基于索引的分布式RDF查询优化算法

汪璟玢 方知立
(福州大学数学与计算机科学学院 福州 350108)

摘 要 在Hadoop平台中采用索引文件来辅助查询是解决海量RDF(Resource Description Framework)查询的一种新思路。目前在Hadoop平台中实现的RDF查询都较少利用索引文件,且主要针对RDF的静态数据,对数据动态更新操作的兼容性都比较差。为了克服这两个缺点,提出IMSQ(using Index in MapReduce to Segment and Query)算法来对RDF文件进行分布式查询。该算法主要分为分割和查询两部分:首先为RDF进行一次星形分割,得到若干个分割,文件并建立索引文件;其次在查询时,按照分层生成连接计划,采用过滤选择策略,先找索引文件,缩小文件集,再对相应的分割文件进行查询;最后进行一次结果合并和输出。在LUBM数据集上进行的测试实验表明,在数据量大的情况下IMSQ方法的查询效率具有明显的优势。

关键词 Hadoop,RDF,索引,MapReduce
中图法分类号 TP391 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2014.11.045

Distributed Optimized Query Algorithm Based on Index

WANG Jing-bin FANG Zhi-li
(College of Mathematics and Computer Science, Fuzhou University, Fuzhou 350108, China)

Abstract Using index file is a new way of solving the large amount of RDF (Resource Description Framework) query problem, which can be a great aid to query optimization. At present, most of the RDF query optimization method based on Hadoop do not use index file, and most of them aim at static data so they perform poorly at dynamic updating of data. In order to overcome these two drawbacks, this paper proposed IMSQ (using Index in MapReduce to Segment and Query) algorithm to perform distributed RDF query. The algorithm can be divided into segment and query execution two parts, firstly, makes a starlike segmentation for RDF data, and obtaines several segment file and corresponding index file, secondly, generates a layered join plan, uses filter method to seek the index file to narrow the result set and then does query on corresponding segment file; finally, merges and outputs the middle result. The results of the experiment on the LUBM test data set show that IMSQ method query efficiency is higher when the amount of the RDF data is large.

Keywords Hadoop, RDF, Index, MapReduce

1 引言

RDF数据是指由本体语言描述、以三元组形式组织起来的数据,又称语义网数据[1]。早期RDF存储系统采用单一机器集中式架构模型,大多通过关系数据库来存储和查询RDF数据[2]。随着链接数据(Linked Data)运动的开展,Web上越来越多的开放数据以RDF格式发布,领域已涉及百科全书、地理信息、生命科学、政府数据和社会媒体等。截至2010年9月,链接数据的总量已达250亿条三元组[3]。因此原始的关系型数据库对海量RDF数据的存储能力和查询响应两方面都不能满足日益增长的需求。目前有不少工作讨论了海量RDF分布式存储和查询问题,RDF分布式存储主要包括HDFS和HBase两种方案[2-9]。研究者针对其存储方式的特点各自提出了分布式RDF查询策略。在文献[2,5]中,Jin Qiang提出在HBase中建立SPO,POS和OSP 3张表来存放数据,在其查询时先确定对应使用哪个表,再进行查询。文献[6-9]中主要是利用MapReduce的框架实现查询算法。Mohammad Farhan Husain等人[6]提出了一个贪心的MapReduce任务生成算法,即多个MapReduce任务迭代处理SPARQL BGP连接操作。Jaeseok Myung等人[7]提出的查询策略中首先需要创建一个MapReduce任务,从HDFS中的RDF数据文件中查找对应的RDF三元组,随后创建多个MapReduce任务迭代处理SPARQL BGP连接操作。Jieru Cheng等人[9]提出的算法中采用PredicateLead进行数据预处理,并采用JobPartitioner算法生成多个MapReduce任务进行查询连接处理得到查询结果。Buwen Wu等人[10]提出

到稿日期:2014-01-02 返修日期:2014-03-14 本文受福州大学科技发展基金项目(2013-XQ-32),空间数据挖掘与信息共享教育部重点实验室开放研究基金项目(201006),2011年福建省科技拥军基金项目(JG2011005),福建省自然科学基金项目(2012J01168)资助。
汪璟玢(1973—),女,硕士,副教授,CCF会员,主要研究领域为海量数据管理、网络数据库和智能技术,E-mail:wjbcc@263.net;方知立(1990—),男,硕士生,主要研究领域为海量数据管理、智能技术。

