Hadoop 环境下的分布式协同过滤算法设计与实现*

肖 强[1] 朱庆华[1] 郑 华[2] 吴克文[1]
[1](南京大学信息管理学院 南京 210093)
[2](南京大学工程管理学院 南京 210093)

【摘要】以开源项目 Hadoop 为实验平台,论证传统协同过滤算法无法适应云平台;从相似度和预测偏好两方面,借鉴共词分析法,将传统协同过滤算法改进为适应 Hadoop 平台的分布式协同过滤算法;实现顺序组合式 MapReduce 协同过滤任务,并做进一步实验分析。

【关键词】Hadoop 协同过滤 大数据 分布式 云计算
【分类号】TP393

Design and Implementation of Distributed Collaborative Filtering Algorithm on Hadoop

Xiao Qiang[1] Zhu Qinghua[1] Zheng Hua[2] Wu Kewen[1]
[1](School of Information Management, Nanjing University, Nanjing 210093, China)
[2](School of Engineering Management, Nanjing University, Nanjing 210093, China)

【Abstract】Based on Hadoop, this paper demonstrates that traditional collaborative filtering algorithm cannot adjust to cloud computing platform, then improves traditional collaborative filtering algorithm to adapt to the Hadoop platform from similarity and prediction, and also achieves sequential modular MapReduce collaborative filtering computing tasks.

【Keywords】Hadoop Collaborative filtering Big data Distributed Cloud computing

1 引 言

网络信息增长迅速,为满足不同用户的信息需求,个性化推荐系统[1]应运而生,协同过滤算法[2]是其中较为成熟的一种推荐算法。近年来学者提出各种高效的协同过滤算法,如 Pan 等[3,4]提出的最小二乘逼近算法,Salakhutdinov 等[5]提出的概率矩阵分解算法等。

当前对协同过滤算法的研究主要侧重于单机模式的设计。然而随着海量数据的获取,大数据时代已经来临[6,7],社区用户信息数据量不断增长,数以亿计的数据给社区数据的存储和计算的速度带来极大的挑战。单机模式的推荐算法逐渐难以满足推送的即时性需求。因此,分布式推荐算法成为推荐算法研究中一个新的研究方向。

本文将协同过滤推荐算法与 Hadoop 的分布式计算特性相结合,探索协同过滤推荐算法在 Hadoop 平台上的实现。这既有分析推荐算法在云环境的可行性的理论意义,又有解决大数据条件下社区数据推送的即时性的现实意义。

收稿日期:2012 - 12 - 27
收修改稿日期:2013 - 01 - 15
*本文系国家社会科学基金项目"互联网用户群体协作行为模式的理论与应用研究"(项目编号:10ATQ004)的研究成果之一。

2 协同过滤与 Hadoop

2.1 传统协同过滤算法

协同过滤算法主要分为基于用户的协同过滤算法和基于项目的协同过滤算法,但其实质是相同的。

该算法通过分析社区所有用户的历史行为,生成 Item – User 矩阵;根据 item 之间的相似度,计算 item 的最近 N 邻居;根据最近 N 邻居和被预测用户的历史行为,预测用户对新 item 的评分,并产生推荐列表[8]。

(1)输　入

社区用户对项目的浏览打分,构成一次信息获取行为。得分代表用户对项目的偏好,例如 1 – 5 分范围内,1 代表很不喜欢,5 代表很喜欢。这样(user,item,preference)元组代表一次实际信息获取行为。由多个这样的元组构成的大量文件就是协同过滤算法的输入。

(2)生成 Item – User 矩阵

分析输入中不同的 user 和 item,并以 item 为行,user 为列,构建用户偏好矩阵,$I_1 - I_m$ 代表 m 个 item,$U_1 - U_n$ 代表 n 个 user,P_{ij}代表用户 j 对项目 i 的偏好,如图 1 所示:

	U_1	U_2	…	U_n
I_1	P_{11}	P_{12}	…	P_{1n}
I_2	P_{21}	P_{22}	…	P_{2n}
⋮	⋮	⋮		⋮
I_m	P_{m1}	P_{m2}	…	P_{mn}

