协同过滤推荐算法在MapReduce与Spark上实现对比

腾讯大数据 浪尖聊大数据 1周前

MapReduce为大数据挖掘提供了有力的支持，但是复杂的挖掘算法往往需要多个MapReduce作业才能完成，多个作业之间存在着冗余的磁盘读写开销和多次资源申请过程，使得基于MapReduce的算法实现存在严重的性能问题。大处理处理后起之秀Spark得益于其在迭代计算和内存计算上的优势，可以自动调度复杂的计算任务，避免中间结果的磁盘读写和资源申请过程，非常适合数据挖掘算法。腾讯TDW Spark平台基于社区最新Spark版本进行深度改造，在性能、稳定和规模方面都得到了极大的提高，为大数据挖掘任务提供了有力的支持。

本文将介绍基于物品的协同过滤推荐算法案例在TDW Spark与MapReudce上的实现对比，相比于MapReduce，TDW Spark执行时间减少了66%，计算成本降低了40%。

算法介绍

互联网的发展导致了信息爆炸。面对海量的信息，如何对信息进行刷选和过滤，将用户最关注最感兴趣的信息展现在用户面前，已经成为了一个亟待解决的问题。推荐系统可以通过用户与信息之间的联系，一方面帮助用户获取有用的信息，另一方面又能让信息展现在对其感兴趣的用户面前，实现了信息提供商与用户的双赢。

协同过滤推荐（Collaborative Filtering Recommendation）算法是最经典最常用的推荐算法，算法通过分析用户兴趣，在用户群中找到指定用户的相似用户，综合这些相似用户对某一信息的评价，形成系统对该指定用户对此信息的喜好程度预测。协同过滤可细分为以下三种：

- User-based CF: 基于User的协同过滤，通过不同用户对Item的评分来评测用户之间的相似性，根据用户之间的相似性做出推荐；
- Item-based CF: 基于Item的协同过滤，通过用户对不同Item的评分来评测Item之间的相似性，根据Item之间的相似性做出推荐；

- Model-based CF: 以模型为基础的协同过滤（Model-based Collaborative Filtering）是先用历史资料得到一个模型，再用此模型进行预测推荐。

问题描述

输入数据格式：Uid，ItemId，Rating （用户Uid对ItemId的评分）。

输出数据：每个ItemId相似性最高的前N个ItemId。

由于篇幅限制，这里我们只选择基于Item的协同过滤算法解决这个例子。

算法逻辑

基于Item的协同过滤算法的基本假设为两个相似的Item获得同一个用户的好评的可能性较高。因此，该算法首先计算用户对物品的喜好程度，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。该算法的详细描述如下：

- 计算用户喜好：不同用户对Item的评分数值可能相差较大，因此需要先对每个用户的评分做二元化处理，例如对于某一用户对某一Item的评分大于其给出的平均评分则标记为好评1，否则为差评0。
- 计算Item相似性：采用Jaccard系数作为计算两个Item的相似性方法。狭义Jaccard相似度适合计算两个集合之间的相似程度，计算方法为两个集合的交集除以其并集，具体的分为以下三步。

1)Item好评数统计，统计每个Item的好评用户数。

2)Item好评键值对统计，统计任意两个有关联Item的相同好评用户　　数。

3)Item相似性计算，计算任意两个有关联Item的相似度。

- 找出最相似的前N个Item。这一步中，Item的相似度还需要归一化后整合，然后求出每个Item最相似的前N个Item，具体的分为以下三步。

1)Item相似性归一化。

2)Item相似性评分整合。

3)获取每个Item相似性最高的前N个Item。

基于MapReduce的实现方案

使用MapReduce编程模型需要为每一步实现一个MapReduce作业，一共存在包含七个MapRduce作业。每个MapReduce作业都包含Map和Reduce，其中Map从HDFS读取数，输出数据通过Shuffle把键值对发送到Reduce，Reduce阶段以<key，Iterator<value>>作为输入，输出经过处理的键值对到HDFS。其运行原理如图1 所示。

