计算机研究与发展
Journal of Computer Research and Development
ISSN 1000-1239/CN 11-1777/TP
50(Suppl.): 23-31, 2013

一种基于 HBase 的 RDF 数据存储模型

朱 敏 程 佳 柏文阳
(计算机软件新技术国家重点实验室(南京大学) 南京 210023)
(南京大学计算机科学与技术系 南京 210023)
(zhuminbit@gmail.com)

A Storage Model for RDF Data Based on HBase

Zhu Min, Cheng Jia, and Bai Wenyang
(*State Key Laboratory for Novel Software Technology (Nanjing University), Nanjing* 210023)
(*Department of Computer Science and Technology, Nanjing University, Nanjing* 210023)

Abstract With the rapid growth of the Semantic Web data, how to efficiently manage the massive RDF data becomes a key issue. The existing centralized relational RDF data storage management system has been difficult to meet this demand. Therefore, more and more researchers use distributed systems and parallel computing techniques to manage the massive RDF data. This work propose a novel RDF data storage model based on HBase and split data according to classes which defined by OWL. Triples that belong to the same class are stored both in the S_PO and O_PS tables of this class. Design Triple Pattern and Basic Graph Pattern query algorithms for this storage model which supports some inferences. Verify the feasibility of the storage model and query algorithm in the Hadoop cluster.

Key words RDF; semantic data storage; SPARQL; basic graph pattern; query processing

摘 要 随着语义网数据的爆炸式增长,如何高效地管理海量 RDF 数据成为一个关键问题.现有的集中式关系型 RDF 数据存储管理系统已难以适应这种需求,越来越多的研究者使用分布式系统和并行计算技术来管理海量 RDF 数据.提出一种基于分布式数据库 HBase 的 RDF 数据存储模型,根据 OWL 本体定义文件,将数据按类划分,同一类的三元组数据保存在该类的 S_PO 和 O_PS 两张表中,实现该存储模型上的 8 种 Triple Pattern 和 Basic Graph Pattern 查询算法,并提供部分推理功能,在 Hadoop 集群环境下对存储模型与查询算法进行了可行性验证.

关键词 资源描述框架;语义数据存储;SPARQL;基本图模式;查询处理

中图法分类号 TP392

资源描述框架(resource description framework, RDF)[1]是 W3C 组织推荐的一个开放元数据框架,用来描述语义网资源及其之间关系. RDF 通常以三元组(主语,谓语,宾语)的形式描述资源之间存在的特定关系,例如,三元组(http://www.w3.org/,author, W3C group)描述了网页 http://www.w3.org/的作者是 W3C group.

随着语义网技术的不断发展,RDF 数据被越来

收稿日期:2013-05-15
基金项目:国家自然科学基金项目(61100040);国家社会科学基金项目(11AZD121);国家"八六三"高技术研究发展计划基金项目(2011AA01A202)

越多地应用于各个领域,如生物技术、社交网络等. RDF 的广泛应用使得其数据量急速增长,如何高效地管理海量 RDF 数据成为一个急待解决的问题. 现有的 RDF 数据管理系统大都采用传统的关系型数据库来存储 RDF 数据[2],如采用三元组表存储的 RDF-3X[3-4] 和 Hexastore[5],采用垂直划分的方法以二元表形式存储的 C-store [6-7] 和 SW-Store[8] 等. 然而随着 RDF 数据的爆炸式增长,传统的基于关系型数据库的 RDF 管理系统已经难以高效地管理海量 RDF 数据. 有研究表明关系型数据库在处理海量 RDF 数据时存储与查询效率比分布式数据库低[9],越来越多的研究者开始利用分布式系统的海量数据存储与并行计算能力解决海量RDF数据管理问题[10-14].

HBase 是构建在 Apache Hadoop 之上的面向列的分布式数据库,是 Google BigTable[15] 的开源实现,适合存储海量稀疏数据,并且已经得到了广泛的应用.

OWL(Web Ontology Language)[16] 是 W3C 推荐的本体描述语言标准,提供明确定义的词汇表,支持丰富的语义表达以及推理.

SPARQL[17] 是 W3C 提出用于 RDF 数据的标准查询语言,其主要特点是基于模式匹配. Triple Pattern 是最基本的匹配单元,其构成类似于 RDF 三元组,但在任何主语、谓语或宾语的位置中可能有变量,如(?*Book*, *dc*:*title*, ?*title*). 多个 Triple Pattern 组成复杂的模式匹配,如基本图模式(basic graph pattern, BGP). 例如下面的 SPARQL 语句包含一个由 3 个 Triple Pattern 组成的 BGP.

