HMSST:一种高效的SPARQL查询优化算法

董书睐 汪璟玢
(福州大学数学与计算机科学学院 福州 350108)

摘 要 在缩小海量数据查询范围的前提下,结合哈希映射和选择策略树提出了一种SPARQL优化算法——HMSST(HashMapSelectivityStrategyTree),实现了SPARQL的查询优化。并针对LUBM 1000所大学的测试数据集对查询策略进行了实验,实验结果表明:提出的HMSST算法以及存储策略相比现有的查询方案,具有更小的存储代价以及更高的查询能力,在大数据集下可以高效地工作,并且该优化方案在查询的元组模式个数较多和语义较复杂时效果更加明显。

关键词 哈希映射,查询优化,RDF,SPARQL
中图法分类号 TP391 **文献标识码** A

HMSST:An Efficient Algorithm for SPARQL Query

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

Abstract The paper proposed a novel efficient algorithm,named HMSST(HashMapSelectivityStrategyTree),to optimiz SPARQL query,combining the Hash Map and Selection Strategy Tree,based on narrowing the range of massive data query. HMSST algorithm is estimated by LUBM Benchmark and it works well when the university number reaches 1000. The experimental results show that the HMSST algorithm and the storage strategy are better than the existing query schemes,its storage cost is smaller,its query performance is higher,and it works effectively in large data sets,especially when SPARQL query contains more triple patterns and more complex semanteme.

Keywords Hash map,Query optimization,RDF,SPARQL

1 引言

随着RDF在各个领域的广泛应用以及RDF数据量的急速增长,如何高效地管理海量RDF数据成为近年来的研究热点[1]。然而传统关系型数据库存储系统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其难以使用关系型数据库管理系统的查询优化策略[2]。因此研究人员开始将目光投向文件存储,以期利用文件存储所具备的海量存储和查询能力来解决当前RDF面临的各项问题。和大多数查询优化一样,语义Web下的SPARQL语言同样存在优化问题[3]。RDF数据的存储和索引是影响SPARQL查询效率的一个方面。Abadi等人提出使用垂直分割的方法处理关系数据库下大规模RDF数据的存储问题[4]。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索引来提高查询效率[5]。该方法有效地提高了SPARQL的查询效率,但是耗费了大量的存储空间。SPARQL优化的另一方面工作,放在对查询问题本身的优化[6]。Harting等人表述了一个SPARQL查询的图模型(SQGM),它支持所有查询过程的解析,对简单的SPARQL查询语句查询效率显著[7],但是对于一个复杂的查询语句,效果不是很明显。Stocker等人通过引入计算选择估值来对查询语句进行排序优化SPARQL查询,并与其它简单的选择估值启发式优化方法进行了对比。该方法对于复杂语句的查询具有比较好的查询效率[8],但是在评估SPARQL中每条三元组模式的选择度时,需要耗费大量计算时间。本文结合以上两种优化策略,将RDF资源的存储采用简单的三元组垂直存储方式,并将数据进行按Type-P划分,索引方式采用操作系统文件目录索引,并建立一张实例-Type映射表辅助索引,而对于查询问题本身,本文提出了一种新的SelectivityStrategyTree(SST)选择算法,实验表明:该选择算法简单高效,在与哈希映射有机结合后对于元组模式个数较多的查询语句具有很好的查询效率。

本文第2节对SPARQL相关概念和系统缓存策略以及哈希映射进行介绍;第3节给出了HMSST混合优化策略,详细表述了该策略的3个组成部分:Type-P数据划分方案、操作系统两级缓存、SST选择算法,进一步给出HMSST混合优化策略的算法实现;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,验证算法的有效性;最

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。
董书睐(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)。

后对本文进行总结。

2 相关基础知识

2.1 RDF

RDF是一种简易的资源描述方式，即用主体(Subject)、谓词(predicate)、客体(Object)构成的三元组来表示资源。随着计算机技术的进步，各种系统所处理的数据量也与日俱增，实际应用中的RDF数据集的三元组数目同早期相比已不可同日而语，在这样爆炸式增长的存储压力之下，原始的简易架构模型不可避免地遭遇了性能瓶颈，在存储能力到查询响应性能方面都不能满足日益增长的需求。在巨大的数据压力面前，研究人员对RDF存储系统的各个方面如RDF存储模型定义、底层存储系统选择、查询连接处理等均进行了详细的分析研究，从不同角度提出了新的解决方案，但当前依然没有一种被业界完全认可的RDF存储系统方案。

