HMSST+:基于分布式内存数据库的HMSST算法优化

董书瞳 汪璟玢 陈 远
(福州大学数学与计算机科学学院 福州 350108)

摘 要 为了解决HMSST(HashMapSelectivityStrategyTree)算法在集中式环境下受限于有限内存的问题,提出了一种新的分布式SPARQL查询优化算法HMSST+。该算法基于Redis提出了一种分布式存储方案,通过平行扩展存储节点和分布式调度,使得海量RDF数据的查询得以在分布集群的内存中实现。采用LUBM1000所大学的测试数据集对查询策略进行了实验,结果表明提出的方法与HMSST算法相比具有更好的扩展能力,与现有的分布式查询方案相比也具有更好的查询效率。

关键词 RDF,Redis,分布式存储,内存数据库,SPARQL
中图法分类号 TP391 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2016.3.040

HMSST+:HMSST Algorithm Optimization Based on Distributed Memory Database

DONG Shu-jian WANG Jing-bin CHEN Yuan
(College of Mathematics and Computer Science, Fuzhou University, Fuzhou 350108, China)

Abstract To solve the bottleneck of HMSST(HashMapSelectivityStrategyTree) algorithm which is limited to the memory in a centralized environment, this paper proposed a novel distributed SPARQL optimized query algorithm named HMSST+. This algorithm presents a distributed storage solution based on the Redis(Remote Dictionary Server), and realizes the query of massive RDF data in the memory of distributed cluster by a parallel expansion of storage nodes and distributed scheduling. The method was tested on LUBM Benchmark and it worked well when the number of universities reaches 1000. The result shows that the method has better scalability than the HMSST algorithm and higher query efficiency than the existing query schemes.

Keywords RDF, Redis, Distributed storage, Memory database, SPARQL

1 引言

随着互联网的高速发展,Web上的数据呈现出指数级的增长,这对大规模的数据存储和查询提出了要求[1]。RDF(Resource Description Framework)作为语义网中资源的描述标准,在语义网中已经占据了重要的地位。RDF格式数据的大量涌现,使得RDF数据存储以及高效的查询面临着巨大的挑战,大规模RDF数据的存储和查询成为当前迫切需要解决的问题。早期的研究主要集中在利用关系型数据库来存储RDF数据,如4Store[2]、RDF-3X[3]等。然而传统关系型数据库在日益增长的数据面前遭遇了难以跨越的存储瓶颈,且RDF数据的无模式特征使得其难以使用关系型数据库管理系统的查询优化策略[4]。因此研究人员开始将目光投向文件存储,以期利用文件系统所具备的海量存储能力来解决当前RDF面临的各项问题。Weiss等人采用基本的三元组方式存储RDF数据,在此基础上根据元组的主语、谓语和宾语的顺序建立6个索引来提高查询效率[5]。该方法在一定程度上提高了SPARQL的查询效率,但是耗费了大量的存储空间;而且基于文件系统的存储方式对于大规模RDF数据的查询效率很低。文献[6]提出的HMSST算法通过结合文件存储和内存高速运算的能力,先通过主语所属的类别缩小查询数据的范围,然后通过优化算法分步加载数据到内存中进行运算,一定程度上解决了文件系统查询效率低以及内存有限这个瓶颈。但其仍然受限于集中式环境下有限的内存,随着数据量的无限增大,内外存交换所消耗的时间会越来越多,因此难以扩展到超大规模的RDF数据集上。本文提出的HMSST+算法通过结合Redis[7,8]内存数据库和分布式存储技术,将HMSST算法应用于Redis分布式集群,解决了HMSST算法在集中式环境下受限于有限内存的瓶颈,同时将集中式的SST(SelectivityStrategyTree)算法应用于分布式环境。实验表明HMSST+算法具有与HMSST算法同样强度的查询性能,同时具有更好的扩展能力。

本文第2节介绍SPARQLR相关概念和Redis;第3节给出HMSST+算法的存储策略以及所采用的查询方案;第4节使用LUBM[9]1000所大学10.6GB的数据集进行相关查询测试,以验证算法的有效性;最后总结全文。

到稿日期:2015-01-19 返修日期:2015-05-15 本文受福州大学科技发展基金资助项目(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),2014年福建省科技拥军基金项目(JG2014001),福建省自然科学基金项目(2012J01168)资助。
董书瞳(1990-),男,硕士生,主要研究领域为海量数据管理、智能技术,E-mail:454884957@qq.com;汪璟玢(1973-),女,硕士,副教授,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(通信作者)。