SELECT ?*X*, ?*Y*
WHERE
{?*X rdf*:*type ub*:*Student*.
?*Y rdf*:*type ub*:*Course*.
?*X ub*:*takesCourse* ?*Y*}

针对目前 RDF 数据管理问题,本文提出一种使用分布式数据库 HBase 存储 RDF 数据,根据 OWL 本体定义文件,将数据按类划分与存储,采用 HBase Java API 实现 SPARQL 查询的方案. 该方案旨在提供具有高可扩展性的海量 RDF 数据存储能力并能快速响应 SPARQL 查询,通过在 Hadoop 集群环境下的实验证明了该方案的可行性.

本文的主要贡献有:

1) 提出一种基于 HBase 的 RDF 数据存储模型;

2) 设计并实现该存储模型上的 8 种 Triple Pattern 和 Basic Graph Pattern 查询算法,提供部分推理功能.

1 相关工作

目前有很多关于分布式 RDF 数据存储与查询的工作,在此进行简要介绍.

文献[10-11]基于 Hadoop 分布式文件系统(HDFS)以文件形式存储 RDF 数据,并采用 MapReduce 处理 SPARQL 查询. 在数据存储方面,文献[10]根据属性及宾语所属的类型将数据划分成多个小文件并存储在 HDFS,而文献[11]则根据主语划分文件. 在查询处理方面,文献[10]采用一种贪心的 MapReduce 任务生成算法,多个任务迭代处理 SPARQL 查询的连接操作,每个任务优先处理出现次数最多的变量所在的子句;文献[11]采用每个 MapReduce 任务处理一条 Triple Pattern 的查询算法,所有 Triple Pattern 按顺序执行.

文献[12-13]基于 HBase 存储 RDF 三元组. 文献[12]将数据存储在 *SPO*, *POS*, *OSP* 三张表中,*SPO* 表以(主语,谓语)为 row-key,宾语为 value,*POS* 表以(谓语,宾语)为 row-key,主语为 value,*OSP* 表以(宾语,主语)为 row-key,谓语为 value,用多个列存放多个值;文献[13]使用三张表 *Ts*, *Tp*, *To* 存储数据,*Ts*, *Tp*, *To* 分别以 *s*, *p*, *o* 为 row-key,*p*|*o*, *s*|*o*, *s*|*p* 为 value,用多个版本存放多个值. 在查询处理方面,文献[12]使用 MapReduce 处理 SPARQL 查询,提出一种贪心的多路连接选择策略,在 MapReduce 多路连接时优先处理具有高选择性的子句进行查询;文献[13]使用 HBase API 实现 SPARQL 查询,提出 3 个查询算法,分别实现 Triple Pattern 与三元组的匹配,在 HBase 数据库表中查询与 Triple Pattern 匹配的三元组以及在 HBase 数据库中查询与 SPARQL BGP 匹配的一组三元组.

文献[14]基于 Accumulo 数据库存储 RDF 数据. 用 3 张表 *SPO*, *POS* 与 *OSP* 分别将三元组以(主语,谓语,宾语)、(谓语,宾语,主语)、(宾语,主语,谓语)形式存储在 Row ID 中. 使用 OpenRDF Sesame SAIL API 实现 SPARQL 查询,并利用 *s*, *p*, *o* 的统计信息优化 SPARQL 连接的执行顺序. 支持子类、子属性、等价属性 3 种推理.

表 1 从存储模型、查询实现方式以及是否支持推理 3 个方面对这 5 种方案进行简单总结.

表 1　文献[10-14]方案总结

参考文献	存储模型	查询实现方式	是否支持推理
文献[10]	HDFS 文件形式,根据谓语和宾语所属类型划分文件	MapReduce	否
文献[11]	HDFS 文件形式,根据主语划分文件	MapReduce	否
文献[12]	HBase 数据库,*SPO*,*POS*,*OSP* 三张表	MapReduce	否
文献[13]	HBase 数据库,*Ts*,*Tp*,*To* 三张表	HBase Java API	否
文献[14]	Accumulo 数据库,*SPO*,*POS*,*OSP* 三张表	OpenRDF Sesame SAIL API	是

2　基于 HBase 的分布式 RDF 数据存储

2.1　HBase 数据模型

HBase 是建立在 Hadoop HDFS 上的稀疏的、面向列的分布式数据库,以 HTable 数据表存储数据.

HTable 是一张多维映射表,表中数据通过 4 个键索引:(row-key,族名:标签,时间戳)->value. row-key 是行标识,数据按照 row-key 字典序排序. 列族支持动态扩展,一个列族可以由任意多个列组成,每个列通过标签识别. 不同版本通过时间戳索引.

HTable 按列族存储,只存放有内容的表格单元,HTable 的稀疏性非常适合存储稀疏的 RDF 数据.