图 1　Item – User 矩阵

(3)计算最近 N 邻居

根据 Item – User 矩阵,利用 Pearson 系数、余弦系数、修正余弦系数等相似度方法计算项目之间的相似度,其中 Pearson 系数计算[9]方法如下:

$$Sim(I_i,I_j)=\frac{\sum_{k\in U_{ij}}(P_{ik}-\overline{P_I})(P_{jk}-\overline{P_J})}{\sqrt{\sum_{k\in U_{ij}}(P_{ik}-\overline{P_I})^2}\sqrt{\sum_{k\in U_{ij}}(P_{jk}-\overline{P_J})^2}} \quad (1)$$

$Sim(I_i,I_j)$表示项目 I_i、I_j 之间的相似度;U_{ij}表示对 I_i 表达过偏好的用户集合与对 I_j 表达过偏好的用户集合的交集;$\overline{P_I}$、$\overline{P_J}$分表表示项目 I_i、I_j 的平均偏好。根据计算结果,与 I_i 最相似的前 N 个项目,就构成 I_i 的最近 N 邻居集合。

(4)产生推荐结果

用户 u 对新项目 i 的预测偏好值 P'_{iu}计算如下[10]:

$$P'_{iu}=\overline{P_I}+\frac{\sum_{k\in Iu}Sim(i,k)(P_{ku}-\overline{P_k})}{\sum_{k\in Iu}Sim(i,k)} \quad (2)$$

Iu 表示用户 u 表达过偏好的项目集合与项目 i 最近 N 邻居集合的交集;$\overline{P_I}$和$\overline{P_k}$分别代表项目 i、k 的平均偏好。

最终,根据计算的预测偏好值排序,形成推荐结果。

2.2 Hadoop

Hadoop 是 Apache 的顶级开源项目,核心是 HDFS 和 MapReduce,分别是 Google 云平台 GFS 和 MapReduce 的开源实现。HDFS 是分布式存储系统,可以冗余存储数以亿计的大数据;MapReduce 是建立在 HDFS 上的分布式并行计算架构。Hadoop 平台基于主从式架构,通过 Namenode、Datanode、Secondary、Jobtracter 和 Tasktracker 管理[11],可以运行在几十台乃至几千台计算机上,能够充分利用集群节点巨大的存储和计算资源。

Hadoop 强调的是移动计算,而不是移动数据。HDFS 将数据分块存储在集群中不同的节点上。计算前,Namenode 分析程序需要的数据存储在集群中的哪些节点;Jobtracter 将 MapReduce 计算任务分配给这些节点上的 Tasktracker;Tasktracker 启动 Map 程序,开启计算任务[12];经过 Combiner、Shuffle 等过程,在 Reduce 阶段生成计算结果。

MapReduce 计算框架式是以文件块的键值对为基础的,其计算流程如下:

Map: $<K1,V1>\rightarrow<K2,V2>$

Combiner: $<K2,V2>\rightarrow<K2,list(V2)>$

Shuffle: Partitioner(K2)→Reduce

Reduce: $<K2,list(V2)>\rightarrow<K2,V3>$

MapReduce 的输入是经过分割的 HDFS 文件块的 $<K1,V1>$ 键值对。默认情况下,K1 是 LongWritable 数据类型,表示数据在原文件的偏移量 Offset,即偏离文件首行的数值;V1 是 Text 数据类型,是该行的文本内容;Map 阶段根据 $<K1,V1>$ 进行相关计算,输出 0 个或多个不同数据类型的 $<K2,V2>$;Combiner 阶段根据 K2 对 V2 数据内容合并,生成 $<K2,list(V2)>$,以减少网络流量;Shuffle 对 K2 排序,根据 K2 的 Partitioner 值(默认为哈希值)将 $<K2,list(V2)>$ 分配给不

同的 Reduce；经过 Reduce 过程，输出 < K3，V3 > 到 HDFS 文件系统上。

Hadoop 集群支持三种运行模式：单机模式，伪分布式模式和完全分布式模式。用户可以通过修改 Hadoop 配置文件以调节集群的不同运行模式。默认情况下，Hadoop 被配置成一个以伪分布单机模式运行的独立 Java 进程。伪分布式模式通过在单机上创建多个线程，模拟分布式运行中的各类节点（Namenode、Datanode、Secondary、Jobtracter and Tasktracker）以达到分布式集群的目的；完全分布式模式是由两台以上的机器组成的真正意义上的分布式集群。单机模式适合调试工程的基本 BUG；伪分布式模式或小集群的完全分布式模式适合在小数据量条件下调试工程运行在分布式平台上的 BUG，通过伪分布式模式调试的工程可以投入完全分布式模式下的计算任务；大集群的完全分布式模式是投入实际使用的实现分布式计算任务的模式[13]。

2.3　传统协同过滤算法在 Hadoop 平台上的可行性

MapReduce 分布式计算的特点是将文件分块计算，Map 只对单独一行或其他特定数据量的数据进行计算[14]。

计算 Item - User 矩阵时，可以在 Map 阶段以（user，item，preference）作为输入 V 值，以 item 作为 K 值，（user，preference）作为 V 值输出，然后在 Reduce 阶段，根据相同的 K 值（即相同 item），合并 V 值，得到的输出 < K，V > 值就是 Item - User 矩阵中的每一行数据。然而，在根据 Item - User 矩阵计算项目之间的相似度时，需要涉及整个 Item - User 矩阵，即整个文件，单行数据无法计算项目之间的相似度。Hadoop 虽然可以将一个文件以 Cache 的方式加载为全局文件，但其前提条件是该文件数据量小，能够被加载到单机内存上，显然，在大数据的计算背景下，这是不现实的。将文件分块计算，使得 MapReduce 对项目相似度计算无能为力。

因此，传统的协同过滤算法在 Hadoop 平台上没有可行性，这就要求研究者设计新的协同过滤算法以适应 Hadoop 平台。

3　Hadoop 平台上的分布式协同过滤算法设计

3.1　相似度算法设计

通过分析可知，要实现 Hadoop 平台上的协同过滤算法，首先要解决其相似度计算的可行性。根据 MapReduce 分布式计算的特点可知，必须通过单行的输入找到项目之间的联系。

以（user，item，preference）为输入，MapReduce 很容易得到以 user 作为 K 值，（item，preference）作为 V 值的输出，也就容易得到数据元组 $\{user,(item_1,preference_1),(item_2,preference_2),\cdots(item_k,preference_k)\}$。该元组表达了用户所有的历史获取信息，所有的获取项目都共同出现在一个元组中。在引文分析中，分析共同出现在一起的词的方法是共词分析法[15]，其主要作用是分析出具有紧密联系的相似的主题词，共同出现的次数代表两个词之间的紧密相似程度。这与分析相似项目的需求相近，因此，本文借鉴共词分析法，通过共现分析法计算相似 item。两个项目共同出现的次数越多，它们就越相关或相似；项目共现矩阵的作用与非分布式协同算法中的相似度作用类似。

以 $\{user,(item_1,preference_1),(item_2,preference_2),\cdots(item_k,preference_k)\}$ 为 V 值初始输入；对于评分项目，如果 $item_i$ 和 $item_j$ 的 preference 值都为正向含义（例如在 1 - 5 分的评分范围内，preference 值都大于等于 3），则在 Map 阶段输出 $< item_i, item_j >$ 键值对，表示 $item_i$，$item_j$ 共同出现过一次；然后根据 K 哈希值，在 Reduce 阶段汇总以后，输出以 item 为 K 值，以 $\{(item_1,n_1),(item_2,n_2),\cdots(item_k,n_k)\}$ 为 V 值的键值对，得到 $\{item,(item_1,n_1),(item_2,n_2),\cdots(item_k,n_k)\}$ 元组，其中，n_k 表示 item 与 $item_k$ 共同出现了 n_k 次。最终，出现在 HDFS 上的相似度文件就是以上述元组为行值组成的文件。其内容实质是一个 Item 共现矩阵，n_{ij} 表示项目 i 与项目 j 共同出现的次数，代表两个项目之间的紧密相似程度，如图 2 所示：

	I_1	I_2	…	I_m
I_1	n_{11}	n_{12}	…	n_{1m}
I_2	n_{21}	n_{22}	…	n_{2m}
⋮	⋮	⋮		⋮
I_m	n_{m1}	n_{m2}	…	n_{mm}

图 2　Item 共现矩阵

3.2　预测算法设计

如果以 $U_j(P_{1u},P_{2u},\cdots P_{mu})^T$ 表示第 j 个用户 u 对 m 个所有项目的原偏好向量，以共现矩阵代替相似矩阵，则用户 u 对项目 i 的预测偏好值 P'_{iu} 计算如下：

$$P'_{iu}=\frac{\sum_1^m n_{ik}P_{ku}}{\sum_1^m n_{ik}} \quad (3)$$

预测偏好向量 $U'_j(P'_{1u},P'_{2u},\cdots P'_{mu})^T$ 计算如图3所示：

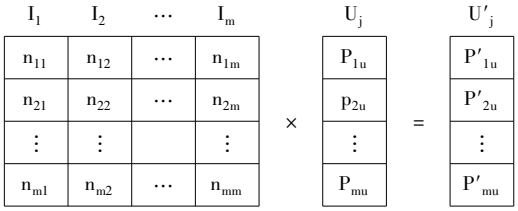

图3 预测偏好向量 U'_j

观察图3所示的计算过程，可以发现，计算 U'_j 仍然需要涉及到整个共现矩阵。然而，在大数据的计算背景下，无法将整个共现矩阵文件加载到内存。因此，目前设计的预测算法仍与 MapReduce 分布式计算的特点相矛盾，需要进一步改进。

公式(3)可分解为两步：

$$P''_{iu}=\sum_1^m n_{ik}P_{ku} \quad (4)$$

$$P'_{iu}=\frac{P''_{iu}}{\sum_1^m n_k} \quad (5)$$

经 MapReduce 过程，很容易实现 $U''_j(P''_{1u},P''_{2u},\cdots P''_{mu})^T$ 到 $U'_j(P'_{1u},P'_{2u},\cdots P'_{mu})^T$ 的转换，因此，算法需要改进的地方主要集中在 U''_j 的实现。

对 U''_j 公式数学变形如图4所示：

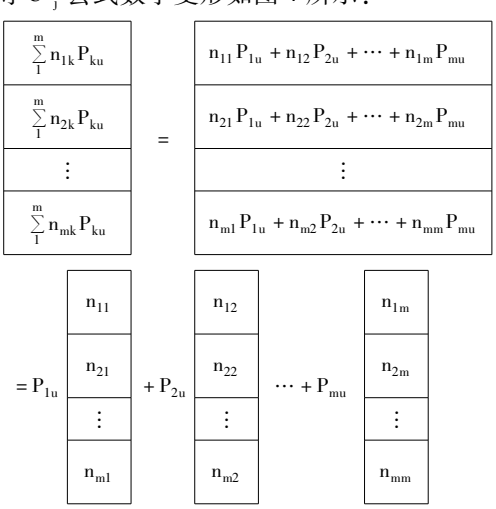

图4 U''_j 公式数学变形

变形所得向量 $(n_{1k},n_{2k},\cdots n_{mk})^T$ 正好是 Item 共现矩阵的第 k 个列向量 $ColI_k$。因此，可得：

$$U''_j=\sum_1^m P_{ku}ColI_k \quad (6)$$

MapReduce 框架的 Map 阶段是以文件块的行为处理单位的，因此，以 Item 共现矩阵的列为输入是不合理的。但是观察发现，Item 共现矩阵是一个对称矩阵，项目 I_i 与项目 I_j 的共现值 n_{ij} 等于项目 I_j 与项目 I_i 的共现值 n_{ji}，因此 Item 共现矩阵的第 k 个列向量 $ColI_k(n_{1k},n_{2k},\cdots n_{mk})^T$ 等于第 k 个行向量 $RowI_k(n_{k1},n_{k2},\cdots n_{km})$ 的转置，即：

$$ColI_k=RowI_k^T \quad (7)$$

因此，公式(6)可转换为：

$$U''_j=\sum_1^m P_{ku}RowI_k^T \quad (8)$$

这样，U''_j 的计算转化为，以每个被用户表达过偏好的项目的偏好值为系数，乘以该项目的共现向量，然后求和。之后，再将 U''_j 转换为 U'_j。由 U'_j 组成的矩阵 $U'[U'_1,U'_2,\cdots U'_n]$，就是所得的预测矩阵。

与图3显示的算法相比较，最终预测算法的改进主要体现在两个方面：将 Item 共现文件按行输入，使预测算法适应 Hadoop 平台上的 MapReduce 框架；将算法时间复杂度 $O(m^3n)$ 降低为 $O(mn)$。如果用户对一个项目没有信息获取行为，则将其偏好值设为0，在图3所示的算法中，单独一个偏好值不能分离，即使为0也要参与运算，而在改进的算法中，偏好值作为系数分离出来，偏好值为0时，可以直接跳出循环，使时间复杂度变为 $O(k^2mn)$，k 为用户表达过偏好的项目的平均数据，k 为常数，因此时间复杂度为 $O(mn)$，这在数据相对稀疏的情况下，性能的提升尤为明显。