2 相关基础知识

2.1 RDF

RDF(Resource Description Framework)是 W3C (World Wide Web Consortium)提出的描述 Web 信息的通用语言。RDF 作为一种 Web 上的知识表示语言,提出用一个简单的模型来表示任意类型的数据,这个模型采用三元组(S,P,O)来描述 Web 信息,其中 S 代表主语(Subject),P 代表谓语(Predicate),O 代表宾语(Object)。从图的角度来看,该模型由节点以及节点之间的边组成,节点分别表示主语 S 和宾语 O,连接节点 S 和节点 O 的边表示谓语 P,这样就可以用节点来表示 Web 上的资源,用边来表示资源的属性。因此,这个数据模型可以方便地描述对象(或者资源)以及它们之间的关系。

2.2 SPARQL 与 Triple Pattern

SPARQL 是 W3C 提出的 RDF 标准查询语言,它的语法同关系数据库查询语言 SQL 接近。图 1 展示了一个 SPARQL 语句,该查询语句包含了一个由 6 个 Triple Pattern[10] 子句组成的 Basic Graph Pattern(BGP)。SPARQL 语言的特点在于其查询构建在模式匹配上。Triple Pattern 是 SPARQL 中最基本的匹配单元,多个 Triple Pattern 子句可以组成更为复杂的模式匹配块,如 BGP 等。

```
BasicGraphPattern{ SELECT ?x?y?z WHERE{
                   ?x rdf:type ub:GraduateStudent.
                   ?y rdf:type ub:University.
                   ?z rdf:type ub:Department.
                   ?x ub:memberOf ?z.
                   ?z ub:subOrganizationOf ?y.
                   ?x ub:undergraduateDegreeFrom ?y.
                   }
```

图 1 SPARQL 示例

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 对应位置可以是绑定了的值或是未绑定的变量,未绑定变量由问号加变量名组成。多个 Triple Pattern 子句间共用同一个变量,表示这些子句间存在连接关系。

2.3 Redis

Redis(Remote Dictionary Server)是一个使用 ANSI C 语言开发的开源的 Key-Value 存储系统,它和目前较流行的 Memcached 类似,都是基于内存(缓存)的数据存储方式,不同的是 Redis 支持的数据类型更加丰富,如可以使用 String、List、Set、Hash 等数据结构,并且对每种数据结构提供了丰富的操作。同时,Redis 不同于 Memcached 之处在于它会将更新的数据异步地持久化到硬盘中或者把进行过修改的操作写入日志文件中。Redis 虽然是 Key/Value 形式的数据库,但是它吸收了部分关系型数据库的优点,如在能保存 List 和 Set 类型的数据的同时,还能完成排序等高级功能,同时在实现 INCR(自增)、SETNX(若不存在 Key 则创建并设值)等功能时保证其操作的原子性。在此基础上,还实现了 Master-Slave(主从)同步。

3 HMSST+存储与查询策略

3.1 基于 Redis 的分布式存储系统的设计

随着大量 RDF 数据的涌现以及云计算技术的发展,当前的大部分研究已向分布式平台转移。Redis 的键值对存储机制使其查询效率几乎可以看成是常数时间,然而缺点是消耗的内存比较多。目前的 Redis 本身不具备分布式集群特性,因此内存大小是其处理数据量大小的瓶颈。本文在使用 Redis 的同时,提出了一种分布式存储方案,通过动态地增加 Redis 节点,来解决单台电脑内存有限的瓶颈。

本方案包含 3 种节点:ManageNode、ProcessNode 和 StorageNode。ManageNode 主要与用户接口进行交互,存储时,ManageNode 获取输入的数据,将数据存储在指定的StorageNode中;查询时,ManageNode 获取输入的查询语句,对查询语句进行分析,对于 S 或 O 已知的简单查询语句,ManageNode 可以直接进行查询,而对于 S 或 O 未知的复杂查询语句,ManageNode 则会生成查询计划,将查询任务分成多个子任务并发送给 ProcessNode 执行。ProcessNode 主要获取从 ManageNode 发送过来的子任务,从对应的 StorageNode 获取子任务所需的数据进行执行,然后返回执行结果。StorageNode 即 Redis 存储节点,主要用于存储数据,与 ManageNode 以及 ProcessNode 进行数据交互。图 2 展示 HMMSST+算法的系统结构。