2.2　基于 HBase 的 RDF 数据存储模型

根据 OWL 本体定义文件,将本体中的类与属性信息存放在 *OWLClass* 与 *OWLProperty* 两张 HTable 表中. *OWLClass* 表存储每个类的子类、等价类等信息,*OWLProperty* 表存储每个属性的定义域、值域、子属性、等价属性、逆属性等信息.

采用将 RDF 数据按类划分的方法,为本体中的每一个类创建两张 HTable 表,表名分别为类名_*S_PO* 与类名_*O_PS*,每张表只有一个列族,名为 *P*. 两张表中的数据是以该类的实例为主语的三元组. 类名_*S_PO* 表以主语作为 row-key,谓语作为列标签,宾语为 value;类名_*O_PS* 表以宾语作为 row-key,谓语作为列标签,主语为 value.

创建 *RDFType* 表记录每个实例所属的类,该表以实例 IRI 作为 row-key,只有一个列族 *Type*,以实例所属的类名为 value.

创建 *RDFInstance* 表记录每个类所拥有的实例,该表以类名作为 row-key,只有一个列族 *Instances*,列族中的每一列存储一个实例,以实例 IRI 作为 value,通过动态地增加列来存储多个实例.

例如,OWL 本体定义文件中定义了类 *Professor*:

〈*owl*: *Class rdf*: *ID* = "*Professor*"〉〈/*owl*: *Class*〉,

并声明了定义域为 *Professor* 类的属性有 *name* 和 *teacherOf*.

在实例数据中,该类有两个实例 *Professor*0 与 *Professor*1,以这两个实例为主语的三元组有 5 个,分别为:

〈*Professor*0〉〈*name*〉"*Jone*";

〈*Professor*1〉〈*name*〉"*Sam*";

〈*Professor*0〉〈*teacherOf*〉〈*Course*0〉;

〈*Professor*0〉〈*teacherOf*〉〈*Course*1〉;

〈*Professor*1〉〈*teacherOf*〉〈*Course*1〉;

则 *Professor* 类的逻辑存储结构如图 1 所示,存储表中省略了时间戳.

Row-key	*P*	
	P:*name*	*P*:*teacherOf*
〈*Professor*0〉	"*Jone*"	〈*Course*0〉,〈*Course*1〉
〈*Professor*1〉	"*Sam*"	〈*Course*1〉

(a) *Professor_S_PO* 表

Row-key	*P*	
	P:*name*	*P*:*teacherOf*
"*Jone*"	〈*Professor*0〉	
〈*Course*0〉		〈*Professor*0〉
〈*Course*1〉		〈*Professor*0〉,〈*Professor*1〉
"*Sam*"	〈*Professor*1〉	

(b) *Professor_O_PS* 表

图 1　*Professor* 类存储结构

3　SPARQL 查询算法

实现上述存储模型上的 8 种 Triple Pattern 查询算法,提供部分推理功能的 Triple Pattern 推理算法和 BGP 查询算法,以下分别介绍这些算法.

3.1　Triple Pattern 与三元组匹配算法 matchTP_T

matchTP_T 算法用于判断 Triple Pattern 与三元组是否匹配,输入为 Triple Pattern *tp* 和三元组 *t*,若 *t* 与 *tp* 匹配,则返回 true,否则返回 false.

三元组与 Triple Pattern 匹配必须满足 3 个条件:1)变量可匹配所有变量与非变量;2)URI 或字面量只能匹配自身;3)出现一次以上的变量必须匹配出现次数与形式均相同的变量或非变量. 此算法设计不依赖于存储模式,由文献[13]提出.

3.2 单个 Triple Pattern 查询算法 QueryTP

QueryTP 算法查询与给定 Triple Pattern 匹配的三元组,支持(S,P,O) ,$(S,P,?O)$,$(S,?P,O)$,$(S,?P,?O)$,$(?S,P,O)$,$(?S,P,?O)$,$(?S,?P,O)$,$(?S,?P,?O)$8 种形式的查询. 具体算法描述如下:

算法 1. *QueryTP*.

输入:Triple Pattern $tp=(sp,pp,op)$;

输出:匹配 Triple Pattern 的三元组集合 R.