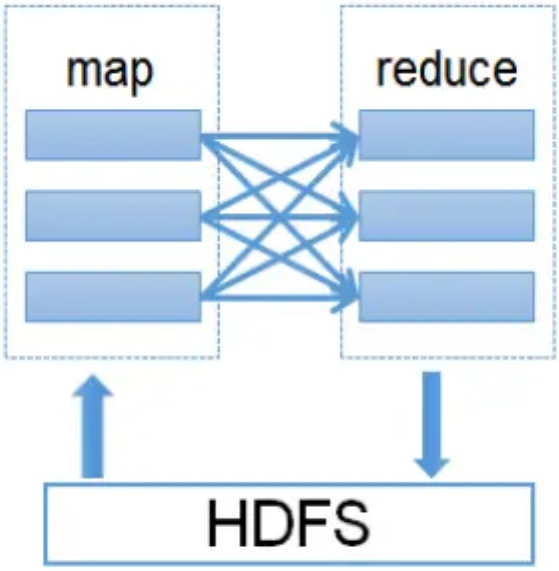

图 1

七个MapReduce作业意味着需要七次读取和写入HDFS，而它们的输入输出数据存在关联，七个作业输入输出数据关系如图2所示。

图 2

基于MapReduce实现此算法存在以下问题：

- 为了实现一个业务逻辑需要使用七个MapReduce作业，七个作业间的数据交换通过HDFS完成，增加了网络和磁盘的开销。
- 七个作业都需要分别调度到集群中运行，增加了Gaia集群的资源调度开销。
- MR2和MR3重复读取相同的数据，造成冗余的HDFS读写开销。

这些问题导致作业运行时间大大增长，作业成本增加。

基于Spark的实现方案

相比与MapReduce编程模型，Spark提供了更加灵活的DAG（Directed Acyclic Graph）编程模型，不仅包含传统的map、reduce接口，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。使用Spark编程接口实现上述的业务逻辑如图3所示。