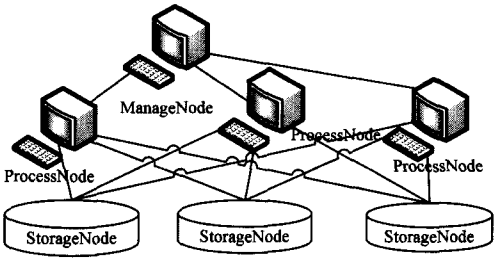

图 2 系统结构

3.2 HMSST+数据分配存储方案

由于现有的大部分查询语句都是在 Type、P 已知的条件下进行查询的,因此本文采用了先根据三元组主语所在的 Type 进行分类、再根据三元组的谓语进行分类的分配方案。具体如下:

1)将 RDF 数据先按三元组的主语所属于的类进行划分,将每个类映射到一个或多个 StorageNode。

2)根据本体中的定义,获取该类所有谓词,为每一个谓词建立以该谓词命名的 Set;然后将所有主语为同一个类且谓语相同的三元组的不重复主语放置于该 Set 中,再为每个主语建立以主语和谓语命名的 Set,以存放相同主语和相同谓语的所有宾语。

3)建立谓语反转备份。根据同一个谓语,建立一个以谓语加 R 为后缀命名的 Set,将主语为同一个类且谓语相同的三元组的不重复宾语放置于该 Set 中,再为每个宾语建立以谓语和宾语命名的 Set,以存放相同宾语和相同谓语的所有主语。

因此无论是对于主语已知还是宾语已知的查询语句,该方案都可以有效地缩小查询范围,提高查询效率。

本文数据存储分配方案的存储示意图如图 3 所示。

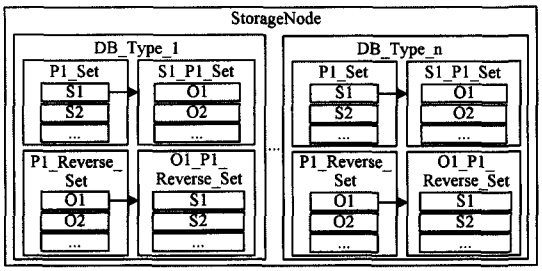

图 3　存储示意图

3.3　连接选择策略

在关于 Basic Graph Pattern(BGP)的模式匹配中，如果通过某种策略能够保证在查询结果正确的前提下使其查询过程代价(这里指时间上)降低，则称该策略是关于 BGP 的一个优化策略。

SST(SelectivityStrategyTree)：SST 连接选择策略是文献[6]提出的一种生成连接计划的方案，该策略可以快捷地计算出最优的连接选择方案。

SST 连接选择策略通过对查询语句的分析，从对应存储节点中获取对应 P 存储集合中不重复主语的个数以及不重复宾语的个数，生成选择策略树，如图 4 所示。

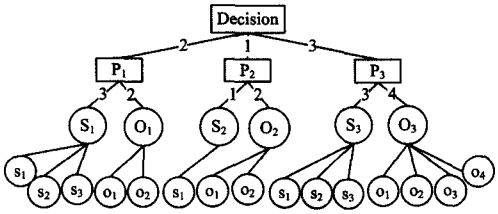

图 4　SST 选择策略树

选择策略树包含一个根节点即 Decision 节点，负责生成连接方案。Decision 节点下一级是每条查询语句中由谓词生成的 P 节点(不包括 Type)，每个 P 节点包含两种子节点：S(主语)节点和 O(宾语)节点。以谓词 P_1 节点为例，P_1 节点的 S(主语)子节点包括所有谓词为 P_1 的主语实例；O(宾语)子节点包括所有谓词为 P_1 的宾语实例。除了 Decision 节点外，每个节点都具有自己的权值，符号定义如下：P_i 为 SST 中第 i 个 P 节点；S_i 为第 i 个 P 节点的 S 子节点；O_i 为第 i 个 P 节点的 O 子节点；s_j 为 S_i 节点下的第 j 个 s 子节点；o_j 为 O_i 节点下的第 j 个 o 节点。权值计算公式如下：

$$\begin{cases} value(s_j)=1; j\in R \\ value(o_j)=1; j\in R \\ value(S_i)=\sum_{j=1}^{n} s_j; i,j\in R, n\in R \\ value(O_i)=\sum_{j=1}^{m} o_j; i,j\in R, m\in R \\ value(P_i)=\mathrm{MIN}(value(S_i), value(O_i)); i\in R \end{cases}$$

SST 根据每个 P_i 节点的 S 节点的权值 $value(S_i)$ 和 O 节点的权值 $value(O_i)$ 获取查询数据集，如果 $value(S_i)>value(O_i)$，则将宾语作为 key，对应的主语集合作为 value 存入 Map 中，且 $value(P_i)=value(O_i)$；如果 $value(S_i)<value(O_i)$，则将主语作为 key，对应的宾语集合作为 value 存入 Map 中，且 $value(P_i)=value(S_i)$；如果 $value(S_i)=value(O_i)$，则任取一种方案。

生成连接计划：SST 将每个 P 节点的 $value(P_i)$ 从小到大排序，权值最小的两条语句先连接，连接生成的结果再和下一条连接。

本文针对 Redis 的分布式集群的存储特点，对 SST 连接选择策略做了进一步的优化，为了生成 SST 连接选择策略中的选择策略树，HMSST 算法需要在数据存储时维护一张数据表，以实时统计存储节点中不重复主语的个数及不重复宾语的个数，对动态数据更新操作的兼容性比较差。而本文提出的方案只需要在生成选择策略树时获取 StorageNode 中对应 Set 的大小即可，不需要实时统计，从而可以节省存储和查询所花费的时间。

3.4　HMSST+查询算法

结合已经给出的基于 Redis 的分布式存储设计、HMSST+数据分配方案、SST 连接选择策略的优化方案，提出了本文的查询算法。其中数据分配是在整个查询计划开始之前预处理完成的。在完成预处理之后，我们的查询数据集就可以根据已知的 Type 定位到数据所在的 StorageNode，然后根据谓语定位到所在的 Set 数据集，从而在缩小数据集的情况下进行查询。查询分成以下 4 种情况：

1)在 P 未知、Type 已知的情况下，在所分类好的 Type 节点中遍历 P 进行查找；

2)当 Type 未知、P 已知时，从本体文件中得到谓语的定义域，所有谓语的定义域的交集即为查询的主语类型，从而转化为 Type 已知、P 已知的情况；

3)当 Type、P 已知时，判断查询语句中是否有 S 或者 O 已知，当 S 或者 O 已知时，ManageNode 直接进行查询；

4)当 Type、P 已知且查询语句不存在已知的 S 或 O 时，ManageNode 查找 ProcessNode 注册表，根据已注册 ProcessNode 的个数，将整个查询任务分割成对应个数的子任务分配给各个 ProcessNode 进行查询，ProcessNode 查询完成后将结果集返回给 Manage Node。

具体查询流程如图 5 所示。

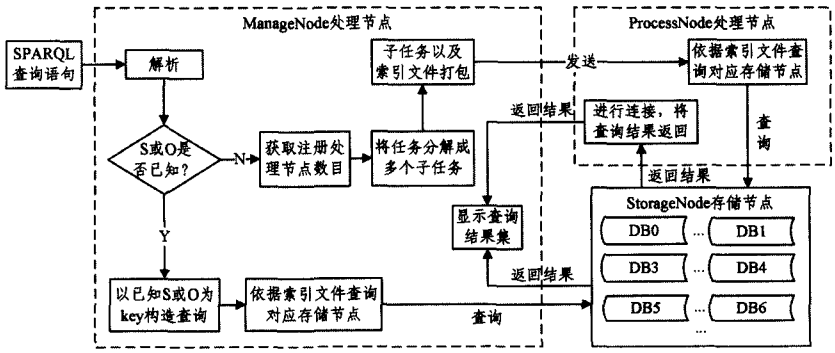

图 5　系统查询流程

对于简单的查询，即主语或宾语已知的查询，ManageNode 将查询分成 3 个阶段：查询语句分析、定位数据集、执行查询操作。以 Q1 为例：

```
SELECT ?x WHERE{
?x rdf:type ub:GraduateStudent.
?x ub:takesCourse
http://www.Department0.University0.edu/
GraduateCourse0.
}
```

针对 Q1 的查询步骤如下：

1）ManageNode 分析查询语句，获取结果集所属的类，type:GraduateStudent；

2）获取谓语，predicate:takesCourse，由于宾语已知，因此 PREDICAT 为 takesCourse_R；

3）根据 type 获取数据集所在的 storageNode；

4）ManageNode 执行查询操作。

HMSST+算法的简单查询伪码如下：

```
1. Begin
2. sparqlQuery:所需要查询的 sparql 查询语句；
3. getDataBaseByType():根据 Type 获取所在的 storageNode；
4. pl:Pipeline 连接数据库的数据管道对象，可以一次打包多条命令一起发出，不需要等待单条命令的响应返回，而 redis 服务端会在处理完多条命令后将多条命令的处理结果打包到一起返回给客户端；
5. pl.smembers(key):根据 key 获取 redis 中对应的 set 集合中的元素；
6. type=sparqlQuery.getType;
7. dataBase=getDataBaseByType(type);
8. predicate=sparqlQuery.getPredicate;
9. IF(subject 已知)
10.    key=subject +"_"+predicate;
11. ELSE(object 已知)
12.    key=object +"_"+predicate+"_R";
13. End IF
14. pl=dataBase.getPipeline;
15. Set<String> response=pl.smembers(key);
16. pl.sync;
17. End
```

对于元组模式个数较多和语义较复杂的查询语句，ManageNode需要将查询任务分成多个子任务交给 ProcessNode 进行执行。以 Q9 为例：

```
SELECT ?x?y?z WHERE{
?x rdf:type ub:Student.
?y rdf:type ub:Faculty.
?z rdf:type ub:Course.
?x ub:advisor ?y.
?y ub:teacherOf ?z.
?x ub:takesCourse ?z.
}
```

查询步骤如下：

1）ManageNode 获取所有 Student 存储节点中 advisor、advisor_R、takeCourse、takeCourse_R 以及所有 Faculty 存储节点中 teacherOf、teacherOf_R 集合的大小，从而构造选择生成树，生成连接选择策略；

2）ManageNode 查找 ProcessNode 注册表，根据已注册 ProcessNode 的个数，将整个查询任务分割成对应个数的子任务，并将子任务以及子任务所需要的数据所在的storageNode 信息打包发送给注册的 ProcessNode 进行计算；

3）ProcessNode 查询完成后，将结果集返回给 ManageNode。

HMSST+算法的 ManageNode 任务分割算法伪码如下：

```
1. Begin
2. dataBase:查询计划中第一条语句主语所在的数据库；
3. predicate:查询计划中第一条语句的谓语；
4. keySet:查询计划中第一条语句和第二条语句的公共主语集合；
5. processNum:连接到 ManageNode 的 ProcessNode 的个数；
6. dataMap:存储 ProcessNode 执行子任务所需的相关数据；
7. separateSet(Set set,int num):将 set 等分成 num 个 set；
8. DataPacket:ManageNode 与 ProcessNode 之间通信的数据包；
9. Pipeline pl=dataBase.getPipeline;//连接数据库的数据管道
10. keySet=pl.smembers(predicate);
11. List<Set<String>> list=separateSet(keySet,processNum);
12. FOR(int i=0;i<processNum;i++)
13. dataMap.put("keySet",l.get(i));
14. DataPacket dataPacket = new DataPacket(DataPacket.search_
    type,dataMap);
15. objectOutputStream[i].writeObject(dataPacket);
16. END FOR
17. End
```

HMSST+算法的 ProcessNode 的连接算法伪码如下：

```
1. Begin
2. dataBasei:数据集 i 所在的数据库；
3. predicatei:查询计划中第 i 条语句的谓语；
4. keySet:查询计划中第一条语句和第二条语句的公共主语集合；
5. Pipeline pli=dataBasei.getPipeline;//连接第 i 个数据库的数据管道
6. FOR(KEY:keySet)
7.    Set L1=pl1.smembers(KEY+"_"+predicate1);
8.    Set L2=pl2.smembers(KEY+"_"+predicate2);
9.    FOR(STRING1:L1)
10.      FOR(STRING2:L2)
11.          IF (pl3.sismember(STRING1 + "_" + predicate3,
             STRING2))
12.          //do anything
13.          END IF
14.      END FOR
15.   END FOF
16. END FOR
17. End
```

4 算法测评

为了更好地测试本文算法，使用利哈伊大学基准（the Lehigh University Benchmark，LUBM）数据集进行测试，利哈伊大学开发的基准是为了便于评价语义 Web 存储库的标准和系统。基准测试的目的是评估这些存储库在一个大的数据集下对外延的查询性能。它包括一个数据生成工具、大学领域本体 univ-bench.owl、一组基准测试查询语句以及对应的查询结果。

LUBM 数据生成工具通过不同的参数配置可以生成不同规模的数据集。本文采用 LUBM 数据生成器分别生成了 50、100、500、1000 所大学的数据，测试了本文算法在这 4 组

数据集下的查询响应情况，并与现有的查询算法的查询性能进行了对比。数据大小如表1所列。

表1 LUBM文件大小说明

学校个数	三元组个数(万)	文件大小(GB)
50	688	0.53
100	1377	1.06
500	6883	5.30
1000	13777	10.70

本文将HMSST+算法与Liu L[11]、Kim HS[12]在1000所大学的情况下进行了对比测试。

4.1 实验环境对比

本文的实验环境为分布式Redis集群，具有1个ManageNode、3个ProcessNode和3个StorageNode节点，硬件环境为4核8GB内存，ProcessNode和StorageNode共处于一台物理节点上，软件环境为操作系统Linux Ubuntu，采用Java作为编程语言，开发环境为Eclipse。ManageNode的软件环境为操作系统Windows7 64位，采用Java作为编程语言，开发环境为Eclipse。

Liu L、Kim HS的实验环境为分布式Hadoop集群，包含1个namenode、4个datanode，集群的机器为24核64GB内存，软件环境为操作系统Linux Ubuntu，采用Java作为编程语言，开发环境为Eclipse。

4.2 实验过程

1)经过本文的数据分配方案预处理后，将LUBM数据集中的主语所属类型为GraduateStudent的三元组存储在第一个StorageNode的DB1中，将主语所属类型为UndergraduateStudent的三元组存储在第二个StorageNoe的DB1中，将主语所属类型为AssistantProfessor、AssociateProfessor、FullProfessor、Department、Publication、Lecturer等的分别存储在第三个StorageNode的DB1－DBn中；并在ManageNode中存储各个类对应的DB号以及StorageNode所在的物理节点的IP及端口号。

2)在3个StorageNode所在物理节点上开启3个ProcessNode并连接ManageNode，ManageNode存储ProcessNode所在IP、端口号以及是否在线等信息。

3)ManageNode获取SPARQL查询请求，执行查询操作。

4.3 性能对比

为了测试在不同查询条件下本文算法的查询效率，挑选了6条具有代表性的LUBM查询语句，分别为Q1、Q2、Q3、Q4、Q5、Q9。其中Q1、Q3、Q4、Q5分别为主语已知或者宾语已知的多条件简单查询语句；Q2、Q9为主语和宾语都未知的多条件复杂查询语句。在不同数据集下利用本文的算法对这6条查询语句的查询响应时间进行了测试，同时在1000所大学的数据集下利用本文的算法以及Liu L、Kim HS对这6条查询语句的查询响应时间进行了多次测试，并取其平均值，测试结果如表2、表3所列。

表2 HMSST+在不同数据集下的查询响应时间(单位：s)

	Q1	Q2	Q3	Q4	Q5	Q9
50	0.3	1.3	0.2	0.2	0.2	3.8
100	0.1	2.2	0.2	0.4	0.5	6.1
500	0.3	12.7	0.4	0.3	0.6	24.2
1000	0.5	43.8	0.3	0.6	0.4	55.0

表3 本文与Liu L、Kim HS的查询响应时间的对比(单位：s)

	Q1	Q2	Q3	Q4	Q5	Q9
Liu L	41.0	162.0	45.0	68.0	38.0	142.0
Kim HS	30.0	108.0	36.0	35.0	32.0	153.0
本文	0.5	43.8	0.3	0.6	0.4	55.0

三者对于LUBM1000个大学数据集的查询响应时间对比如图6所示。

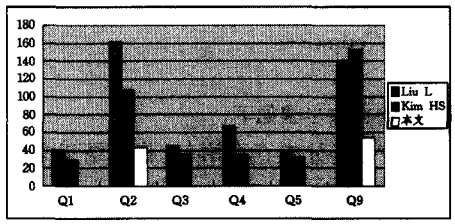

图6 查询响应时间对比

由图6可以看出，与Liu L以及Kim HS相比，本文算法由于在分布式的环境下利用了Redis键值对的数据存储方式，因此在Q1、Q3、Q4、Q5等主语或者宾语已知的简单查询语句中具有明显的优势，查询时间随数据集的增大改变不大，接近常数时间，且都在毫秒级别；而Liu L以及Kim HS每次都要开启1个Job，且随着数据量的增大，文件读写的I/O时间增加，因此需要消耗比较多的查询时间。此外，对于Q2、Q9等复杂的查询语句，因为子任务的分配牵涉到比较大的数据调度，而跨节点调度数据需要消耗比较多的时间，因此本文的算法需要消耗比较多的查询时间，但是相比Liu L以及Kim HS还是具有很明显的优势。