① $R=\varnothing$
② if $tp.pp==rdf:type$
③ then $R=QueryTP\_Type(tp)$
④ else if $tp.sp$ 非变量
⑤ then $R=QueryTP\_S(tp)$
⑥ else if $tp.pp$ 非变量
⑦ then $R=QueryTP\_P(tp)$
⑧ else $R=QueryTP\_O(tp)$
⑨ end if
⑩ end if
⑪ end if
⑫ return R

QueryTP 算法分 4 种情况处理 tp:

1) 若谓语为 $rdf:type$,则在 *RDFType* 表或 *RDFInstance* 表中查询匹配的三元组,由 *QueryTP_Type* 算法实现;

2) 若主语非变量,则先在 *RDFType* 表中查找到主语所属的类 Ci,然后在 Ci_S_PO 表中查询匹配的三元组,由 *QueryTP_S* 算法实现;

3) 若谓语非 $rdf:type$ 且非变量,则先在 *OWLClass* 与 *OWLProperty* 两张表中查找到谓语的定义域 C 和 C 的所有等价类 Ce 和子类 Cs,然后在所有 C,Ce 与 Cs 类存储表中查找匹配的三元组,由 *QueryTP_P* 算法实现;

4) 若非以上 3 种情况,则遍历所有类存储表,由 *QueryTP_O* 算法实现.

以上 4 种算法具体描述如下:

算法 2. *QueryTP_Type*.

输入:Triple Pattern $tp=(sp,pp,op)$;

输出:匹配 Triple Pattern 的三元组集合 R.

① $R=\varnothing$
② if $tp.sp$ 非变量
③ then 查找 *RDFType* 表中 row-key== $tp.sp$ 的记录,得到列族 *Type* 的值 Ci
④ for each $t=(sp,pp,Ci)$ do
⑤ if $matchTP_T(tp,t)==$ture
⑥ then 将 t 加入 R end if
⑦ end for
⑧ else if $tp.op$ 非变量
⑨ then 查找 *RDFInstance* 表中 *row-key* == $tp.op$ 的记录,得到列族 *Instances* 所有值 si
⑩ for each $t=(si,pp,op)$ do
⑪ if $matchTP_T(tp,t)==$ture
⑫ then 将 t 加入 R end if
⑬ end for
⑭ else 遍历 *RDFType* 表中所有记录,得到每条记录的 row-key si 及列族 *Type* 值 Ci
⑮ for each $t=(si,pp,Ci)$ do
⑯ if $matchTP_T(tp,t)==$ture
⑰ then 将 t 加入 R end if
⑱ endfor
⑲ end if
⑳ end if
㉑ 返回 R

QueryTP_Type 算法分 3 种情况处理 tp:

1) 若主语 sp 非变量,则在 *RDFType* 表中查找 sp 所属的类 Ci,用 Ci 替代宾语 op 组成新的三元组 t,调用 $matchTP\_T$ 判断 tp 与 t 是否匹配;

2) 若 op 非变量,则在 *RDFInstance* 表中查找类 op 的所有实例 si,用 si 替代 sp 组成新的三元组 t,调用 $matchTP\_T$ 判断 tp 与 t 是否匹配;

3) 若 sp 与 op 都为变量,则遍历 *RDFType* 表,用每条记录的 row-key 和 value 替代 sp 和 op 组成新的三元组 t,调用函数 $matchTP\_T$ 判断 tp 与 t 是否匹配.

算法 3. *QueryTP_S*.

输入:Triple Pattern $tp=(sp,pp,op)$;

输出:匹配 Triple Pattern 的三元组集合 R.

① $R=\varnothing$
② 查找 *RDFType* 表中 row-key== $tp.sp$ 的记录,得到列族 *Type* 值 Ci
③ if $tp.pp$ 非变量
④ then 查找 Ci_S_PO 表中 row-key== $tp.sp$

的记录,得到列 P:$tp.pp$ 值 oi

⑤ for each $t=(sp,pp,oi)$ do

⑥ if $matchTP\_T(tp,t)$==ture

⑦ then 将 t 加入 R end if

⑧ end for

⑨ else 查找 Ci_S_PO 表中 row-key==$tp.sp$ 的记录,得到所有列 P:pi 所有值 oi,

⑩ for each $t=(sp,pi,oi)$ do

⑪ if $matchTP\_T(tp,t)$==ture

⑫ then 将 t 加入 R end if

⑬ end for

⑭ end if

⑮ 返回 R

QueryTP_S 首先在 *RDFType* 表中查找主语 sp 所属的类 Ci,然后分两种情况处理:

1) 若谓语 pp 非变量,则在 Ci_S_PO 表中查询以 sp 为 row-key 的记录,取以 pp 为列标签的列值,用该值替代宾语 op 组成新的三元组 t,调用函数 $matchTP\_T$ 判断 tp 与 t 是否匹配;

2) 若 pp 为变量,则在 Ci_S_PO 表中查询以 sp 为 row-key 的记录,用每个列的标签和列值替代 pp 和 op 组成新的三元组 t,调用函数 $matchTP\_T$ 判断 tp 与 t 是否匹配.