使用混合划分的方法来减少 MapReduce 的次数，从而达到减少频繁的 I/O 通信的目的。Liu L 等人[11]提出使用多重连接过滤的方法替代传统的 SQL 连接查询，然后在工作流中合并不同的连接计划。Mohammad Farhan Husain 等人[8]在文献[6]的基础上提出查询计划优化算法，并用多个 MapReduce 任务迭代处理 SPARQL BGP 连接操作。其中 Husain 提出的方法相比于其他文献只优化连接计划，还包含了按照谓语 P 进行分割来减少文件输入，但是以上几种方案都较少使用索引文件来辅助查询，利用 MapReduce 的查询时间相对较长，且对于数据动态更新要进行的操作都比较复杂。

为了解决上述查询效率不高和更新操作复杂的问题，本文提出了 IMSQ 算法，该算法建立了多个索引文件来辅助查询。首先将 RDF 文件以星形解析成三元组形式存储在 HDFS 文件系统中，并在此过程中建立索引文件，然后将 SPARQL 查询语言以分层生成连接计划，再利用 MapReduce 并行计算得到查询结果，最后进行一次结果的汇总输出。文章最后通过对比实验验证了 IMSQ 方法在大数据量的情况下查询效率较高。

本文首先介绍 RDF 数据模型、SPARQL 查询语言与 MapReduce 计算框架；第 3 节详细论述 IMSQ 算法，包括把 RDF 解析成三元组后分割存储到 HDFS 文件系统和建立索引的算法过程，以及采用分层生成连接计划和利用 MapReduce 并行计算的算法，并讨论在需要更新数据时 IMSQ 算法的优点；第 4 节给出实验结果并与未使用索引文件的方法及 Husain 提出的方法进行对比分析；最后总结本文工作。

2 基本概念

2.1 RDF 数据模型

RDF 的基本思想是通过统一资源标识符(Uniform resource Idetifier，URI)来标识 Web 上的资源，用简单的属性(property)以及属性值(value)来描述资源。RDF 模型可以采用不同的语法形式来描述。大多情况下都是用 XML 来刻画 RDF，其特点是简单、易于掌握和使用。

三元组是 RDF 模型的另外一种常见描述方式。每一个三元组包括一个主语(subject)、一个谓语(predicate)和一个宾语(object)。陈述的主语(subject)通常是指向相应资源(resource)的 URI，谓语(predicate)是表示某一属性(property)的 URI，而宾语(object)既可以是指向某一个资源的 URI，也可以单纯地以一个文字(literal)作为属性值(property value)。除此之外，主语和宾语也可以是没有 URI 表示的匿名资源，亦称空白节点(blank node)。本文采用以三元组形式表示的 RDF 数据。

除了以上两种方式之外，RDF 还可以用有向图的形式来描述。一个 RDF 图的结点就是它包含的所有三元组的主语和宾语，而边的方向总是指向宾语[12]。在 RDF 图建立完成之后便可通过图中内容来了解 RDF 所描述的资源的信息。基本的 RDF 图的表示形式如图 1 所示。

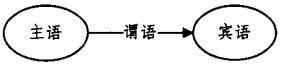

图 1 包含一个三元组的 RDF 图

2.2 SPARQL 查询语言

SPARQL(Simple Protocol and RDF Query Language)是目前 W3C(World Wide Web Consortium)针对 RDF 查询语言的推荐标准，它定义了 RDF 查询语言的语法和语义，它的语法同关系数据库查询语言 SQL 接近[2]。Triple Pattern 作为 SPARQL 中最基本的匹配单元，其构成与 RDF 三元组表示形式相对应。RDF 数据以(S，P，O)三元组表示，Triple Pattern 也由这 3 部分构成。Triple Pattern 对应位置可以是已绑定的值或是未绑定的变量，未绑定变量由问号加变量名组成。例如，以下 SPARQL 语句表示查询“工作于University1并且是 Department1 成员的教授”。

```
SELECT ?x
Where {
?x rdf:type Professor.
?x worksFor University1.
?x memberOf Department1.
}
```

2.3 MapReduce 计算框架

MapReduce 是一个编程模式，充分利用了“分而治之”的理念，使得在处理海量数据时可以对任务并行处理，这样通过协作能够轻松地完成更多任务[13]。MapReduce 并行计算框架将任务处理划分为 Map 阶段与 Reduce 阶段。Map 与 Reduce 函数的输入都是(Key，Value)值。

用户指定一个 map 函数，通过这个 map 函数处理 key/value(键/值)对，产生一系列的中间 key/value 对，并且使用 reduce 函数来合并所有相同 key 值的 value 部分后得出最终结果。

3 IMSQ 算法

IMSQ 算法的主要思想是建立索引文件，配合优化的连接计划，对用 MapReduce 输入的文件进行过滤，以减少 MapReduce输入文件的大小，从而达到减少查询时间的目的。IMSQ 算法主要包含把 RDF 解析成三元组后分割存储到 HDFS 文件系统和建立索引的算法过程、采用过滤选择策略分层生成连接计划及 SPARQL 查询执行算法两部分，算法框架如图 2 所示。

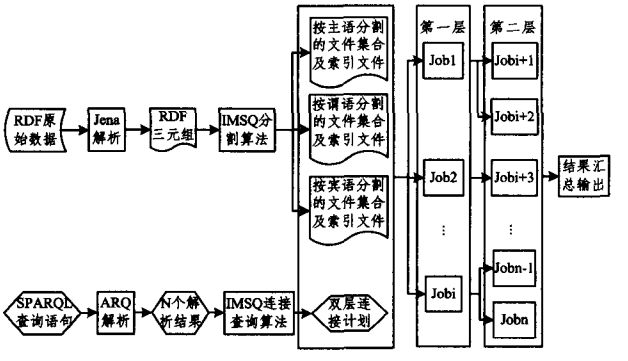

图 2 IMSQ 算法框架

首先，定义几个变量和函数：

BestMapCount：最优的 Map 个数。它与实验的节点数及其 CPU 有关，可以通过控制 *mapred. min. split. size* 的值来控制。

RDFWeight：RDF 的总文件大小。

$W(File_i)$：文件 i 的文件大小。$Best(W(File))$为最优文件大小。每个文件最优的大小为内存页大小，因为这样在加

载数据时能进一步减少磁盘 I/O 次数，在读取数据时可以减少系统在地址翻译时的页表查询次数，即提高 TLB 的命中[14]。因此在这里我们设置 $Best(W(File)) = dfs.block.size$。

FileCount：文件分割个数。MapReduce 并不适合处理数量众多的小文件，因此分割的文件个数不能太多且每个文件不能太小。最优的文件分割个数 $Best(FileCount) = RDF\text{-}Weight / Best(W(File))$。

$Write(X_i, newFile_t, indexFile)$：把 X_i 和其最终放置的文件名 $newFile_t$ 根据 X_i 的值有序地写到 *indexFile* 中，以便于二分查找。

3.1 IMSQ 分割算法

首先使用 Jena 作为 RDF 数据解析工具，将 RDF XML 格式解析成三元组格式的文件 F，然后对 F 中的所有三元组的主语(S)、谓语(P)、宾语(O)做 3 步操作(以宾语为分割对象为例)：第 1 步，把相同分割对象的三元组放在一个文件当中，得到多个文件；第 2 步，通过文件大小对所有文件进行排序；第 3 步，对排序后的文件进行合并，使文件尽量接近 $Best(W(File))$。具体算法如下：

1. Beign;
2. 步骤 1：对于任意一个三元组，$\forall(S_i, P_j, O_k)$，
3. if $\exists File_{ot}$ and$(S_u, P_v, O_k) \in File_{ot}$，
4. then$(S_i, P_j, O_k) \in File_{ot}$；
5. else new $File_{om}$，let $(S_i, P_j, O_k) \in File_{om}$。
6. 步骤 2：new List
7. $\forall File_i (i=1,2,\cdots,n)$，Put $File_i$ into List，
8. Sort(W (List)) 使 W(List(i))<W(List(i+1)) (i=0,1,…,n−2).
9. 步骤 3：Let t=0
10. While t<=n do
11. if $\sum_{i=t}^{j} W(List(i)) < Best(W(File))$ and $\sum_{i=t}^{j+1} W(List(i)) > Best(W(File))$
12. Then $newFile_k$
13. put List(i)(i=t,t+1,…,j) into $newFile_k$，
14. Write $(O_i, newFile_k, o\_indexFile) (i=t,\cdots,j \&\& (S_j, P_k, O_i) \in List(i))$
15. and Let t=j+1
16. done
17. End

通过这 3 步就可以建立以宾语(O)为分割对象的分割文件集合 o_set 及对应的索引文件 o_indexFile。同理分别建立以主语(S)为分割对象的分割文件集合 s_set 及对应的索引文件 s_indexFile 和以谓语(P)为分割对象的分割文件集合 p_set 及对应的索引文件 p_indexFile。

3.2 IMSQ 连接查询算法

定义 1(连接计划) 给定 SPARQL 查询 Q，经过 ARQ 解析后生成 Triple Pattern 子句集 $T=\{T_1, T_2, T_3, \cdots, T_n\}$。分析所有 Triple Pattern 子句集中变量间的关系，得到一个最优的连接方案，称为该 SPARQL 查询的连接计划。

定义 2(单变量语句、多变量语句) 只有一个变量未知的语句称为单变量语句。多个变量未知的语句称为多变量语句。

在进行查询之前首先选用开源工具 ARQ 来进行 SPARQL 查询语句解析。以 LUBM 开源查询语句中的 Q8 为例，图 3 是 LUBM Q8 的语句，图 4 是 LUBM Q8 的 ARQ 的解析结果。

```
SELECT ?X,?Y
WHERE
{?X rdf:type ub:Student. //1
?Y rdf:type ub:Department. //2
?X ub:memberOf ?Y. //3
?Y ub:subOrganizationOf <http://www.University0.edu>. //4
?X ub:emailAddress ?Z//5
}
```

图 3 LUBM Q8

```
(Project(?X ?Y)
(bgp
(triple ?X rdf:type ub:Student)
(triple ?Y rdf:type ub:Department)
(triple ?X ub:memberOf ?Y)
(triple ?Y ub:subOrganizationOf <http://www.University0.edu>)
(triple ?X ub:emailAddress ?Z)
))
```

图 4 LUBMQ8 的 ARQ 解析结果

Triple Pattern 间的关联性通过它们之间是否存在共同的未绑定变量来判定。LUBM Q8 共有 5 个 Triple Pattern 子句，其中 1、3、5 拥有共同变量 x，故可连接得到(x,y,z)；2、3、4 拥有共同变量 y，也可连接得到(x,y)。由此可见，同一个查询语句存在多种不同的连接方案。如 LUBMQ8 连接方案可以是 1→3→5→2→4 或者 2→3→4→1→5 等多种方案。然而连接方案的优劣很大程度上决定了 SPARQL 查询的效率。为了提高 RDF 查询效率，本文提出的 IMSQ 算法中使用分层连接计划，把单变量语句和多变量语句分开，单变量语句并行查询，查询的中间结果经过过滤选择策略之后作为多变量语句查询的输入，完成多变量语句并行查询。

本文提出的 IMSQ 连接查询算法结合 3.1 节提到的在分割阶段建立的 3 个索引文件以及相应的文件集，主要包含 3 个步骤：1)文件过滤；2)单变量查询；3)多变量连接查询。3 个步骤及其输入输出流程如图 5 所示。

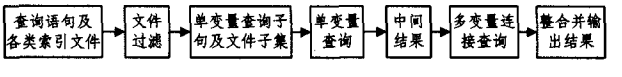

图 5 IMSQ 连接查询算法框架

3.2.1 文件过滤

MapReduce 中 Map 正常的并行规模大致是每个节点(node)大约 10 到 100 个，HDFS 的 Block 默认大小是 64MB，因此如果当数据量达到 10GB、100GB，甚至 TB 级别的时候，产生的 Map 数量巨大，因此如果希望提高查询效率，则需缩小 Input 的文件大小。

本文的文件过滤，主要根据常量的位置使用 3.1 节提到的索引文件和对应的分割的文件集，使用 binarySearch 二分查找的方法，输入已知的宾语或主语，输出其所在的文件子集，达到缩小查询范围的效果。设变量为?x。

输入：查询语句 Q。

输出：文件子集。

算法：

1. Begin
2. if query=?x p_j o_i, $File_i$=binarySearch(o_i) in o_indexFile;
3. else if query=s_i p_j?x, $File_i$= binarySearch (s_i) in s_indexFile;
4. else if query=s_i?x o_j, $File_i$=binarySearch(s_i) in s_indexFile or $File_i$=binarySearch(o_j) in o_indexFile;
5. output $File_i$;
6. End

例如，LUBM Q8 通过图 4 的 ARQ 解析之后的查询语句 Q，选出其中的单变量语句为：1.（triple ?X rdf:type ub:Student）；2.（triple ?Y rdf:type ub:Department）；3.（triple ?Y ub:subOrganizationOf〈http://www.University0.edu〉）。对语句 1 在 o_indexFile 中过滤得到 ub:Student 所在的文件 $File_i$，对语句 2 在 o_indexFile 中过滤得到 ub:Department 所在的文件 $File_j$，对语句 3 在 o_indexFile 中过滤得到〈http://www.University0.edu〉所在的文件 $File_k$。

3.2.2 单变量查询

通过 ARQ 解析之后的查询语句 Q，选出其中的单变量语句进行查询。对每个单变量语句起一个 MapReduceJob 任务，由于各个 MapReduceJob 查询的范围不同，各个 MapReduceJob 可以并行处理。单个 MapreduceJob 的具体过程如下：

输入：经过文件过滤后的 RDF 文件子集。

Map 阶段：输入查询的 RDF 三元组文件，遍历每个三元组过滤所有 Triple Pattern 子句，若三元组满足查询子句的条件，则生成一对(key,value)。其中，key 为变量，value 为三元组。

Reduce 阶段：完成该变量对应的多个 Triple Pattern 子句的连接，得到一个变量中间结果集。

最后并行做完所有的 MapReduceJob，便可以得到所有单变量语句查询的中间结果集。

例如，由 3.2.1 节可知，LUBM Q8 一共有 3 条单变量语句，所以对语句 1 起一个 MapReduceJob，输入为 $File_i$，在 Map 阶段遍历每个三元组，满足谓语为 rdf:type，宾语为 ub:Student，则输出(X，主语)；在 Reduce 阶段则只连接 Map 的输出，输出(X，主语集合)，设主语集合为 $X1$。同理，对语句 2 起一个 MapReduceJob，输入为 $File_j$，在 Map 阶段遍历每个三元组，满足谓语为 rdf:type，宾语为 ub:Department，则输出(Y，主语)，在 Reduce 阶段则只连接 Map 的输出，输出(Y，主语集合)，设主语集合为 $Y1$；对语句 3 起一个 MapReduceJob，输入为 $File_k$，在 Map 阶段遍历每个三元组，满足谓语为 ub:subOrganizationOf，宾语为〈http://www.University0.edu〉，则输出(Y，主语)，在 Reduce 阶段则只连接 Map 的输出，输出(Y，主语集合)，设主语集合为 $Y2$。

3.2.3 多变量连接查询

假设 3.2.2 节取得的变量为?x,?y,?z。其中间结果集分别为 $X1,X2,\cdots,Xm,Y1,Y2,\cdots,Yn,Z1,Z2,\cdots,Zt$。

步骤 1 对 3.2.2 节中得到的同一变量的多个结果集分别取交集，即 $X=X1\cap X2\cap\cdots\cap Xm$，$Y=Y1\cap Y2\cap\cdots\cap Yn$，$Z=Z1\cap Z2\cap\cdots\cap Zt$。这样就可以在多变量连接前减小各自结果集大小。

步骤 2 此时，我们可以把?x,?y,?z 当成已知中间结果的变量，对?x,?y,?z 分别做 3.1.1 节中提到的文件过滤，输入分别为 X,Y,Z，得到文件子集 FX,FY,FZ。

步骤 3 分别对多变量查询语句选择 MapReduce 查询的文件子集。

由于不同谓语的数量较少，而同个谓语的三元组数量很多，因此如果谓语是已知中间结果集的变量，选择查询以谓语为分割对象的索引文件和文件子集其过滤效果不如选择非谓语的变量好，所以选择文件子集的策略是忽略谓语变量，选择已知中间结果并且通过步骤 2 得到的文件子集较少的文件子集作为输入。

步骤 4 分别对每一条多变量查询语句起一个 MapReduceJob 任务，各个 MapReduceJob 并行处理。查询过程与单变量查询一致。具体过程如下：

输入：把步骤 3 得到的文件子集作为输入。把步骤 1 得到的变量的中间结果集作为配置的参数传入。

Map 阶段：输入查询的 RDF 三元组文件，用变量的中间结果集替换多变量查询语句的相应变量，遍历每个三元组过滤所有 Triple Pattern 子句，若三元组满足查询子句的条件，则生成一对(key,value)。其中 key 为多变量，value 为三元组。

Reduce 阶段：完成变量对应的多个 Triple Pattern 子句的连接，得到多变量结果集。

最后并行做完所有的 MapReduceJob，便可得到所有多变量语句查询的结果集 Q_1-Q_n。再对 Q_1-Q_n 相同变量取交集 L，再在 $Q_1\sim Q_n$ 中找到 L 对应的变量集合 $Q_{V1}-Q_{Vn}$，得到最终的结果 $Q=Q_{V1}\cup Q_{V2}\cdots\cup Q_{Vn}$，输出结果 Q。

例如，通过 3.2.2 节我们可以得到 LUBM Q8 的单变量中间结果集为 $X1,Y1,Y2$。

1. $Y1$ 和 $Y2$ 取交集，设为 Y，即 $Y=Y1\cap Y2$。

2. 在 s_indexFile 中过滤得到 $X1$ 和 Y 对应的文件集合分别为 $File\_X1$ 和 $File\_Y$。

3. 对于多变量查询语句：4.（triple ?X ub:memberOf ?Y），5.（triple ?X ub:emailAddress ?Z），如果 $File\_X1$ 的元素个数小于 $File\_Y$ 的元素个数，则语句 4 输入的文件为 $File\_X1$，否则输入的文件为 $File\_Y$。对于语句 5，输入的文件为 $File\_X1$。