结束语 本文提出的HMSST+算法有如下的特点与创新：1)将RDF数据以Key-Value的形式存储在Redis中，使简单的查询语句的查询效率接近常数时间；2)提出了一种Redis分布式集群构建方案，并且将HMSST算法应用于该方案，解决了HMSST算法在集中式环境下受限于有限内存的瓶颈，同时对SST算法进行改进，使其更好地应用于分布式环境。实验表明，本文的存储策略与现有的查询存储方案相比，对于简单的查询语句具有明显的查询优势，对于复杂的查询语句也具有较快的查询能力，且可扩展性强，在大数据集下可以高效地工作。

本文的不足在于：1)未充分考虑查询时节点间流量的负载均衡，因此以后的方向在于研究分布式存储时如何充分考虑到分布式查询时节点间流量的负载均衡问题，使其在复杂的查询语句下具有更高效的查询效率。2)本文的算法受限于集群整体的内存大小，不能无限制地扩大数据集，因此在实际的应用中，需要在Redis中设置数据的生命周期，同时添加文件存储来进行数据固化操作，缓解内存压力。

参考文献

[1] He Shao-peng, Li Jian-hui, Shen Zhi-hong, et al. Overview of the Storage Technology for Large-scale RDF Data[J]. Network New Media, 2013(1): 8-16(in Chinese)
何少鹏，黎建辉，沈志宏，等. 大规模的RDF数据存储技术综述[J]. 网络新媒体技术，2013(1)：8-16

[2] Harris S, Lamb N, Ltd N S G. 4store: The Design and Implementation of a Clustered RDF Store[C]// The 5th International Workshop on Scalable Semantic Web Knowledge Base Systems. 2009

(下转第230页)

for Top-k Ranking in Uncertain Databases[J]. Proceedings of the VLDB Endowment,2011,4(10):598-609

[11] Lu Xin,Chen Hua-hui,Dong Yi-hong,et al. Top-k Query Calculations on Uncertain Dataset under MapReduce Framework[J]. Pattem Recognition and Aitificial Intelligence,2013(7):695-704 (in Chihese)
卢鑫,陈华辉,董一鸿,等. MapReduce 框架下的不确定数据 Top-k 查询计算[J]. 模式识别与人工智能,2013(7):695-704

[12] Liu B. Uncertainty Theory(2nd edition)[M]. Springer-Verlag, Berlin,2007

[13] Liu B. Uncertainty Theory: A Branch of Mathematics for Modeling Human Uncertainty[M]. Springer-Verlag,Berlin,2010

[14] Liu Bao-ding. Uncertainty Distribution and Independence of Uncertain Processes[J]. Fuzzy Optimization and Decision Making, 2014,13(3):259-271

[15] Zhou Jian,Chen Lu,Wang Ke. Path Optimality Conditions for Minimum Spanning Tree Problem with Uncertain Edge Weights,International Journal of Uncertainty[J]. Fuzziness and Knowledge-Based Systems,2015,23(1):49-71

[16] Gao X L. Uncertain relations on a finite set and their properties [J]. Pure and Applied Mathematics Journal,2014,3(1):13-19

[17] Gao X L,Gao Y. Connectedness Index of Uncertainty Graphs, International Journal of Uncertainty[J]. Fuzziness and Knowledge-Based Systems,2013,21(1):127-137

[18] Gao X. Some properties of continuous uncertain measure,International Journal of Uncertainty[J]. Fuzziness and Knowledge-Based Systems,2009,17(3):419-426

[19] Gao X,Gao Y,Ralescu D. On Liu's Inference Rule for Uncertain Systems,International Journal of Uncertainty[J]. Fuzziness and Knowledge-Based Systems,2010,18(1):1-11

[20] Gao Y,Yang L X,et al. On Distribution Function of the Diameter in Uncertain Graph[J]. Information Sciences,2015,296(1): 61-74

[21] Gao Y. Shortest Path Problem with Uncertain Arc Lengths [J]. Computers and Mathematics with Applications,2015,296 (1):61-74

[22] Liu B. Some research problems in uncertainty theory[J]. Journal of Uncertain Systems,2009,3(1):3-10

[23] Luo Jun-zhou, Jin Jia-hui, Song Ai-bo, et al. Cloud computing: architecture and key technologies[J]. Journal on Communications,2011,32(7):3-21(in Chinese)
罗军舟,金嘉晖,宋爱波,等. 云计算:体系架构与关键技术[J]. 通信学报,2011,32(7):3-21