算法 4. *QueryTP_P*.

输入:Triple Pattern $tp=(sp,pp,op)$;

输出:匹配 Triple Pattern 的三元组集合 R.

① $R=\varnothing$

② 查找 *OWLProperty* 表中属性 $tp.pp$ 的定义域 Ci,查找 *OWLClass* 表中 Ci 的所有等价类 Cei 和子类 Csi

③ if $tp.op$ 非变量

④ then 查找 Ci_O_PS 表、Cei_O_PS 表及 Csi_O_PS 表中 row-key==$tp.op$ 的记录,得到列 P:$tp.pp$ 的所有值 si

⑤ for each $t=(si,pp,op)$ do

⑥ if $matchTP\_T(tp,t)$==ture

⑦ then 将 t 加入 R end if

⑧ end for

⑨ else 遍历 Ci_S_PO 表、Cei_S_PO 表及 Csi_S_PO 表中所有记录,得到每条记录的 row-key si 及列 P:$tp.pp$ 的所有值 oi

⑩ for each $t=(si,pp,oi)$ do

⑪ if $matchTP\_T(tp,t)$==ture

⑫ then 将 t 加入 R end if

⑬ end for

⑭ end if

⑮ 返回 R

QueryTP_P 算法首先查找 *OWLProperty* 表中谓语 pp 的定义域 Ci,查找 *OWLClass* 表中类 Ci 的所有等价类 Cei 和子类 Csi,然后分两种情况处理:

1) 若宾语 op 非变量,则在 Ci_O_PS 表、Cei_O_PS 表及 Csi_O_PS 表中查询以 op 为 row-key 的记录,取以 pp 为列标签的列值,用该值替代主语 sp 组成新的三元组 t,调用函数 $matchTP\_T$ 判断 tp 与 t 是否匹配;

2) 若 op 为变量,则遍历 Ci_S_PO 表、Cei_S_PO 表及 Csi_S_PO 表,用每条记录的 row-key 替代 sp,取以 pp 为列标签的列值,用该值替代 op 组成新的三元组 t,调用函数 $matchTP\_T$ 判断 tp 与 t 是否匹配.

算法 5. *QueryTP_O*.

输入:Triple Pattern $tp=(sp,pp,op)$;

输出:匹配 Triple Pattern 的三元组集合 R.

① $R=\varnothing$;

② if $tp.op$ 非变量

③ then 对 *OWLClass* 表中每个类 Ci,查找 Ci_O_PS 表中 row-key==$tp.op$ 记录,得到每列 P:pi 所有值 si

④ for each $t=(si,pi,op)$ do

⑤ if $matchTP\_T(tp,t)$==ture

⑥ then 将 t 加入 R end if

⑦ end for

⑧ else 对 *OWLClass* 表中每个类 Ci,遍历 Ci_S_PO 表的所有记录,得到每条记录的 row-key si、列标签 pi 和值 oi

⑨ for each $t=(si,pi,oi)$ do

⑩ if $matchTP\_T(tp,t)$==ture

⑪ then 将 t 加入 R end if

⑫ end for

⑬ end if

⑭ 返回 R

QueryTP_O 算法分两种情况处理 tp:

1) 若宾语 op 非变量,则对 *OWLClass* 表中每个类 Ci,查找 Ci_O_PS 表中以 op 为 row-key 的记录,用每个列的标签和列值替代谓语 pp 和主语 sp 组成新的三元组 t,调用函数 $matchTP\_T$ 判断 tp 与 t 是否匹配;

2) 若 op 为变量,则对 *OWLClass* 表中每个类 Ci,遍历 Ci_S_PO 表的所有记录,用每条记录的 row-key 及每个列的标签和列值替代 sp,pp 和 op 组成新的三元组 t,调用函数 $matchTP\_T$ 判断 tp 与 t 是否匹配.

3.3 推理算法 ReasonTP

ReasonTP 算法实现 OWL 定义的 rdfs:subClassOf, owl: equivalentClass, rdfs: subPropertyOf, owl: equivalentProperty 和 owl:inverseOf 五种推理,由给定 Triple Pattern 推理出满足以上 5 种推理的一组 Triple Pattern.

推理算法具体描述如下:

算法 6. *ReasonTP*.

输入:Triple Pattern $tp=(sp,pp,op)$;

输出:结果集 R、包含 tp 及由 tp 推理出的所有 Triple Pattern.