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。

```
BasicGraph Pattern {SELECT ?x ?y ?z WHERE {
?x rdf :type ub :GraduateSt udent.
?y rdf :type ub :University .
?z rdf :type ub :Department .
?x ub :memberOf ?z.
?z ub :subOrganiz ationOf ?y.
?x ub :undergradu ateDegreeF rom ?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，未绑定的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则表示这些子句间存在连接关系。Triple Pattern子句的表现形式及语义如表1所列。

表1 Triple Pattern定义

	表达式	语义
Q1	(S P O)	若存在则返回该三元组否则返回空
Q2	(S P ?O)	给定S,P,返回对应三元组中O的值
Q3	(S ?P O)	给定S,O,返回对应三元组中P的值
Q4	(S ?P?O)	给定S,返回对应三元组中O和P的值
Q5	(?S P O)	给定P,O,返回对应三元组中S的值
Q6	(?S P ?O)	给定P,返回对应三元组中S和O的值
Q7	(?S ?P O)	给定O,返回对应三元组中S和P的值
Q8	(?S ?P ?O)	返回所有三元组

2.3 缓存策略

缓存法通过设计良好的数据分块、预取、顺序预取、缓存替换等算法来提高对缓存内容的命中率，从而在程序运行读取数据时，缓解内存和外存之间的速度差。缓存算法可以分为基于访问时间的策略，如LRU (Least Recently Used)，该算法维护一个缓存项队列，队列中的缓存项按每项的最后被访问时间排序；基于访问频率的策略，如LFU(Least Frequently Used)；访问时间与频率兼顾策略，如LRFU(Least Recently/Frequently Used)。

缓存策略主要有3方面内容：①缓存什么内容；②何时进行缓存；③当缓存空间已满时如何进行替换，即缓存替换算法。对于第二方面，大部分缓存算法使用预取策略来提前将部分磁盘数据放入缓存，以进一步减少磁盘I/O，加大缓存命中率。通过记录、分析以往的数据请求模式来预测将来可能被请求到的数据段，将访问可能性大的数据段放入缓存。

2.4 Hash Map

基于哈希表的Map接口的实现。哈希表最大的优点，就是把数据的存储和查找消耗的时间大大降低，几乎可以看成是常数时间；Hash Map的实例有两个参数影响其性能：初始容量和加载因子。容量是哈希表中桶的数量，初始容量只是哈希表在创建时的容量。加载因子是哈希表在其容量自动增加之前可以达到多满的一种尺度。当哈希表中的条目数超出了加载因子与当前容量的乘积时，则要对该哈希表进行rehash操作(即重建内部数据结构)，使哈希表具有大约两倍的桶数。总哈希桶数、总记录数、哈希桶容量以及装载因子之间的关系如下：

$$\text{总哈希桶数}=\frac{\text{总记录数}}{\text{哈希桶容量}\times\text{装载因子}}$$

3 结合哈希映射和SST的SPARQL优化算法——HMSST

本文提出的HMSST算法主要通过哈希映射来加快数据的查询速度，哈希表最大的优点就是把数据的存储和查找消耗的时间大大降低，几乎可以看成是常数时间；而缺点就是消耗的内存较多。因此HMSST算法在使用哈希映射的基础上结合Type-P的数据划分存储方案以及操作系统两级缓存策略，最大限度地利用有限的内存达到优化查询效率的目的。并通过与SST连接选择策略的有机结合，实现在有限的内存中最大限度地利用哈希映射达到优化查询效率的目的。图2展示了HMSST算法的模型图。

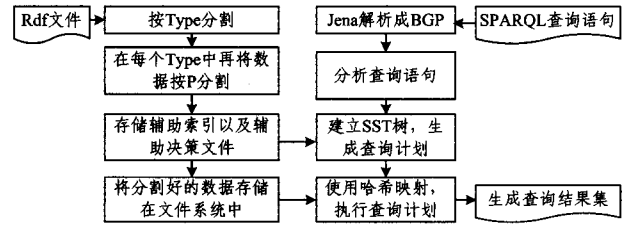

图2 HMSST算法的模型图

3.1 Type-P数据划分方案

因为现在大部分的查询语句都是在Type、P已知的条件下进行查询的，所以本文采用了将RDF数据先按三元组的主语所属于的类进行划分，建立以类命名的文件夹；再根据本体中的定义，获取该类所有谓词，为每一个谓词建立以该谓词命名的文件夹；然后再将所有主语归为同一个类，且谓词相同的的三元组放置于该文件夹中。该方案可以有效地缩小查询范围，提高查询效率。以三元组：UndergraduateStudent476、takesCourse、Department0. University0. edu/Course2为例，根

