.3 Spark 平台优化效率

实验过程中发现针对平台做优化可提高算法的执行效率（算法在不同环境下的计算时间），常用的平台优化方法如表 3 所示。

如图 5 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 2 组，节点规模分别为 10 与 20，每组实验分别运行优化前后的算法 10 次进行对比实验。

图 5 及图 6 分别表示在不同集群规模环境下，平台优化前后的算法执行时间对比。表 5 对不同集群规模的优化效率进行了对比，其中当集群规模为 10 节点时，优化平均值为 10.62%；规模为 20 节点时，平均优化率为 11.57%。随着集群规模的增加，优化后的集群执行效率越高。

表 5　平台优化率对比

Table 5　Comparison of the rate of optimization platform

实验编号	10 节点优化率/%	20 节点优化率/%
1	8.98	17.20
2	11.73	15.35
3	12.92	16.35
4	10.45	7.71
5	14.45	6.85
6	5.10	8.95
7	8.42	13.69
8	12.75	7.86
9	7.16	7.39
10	14.20	14.37
平均值	10.62	11.57

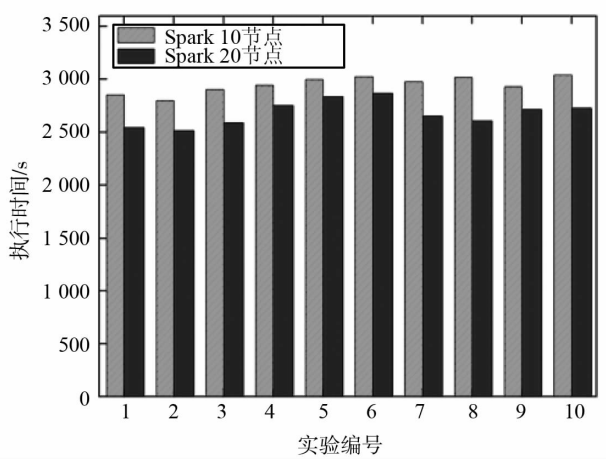

图 5　优化前后的任务完成时间对比
（10 节点集群规模）

Fig. 5　The task completion time comparison
(10 node clusters)

4.4　算法能耗优化

无论是 MapReduce 还是 Spark 平台，任务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。其中，CPU 的能耗模型受到处理器的活动状态、指令执行情况、高速 cache 使用情况、当前执行频率及制造工艺等因素的影响。设 P_{cpu} 表示 CPU 利用率为 ucpu 时的功率，根据 Kansal 等在文献［35］中提出的能耗模型，P_{cpu} 与 u_{cpu} 关系可表示如下：

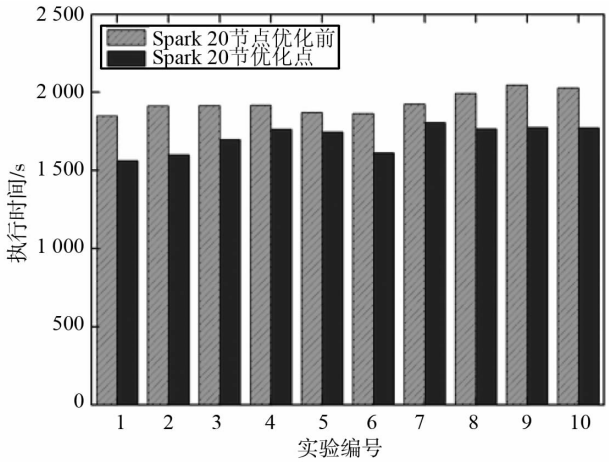

图 6　优化前后的任务完成时间对比
（20 节点集群规模）

Fig. 6　The task completion time comparison
(20 node clusters)

$$P_{cpu} = a_{cup} \cdot u_{cpu} + \lambda_{cpu} \tag{4}$$

其中，a_{cup} 与 λ_{cpu} 为 CPU 能耗模型的常数，不同的 CPU 之间存在着差异，其值可通过大量的训练获得。

已有大量工作针对内存的能耗模型进行了研究，发现影响内存能耗的主要因素是内存读写数据的吞吐量[23]。记录内存最后一层 Cache（Last Level Cache）的缺失次数，是一种轻量级的内存吞吐量评估方法。根据吞吐量指标，设 $E_{mem}(T)$ 表示内存在任务执行的 T 时间区间内的能耗，N 表示最后一层 Cache 缺失次数，内存模型表达如下[35]：

$$E_{mem}(T) = a_{mem} \cdot N + \lambda_{mem} \tag{5}$$

其中，a_{mem} 与 λ_{mem} 为内存能耗模型的常数。

磁盘能耗与读写数据量密切相关，设 $E_{disk}(T)$ 表示一条 SQL 执行过程中 T 时间区间内磁盘的能耗，r 与 w 分别表示在 T 时间区间内磁盘的读写数据量，磁盘能耗模型可表达如下[36-37]：

$$E_{disk}(T) = a_r \cdot r + a_w \cdot w + \lambda_{disk} \tag{6}$$

其中，a_r 与 a_w 分别表示磁盘读写参数，λ_{disk} 表示磁盘的静态能耗，3 个参数的值可通过大量的能耗数据分析与训练获得。

与磁盘能耗模型类似，网卡的能耗与发送、接收到的数据量成正比。设 $E_{net}(T)$ 表示网卡在 T 时间区间内（一条 SQL 的执行时间）的能耗，s 与 v 分别表示在 T 时间区间内网卡发送与接收到的数据量，

网卡能耗模型可表达如下：

$$E_{\text{net}}(T) = a_{\text{s}} \cdot s + a_{\text{v}} \cdot v + \lambda_{\text{net}} \quad (7)$$

其中，a_{s} 与 a_{v} 分别表示网卡在发送、接收数据时的能耗参数，λ_{net} 为网卡的静态能耗参数。

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，所以在任务执行完毕的时间周期 $T = [t_{\text{s}}, t_{\text{e}}]$ 内，其能耗可由下式得到：

$$\begin{aligned} E = E_{\text{cpu}} + E_{\text{mem}} + E_{\text{disk}} + E_{\text{net}} = \\ \int_{ts}^{te} (a_{\text{cup}} \cdot u_{\text{cpu}} + \lambda_{\text{cpu}}) + \\ \int_{ts}^{te} (a_{\text{mem}} \cdot N + \lambda_{\text{mem}}) + \\ \int_{ts}^{te} (a_{\text{r}} \cdot r + a_{\text{w}} \cdot w + \lambda_{\text{disk}}) + \\ \int_{ts}^{te} (a_{\text{s}} \cdot s + a_{\text{v}} \cdot v + \lambda_{\text{net}}) \end{aligned} \quad (8)$$

通过（8）式可知，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两方面达到优化任务能耗利用率的目的。本文中通过电力监测仪动态的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与优化后的能耗效率（不同环境完成相同任务所需能耗的比值）对比。如图 7 所示为 6 组实验在 3 种（Hadoop 平台下、Spark 平台下及优化后的 Spark 平台下）不同的情况下的能耗对比。

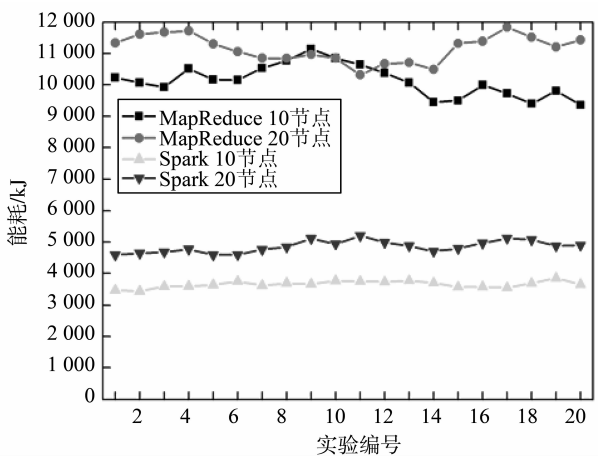

图 7 实验能耗对比

Fig. 7 The energy consumption comparison

如图 7 所示，结合公式（8）的分析，虽然 Hadoop 平台上的平均功耗在 101 ~ 105 W 之间，相比 Spark 平台的平均功耗 121 ~ 125 W 要小不少，但是由于 Spark 平台上的执行时间比 Hadoop 时间大大缩短。所以，Spark 平台的算法能耗效率比 Hadoop 有所提高，并且由于系统的优化，能耗效率进一步的提升。当集群节点数为 10 时，Spark 平台相比 MapRedcue能耗效率提高 63.92%；当集群节点数为 20 时，Spark 平台相比 MapRedcue 能耗效率提高 56.45%。在执行相同规模的任务时（5 亿数据量），执行平台相同时，MapReduce 平台节点规模为 20 时相对规模为 10 时能耗值有所提高，导致能耗效率下降 10.45%；Spark 平台与 MapReduce 现象一致，并且能耗效率下降了 33.03%。这表明执行相同任务时，集群规模越大，能耗效率越低，这是因为当集群规模增大时，任务之间的数据传输增加，任务之间的关联变得复杂，任务协调成本增加，从而导致作业执行能耗增加。

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不少。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。

5 结论及展望

虽然 MapReduce 逐渐成为工业与学术界事实上的海量数据并行处理标准，但 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 计算效率并不理想。并且，由于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工作效果并不显著。由于在前期研究中发现 MapReduce 框架下的协同过滤算法执行效率较为低下。本文首先对 MapReduce 框架下的协同过滤算法的实现步骤进行分解，在此基础上对算法的效率缺陷进行了分析；进而在对 MapReduce 效率分析的基础上将算法从 MapReduce 移植到 Spark 平台，并对实现流程、移植方法及平台优化进行了研究；最后，通过实现证明了算法移植后在任务执行时间及能耗利用率方面的有效性。下一步工作主要是在本文的基础上，将 MapReduce 平台中，基于内容、基于规则及基于网络结构等更多的协同过滤算法移植到 Spark 平台。

参考文献：

[1] 孟小峰，慈祥. 大数据管理：概念、技术与挑战[J]. 计算机研究与发展，2013，50(1)：146 - 149.
MENG X F, CI X. Big data management: concepts, techniques and challenges [J]. Journal of Computer Research

and Development, 2013: 50(1): 146 - 149.

[2] GANTZ J, CHUTE C, MANFREDIZ A, et al. The diverse and exploding digital universe: An updated forecast of worldwide information growth through 2011 [EB/OL]. [2015 - 2 - 25]. http: // wwww. ifap. ru/ library/ book268. pdf.

[3] TIMES N Y. Power, pollution and the internet [EB/OL]. [2016 - 5 - 20]. http: //www. nytimes. com/2012/09/23/ technology/data-ceneters-waste-vast-amounts-of-energy-belying-industry-image. html.

[4] 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.

[5] 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.
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.

[6] 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.
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.

[7] CHEN Y, KEYS L, KATZ R H. Towards energy effcient mapreduce [R]. Berkeley: EECS Department, University of California, 2009 - 10 - 9.

[8] LEVERICH J, KOZYRAKIS C. On the energy (in) efficiency of hadoop clusters [J]. ACM SIGOPS Operating Systems Review, 2010, 44 (1): 61 - 65.

[9] KAUSHIK R T, BHANDARKAR M. GreenHDFS: Towards an energy-conserving, storage-efficient, hybrid hadoop compute cluster [C]//Proceedings of the 2010 International Conference on Power Aware Computing and Systems. Piscataway, NJ: IEEE, 2010: 1 - 9.

[10] KAUSHIK R T, BHANDARKAR M, NAHRSTEDT K. Evaluation and analysis of GreenHDFS: A self-adaptive, energy conserving variant of the hadoop distributed file system [C]//Proceedings of the 2nd IEEE International Conference on Cloud Computing Technology and Science. Piscataway, NJ: IEEE, 2010: 274 - 287.

[11] 彭利民. 高效节能的虚拟网络重构算法[J]. 中山大学学报(自然科学版), 2015, 54(5): 5 - 10.
PENG L M. An energy efficient virtual network reconfiguration algorithm [J]. Acta Scientiarum Naturalium Universitatis Sunyatseni, 2015, 54(5): 5 - 10.

[12] WIRTZ T, GE R. Improving MapReduce energy efficiency for computation intensive workloads [C] // Green Computing Conference and Workshops (IGCC), 2011 International. IEEE, 2011: 1 - 8.

[13] 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.

[14] CHEN Y, GANAPATHI A, KATZ R H. To compress or not to compress-compute vs. IO tradeoffs for mapreduce energy efficiency [C] // ACM SIGCOMM Workshop on Green Networking, New Delhi, 2010: 23 - 28.

[15] 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.
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.

[16] 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.
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.

[17] 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.
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.

[18] LIN J C, LEU F Y, CHEN Y P. Impact of MapReduce policies on job completion reliability and job energy consumption [J]. IEEE Transactions on Parallel & Distributed Systems, 2015, 26(5): 1364 - 1378.