① $R=\varnothing$
② 将 tp 加入 R
③ do 顺序取 R 中的每个 tp
④ if($tp.sp$ 是某个类 ‖ $tp.op$ 是某个类)
⑤ then 在 *OWLClass* 表中查找此类的子类与等价类
⑥ 用该类的子类或等价类替换原类在 tp 中的位置,组成新的 Triple Pattern tp'
⑦ 将 tp' 加入 R
⑧ end if
⑨ if($tp.pp$ 非变量)
⑩ then 在 *OWLProperty* 表中查找该属性的子属性、等价属性与逆属性
⑪ 用该属性的子属性或等价属性替换原属性,组成新的 Triple Pattern tp'
⑫ 将 tp' 加入 R
⑬ 用该属性的逆属性 pp' 替换原属性,并交换 tp 的主语与宾语,组成新的 Triple Pattern $tp'=(tp.op,pp',tp.sp)$
⑭ 将 tp' 加入 R
⑮ end if
⑯ until tp 为 R 的最后一个 Triple Pattern
⑰ 返回 R

例如: Triple Pattern (? *prof*, *rdf*: *type*, *Faculty*),本体定义文件中定义了类 *Faculty* 有子类 *Professor*,类 *Professor* 有子类 *FullProfessor*,则由(? *prof*,*rdf*:*type*,*Faculty*)可推理出(? *prof*, *rdf*: *type*, *Faculty*), (? *prof*, *rdf*: *type*, *Professor*),(? *prof*,*rdf*:*type*,*FullProfessor*)3 个 Triple Pattern.

3.4 BGP 查询算法 *QueryBGP*

QueryBGP 算法查询与 SPARQL BGP 匹配的一组三元组.算法的步骤如下:

1) 将 BGP 中的所有 Triple Pattern 重新排序,排序的依据为(1)与之前已排好序的 Triple Pattern 有共享变量的 Triple Pattern 优先;(2)选择性最高.

Triple Pattern 选择性由高到低的顺序如下:

① 主语非变量;

② 主语为变量,谓语非变量且非 *rdf*:*type*,宾语非变量;

③ 主语为变量,谓语为 *rdf*:*type*;

④ 主语为变量,谓语非变量且非 *rdf*:*type*,宾语为变量;

⑤ 主语与谓语均为变量,宾语为非变量;

⑥ 主语、谓语、宾语均为变量.

2) 对重排序后的第 1 个 Triple Pattern tp_1 进行推理 $ReasonTP(tp_1)$得推理出的结果集 S_1.

3) 调用算法 2 中的函数 *QueryTP* 查询 S_1 中的每个 Triple Pattern,将查询结果合并得结果集 R,若 R 为空则直接返回空集.

4) 否则取下一个 Triple Pattern tp_i,查询前先判断 tp_i 与之前的 Triple Pattern 是否有共享变量,若没有,则调用 *QueryTP* 查询 $ReasonTP(tp_i)$推理出的每个 Triple Pattern,将得到的结果并入结果集 R;若有共享变量,则用 R 中该变量已有的值替代 tp_i 中的共享变量得到新的 Triple Pattern tp'_i,调用 *QueryTP* 查询 $ReasonTP(tp'_i)$ 推理出的每个 Triple Pattern,将查询结果合并得结果集 R',若 R' 为空,则删除 R 中该已有值所在的三元组,若 R' 不为空,则将 R' 并入 R.

5) 重复 4)直至所有 Triple Pattern 查询结束.

6) 返回结果集 R.

具体算法描述如下:

算法 7. *QueryBGP*.

输入:BGP $bgp=\{tp_1,tp_2,\cdots,tp_n\},n\rangle=1$;

输出:匹配 BGP 的三元组集合 R.

① 根据共享变量优先与选择性最高原则将 bgp 中的所有 tp 重新排序,设重新排序后 $bgp=\{tp_1,tp_2,\cdots,tp_n\}$
② $R=\varnothing$

③ $S_1=ReasonTP(tp_1)$
④ for each tp_i in S_1 do $R=R\cup QueryTP(tp_i)$
　end for
⑤ if $R==\varnothing$ then return R end if
⑥ for each tp_i in $(tp_2,\cdots,tp_n)$ do
⑦　if tp_i 与 $tp_{i-1},\cdots,tp_1$ 有共享变量,
⑧　then 用 R 中该共享变量的已有值替代 tp_i 中的共享变量,设集合 TP 由替代后的 Triple Pattern tp' 组成
⑨　　for each tp' in TP do
⑩　　$R'=\varnothing$
⑪　　$S_i=ReasonTP(tp')$
⑫　　for each tp'_i in S_i do
⑬　　$R'=R'\cup QueryTP(tp'_i)$
⑭　　end for
⑮　　if $R'!=\varnothing$ then $R=R\cup R'$
⑯　　else 删除 R 中该已有值所在的三元组
⑰　　if $R==\varnothing$ then return R end if
⑱　　end if
⑲　end for
⑳　else $R'=\varnothing$
㉑　　$S_i=ReasonTP(tp_i)$
㉒　　for each tp'_i in S_i do
㉓　　$R'=R'\cup QueryTP(tp'_i)$
㉔　　end for
㉕　　if $R'!=\varnothing$ then $R=R\cup R'$ end if
㉖　end if
㉗ end for
㉘ 返回 R