4 Hadoop 平台上的分布式协同过滤算法实现

4.1 实现步骤

Hadoop 平台上的协同过滤算法思想设计完成以后，还要转化为 MapReduce 分布式计算框架可执行的 MapReduce 流程，才能真正实现 Hadoop 平台上的分布式协同过滤，其实现步骤如下：

(1)生成用户向量 $U[u,(I_1,P_1),(I_2,P_2),\cdots(I_t,P_t)]$。用户向量中仅包括该用户表达过偏好的用户，因此，$t\leqslant m$(项目总数目)。以初始文件为输入，MapReduce 流程如下：

Map：$<Offset,(u,I,P)>\rightarrow<u,(I,P)>$

Reduce：$<u,(I,P)>\rightarrow<u,list(I,P)>$

(2)生成项目共现向量 $I[i,(I_1,n_1),(I_2,n_2),\cdots(I_k,n_k)]$ 和项目共现次数和 I_{sum}，即公式(5)中的 $\sum_1^m n_{ik}$。共现向量中仅包括与该项目共同出现过的项目，因此，$k\leqslant m$。以用户向量文件为输入，利用 Hadoop

下的 MultipleOutputs 类输出两个不同的文件，MapReduce 流程如下：

Map：$<Offset,U>\rightarrow<i,(I,1)>$

Combiner：$<i,(I,1)>\rightarrow<i,list(I,n')>$

Reduce：$<i,list(I,n')>\rightarrow<i,list(I,n)>/<i,sum>$

（3）生成 U″公式数学变形过程中的分向量 ColU″[u,(I_1,Col_1),(I_2,Col_2),…]，即公式（8）累加前的 $P_{ku}RowI_k^T$ 列向量。这一步的难点是需要连接步骤（1）和步骤（2）生成的用户向量和项目共现向量两个中间文件。Hadoop 没有关系数据库强大的连接处理能力，多数据源的连接处理相对复杂。由于两个中间文件都比较大，不可能以 Cache 的方式连接，因此，本文利用 DataJoin 类库实现数据源在 Reduce 端的连接：为不同数据源下的每个数据记录定义一个数据标签 Tag；为每个待连接的数据记录确定一个连接键 GroupKey；将数据传输到 Reduce 端进行连接。连接过程中，传输数据的值的格式必须是定义好的 TaggedMapOutput 格式，其包含数据标签 Tag 和数据内容；Map 和 Reduce 过程也必须继承 DataJoin 类库中的 DataJoinMapperBase 抽象类和 DataJoinReducerBase 抽象类。在协同算法中，根据步骤（1）和步骤（2）生成的两个中间文件名为 Tag，项目为 GroupKey，经过 Reduce 过程实现 $P_{ku}RowI_k$ 的计算，输出以用户为 K 值、计算结果为 V 值的键值对。以步骤（1）和步骤（2）生成的两个中间文件为输入，MapReduce 流程如下：

Map：$<Offset,U/I>\rightarrow<i,TaggedMapOutput>$

Reduce：$<i,TaggedMapOutput)>\rightarrow<u,ColU''>$

（4）生成中间预测向量 U″[u,(I_1,Col_1),(I_2,Col_2),…]。以分向量 ColU″文件为输入，MapReduce 流程如下：

Map：$<Offset,(u,ColU'')>\rightarrow<u,ColU''>$

Reduce：$<u,ColU'')>\rightarrow<u,U''_j>$

（5）生成按项目组织的预测向量 I′[i,(U_1,P_1),(U_2,P_2),…(U_n,P_n)]。这一步需要连接步骤（2）和步骤（4）生成的项目共现次数和 I_{sum} 和中间预测向量 U″两个中间文件，连接方法同步骤（3），但是因为需要根据项目共现次数对 U″的每个元素实现公式（4）到公式（5）的变换，所以选取项目为连接键 GroupKey。以上述两个文件为输入，MapReduce 流程如下：