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

[25] Zaharia M, Chowdhury M, Franklin M J, et al. Spark: cluster computing with working sets[C]//Proceedings of the 2nd USENIX Conference on Hot Topics in Cloud Computing. 2010:10

[26] Neumeyer L, Robbins B, Nair A, et al. S4: Distributed stream computing platform[C]// 2010 IEEE International Conference on Data Mining Workshops (ICDMW). IEEE,2010:170-177

[27] Wieder A,Bhatotia P,Post A,et al. Brief announcement:modelling MapReduce for optimal execution in the cloud[C]// Proceedings of the 29th ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing. New York, NY, USA: ACM,2010:408-409

[28] Zheng Q. Improving MapReduce fault tolerance in the cloud[C]// 2010 IEEE International Symposium on Parallel & Distributed Processing,Workshops and Phd Forum (IPDPSW). Atlanta, GA:IEEE,2010:1-6

[29] Ding Lin-lin,Xin Jun-chang,Wang Guo-ren,et al. Efficient Skyline Query Processing of Massive Data Based on Map-Reduce [J]. Chinese Journal of Computers,2011,34(10):1785-1796(in Chinese)
丁琳琳,信俊昌,王国仁,等. 基于 Map-Reduce 的海量数据高效 Skyline 查询处理[J]. 计算机学报,2011,34(10):1785-1796

[30] Li Ling-juan,Zhang Min. Research on Algorithms of Mining AssociationRule under Cloud Computing Environment[J]. Computer Technology and Development,2011,21(2):43-46(in Chinese)
李玲娟,张敏. 云计算环境下关联规则挖掘算法的研究[J]. 计算机技术与发展,2011,21(2):43-46

[31] Zeng Qing-sen, Huang Xian-ying. Fast Data Mining Algorithm Based on FP-tree[J]. Journal of Chongqing Institute of Technology(Natural Science),2009,23(10):72-76(in Chinese)
曾庆森,黄贤英. 基于 FP-tree 的快速数据挖掘算法[J]. 重庆工学院学报(自然科学版),2009,23(10):72-76

(上接第 224 页)

[3] Neumann T, Weikum G. The RDF-3X engine for scalable management of RDF data[J]. The VLDB Journal,2010,19:91-113

[4] Wang Yan,Tian Cui-hua,Zhu Shun-zhi,et al. RDF Data Index Method Based on Association of SPARQL Query Twis[J]. Journal of Xiamen University(Natural Science),2014,53(3):322-329(in Chinese)
王琰,田翠华,朱顺痣,等. 基于 SPARQL 查询小枝关联的 RDF 数据索引方案[J]. 厦门大学学报(自然科学版),2014,53(3): 322-329

[5] Weiss C, Karras P, Bemstein A. Hexastore: sextuple indexing for semantic web data anagementl [C]//Proceedings of the 34rd International Conference on Very Large Data Bases. New York: ACM,2008:1008-1019

[6] Dong Shu-jian,Wang Jing-bin. HMSST:An efficient algorithm for SPARQL query[J]. Computer Science,2014,41(S2):323-326,336(in Chinese)
董书琳,汪璟玢. HMSST:一种高效的 SPARQL 查询优化算法[J]. 计算机科学,2014,41(S2):323-326,336

[7] Zeng Chao-yu,Li Jin-xiang. Redis application in cache system [J]. Microcomputer&Its Applications,2013,12:11-13(in Chinese)
曾超宇,李金香. Redis 在高速缓存系统中的应用[J]. 微型机与应用,2013,12:11-13

[8] Gao X,Fang X. High-Performance Distributed Cache Architecture Based on Redis[C]//Proceedings of the 9th International Symposium on Linear Drives for Industry Applications,Volume 1. Springer Berlin Heidelberg,2014:105-111

[9] Guo Y, Pan Z, Heflin J. LUBM: A benchmark for OWL knowledge base systems[J]. Web Semantics: Science, Services and Agents on the World Wide Web,2005,3(2):158-182

[10] Huang H,Liu C. Selectivity estimation for SPARQL graph pattern[C]//Proceedings of the 19th international conference on World Wide Web. ACM,2010:1115-1116

[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] Kim H S,Ravindra P,Anyanwu K. From SPARQL to MapReduce:The journey using a nested TripleGroup algebra[J]. Proc. of the VLDB Endowment,2011,4(12):1426-1429