3.5 BGP 查询示例

下面举例说明整个 BGP 查询过程:

BGP:{tp_1:?X,rdf:$type$,$Faculty$.
　　tp_2:?X,$teacherOf$,?Y.
　　tp_3:?X,$name$,$Jone$. }

此查询的功能是找出名字为 $Jone$ 的教职工以及他所教的课程.

查询过程如下:

1) 根据共享变量优先与选择性最高原则将 BGP 中的所有 Triple Pattern 重新排序,重新排序后的 BGP 变为

BGP:{tp_1:?X,$name$,$Jone$.
　　tp_2:?X,rdf:$type$,$Faculty$.
　　tp_3:?X,$teacherOf$,?Y. }

然后按顺序查询每条 Triple Pattern.

2) 查询 tp_1:调用函数 $ReasonTP(tp_1)$ 对 tp_1 进行推理得到推理出的结果集 S_1 {(?X,$name$,$Jone$)}. 调用函数 $QueryTP$ 查询 S_1 中的每个 Triple Pattern,(?X,$name$,$Jone$)主语未知,谓语已知且非 rdf:$type$,在 $QueryTP$ 时实际调用 $QueryTP\_P$ 函数查询.由 2.2 节图 1 中的数据可得 S_1 的查询结果为{($Professor0$,$name$,$Jone$)},将该结果并入结果集 R.

3) 查询 tp_2:tp_2 与 tp_1 有共享变量? X,用 R 中变量? X 的已有值 $Professor0$ 替代 tp_2 中的变量 ?X 得到 tp'_2:($Professor0$,rdf:$type$,$Faculty$). 调用函数 $ReasonTP(tp'_2)$ 对 tp'_2 进行推理,本体中定义了类 $Faculty$ 有子类 $Professor$,由子类推理可得推理结果集 S_2:{($Professor0$,rdf:$type$,$Faculty$),($Professor0$,rdf:$type$,$Professor$)}. 调用函数 $QueryTP$ 查询 S_2 中的每个 Triple Pattern, S_2 中的两个 Triple Pattern 谓语均为 rdf:$type$,在 $QueryTP$ 时实际调用 $QueryTP\_Type$ 函数查询,将两者的查询结果合并,由 $RDFType$ 表数据可知 S_2 查询结果 R_2 为{($Professor0$,rdf:$type$,$Professor$)},R_2 非空,将 R_2 并入结果集 R,此时 R 为{($Professor0$,$name$,$Jone$),($Professor0$,rdf:ype,$Professor$)}.

4) 查询 tp_3:tp_3 与 tp_2,tp_1 有共享变量? X,用 R 中变量? X 的已有值 $Professor0$ 替代 tp_3 中的变量? X 得到 tp'_3:($Professor0$,$teacherOf$,? Y).调用函数 $ReasonTP(tp'_3)$ 对 tp'_3 进行推理得到推理结果集 S_3:{($Professor0$,$teacherOf$,? Y)}.调用函数 $QueryTP$ 查询 S_3 中的每个 Triple Pattern,($Professor0$, $teacherOf$,? Y) 主语已知,在 $QueryTP$ 时实际调用 $QueryTP\_S$ 函数查询.由 2.2 节图 1 中的数据得到 S_3 的查询结果 R_3 为{($Professor0$, $teacherOf$, $Course0$),($Professor0$, $teacherOf$,$Course1$)},R_3 非空,将 R_3 并入结果集 R,此时 R 为{($Professor0$, $name$, $Jone$),($Professor0$, rdf: $type$, $Professor$), ($Professor0$, $teacherOf$, $Course0$),($Professor0$,$teacherOf$,$Course1$)}.

5) BGP 查询结束,查询结果 R 为{($Professor0$, $name$, $Jone$),($Professor0$, rdf:$type$, $Professor$), ($Professor0$, $teacherOf$, $Course0$), ($Professor0$, $teacherOf$,$Course1$)},可得变量? X 值为 $Professor0$, 变量? Y 值为 $Course0$,$Course1$.

4 实验结果与分析

4.1 实验环境

实验硬件环境基于一个包含 19 个节点的

Hadoop 集群. 两个主节点 NameNode, JobTracker 以及 17 个从节点的配置均为 Intel Xeon® E5620 2.4 GHz、24 GB 内存、2 TB 硬盘. 集群使用操作系统为 Redhat Enterpreise Linux Server 6.0, 开发环境为 Java SE 1.6, Hadoop 0.20.205.0, HBase 0.94.3.