4. 对语句 4 起一个 MapReduceJob，输入为 $File\_X1$ 和 $File\_Y$ 元素个数较少者，把 $X1$ 和 Y 作为配置参数传入，在 Map 阶段先用 $X1$ 和 Y 替换?X 和?Y，遍历每个三元组，满足主语属于 $X1$ 谓语为 ub:memberOf 宾语属于 Y，则输出(XY，三元组)，在 Reduce 阶段则只连接 Map 的输出，输出(XY，三元组集合)，设三元组集合为 $Q1$。同理对语句 5 起一个 MapReduceJob，输入为 $File\_X1$，把 $X1$ 作为配置参数传入，在 Map 阶段先用 $X1$ 替换?X，遍历每个三元组，满足主语属于 $X1$ 谓语为 ub:emailAddress，则输出(XZ，三元组)，在 Reduce 阶段则只连接 Map 的输出，输出(XZ，三元组集合)，设三元组集合为 $Q2$。在 $Q1$ 和 $Q2$ 中找到与 X 相同的三元组，$Q1$ 输出对应的 XY，$Q2$ 输出对应的 XZ，则最终结果 XYZ 为两者的并集。

3.3 IMSQ 算法在动态数据更新中的运用

我们可以假设几种情况：LUBM 数据集中的 student 与 course 并不是固定不变的，可能出现新增某些 student 与 course 的关系，或者有些 student 一个学期可能会变更其对应的 course，或者某个 course 可能出现老师有事没法上课必须删除等多种情况。过去的海量 RDF 查询算法都较少涉及动

态数据更新问题，在遇到数据更新（如增加，删除，修改）时所需要的耗时都比较大，本文提出的 IMSQ 算法在动态数据更新上，特别是小范围的数据更新上具有其独特的优势。

（1）新增数据。每条三元组数据只需要先查找 3 个索引文件，找到对应的文件子集，添加数据。若找不到对应的文件子集，则新增一个文件，写入对应的索引文件。

（2）修改数据。每条三元组数据先根据原始三元组查找 3 个索引文件，找到对应的文件子集，查找文件子集找到原始的三元组并删除。再按照新增操作添加新三元组。

（3）删除数据。根据要查找的三元组查找 3 个索引文件，找到对应的文件子集，查找文件子集找到原始的三元组并删除。

建立主语（S）、谓语（P）和宾语（O）的分割文件集合及对应的索引文件，表面上存储量变大，但是对于数据更新操作，只需要查询索引文件及其中的一些文件子集。这相对于全部数据都删除并重新添加，特别是小范围的数据更新要快很多。

4 实验分析

实验所使用的硬件环境为 Intel（R）Core（TM）i5-3570，CPU 3.40GHz 双核双线程，内存 2GB，磁盘 500 GB，转速 7200 rpm。软件环境操作系统为 Linux Ubuntu，采用 java 作为编程语言，开发环境为 eclipse。在实验环境中，用表 1 所列配置作为本系统 Hadoop 集群的配置，共计 5 台，其中 1 台作为 HDFS 的名称节点兼 MapReduce 的主节点，4 台为 HDFS 的数据节点兼 MapReduce 的从节点。集群工作站的基本配置如表 1 所列。

表 1　集群工作站的基本配置

CPU	Intel(R) Core(TM) i5-3570M
内存	2G
硬盘	500 GB SATA
Java	JDK1.6
Hadoop	Hadoop 1.0.4

本文采用 LUBM 工具分别生成了 100、200、500 所大学的测试。数据集的三元组数目、XML 文件大小以及解析后的三元组文件大小如表 2 所列。

表 2　LUBM 文件大小说明

LUBM 学校个数	三元组个数	XML 文件大小	解析后文件大小
100	1340 万	1.1G	2.04G
200	2754 万	2.13G	4.20G
500	6884 万	5.33G	10.5G

本文针对 LUBM 数据集建立了 3 个索引，表 3 列出了针对 LUBM100、LUBM200、LUBM500 的主语索引文件和宾语索引文件的大小。

表 3　索引文件大小说明

LUBM 学校个数	主语索引文件	宾语索引文件
100	143MB	88MB
200	268MB	180MB
500	639MB	387MB

在此数据集上将 IMSQ 算法与文献[6]中提出的按 P 分割、MapReduce 迭代查询和不使用索引并只启用单个 MapReduce 的方法进行了对比实验。

本实验重点对比了算法的查询性能。在 100 所、200 所、500 所学校的数据集上，分别从 LUBM 中 14 个标准查询选取了其中 8 个查询语句进行了测试实验，对比的实验结果如表 4—表 6 所列，查询时间单位为秒（s）。

表 4　LUBM(100) 对比实验

LUBM(100)	Q1	Q2	Q4	Q6	Q7	Q8	Q9	Q10	Q14
M. F. Husain	99	311	364	127	296	424	466	124	93
不使用索引	82	238	264	95	256	317	331	102	75
IMSQ	89	252	271	98	280	345	348	103	87

表 5　LUBM(200) 对比实验

LUBM(200)	Q1	Q2	Q4	Q6	Q7	Q8	Q9	Q10	Q14
M. F. Husain	175	551	604	226	356	694	763	231	171
不使用索引	157	466	509	180	417	578	595	192	155
IMSQ	147	427	451	172	378	533	545	182	147

表 6　LUBM(500) 对比实验

LUBM(500)	Q1	Q2	Q4	Q6	Q7	Q8	Q9	Q10	Q14
M. F. Husain	732	1723	1873	772	1580	2107	2186	921	756
不使用索引	690	1583	1704	708	1242	1862	1938	804	715
IMSQ	300	867	972	405	823	1213	1359	407	373

从实验对比结果可见，在选取的 8 个查询语句中，IMSQ 算法与文献[6]中 M. F. Husain 等人提出的方法以及不使用索引并只启用单个 MapReduce 的方法相比，在数据量较小的情况下，查询效率相差不大；但是在数据量大的情况下，查询效率高很多。这是因为在查询数据量不大的情况下，使用 IMSQ 算法分割数据的优势并不十分明显，虽然进行了分割，但是对于 Q2、Q4、Q8、Q9 仍需启动多个 job，当 job 并行运算时有可能会出现资源竞争的情况，从而导致某个资源在同一时间只能让某个 job 独占，从而耗费了一定时间。但是当数据量增大时，IMSQ 算法分割数据的粒度更细，且在查询时结合了索引文件来优化，把中间结果替换成多变量查询语句的变量，输入较小的数据文件进行连接查询，从而大大提高了查询效率。下面以 Q2 为例分析。

```
SELECT ?X,?Y,?Z
WHERE
{ ?X rdf:type ub:GraduateStudent. //1
  ?Y rdf:type ub:University. //2
  ?Z rdf:type ub:Department. //3
  ?X ub:memberOf ?Z. //4
  ?Z ub:subOrganizationOf ?Y. //5
  ?X ub:undergraduateDegreeFrom ?Y} //6
```

Q2 查询中共有 6 个查询子句、3 个查询变量。IMSQ 算法在单变量查询时可以查出 1，2，3 查询子句中 X，Y，Z 的取值集合，然后通过比较集合大小可以知道 Z 的取值最少，Y 次之，X 最多，因此对于查询子句 4 选择查询 Z 取值对应的文件集合比 X 取值的文件集合少，从而达到更好地缩小查询范围的效果。同理查询子句 5 选择查询 Z 取值对应的文件集合，查询子句 6 选择查询 Y 取值对应的文件集合。IMSQ 算法因为在缩小查询范围时的效果比 M. F. Husain 提出的方案要好，所以查询时间优于 M. F. Husain 提出的方案和不使用索引并只启用单个 MapReduce 的方案。

结束语　本文主要提出了一种在 Hadoop 平台上利用索引文件对大规模 RDF 数据进行查询的方法，通过对比实验验证了本文提出的 IMSQ 算法在数据规模越大的情况下越具备一定的优越性。本文在建立索引的过程中耗时相对较大，在后续研究过程中将重点研究如何更快地建立索引文件。

参 考 文 献

[1] 李慧颖,瞿裕忠. 基于关键词的语义网数据查询研究综述[J]. 计算机科学,2011,38(7):18-23

[2] 金强. 基于 Hase 的 RDF 存储系统的研究与设计[D]. 杭州:浙江大学,2011

[3] 王鑫,冯志勇,杜朴风,等. Jingwei:一种分布式大规模 RDF 数据服务器[J]. 计算机研究与发展,2011,48(Suppl.):451-455

[4] Li L,Song Y. Distributed Storage of Massive RDF Data Using HBase[J]. Journal of Communication and Computer, 2011, 8(5):325-328

[5] Sun J,Jin Q. Scalable rdf store based on hbase and mapreduce [C]// 2010 3rd International Conference on Advanced Computer Theory and Engineering (ICACTE). IEEE,2010:633-636

[6] Husain M F, Doshi P, Khan L, et al. Storage and retrieval of large rdf graph using hadoop and mapreduce[M]. Cloud Computing. Springer Berlin Heidelberg,2009:680-686

[7] Myung J,Yeon J,Lee S G. SPARQL Basie Graph Pattern Processing with Iterative MapReduce [C] // Proceedings of the Workshop on Massive Data Analytics on the Cloud (MDAC' 10). 2010:6-12