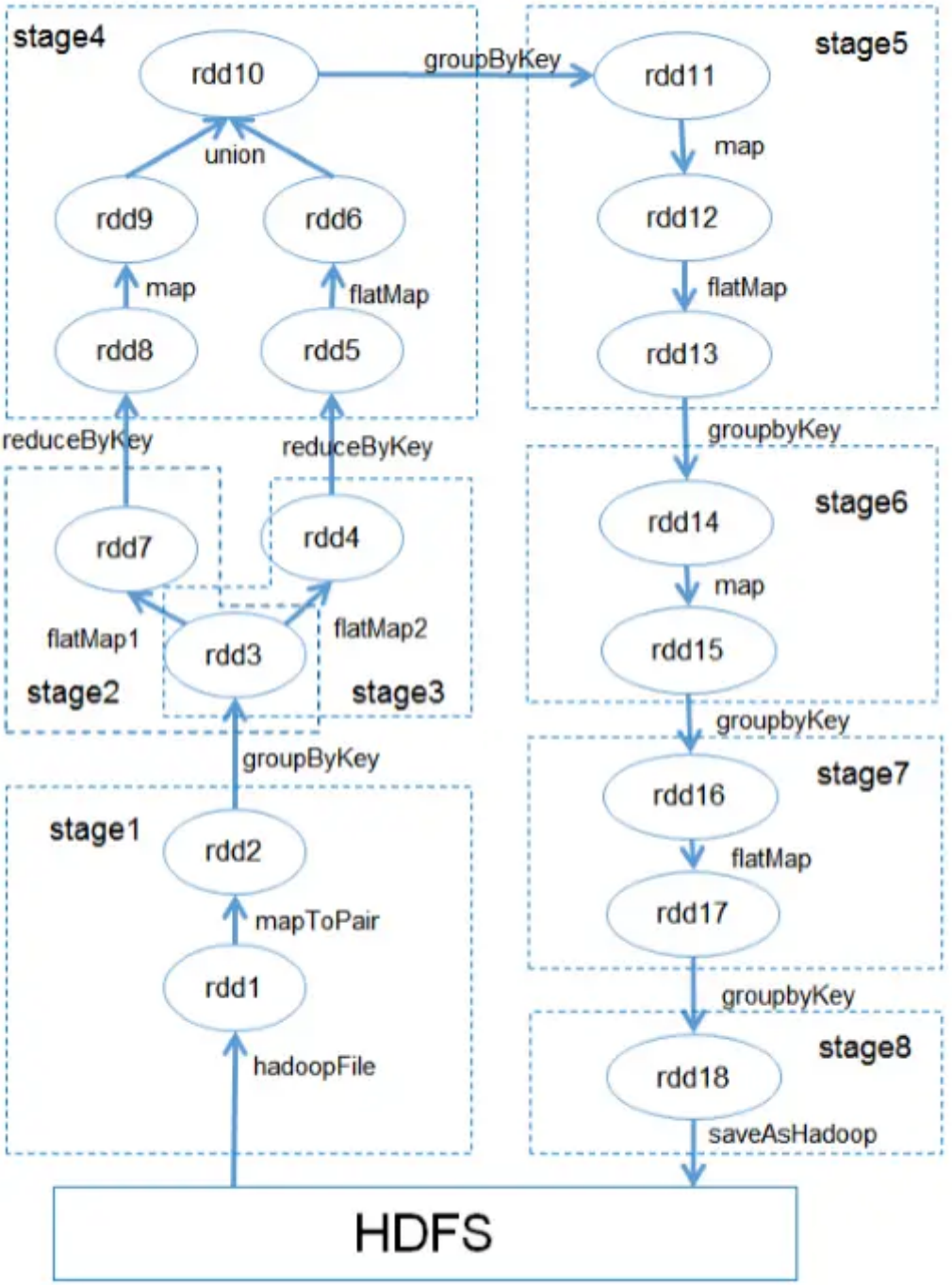

图 3

相对于MapReduce，Spark在以下方面优化了作业的执行时间和资源使用。

- DAG编程模型。通过Spark的DAG编程模型可以把七个MapReduce简化为一个Spark作业。Spark会把该作业自动切分为八个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。最终只需要读取和写入HDFS一次。减少了六次HDFS的读写，读写HDFS减少了70%。
- Spark作业启动后会申请所需的Executor资源，所有Stage的Tasks以线程的方式运行，共用Executors，相对于MapReduce方式，Spark申请资源的次数减少了近90%。

- Spark引入了RDD（Resilient Distributed Dataset）模型，中间数据都以RDD的形式存储，而RDD分布存储于slave节点的内存中，这就减少了计算过程中读写磁盘的次数。RDD还提供了Cache机制，例如对上图的rdd3进行Cache后，rdd4和rdd7都可以访问rdd3的数据。相对于MapReduce减少MR2和MR3重复读取相同数据的问题。

效果对比

测试使用相同规模的资源，其中MapReduce方式包含200个Map和100个Reduce，每个Map和Reduce配置4G的内存；由于Spark不再需要Reduce资源， 而MapReduce主要逻辑和资源消耗在Map端，因此使用200和400个Executor做测试，每个Executor包含4G内存。测试结果如下表所示，其中输入记录约38亿条。

运行模式	计算资源	运行时间（min）	成本（Slot*秒）
MapReduce	200 Map+100 Reduce（4G）	120	693872
Spark	200 Executor（4G）	33	396000
Spark	400 Executor（4G）	21	504000

对比结果表的第一行和第二行，Spark运行效率和成本相对于MapReduce方式减少非常明显，其中，DAG模型减少了70%的HDFS读写、cache减少重复数据的读取，这两个优化即能减少作业运行时间又能降低成本；而资源调度次数的减少能提高作业的运行效率。

对比结果表的第二行和第三行，增加一倍的Executor数目，作业运行时间减少约50%，成本增加约25%，从这个结果看到，增加Executor资源能有效的减少作业的运行时间，但并没有做到完全线性增加。这是因为每个Task的运行时间并不是完全相等的， 例如某些task处理的数据量比其他task多；这可能导致Stage的最后时刻某些Task未结束而无法启动下一个Stage，另一方面作业是一直占有Executor的，这时候会出现一些Executor空闲的状况，于是导致了成本的增加。

小结

数据挖掘类业务大多具有复杂的处理逻辑，传统的MapReduce / Pig类框架在应对此类数据处理任务时存在着严重的性能问题。针对这些任务，如果利用Spark的迭代计算和内

存计算优势，将会大幅降低运行时间和计算成本。TDW目前已经维护了千台规模的Spark集群，并且会在资源利用率、稳定性和易用性等方面做进一步的提升和改进，为业务提供更有利的支持。