据 HMSST 算法，该三元组的主语 UndergraduateStudent476 属于 UndergraduateStudent 类，则在按 Type 分割的时候将其分割到 UndergraduateStudent 所属文件夹中，然后在按 P 分割的时候，再将该三元组放置在 takesCourse 文件夹中的 takesCourse_n. txt 中。

分割数据的同时，HMSST 算法还会统计一些数据作为辅助决策以及辅助索引，如：1）实例-Type 映射文件中记录了所有属于这个类的实例，用来在 P 以及 type 都未知的情况下进行辅助索引。2）AssistFile 文件统计了每个类文件夹中各个 P 文件中不重复主语的个数以及不重复谓语的个数，用来辅助 SST 进行决策。3）Typelist 统计了这个数据集中存在的类。具体的划分示意图如图 3 所示，划分完成后只要通过操作系统的文件系统进行索引，就可以迅速定位到要查询的数据模块。

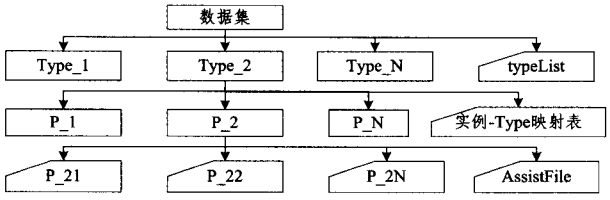

图 3　Type-P 划分示意图

为了应对 Type 未知或者 P 未知的查询语句，HMSST 会执行以下策略：

1）在 P 和 Type 未知的情况下，根据已知的 S 或者 O，查询实例-Type 映射文件，获取 S 或 O 所属于的 Type 文件夹，然后转化为在 P 未知、Type 已知的情况下进行查询；

2）在 P 未知、Type 已知的情况下，在所分类好的 Type 文件夹中查找；

3）当 Type 未知、P 已知的时候，则根据 TypeList 表查询所有 Type 文件夹中对应的 P 文件。

3.2　两级缓存策略

为了最大限度地发挥哈希映射的高效查询效率，我们需要将部分数据加载入内存，但是基于内存的限制，这部分的数据不能太大，因此我们需要采取两级缓存策略，在程序运行时，将需要查询的数据集加载到系统内存中，此为一级缓存；在执行查询语句的时候将需要的数据集加载到 JVM 内存中，此为二级缓存；当这部分数据执行完查询后，释放 JVM 内存，重新从系统内存中快速获取数据。以 32 位操作系统的 JVM 程序运行时的内存限制为 1. 5GB 左右为例，具体图示如图 4 所示。

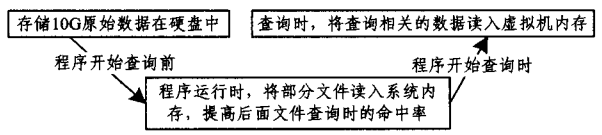

图 4　两级缓存策略

3.3　SST 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。HMSST 算法通过对查询语句的分析，从辅助决策文件中获取对应的 P 文件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 5 所示。

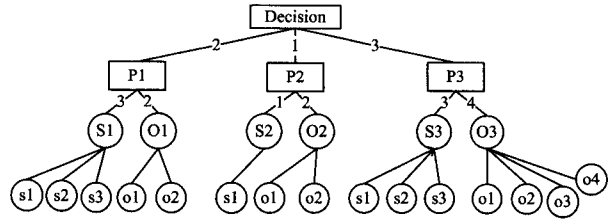

图 5　SST 选择策略树

选择策略树包含一个根节点：Decision 节点，负责生成 Hash Map 的存储方案以及 HMSST 的连接方案。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 $P1$ 节点为例，$P1$ 节点的 S(主语)子节点包括所有谓词为 $P1$ 的主语实例；O(宾语)子节点包括所有谓词为 $P1$ 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：

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

生成 Hash Map 存储方案：SST 根据每个 P 节点的 S 节点的权值 $value(S)$ 和 O 节点的权值 $value(O)$，生成存储方案，如果 $value(S)>value(O)$，则将宾语作为 key，对应的主语存为 list 作为 value 存入 Hash Map 中；如果 $value(S)<value(O)$，则将主语作为 key，对应的宾语存为 list 作为 value 存入 Hash Map 中，如果两者相等，则任取一种方案。

生成连接计划：SST 将每个 P 节点的 $value(Pi)$ 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。

我们以 LUMB 中最复杂的查询语句 query9 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.              --//1
  ?Y ub:teacherOf ?Z.            -- //2
  ?X ub:takesCourse ?Z}          -- //3
```

通过对查询语句的分析，我们可以生成 3 个 Hash Map：advisorMap、teacherOfMap、takesCourseMap，其中 advisorMap 是以 advisor 数据集中不重复宾语作为 key，对应的主语存为 list 作为 value；teacherOfMap 是以 teacherOf 数据集中不重复主语作为 key，对应的宾语存为 list 作为 value；takesCourseMap 是以 takesCourse 数据集中不重复宾语作为 key，对应的主语存为 list 作为 value。对应的连接计划为：2→1→3。

4　HMSST 查询优化算法

我们结合已经给出的 Type-P 数据划分存储方案、操作系

统两级缓存方案、SST 生成的连接选择策略以及哈希存储查询方法，提出了 HMSST 查询优化算法。其中 Type-P 的数据划分存储方案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 和 P，生成文件路径，然后通过操作系统的文件系统定位到数据集。我们把 HMSST 优化算法分成 4 个阶段：查询语句分析、SST 生成 Hash Map 存储方案与连接计划、哈希映射连接查询。算法描述如下：

输入参数：

获取查询语句 query；

分析查询语句，如果 Type、P 已知，查询 S，O。则根据查询语句，读入辅助决策文件，建立 SST 树，生成连接计划与 Hash Map 存储方案；

根据查询计划，执行 SearchObjectAndSubjectByJoin；

FUNCTION SearchObjectAndSubjectByJoin

BEGIN

1)通过 Type-P 定位到对应的文件模块，将文件系统中的数据载入系统内存；

2)根据查询语句中谓语个数(不包含 type)创建对应个数的 Hash Map；

3)根据 SST 生成的 Hash Map 存储方案将系统内存中的数据分块载入 JVM 内存；

4)执行连接计划：

```
FOR(KEY:MAP1)
  IF(MAP2 CONTAIN KEY)
     LIST l1 = MAP2 GET KEY;
     LIST L2 = MAP1 GET KEY;
  FOR(STRING1:L1)
     FOR(STRING2:l2)
    IF(MAP3 CONTAIN STRING1)
        LIST L3 = MAP3 GET STRING1;
  IF(L3 CONTAIN STRING2))
  //do anything;
  END IF
    END IF
  END FOR
   END FOF
   END IF
END FOR
END
```

5 算法测评

本章对前文提出的 RDF 查询处理方法采用 LUBM(Lehigh University Benchmark)标准测试数据集进行了实验验证。测试了 6 组不同 LUBM 数据集下的 RDF 查询响应情况，并与现有的查询算法进行存储方案以及查询性能的对比。

5.1 实验环境

实验所使用的硬件环境为 Intel(R) Core(TM) i5-3470 CPU @3.20GHz，内存 4GB，磁盘 500 GB。软件环境为操作系统 windows7 32 位，采用 Java 作为编程语言，开发环境为 eclipse。其中分配给 Java 虚拟机的内存为 1.5GB。

5.2 存储方案实验对比

实验将本文提出的存储模型与查询算法与 H2RDF 和 ProvBase 采用的方案作对比。三者对于 LUBM 100 个大学数据集的存储开销如图 6 所示。H2RDF[11]和 ProvBase[12]均采用了 3 张存储表的方式存储 RDF 数据，每张表都存储所有三元组数据，重复存储了 3 份数据，不同的是 H2RDF 采用多列存储多值，ProvBase 采用多版本存储多值，两者的存储开销差别不大。本文采用的存储模型是基于原始数据集的数据划分，不仅不需要扩展数据集，而且还可以删除一些冗余的信息，存储开销比 H2RDF 和 ProvBase 要少得多。

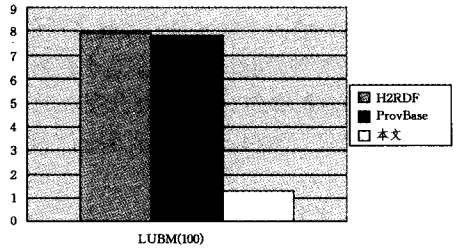

图 6 存储开销对比(单位：GB)

5.3 RDF 查询效率对比

我们针对如下几组不同大小的 LUBM 数据集进行测试，数据大小如表 2 所列。表 2 中的 RDF 三元组是经过 Jena[13]处理后的数目。

表 2 测试集大小(单位：万个)

Univ.	1	50	100	300	500	1000
RDF Triples	15	1011	2035	6055	10131	20363

我们在不同数据集下分别测试了 9 条 LUBM 查询语句，各查询的响应时间如表 3 所列。

表 3 LUBM 查询时间响应(单位：秒)

学校数目 \ 查询语句	Q1	Q2	Q3	Q4	Q5
1	0.094	0.104	0.164	0.048	0.178
50	0.596	0.681	1.570	0.317	1.163
100	1.010	1.506	3.120	0.419	1.695
300	3.061	7.783	10.772	1.359	5.465
500	5.498	19.245	17.036	2.279	8.619
1000	11.437	69.211	35.688	4.993	20.748

学校数目 \ 查询语句	Q6	Q7	Q8	Q9
1	0.135	0.142	0.241	0.287
50	1.103	2.456	2.731	6.895
100	1.929	5.154	4.434	8.528
300	6.612	19.650	14.448	44.372
500	12.439	41.100	27.547	80.225
1000	33.084	130.039	100.783	164.086

实验对 LUBM 14 个查询中具有代表性的 7 个查询与 H2RDF 和 ProvBase 采用的方案进行了对比测试。LUBM 测试查询中的部分查询语句要求查询处理引擎具备在 subProperty 和 subClassof 关系上的推理能力。然而当前 HMSST 算法、对比的 H2RDF 以及 ProvBase 算法都不支持对 RDF 数据进行推理。为了让这些查询语句能够有效工作，我们需要对数据进行预处理，将所有隐含的三元组推理出来，然后再进行计算。在进行测试时，我们通过 Jena 处理 LUBM 数据集来获得最终测试数据。三者对于 LUBM 100 个大学

(下转第 336 页)

利用 Hadoop 框架的 MapReduce 分布式处理模式对高校教学资源平台产生的海量日志进行分析，通过在该框架下反复的实验以及减少 I/O 访问次数和调整 MapReduce 参数等方法，来提高整个集群并行处理速度，从而更好地使用 MapReduce 分布式作业计算策略进行海量数据分析处理。

参 考 文 献

[1] 汤姆. Hadoop 权威指南[M]. 北京：清华大学出版社，2010：63-65

[2] HDFShttp://hadoop. apache. org

[3] MapReducehttp://hadoop. apache. org

[4] 徐子沛. 大数据：正在到来的数据革命[M]. 桂林：广西师范大学出版社，2012：23-30

[5] 李小庆. 银行面向大数据分析决策系统的构建[J]. 金融科技时代，2013：1-2

[6] 刘欢，张瑾. 数据挖掘改善校园网体验 [J]. 中国教育网络，2012(1)：27-30

[7] 范范. 大数据前景展望[N]. 网络世界，2012，(5)

[8] 李开复. 云计算[J]. 中国教育网络，2008(6)：34

[9] NfcKinsey Global Institute. Big data：The next frontier for innovation ompetition and productivity [R]. 2011(1)

[10] 白云川. 迎接大数据的时代[J]. 中国制造业信息化，2011(2)

[11] 蒋杰. Big Data 技术综述[J]. 程序员，2011：2-3

[12] 董彩云，等. 数据挖掘及其在高校教学系统中的应用[J]. 济南大学学报：自然科学版，2004：1-2

[13] 大数据时代下. 企业信息管理的新革命[J]. 网络与信息，2012(4)：7

(上接第 326 页)

数据集的查询响应时间如图 7 所示。

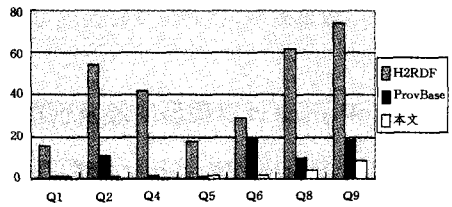

图 7　LUBM 100 个大学数据集的查询响应时间(单位：秒)

由图 7 可以看出，和 ProvBase 相比：1. HMSST 在 Q1、Q4、Q5 3 条简单的查询语句中和 ProvBase 的查询性能相当，ProvBase 利用 HBase Java API 查询时有效地利用了 HBase 提供的 row-key 索引，因此对于简单的查询语句具有很快的查询效率；2. 在 Q6 中，HMSST 的查询性能要比 PorvBase 好，因为 HMSST 是按 Type-P 划分存储的，对于 Type 已知、直接查询 S 的查询语句，具有很高的查询效率，而 ProvBase 的每张表都过于庞大，因此对于此类查询性能会略低。3. 在 Q2、Q8、Q9 这 3 条具有多个元组模式且连接操作较多的查询语句中，SST 提供的选择连接策略就发挥了很大的查询优化效果，使 HMSST 算法的查询性能在这 3 条上优于 ProvBase。而对于 H2RDF，因为 H2RDF 采用了 MapReduce 处理 BGP 连接，而一个 MapReduce 作业的启动时间就要 3 秒，对于选择度很高的查询来说，启动时间就占了整体查询时间的大部分，所以其查询效率很低。

结束语　本文提出的 HMSST 算法有如下的特点与创新：1. RDF 存储系统通常都需要面对数据划分策略的选择，而本文采取的 Type-P 划分存储策略可以有效地缩小查询范围，且划分简单，速度快；2. 本文采取的两级缓存策略弥补了哈希表在高效存储查询时需要耗费较大内存的缺点；3. 我们提出了与哈希映射高效结合的 SST 选择策略，实现了复杂连接操作语句的快速查询。实验表明，我们的存储策略相比现有的查询存储方案，具有更小的存储代价、更快的查询速度，以及更高的查询能力，在大数据集下可以高效地工作，并且该优化方案在查询的元组模式个数较多和语义较复杂时效果更加明显。

本文的不足在于：因为是集中式查询，HMSST 算法局限于有限的内存下，数据的内外存交换耗费了大量的时间，90% 的查询时间要消耗在内外存交换中。因此本文以后的方向在于研究分布式缓存技术，将 HMSST 算法应用到分布式文件系统中，从而让其在更大的数据量下发挥出更有效的查询效率。

参 考 文 献

[1] 朱敏，程佳，柏文阳. 一种基于HBase的RDF数据存储模型[C]// Proceedings of the NDBC. 2013

[2] 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

[3] Schmidt M，Meier M，Lausen G. Foundations of SPARQL query optimization [DB/OL]. [2009-01-26]. The Computing Research. Repository，http://arxiv. org/abs/0812. 3788

[4] Abadi D J，Marcus A，Madden S R，et al. Scalable semantic web data management using vertical partitioning [C]// Chan C Y. Proceedings of the 33rd International Conference on Very Large Data Bases. New York：ACM，2007：411-422

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

[6] 叶育鑫. 混合语义约简和选择估值优化 SPARQ[J]. 电子学报，2010，38(5)：1205-1210

[7] Hatting O. Querying trust in RDF Data with tSPARQL[C]// Proceedings of 6th European Semantic Web Conference (ESWC'09). Berlin：Springer，2009：5-20

[8] Stocker M，Seaborne A，et al. SPARQL basic graph pattern Optimization using selectivity estimation[C]// Liu Y H. www' 08. New York：ACM，2008：595-604

[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] Papailiou N，Konstantinou I，Tsoumakos D，et al. H2RDF：adaptive query processing on RDF data in the cloud[C]// Proceedings of the 21st International Conference Companion on World Wide Web. ACM，2012：397-400

[12] Abraham J，Brazier P，Chebotko A，et al. Distributed storage and querying techniques for a Semantic Web of scientific workflow provenance[C]// Services Computing (SCC)，2010 IEEE International Conference on. IEEE，2010：178-185

[13] McBride B. Jena：A Semantic Web Toolkit [J]. IEEE Internet Computing，2002，6(6)：55-9