4.2 数据加载

实验采用 LUBM(Lehigh University Benchmark)[18] 测试集. LUBM 包括 OWL 本体定义文件、数据生成器 UBA 以及 14 种 SPARQL 查询.

数据导入过程分 3 步:

1) 对 OWL 本体定义文件进行解析, 创建 *OWLClass* 与 *OWLProperty* 表存储本体中的类与属性信息; 为每个类创建两张存储表, 表名为类名_*S*_*PO* 和类名_*O*_*PS*; 创建 *RDFType* 表与 *RDFInstance* 表.

2) 用数据生成器 UBA 生成 RDF 数据. 实验采用 5 组不同大小的数据集进行测试, 数据集大小如表 2 所示:

表 2 实验数据集

数据集	大学数	实例数	RDF 三元组数
D_1	1	20 659	82 415
D_2	5	129 533	516 116
D_3	10	263 427	1 052 895
D_4	50	1 381 216	5 507 426
D_5	100	2 779 262	11 096 694

3) 将生成的数据集放到 HDFS, 然后通过一次 MapReduce 任务将数据加载到每个类的两张存储表以及 *RDFType* 表与 *RDFInstance* 表.

4.3 实验结果分析

实验在不同数据集上分别测试 LUBM 提供的 14 种查询. 每个查询在每个数据集上运行 5 次取响应时间的平均值, 各查询的平均响应时间如表 3 所示, 时间单位为毫秒(ms).

实验结果表明:

1) 对于语句与变量较少且具有较高选择性的查询, 如 Q_1, Q_3, Q_6, Q_{10}, Q_{11}, Q_{13}, Q_{14} 响应时间最快, 在数据集成倍增长的情况下响应时间只少量增加;

2) 有多条语句多个变量, 但其中一条语句有较高选择性的查询, 如 Q_4, Q_7, Q_{12}, 响应时间较快, 随着数据集的增长, 响应时间增长较慢;

3) 语句与变量虽少但选择性较低或有较复杂推理关系的查询, 如 Q_5 响应时间较慢, 随着数据集的增长, 响应时间增长较快;

4) 语句与变量较多且有复杂推理关系的查询, 如 Q_2, Q_8, Q_9 则响应时间最慢, 随着数据集的增长, 响应时间增长最快.

由此可见本文的存储与查询方案对选择性较高的语句查询效率高, 但对多个变量以及需要复杂推理的语句由于查询过程中增加了推理时间以及共享变量替代时间, 所以查询效率较低.

表 3 查询响应时间 ms

查询	D_1	D_2	D_3	D_4	D_5
Q_1	176	209	255	427	694
Q_2	9 484	15 550	26 638	48 941	79 262
Q_3	161	182	196	226	267
Q_4	770	791	858	953	1 095
Q_5	13 141	14 648	16 111	22 764	34 403
Q_6	89	130	221	294	376
Q_7	396	493	572	703	927
Q_8	20 457	25 441	32 611	53 251	76 149
Q_9	85 602	103 367	147 352	211 045	294 396
Q_{10}	54	61	72	96	121
Q_{11}	32	37	48	57	72
Q_{12}	1	2	3	5	7
Q_{13}	134	151	179	219	266
Q_{14}	62	78	98	119	154

5 结束语

本文针对如何高效管理海量 RDF 数据问题提出一种基于 HBase 的 RDF 数据存储模型, 按类划分并存储数据, 实现该存储模型上的 Triple Pattern 和具有部分推理功能的 BGP 查询算法, 并在 Hadoop 集群上验证了存储模型与查询算法的可行性. 更进一步工作是利用 MapReduce 计算模型对查询算法进行优化以及增强语义推理功能, 并与其他存储模型进行比较.

参 考 文 献

[1] W3C Recommendation. Resource description framework (RDF): Concepts and Abstract Syntax. (2004-02-10). http://www.w3.org/rdf/

[2] 杜小勇, 王珊, 吕彬. 语义 Web 数据管理研究进展. 软件学报, 2009, 20(11): 2950-2964

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

[4] Neumann T, Weikum G. x-rdf-3x: Fast querying, high update rates, and consistency for rdf databases // Proc of VLDB'10. Trondheim, Norway: VLDB Endowment, 2010: 256-263

[5] Weiss C, Karras P, Bernstein A. Hexastore: Sextuple indexing for semantic web data management //Proc of VLDB'08. Trondheim, Norway: VLDB Endowment, 2008: 1008-1019

[6] Abadi D J, