文章编号:1671-4598(2018)08-0238-05 DOI:10.16526/j.cnki.11—4762/tp.2018.08.053 中图分类号:TP311 文献标识码:A

MapReduce 框架下一种负载均衡的 Top—k 连接查询算法

胡东明[1], 刘旭敏[1], 徐维祥[2]
(1. 首都师范大学 信息工程学院, 北京 100048; 2. 北京交通大学 交通运输学院, 北京 100044)

摘要: 针对传统 Top—k 连接查询算法在处理海量数据时的时效问题, 提出一种基于 MapReduce 框架的负载均衡的并行 Top—k 连接查询算法 (P—TKJ); 使用直方图形式来存储数据, 有助于提高 CPU 的利用率; 同时融入了提前终止策略和磁盘数据的选择性访问, 以便提高对 HDFS 数据访问的性能; 另外, 融入了数据过滤和基于最长处理时间优先 (LPT) 算法的负载均衡策略来减少和均衡 Reduce 任务, 以此设计出高效的并行 Top—k 连接算法; 一个集群实验结果表明, 该方法能够有效缩短算法的执行时间。

关键词: Top—k 连接查询; MapReduce 框架; 数据过滤; 负载均衡; 执行时间

A Load Balancing Top—k Join Query Algorithm in MapReduce Framework

Hu Dongming[1], Liu Xumin[1], Xu Weixiang[2]
(1. College of Information Engineering, Capital Normal University, Beijing 100048, China;
2. College of Traffic and Transportation, Beijing Jiaotong University, Beijing 100044, China)

Abstract: For the issues that the time efficiency problem of traditional Top—k join algorithm when dealing with massive data, a load—balanced parallel Top—k join query algorithm (P—TKJ) based on MapReduce framework is proposed. It used histograms to store data helps to increase CPU utilization. An early termination strategy and disk data selective access mechanism is incorporated to improve the performance of HDFS data access. In addition, data filtering and load—balancing strategies based on the longest processing time priority (LPT) algorithm are incorporated to reduce and equalize reduce tasks, so that to design an efficient parallel Top—k connection algorithm. A cluster experiment shows that this method can shorten the execution time of the algorithm effectively.

Keywords: top—k join query; MapReduce framework; data filtering; load balancing; execution time

0 引言

排序查询处理对于大规模数据分析至关重要, 通常使用的排序查询方法称为 Top—k 连接查询算法[1]。Top—k 查询中, 根据每个对象的属性计算一个权重, 再通过给定的评分函数为对象进行评分, 返回 k 个最重要的结果[2]。在一十大数据时代, 用户检查大量未排序的查询结果集是不现实的。并行化执行不仅可以实现高效地运行, 并且可以返回精准的结果。目前 MapReduce 是一种广泛应用的并行编程环境[3]。

目前, 学者也提出了一些并行的 Top—k 连接查询算法。例如, 文献 [4] 在 MapReduce 的背景下, 提出了两种关于 Top—k 连接的方法。一种称为 RanKloud 的算法, 其在扫描记录期间计算统计数据, 并使用这些统计数据计算提前终止的阈值 (Top—k 结果的最低分数)。此外, 还提出了一种新的分区方法, 称为 uSplit, 旨在以使用敏感方式对数据进行重新分区。然而, RanKloud 不能保证正确的返回 k 个检索结果。另外, 常用的一种基于 MapReduce 框架的用来计算 Top—k 连接结果的通用二路连接算法为 Reduce—side join, 简称为 RSJ[5], 其连接是在 Reduce 函数中实现。

本文在 MapReduce 编程模型中实现并行 Top—k 连接查询算法 (Parallel Top—k Join, P—TKJ), 同时融入提前终止机制和负载均衡机制来增强 Top—k 连接处理的性能。主要创新点为: 在 MapReduce 中提出了一个新的 Top—k 连接处理框架, 尽可能地利用并行性, 并避免链接 MapReduce 作业的初始化开销; 使用直方图形式的数据表示, 并融入了提前终止策略、数据过滤和负载平衡策略, 以便设计出高效的并行 Top—k 连接算法。

1 MapReduce 编程模型

MapReduce 是 Hadoop 中的一个编程框架, 为并行算法提供了一个容错和可靠的编程环境。为了处理大量的数据, 该框架支持一个可扩展的文件系统, 称为 Hadoop 分布式文件系统 (HDFS), 用于在硬件群集中的机器上存储大量文件。

MapReduce 计算过程分成 Map 和 Reduce 两个阶段[6], 其中数据的格式以键值对<key, value >呈现, 其处理过程如图 1 所示。

收稿日期:2018-01-11; **修回日期**:2018-01-29。
基金项目:国家自然科学基金(61672002);北京市长城学者项目(CIT&TCD20170322)。
作者简介:胡东明(1987-),男,河北盐山人,硕士研究生,主要从事云计算、数据挖掘等方向的研究。
刘旭敏(1956-),女,辽宁锦州人,博士,教授,主要从事数据挖掘、图形图像处理、计算机辅助几何设计等方向的研究。
徐维祥(1956-)男,辽宁锦州人,博士,教授,主要从事数据挖掘、云计算、交通运输系统分析集成等方向的研究。

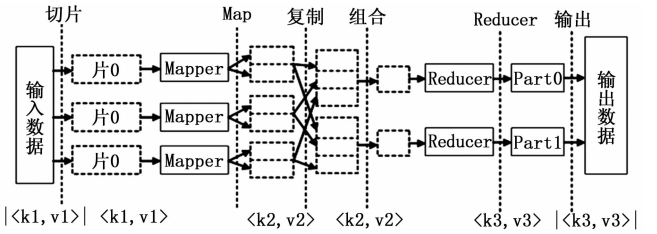

图 1　MapReduce 编程模型的处理过程

2　问题描述

2.1　Top－k 连接查询

给定具有 n 个得分属性的输入表或关系 T，使用 τ 代表 T 的记录（或元组），$\tau[i]$ 是指第 i 个得分属性（$i \in [1, n]$）。Top－k 查询 $q(k, f)$ 基于单调评分函数 f 返回 k 个最佳查询结果。当应用于关系 T 时，Top－k 查询 $q(k, f)$ 的结果是 T 中一组 k 个记录 τ_i，…，τ_k 中得分最小的 Δk，即 $f(\theta)$ 的值。在不失一般性的情况下，分数最低的记录被认为是最好的[7]。

通常在排名感知处理中，需要两个（或更多）输入关系连接的 Top－k 结果，视为一个运算符，称之为 Top－k 连接查询[8]。可以通过先执行连接，然后通过评分函数对连接记录进行排名，并输出前 k 个排名结果。然而，这会导致处理过程的资源浪费，所以需要提出高效的算法来解决交织排序和连接的问题[9]。

在本文中，认为输入关系 T_i 包含了一个连接属性 a_i，一个得分属性 s_i，以及其他一些属性。因此，T_i 由唯一标识符（τ，id）、连接属性值或连接值（τ，a_i）和得分属性值（τ，s_i）所描述的记录组成。本文关注二元多对多的 Top－k 等值连接，其中输入表 T_0 和 T_1 连接在连接属性 $a_0 = a_1$ 上，得分属性（s_0 和 s_1）的组合是为了生成 Top－k 连接记录，作为得分函数 f 的输入。

2.2　问题描述

考虑两个输入表 T_0 和 T_1，它们分别在一组机器上被水平分割，并具有连接属性 a_0，a_1 和得分属性 s_0，s_1。给定由整数 k 定义的 Top－k 连接查询 $q(k, f, T_0, T_1)$，和用于组合得分属性 s_0 和 s_1 产生连接记录的单调得分函数 f。并行 Top－k 连接问题要求产生具有最低分数的 Top－k 连接记录。

在 MapReduce 环境中，输入表 T_0 和 T_1 被拆分为 HDFS 块，并按照水平分区的概念存储在 HDFS 中。一个记录 τ 在每个文件中都是（$\tau.id$，$\tau.a_i$，$\tau.s_i$）形式，其中 $\tau.id$ 是唯一标识符，$\tau.a_i$ 是连接属性，$\tau.s_i$ 是得分属性。除了这个三元组之外，每一行可能都包含其他任意长度的记录元素的属性 τ。因此，在一般情况下，每个节点只存储每个关系记录的一个子集。问题在于设计一个由 Map 和 Reduce 阶段组成的算法，通过并行方式有效计算 Top－k 连接方式。

最后，本文注意到 Top－k 连接并行处理中最昂贵的部分是计算每个连接值的 Top－k 连接记录。因此，在本文中，我们着重于提供一个完全并行的解决方案来解决这个问题。获得 Top－k 连接结果的最后一步需要处理 $k \cdot m$ 个连接记录（其中 m 表示不同连接值的个数），这通常比初始表 T_i 的值小几个数量级，即 $k \cdot m << |T_i|$。因此，可利用一个集中程序来处理这些单独的 Top－k 结果，而没有显著的开销。

3　提出的并行 Top－k 连接查询算法

3.1　方法概述

上传两个输入表 T_0 和 T_1，并作为单独的文件存储在 HDFS 中，根据得分属性以升序排序。此外，对于每个输入表，计算并存储在 HDFS 直方图 $H(T_0)$ 和 $H(T_1)$ 中，它们维护一系列连接属性值的记录数。需要注意的是，这些信息可以在输入表上传到 HDFS 的过程中构建，而开销可以忽略不计。

给定一个 Top－k 连接查询，计算每个输入表（基于直方图）的分数范围，这些范围决定了作业执行前足以产生正确结果的记录子集。因此，可以选择性地在 Map 阶段加载和处理存储数据的一小部分，一旦遇到分数值大于边界的记录，就终止 Mappers 的处理。此外，通过引入数据过滤和负载均衡机制来优化 Reduce 端连接的性能，该机制均衡地将连接值分配给 Reduce 任务。

3.2　直方图构建

在 Hadoop 中处理数据需要上传数据，整个数据集从外部源按顺序读取并存储在 HDFS 中[10]。这个阶段主要是 I/O 密集型任务，CPU 没有充分利用，可以利用这个阶段在后台建立直方图。通常情况下，直方图的大小比初始数据集要小几个数量级，但是在准确性和磁盘大小之间权衡，即在构建过程中更大直方图可以实现更高的准确性，同时会消耗更多磁盘空间的。

为达到预期的目的，本文选择了构建等宽直方图，其构造简单且符合一次通过的要求。更详细地说，当一个记录 $\tau(\tau.a_i, \tau.s_i)$ 在上传阶段被读取，可以通过增加对应分数值 $\tau.s_i$ 的 bin 的内容来更新连接值为 $\tau.a_i$ 的直方图。

图 2 描绘了相同连接属性值下，T_0 和 T_1 的等宽直方图。对于每个输入表 T_i，创建与连接属性中单独值数量一样多的直方图。每个直方图被表示为 $H(T_i)$。例如，所描述的 T_1 的直方图 $H(T_1)$ 表示它总共包含 11 个具有连接值 $a_1 = x$ 的记录。此外，第一个直方图框表示存在 2 条记录，得分在 0－10 之间（表示为 [0－10]：2），剩下的 bin 是：[10－20]：3，[20－30]：2 以及 [30－40]：4。

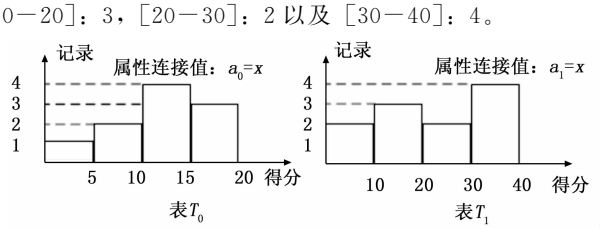

图 2　相同的连接属性值（$a_0 = a_1 = x$）下，T_0 和 T_1 的等宽直方图的例子

3.3　提前终止机制

为了减少连接的处理成本，本文只处理两个表的输入记录子集，来保证提供正确的 Top－k 连接结果。直观地

说，只有表 T_i 中分数低于 b_i 的记录才会参与连接，用来产生 Top－k 连接结果。因此，为了实现提前终止操作，需要有一种方法来确定分数范围 b_0 和 b_1，以便尽可能早地放弃高于 b_i 分数的记录。

1）分数界限估计：将两个表的直方图作为输入，问题在于要计算每个表 T_i 中输入记录得分的正确分数界限 b_i。为此，本文使用文献［11］中提出的算法来进行分数界限估计。在实践中，这个算法对两个表格的直方图执行连接，并估计连接结果的数量和分数范围。这个算法的用处为：第一，识别直方图 bin 和相应分数范围用来产生 k 个连接记录；第二，确保没有其他直方图 bin 组合可以产生具有比这第 k 个连接记录更小分数值的连接记录。为此直方图 bin 不断被访问和加入，直到加入记录的数量超过 k，或者任何直方图 bin 产生的连接记录得分都不小于当前第 k 个记录的得分。用一个例子来解释算法的操作，描述如下。

示例 1：考虑图 2 中描述的直方图，并假设 Top－k 连接结果（$k=1$）被要求使用作为评分函数的总和。通过检查每个直方图的第一个 bin，可以知道在［0－15］范围内存在 2（＝1×2）个连接记录，即［0－15］：2。通过每个直方图，还可以知道存在［10－25］：3，［5－20］：4 和［15－30］：6。只有在 T_0 的第三个 bin 被检查后（产生的连接记录没有显示在这里），才可以安全地停止处理，并且报告得分范围 $b_0=15$ 和 $b_1=20$。这是因为得分［0－15］内已经有至少 2 条记录（即多于 $k=1$），并且 T_0 或 T_1bin 组合产生的任何连接记录的分数都将大于 15。

2）在 Hadoop 中实现提前终止操作：假设输入表以 HDFS 格式存储，并且直方图也可用，创建一个提前终止机制，在 Map 阶段有选择地只处理分数比各自界限低的输入记录。需要注意的是，提前终止机制是通过扩展 Hadoop 来实现的，也就是说，不会更改 Hadoop 核心。

3.4 数据过滤

Map 任务会处理一组输入记录（以键值对的形式）并生成一组输出记录。限制输出记录的数量非常重要，这会影响整体性能，因为这些记录需要通过 Reduce 任务进行混洗（消耗通信成本）和处理（消耗处理成本）。数据过滤技术通常是通过消除不影响结果的输入记录来限制 Map 输出记录的数量。应该注意的是，数据过滤是依赖于作业的，这意味着每个作业都需要基于查询类型的不同过滤机制。

Top－k 查询的过滤过程中，考虑在 n 维空间 R^n 中定义的多维数据集 S（例如，$p\in S$ 且 $p=[p_1,\cdots p_n]$），以及一个 Map 任务，即访问完整数据集 S 的子集 S'。另外，让一个偏好函数 $f(p)=\omega_1\cdot p_1+\cdots+\omega_n\cdot p_n$ 为数据对象赋值。目标是检索出得分最高的 top－k 对象。对于由 Map 任务读取的每个对象 $p\in S'$，分配一个分数 $f(p)$。通过在优先队列中保存 k 个最高得分对象来执行 Map 任务中的过滤。只有这些 k 个对象需要发送到 Reduce 阶段，而不是由 Map 任务访问的 $|S'|$ 个对象。

图 3 所示为一个 2 维数据集中的 Top－k 查询过滤例子。白点和黑点对应于由两个不同 Map 任务访问的对象。黑点对象的局部 Skyline 集合用虚线连接。这些是一个 Map 任务中唯一需要发送到 Reduce 阶段的象，而剩余的黑点则被过滤。

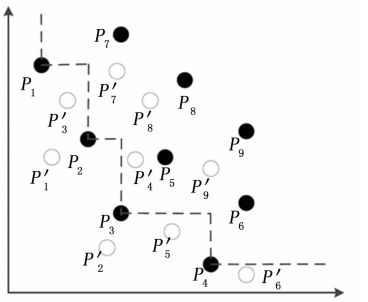

图 3 具有两个 Map 任务的 Top－k 示例，空心点对应第 1 个 Mapper，实心点对应第 2 个 Mapper

3.5 负载均衡机制

Reduce 任务的工作量由其需要处理和连接的记录数决定[12]。为了执行负载均衡，本文目标是将一些连接值分配给 Reduce 任务，以最小化每个 Reduce 任务的最大记录数，这个问题相当于多处理器调度问题。然而，多处理器调度问题是一种 NP－hard 问题，因此本文使用了一种名为 LPT（最长处理时间）的启发式算法来进行调度。该算法根据连接记录的数量对连接值进行排序，然后将它们分配给迄今为止连接总数最低的处理器（Reducer）。

3.6 基于 MapReduce 的并行实现

算法 1 展示了如何在 Map 阶段实现提前终止、数据过滤和负载均衡机制。该算法将每个输入表的分数界限作为输入，并访问排序的输入表。另外，如上所述，HashMap H 用来捕获一些连接值分配给 Reduce 任务。只要表 T_1 中的输入记录 τ 的得分低于得分边界 b_1，即 $\sum s_i\leqslant b_i$，则将该记录传递给 Reduce 任务。以此确保没有得分高于边界的记录可以产生属于 Top－k 连接的连接结果，从而可以弃用高得分记录的连接结果，显著减少需要传递和处理的记录数量。

算法 1：P－TKJ Map 阶段输入：T_0,T_1,b_0,b_1,H
输出：T_0,T_1 中分数低于 b_0,b_1 的记录
Function Map（$\tau(\tau.a_i,\tau.s_i)$）//表 T_i 中的记录
1：$r\leftarrow H.get(\tau.a_i)$
2：if（$\tau\in T_0$）then
3：if（$\tau.s_0\leqslant b_0$）then
4：$\tau.tag\leftarrow 0$
5：output［$(\tau.a_i,\tau.s_i,\tau.tag,r),\tau$］
6；else
7：if（$\tau.s_0\leqslant b_1$）then
8：$\tau.tag\leftarrow 1$
9：output［$(\tau.a_i,\tau.s_i,\tau.tag,r),\tau$］
10：执行数据过滤
11；end

算法 2 展示了 Reduce 阶段的流程。将 Map 阶段的输出键值对根据连接值（$\tau.a_i$）分组，并使用自定义分区程序分配给 Reduce 任务。在每个 Reducer 中，需要按照得分（$\tau.s_i$）的升序对每个组中的记录进行排序，这是通过使用组合键排序来实现的。Reduce 阶段的输出形式为 a，$\tau.id$，$\tau'.$

id，$f(\tau.\ \tau')$。

每个 Reduce 任务将与特定连接属性值相关的所有记录作为输入，并独立于其他 Reduce 任务，对每个这样的连接值执行 Top－k 连接。而且，由于按升序对记录进行排序访问，因此只要在存储器（M_0 和 M_1）中，从每个输入表（第 6 行）中只读取与 k 相同数量的记录即可，因为任何其他记录都不能产生 Top－k 连接结果。

算法 2 ：P－TKJ Reduce 阶段

输入：连接值 key_1，key_2，・・・的子集与相关联的记录集合 V_1，V_2・・・。

输出：连接值 key 的 Top－k 记录。

Function Reduce(key，V)

1：for ($\tau \in V$) do

2：if ($\tau.tag = 0$) then

3：载入 τ in M_0

4：else

5：载入 τ in M_1

6：if ($M_0.size() \geqslant k$)　and　($M_1.size() \geqslant k$) 则

7：执行提前终止机制

8：output $[RankJoin(k, f, M_0, M_1)]$

9：end

4　实验评估

4.1　实验设置

将算法部署在由 8 个服务器节点组成的内部 Hadoop 集群[13]中。对于 Map 和 Reduce 任务，JVM 堆大小设置为 2GB。HDFS 大小配置为 128MB，默认复制因子为 3。

使用了两种 Hadoop 平台上计算 Top－k 连接的算法进行比较，分别为传统 RSJ 算法和本文提出的 P－TKJ 算法。这两种算法的区别在于，本文 P－TKJ 算法具有提前终止、数据过滤和负载均衡机制。

对于记录数据集，使用了一个合成数据生成器来生成大量的输入数据集。输入表 T_i 的大小从 1 GB 到 50 GB。根据偏态分布（ZIPF 分布）来生成评分属性，其中偏度为 0.5，表示为 ZI0.5。改变每个表中不同连接值的数量（从 100 到 2000），从而影响连接选择性，以研究它对本文算法的影响。为了验证算法的可扩展性，本文创建了 4 个不同大小的数据集，记为 DS1－DS4。这些数据集的各个参数显示在表 1 中。另外，各种算法中都设置 Top－k 中的 $k = 10$。

对于性能指标，本文使用的主要度量是每个作业的总执行时间。另外，还测量了在 Map 和 Reduce 阶段消耗的 CPU 时间。

表 1　用于可扩展性研究的数据集

数据集	T_0 大小	T_1 大小	不同的连接值
DS1	5.5GB	6GB	500
DS2	11GB	12GB	1000
DS3	22GB	24GB	2000
DS4	44GB	48GB	2000

4.2　实验结果

图 4 给出不同数据集大小下，两种算法的总执行时间。图 5 给出了分别在 Map 和 Reduce 阶段所消耗的 CPU 处理时间。

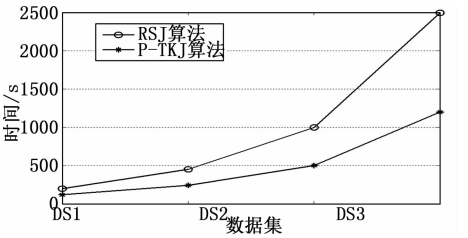

图 4　算法的总执行时间

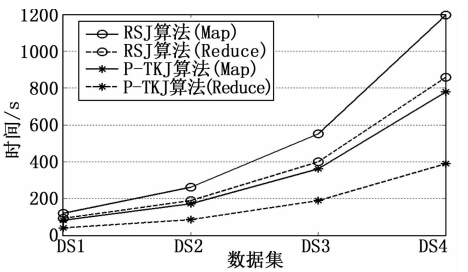

图 5　Map 和 Reduce 阶段所消耗的 CPU 处理时间

可以看出，P－TKJ 算法的执行时间优于 RSJ 算法将近 1 倍。而且，当数据集的大小增加时，优势更加明显。以上实验这有力证明了本文算法支持大量输入的可扩展性。

这是因为 RSJ 虽然为并行 Top－k 连接问题提供了一个正确的解决方案，但是它在性能方面有严重的局限性。首先，尽管直观上一小部分列表记录就足以产生正确的结果，但是它需要完整地访问两个输入表。换句话说，就磁盘访问、处理成本以及通信而言，这明显导致资源的浪费。理想情况下，如果确定已经访问过的记录能够产生正确的结果，只需要有选择地只访问几个 HDFS 块，并终止 Map 阶段的处理。其次，由于 RSJ 不使用与每个连接值关联的记录数量知识，为此其将 Map 输出键（连接值）分配给 Reduce 任务是随机执行的，这可能会导致不均衡的工作分配，从而延迟了工作的完成。

相比而言，本文使用了提前终止策略，使 Map 阶段输入记录的数量减少，所以算法比 RSJ 执行更快。另外，由于本文方法很好地对 Reducer 任务进行了负载平衡。在没有负载均衡机制时，使用 Hadoop 默认的基于散列的分区，将 Map 输出键分配给 Reducers，这本质上是一种随机分区。而由于本文的负载均衡机制，以更统一的方式将连接结果分配给 Reducers，从而以更公平的方式分配工作。另外，本文融入了数据过滤操作，减少了 Reducer 任务数量，这也一定程度上提高了算法执行速度。

为了验证不同 k 对算法性能的影响，这里设定 $k=5$、10、15、20 和 25。在 DS1 上分别进行实验，并统计相应的执行时间，结果如图 6 所示。可以看出，不同 k 值下两种算法的执行时间几乎不受影响。这是因为连接查询是消耗时间最高的操作。但 Top－k 通过在连接阶段实行部分合并，不同 k 值下所维护的元组数量基本相同，所以执行时间也基本不变。

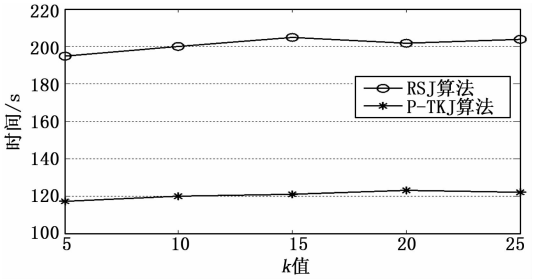

图 6 不同 k 值下的执行时间

5 结论

本文介绍了一种在 MapReduce 框架上处理 Top－k 连接的并行化计算框架。使用数据汇总，以直方图的形式表示，并将这些操作在数据上传过程中通过后台 CPU 处理，以此提高 CPU 利用率。同时利用提前终止策略、数据过滤和负载均衡策略提高了算法对数据分析访问和处理的效率。实验结果证明了提出算法的可扩展性和有效性。

参考文献：

[1] 马友忠，慈 祥，孟小峰. 海量高维向量的并行 Top－k 连接查询 [J]. 计算机学报，2015，38 (1)：86－98.

[2] Xiao Z，et al. ERA－RJN：A SPARQL－Rank Based Top－k Join Query Optimization [A]. Joint International Semantic Technology Conference [C]. Springer，Cham，2015：82－88.

[3] 屈 洁. 虚拟环境下大数据智能并行聚类方法研究 [J]. 计算机测量与控制，2017，25 (6)：257－260.

[4] Candan K S，Kim J，Nagarkar P，et al. RanKloud：Scalable Multimedia Data Processing in Server Clusters [J]. IEEE Multimedia，2011，18 (1)：64－77.

[5] Xiao S，ShengqiYang，He J，et al. A distribute Reduce Side Join algorithm [A]. International Conference on Cloud Computing and Internet of Things [C]. IEEE，2017：21－24.

[6] 慈 祥，马友忠，孟小峰. 一种云环境下的大数据 Top－K 查询方法 [J]. 软件学报，2014，25 (4)：813－825.

[7] Nakayama M，Yamazaki K，Tanaka S，et al. Dynamic profiling and feedback framework for reduce－Side Join [A]. IEEE，International Conference on Computational Science and Engineering. IEEE，2014：1255－1262.

[8] Yu Z，Yu X，Chen Y，et al. Distributed top－k keyword search over very large databases with MapReduce [A]. IEEE International Congress on Big Data [C]. IEEE，2016.

[9] 庞 俊，于 戈，许 嘉，等. 基于 MapReduce 框架的海量数据相似性连接研究进展 [J]. 计算机科学，2015，42 (1)：1－5.

[10] 刘 义，陈 荦，景 宁，等. 海量空间数据的并行 Top－k 连接查询 [J]. 计算机研究与发展，2011，48 (3)：163－172.

[11] Doulkeridis C，et al. Processing of Rank Joins in Highly Distributed Systems [Z]. 2012，41 (4)：606－617.

[12] Li D，Bai L，Wang Z F，et al. Top－k Query Based on Map Reduce [A]. International Conference on Information System and Artificial Intelligence [C]. IEEE，2017：1－4.

[13] 黄富平，梁卓浪，邢英俊，等. 云计算 Hadoop 平台的异常数据检测算法研究 [J]. 计算机测量与控制，2017，25 (7)：260－263.

(上接第 207 页)

图 2 给出了修正钟差和方位误差的 X 射线脉冲星导航和修正钟差 X 射线脉冲星导航的对比结果。两种导航方法都修正了时钟钟差对导航的影响，突出了脉冲星方位偏差对导航精度的影响。在钟差修正的情况下，修正钟差和方位误差的 X 射线脉冲星导航有效的抑制了脉冲星方位造成的系统偏差对导航精度影响。同修正钟差的 X 射线脉冲星导航，该导航方法提供了更好的导航精度。

4 总结

本文研究了时钟漂移及脉冲星方位误差对导航精度的影响，提出了一种修正钟差和脉冲星方位误差的导航方法。本文将时钟钟差与方位误差造成的系统偏差作为增广状态变量并使用多普勒差分量测，利用联邦 UKF 融合导航信息。仿真结果表明，该导航方法能够有效的抑制方位误差，提高星载时钟钟差，并提供更高的导航估计精度。

参考文献：

[1] Paul S R，Kent S W，Michael N L. Wolff. Spacecraft Navigation Using X－Ray Pulsars [J]. Journal of Guidance Control and Dynamics，2006，29 (1)：49－63.

[2] Liu J，Ma J，Tian J W. X－ray pulsar navigation method for spacecraft with pulsar direction error [J]. Advances in Space Research. 2010，46 (11)：1409－1417.

[3] Zhou Y，Wu P L，Li X X. Autonomous navigation for lunar satelliteusing X－ray pulsars with measurement faults [J]. IET Sci Meas. Technol. 2016，10 (3)：239－246.

[4] 吴 萌，刘 劲，马 杰，等. 基于脉冲星的航天器定位算法 [J]. 计算机工程与应，2010，46 (1)：203－205.

[5] Feng D Z，et al. Autonomous orbit determination and its error analysis for deep space using X－ray pulsar [J]. Aerospace Science and Technology，2014，32 (1)：35－41.

[6] Xiong K，Wei C L，Liu L D. The use of X ray pulsars for aiding navigation of satellites in constellations [J]. Acta Astronautica，2009，64 (4)：427－436.

[7] Wang Y D，Zheng W，Sun S M. X－ray pulsar－based navigation using time－differenced measurement [J]. Aerospace Science and Technology. 2014，36：27－35.

[8] Wen Y Z，Zhang Z J，Wu J. High－precision navigation approach of high－orbit spacecraft based on retransmission communication satellites [J]. Journal of Navigation，2012，65 (2)：315－362.

[9] Liu J，Fang J C. X－ray pulsar/Doppler difference integrated navigation for deep space exploration with unstable solar spectrum [J]. Aerospace Science and Technology，2015，41：144－150.

[10] 杨成伟，邓新坪，郑建华，等. 含钟差修正的脉冲星和太阳观测组合导航 [J]. 北京航空航天大学学报，2012，38 (11)：1469－1473.

[11] Liang H，Zhan Y F，Yin H L. New observation strategy for X－ray pulsar navigation using a single detector [J]. 2015，10 (6)：1107－1111.

[12] 谷树文，王安国. 基于 X 射线脉冲星的空间定位误差仿真研究 [J]. 科技导报 [J]. 2008，26 (20)：0055－0059.