[8] Husain M, McGlothlin J, Masud M M, et al. Heuristics-Based Query Processing for Large RDF Graphs Using Cloud Computing[J]. IEEE Transactions on Knowledge and Data Engineering,2011,23(9):1312-1327

[9] Cheng J,Wang W,Gao R. Massive RDF Data Complicated Query Optimization Based on MapReduce [J]. Physics Procedia, 2012,25:1414-1419

[10] Wu B,Jin H,Yuan P. Scalable SAPRQL querying processing on large RDF data in cloud computing environment[C]// Pervasive Computing and the Networked World. Berlin Heidelberg: Springer,2013:631-646

[11] Liu L,Yin J,Gao L. Efficient Social Network Data Query Processing on MapReduce[C]// Proc of the 5th ACM workshop. New York:ACM,2013:27-32

[12] 刘翔宇,吴刚. 基于 Prôfer 序列的 RDF 数据索引与查询[J]. 计算机学报,2011,34(10):1997-2008

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

[14] 袁平鹏,刘谱,张文娅,等. 高可扩展的 RDF 数据存储系统[J]. 计算机研究与发展,2012,49(10):2131-2141

(上接第 211 页)

长度为固定值的无证书环签密,且进一步提高方案的效率,是笔者下一步将继续展开的工作,同时对它的研究也具有重要的理论和现实意义。

参 考 文 献

[1] Al-Riyami S S, Paterson K G. Certificateless public key cryptography[C]// Proceedings of ASIACRYPT 2003. Berlin:Springer-Verlag,2003:452-473

[2] Waters B. Efficient identity-based encryption without random oracles[C]// Proceedings of EUROCRYPT 2005. Berlin:Springer-Verlag,2005:114-127

[3] Gentry C. Practical identity-based encryption without random oracles[C]// Proceedings of EUROCRYPT 2006. Berlin:Springer-Verlag,2006:445-464

[4] Zheng Y L. Digital signcryption or how to achieve cost(signature & encryption) << cost(signature)+cost(encryption)[C]// Proceedings of CRYPTO 1997. Berlin: Springer-Verlag, 1997: 165-179

[5] Rivest R L,Shamir A,Tauman Y. How to leak a secret[C]// Proceedings of ASIACRYPT 2001. Berlin: Springer-Verlag, 2001:552-565

[6] Huang X Y,Susilo W,Mu Y,et al. Identity-based ring signcryption schemes: cryptographic primitives for preserving privacy and authenticity in the ubiquitous world[C]// Proceedings of the 19th International Conference on Advanced Information Networking and Applications. Washington DC,IEEE Computer Society,2005:649-654

[7] Zhang M, Yang B, Zhu S, et al. Efficient secret authenticatable anonymous signcryption scheme with identity privacy[C]// Proceedings of Intelligence and Security Informatics(ISI 2008). Berlin:Springer-Verlag,2008:126-137

[8] Li F G, Masaaki S, Tsuyoshi T. Analysis and improvement of authenticatable ring signcryption scheme[J]. Journal of Shanghai Jiaotong University(Science),2008,13(6):679-683

[9] Zhun L J,Zhang F T. Efficient id-based ring signature and ring signcryption schemes [C] // Proceedings of CIS 2008. IEEE Press,2008:303-307

[10] Zhu Z C,Zhang Y Q,Wang F J. An efficient and provable secure identity-based ring signcryption scheme[J]. Computer Standards & Interfaces,2009,31(6):1092-1097

[11] Sharmila D S S,Sree V S,Pandu R C. On the security of identity based ring signcryption schemes[C]// Proceedings of ISC 2009. Berlin:Springer-Verlag,2009:310-325

[12] Zhao Z M, Yu T, Ren X F. Efficient identity-based ring signcryption scheme in the standard model[EB/OL]. [2013-03-20]. http://www.joics.com/publishedpapers/2013_10_5_1471_1478.pdf

[13] Sun H,Wang A M,Zheng X F. Provably secure identity-based threshold ring signcryption scheme in standard model[J]. Computer Science,2013,40(5):131-135

[14] Zhu L J, Zhang F T, Miao S Q. A provably secure parallel certificateless ring signcryption scheme[C]// Proceedings of 2010 International Conference on Multimedia Information Networking and Security. IEEE Press,2010:423-427

[15] Qi Z H,Yang G,Ren X Y. Provably secure certificateless ring signcryption scheme[J]. China Communications,2011,8(3):99-106

[16] Qin H S,Zhang L,Feng Y Q,et al. Certificateless ring signcryption scheme without paring design[J]. Computer Engineering and Design,2013,34(3):841-844