[19] LIAO B, YU J ZHANG T, 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.

[20] 杨兴耀, 于炯, 吐尔根·依布拉音, 等. 融合奇异性和扩散过程的协同过滤模型[J]. 软件学报, 2013,24 (8):1868 - 1884.
YANG X Y, YU J, IBRAHIM T, et al. Collaborative filtering model fusing singularity and diffusion process [J]. Journal of Software, 2013, 24(8): 1868 - 1884.

[21] GHAUTH K I, ABDULLAH N A. Learning materials recommendation using good learners' ratings and content-based filtering [J]. Educational Technology Research and Development, 2010, 58(6):711 - 727.

[22] UDDIN M N, SHRESTHA J, JO G S. Enhanced content-based filtering using diverse collaborative prediction for

movie recommendation [C]// Intelligent Information and Database Systems, 2009. ACIIDS 2009. First Asian Conference on. IEEE, 2009: 132 – 137.

[23] NGUYEN A T, DENOS N, BERRUT C. Improving new user recommendations with rule-based induction on cold user data [C]// ACM Conference on Recommender Systems. ACM, 2007: 121 – 128.

[24] CHUN J, OH J Y, KWON S, et al. Simulating the effectiveness of using association rules for recommendation systems [C]// Asian Simulation Conference on Systems Modeling and Simulation: Theory and Applications. Springer-Verlag, 2004: 306 – 314.

[25] QIU L Y, BENBASAT I. A study of demographic embodiments of product recommendation agents in electronic commerce [J]. International Journal of Human-Computer Studies, 2010, 68(10): 669 – 688.

[26] CHEN T, HE L. Collaborative Filtering Based on Demographic Attribute Vector [C]// International Conference on Future Computer and Communication. IEEE, 2009: 225 – 229.

[27] JIA C X, LIU R R, SUN D, et al. A new weighting method in network-based recommendation [J]. Physica A Statistical Mechanics & Its Applications, 2008, 387(23): 5887 – 5891.

[28] ZHOU T, REN J, MEDO M, et al. Bipartite network projection and personal recommendation [J]. Physical Review E Statistical Nonlinear & Soft Matter Physics, 2007, 76(2): 046115.

[29] YAMASHITA A, KAWAMURA H, SUZUKI K. Adaptive fusion method for user-based and item-based collaborative filtering [J]. Advances in Complex Systems, 2011, 14(2): 133 – 149.

[30] LIU Z B, QU W Y, LI H T, et al. A hybrid collaborative filtering recommendation mechanism for P2P networks [J]. Future Generation Computer Systems, 2010, 26(8): 1409 – 1417.

[31] BARRAGANS-MARTINEZ A B, COSTA-MONTENEGRO E, BURGUILLO J C, et al. A hybrid content-based and item-based collaborative filtering approach to recommend TV programs enhanced with singular value decomposition [J]. Information Sciences, 2010, 180(22): 4290 – 4311.

[32] PHAM M C, CAO Y W, KLAMMA R, et al. A clustering approach for collaborative filtering recommendation using social network analysis [J]. Journal of Universal Computer Science, 2011, 17(4):583 – 604.

[33] LIU F K, LEE H J. Use of social network information to enhance collaborative filtering performance [J]. Expert Systems with Applications, 2010, 37(7): 4772 – 4778.

[34] KAGIE M, van der LOOS M, VAN WEZEL M. Including item characteristics in the probabilistic latent semantic analysis model for collaborative filtering [J]. AI Communications, 2009, 22(4): 249 – 265.

[35] KANSAL A, ZHAO F, LIU J, et al. Vitrual machine power metering and provisioning [C]// Proceedings of the 1st ACM Symposium on Cloud Computing. Indianaposlis, USA, 2010:39 – 50.

[36] BAO Y, CHEN M, RUAN Y, et al. HMTT: A platform independent full-system memory trace monitoring system [J]. ACM SIGMETRICS Performance Evaluation Review, 2008, 36(1): 229 – 240.

[37] 廖彬, 张陶, 于炯, 等. 适应节能与异构环境的MapReduce数据布局策略[J]. 中山大学学报(自然科学版), 2015, 54(6): 55 – 66.
LIAO B, ZHANG T, YU J, et al. An energy-efficient and heterogeneous environment adaptive data layout strategy for MapReduce [J]. Acta Scientiarum Naturalium Universitatis Sunyatseni, 2015, 54(6): 55 – 66.