Map：$<Offset,I_{sum}/U''_j>\rightarrow<i,TaggedMapOutput>$

Reduce：$<i,TaggedMapOutput)>\rightarrow<i,(U,P)>$

（6）生成最终预测向量 U′[u,(I_1,P_1),(I_2,P_2),…]。以步骤（5）产生的中间文件为输入，MapReduce 流程如下：

Map：$<Offset,I'>\rightarrow<u,(I,P)>$

Combiner：$<u,(I,P)>\rightarrow<u,list(I,P)>$

Reduce：$<u,list(I,P)>\rightarrow<u,list(I,P)>$

经过以上 6 步，实现了 Hadoop 平台上的协同过滤算法，每一步都是一个完整的 MapReduce 流程。6 个 MapReduce 流程组成了 Hadoop 平台上的顺序组合式 MapReduce 计算任务。顺序组合式 MapReduce 作业将按顺序逐个执行每个子任务，每一个子任务的输入都是前项子任务的输出。

4.2　实验分析

经分析，在小数据量条件下通过伪分布式模式或小集群的 Hadoop 分布模式调试的工程可以胜任完全分布式模式下的大数据分布式计算任务，因此，算法实现后，笔者以三台机器作为硬件配置，以 Ubuntu12.10 作为操作系统，Hadoop1.1.0 平台作为底层架构，Eclipse 作为编程环境，MovieLens 作为数据源（自 http://www.grouplens.org/站点下载），实现分布式协同过滤算法的调试和运行。

本次下载的 MovieLens 数据共有 90 571 条评分记录，如表 1 所示：

表 1　MovieLens 数据量

用户数量	电影数量	评分记录数量
50	1 084	4 855
200	1 420	17 748
400	1 542	40 318
800	1 671	77 533
943	1 682	90 571

分别选取 50、200、400、800 和 943 个用户作为实验数据。当预测前两个用户对所有项目的评分时，在不同条件下算法的响应时间，如图 5 所示：

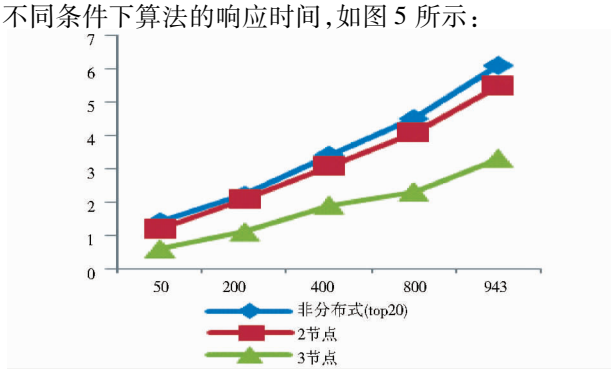

图 5　响应时间

其中，纵轴表示响应时间，以分钟（min）为单位；横轴表示用户数量。可以看出，包含两个节点的 Hadoop 集群响应时间和传统的基于项目的算法（top20）响应时间基本相同，这是因为两个节点的 Hadoop 集群包括一个主节点和一个从节点，这两个节点除了负责计算任务外，还要负责集群资源调度；而包含 3 个节点的集群的响应时间明显优于传统算法。可以想象如果集群包含数十个节点，计算几乎可以瞬间完成，体现分布式协同过滤算法的即时性优势。

在算法的精确度方面，选取平均绝对误差（MAE）作为评价指标，其计算公式[16]为：

$$MAE = \frac{\sum_1^m |p1 - p2|}{m} \qquad (9)$$

其中，m 表示项目数量，p1 表示实际分数，p2 表示预测分数。MAE 越小，表示精度越高。分布式协同过滤算法与传统算法的 MAE 比较，如图 6 所示：

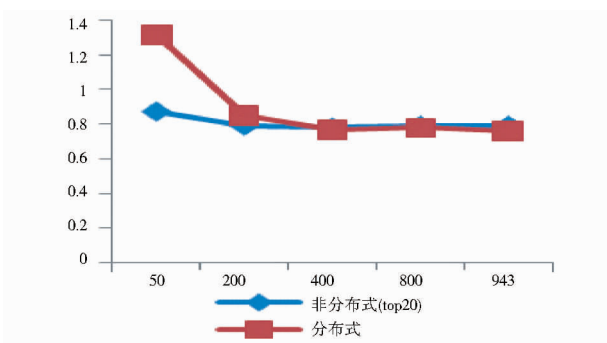

图 6　MAE 比较

可以看出，在数据相对稀疏的情况下，分布式算法的精度较差，而数据量逐渐增大时，其精度逐渐提高，并稍优于传统算法。这是因为根据算法设计，仅仅在 $item_i$ 和 $item_j$ 的 preference 值都为正向含义时，才在 Map 阶段输出 $<item_i, item_j>$ 键值对，当数据相对稀疏时，输出键值对较少，计算误差大。这说明，本文提出的分布式协同算法比较适合海量数据背景下的推荐。

5　结　语

本文介绍了传统协同过滤算法和 Hadoop 云平台的概况，并论证了传统协同过滤算法无法适应 Hadoop 分布式平台；根据 Hadoop 的分布式特性，从相似度和预算偏好两方面，借鉴共词分析法，将传统协同过滤算法改进为适应 Hadoop 平台的分布式协同过滤算法；实现了顺序组合式 MapReduce 协同过滤计算任务，并做进一步实验分析。本文对网络社区中大数据的处理有一定的借鉴意义。但是也存在一定的不足之处，在数据相对稀疏的情况下，分布式算法的精度较差，这说明分布式算法在应对冷启动方面存在明显不足，还有待进一步的改进。

参考文献：

[1] 李树青. 个性化信息检索技术综述[J]. *情报理论与实践*，2009，32(5)：107－113.（Li Shuqin. Review of Personalized Information Retrieval Technology[J]. *Information Studies: Theory & Application*, 2009, 32(5): 107－113.）

[2] 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.

[3] Pan R, Scholz M. Mind the Gaps: Weighting the Unknown in Large－Scale One－Class Collaborative Filtering[C]. In: *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Paris, France. New York: ACM, 2009: 667－676.

[4] Pan R, Zhou Y H, Cao B, et al. One－Class Collaborative Filtering[C]. In: *Proceedings of the 8th IEEE International Conference on Data Mining*, Pisa. Washington, DC, USA: IEEE Computer Society, 2008: 502－511.

[5] Salakhutdinov R, Mnih A. Probabilistic Matrix Factorization[C]. In: *Proceedings of the 25th International Conference on Machine Learning*. New York: ACM, 2008: 880－887.

[6] 侯经川，方静怡. 数据引证研究：进展与展望[J]. *中国图书馆学报*，2013，39(1)：112－118.（Hou Jingchuan, Fang Jingyi. Review on Data Citation in the Context of Big Data[J]. *Journal of Library Science in China*, 2013, 39(1): 112－118.）

[7] 韩翠峰. 大数据带给图书馆的影响与挑战[J]. *图书与情报*，2012(5)：37－40.（Han Cuifeng. The Impact and Challenges of the Library Based on Big Data[J]. *Library & Information*, 2012(5): 37－40.）

[8] Sarwar B, Karypis G, Konstan J, et al. Item－based Collaborative Filtering Recommendation Algorithms[C]. In: *Proceedings of the 10th International Conference on World Wide Web*. New York, NY, USA: ACM, 2001: 285－295.

[9] Resnick P, Iacovou N, Suchak M et al. GroupLens: An Open Architecture for Collaborative Filtering of Netnews[C]. In: *Proceedings of the 1994 ACM Conference on Computer Supported Cooperative Work*. New York, NY, USA: ACM, 1994: 175－186.

[10] Sarwar B, Karypis G, Konstan J, et al. Item－based Collaborative Filtering Recommendation Algorithms[C]. In: *Proceedings of the*

10*th International Conference on World Wide Web*. New York, NY, USA: ACM, 2001: 285 – 295.

[11] White T. Hadoop: The Definitive Guide[M]. The 3rd Edition. USA: O'Reilly Media, 2012.

[12] Dean J, Ghemawat S. MapReduce: Simplified Data Processing on Large Clusters[J]. *Communications of the ACM*, 2008, 51(1): 107 – 113.

[13] Hadoop. HDFS Users Guide[EB/OL]. [2012 – 12 – 02]. http://hadoop.apache.org/docs/stable/hdfs_user_guide.html.

[14] Bahga A, Madisetti V K. Analyzing Massive Machine Maintenance Data in a Computing Cloud[J]. *IEEE Transactions on Parallel and Distributed Systems*, 2012, 23(10): 1831 – 1843.

[15] 冯璐,冷伏海. 共词分析方法理论进展[J]. *中国图书馆学报*, 2006, 32(2): 88 – 92. (Feng Lu, Leng Fuhai. Development of Theoretical Studies of Co – word Analysis[J]. *Journal of Library Science in China*, 2006, 32(2): 88 – 92.)

[16] Sarwar B, Karypis G, Konstan J, et al. Analysis of Recommendation Algorithms for E – commerce[C]. In: *Proceedings of the 2nd ACM Conference on Electronic Commerce*. New York: ACM, 2000: 158 – 167.

(作者 E – mail: njhnxq@163.com)

Facebook 图谱搜索将改变搜索生态

2011 年 2 月,马克·扎克伯格曾在 Facebook 上传了一张他办公室的照片,照片中的电脑浏览器上是一张 Facebook 页面,商业周刊从这幅页面中敏锐地捕捉到一个细节:Facebook 正在开发搜索引擎。一年之后,商业周刊这一发现得到证实:Facebook 在 2012 年 1 月举办的发布会上宣布推出"社交图谱搜索"(Facebook Graph Search, FGS)。用扎克伯格的话说,这是 Facebook 继"动态消息"(News Feed)和"时间线"(Timeline)之后第三大支柱,也是 Facebook"第一个重大的产品发布"。

简单来说,图谱搜索通过人、社交关系以及社交链——即"10 亿人、1 万亿连接关系和 2 400 亿图片"的数据池为基础,为用户的搜索提供结果。比如,"我去年吃过的北京粤菜馆子","去过丽江的朋友","我去年参加过的聚会的照片"等,都是图谱搜索负责帮助用户解决的问题。用户每一次分享、每一次"喜欢"(Like)过的内容、每建立一个好友关系,都会成为图谱搜索数据池的一部分,用于改善用户的搜索体验。

从 Facebook 图谱搜索的功能可以看出,它与 Google 搜索有很大的不同,主要表现在两个方面:

(1)相比于 Google 网页搜索给出一堆链接信息,Facebook 的图谱搜索则直接给出答案。

(2)更关键的差异在于,Google 搜索基于网页质量和网页链接关系——通过给网页赋予 Page Rank 以及分析不同权值的网页之间的链接关系之后给出结果;Facebook 分析的则是人,对人分享了什么新闻、以及对人与人之间发生的关系的判断。

不过,在很大程度上来说,这些只是 Facebook 图谱搜索的愿景。目前而言,图谱搜索只对少数英文用户开放,就内容来看,只能搜索"人"、"图片"、"地点"和"主页"4 种信息。也就是说,这还是一个相当早期的测试版产品。在发布会上,图谱搜索主管 Lars Rasmussen 也说,他们还需要收集更多的数据来改进搜索结果,并鼓励用户多为餐馆打分,贡献数据。

从图谱搜索目前的功能以及 Lars Rasmussen 的表态来看,Facebook 的搜索还相当简陋。就数据积累和需求角度而言,Facebook也有先天的劣势。Google 数据池的来源是几乎所有开放的网页信息,Facebook 所有的数据来源仅仅是它的用户;同时,有图谱搜索需求的只能来自它的用户,尤其来自那些贡献了很多内容、分享了大量信息、拥有许多朋友的人,而这部分人是少数。而且,即使在社交网络上,信息消费者也远远多于信息创造者。无论采取什么样的定义,以及哪怕采用最乐观的估计,信息创造者与信息消费者的比例很可能小于 2:8(即网络上所谓的"沉默的大多数"超过 80%)。比如 2011 年 2 月 Facebook 的总搜索量是 3 亿次,Google 高达 176 亿次,用户数相差不大的情况下,搜索请求相差 2 个数量级。不过,Facebook 推出图谱搜索后,也许会激发用户更多的需求。

图谱搜索与 Google 搜索没有直接的竞争关系,它让 Facebook 与 Google 之间变得更加微妙,也很可能对搜索生态产生重大影响。扎克伯格在发布会上也说,"如果 Google 尊重用户在 Facebook 分享内容的隐私,那么 Facebook 很愿意同 Google 合作"。

Facebook 的高速成长促使 Google 推出自己的社交网站 Google +,这两年 Google 也在大力将 Google + 的信息整合进 Google 搜索中,为登录用户提供更具个性化的搜索结果。在图谱搜索已经推出之后,这一趋势只会加深。可以肯定的是,Google 衡量信息质量的天平随着 Facebook 从"Page"逐步转到"People"上的时候,信息发布、信息传播及信息获取方式都将被改变。

(编译自:http://shibeichen.com/post/40833850372)

(本刊讯)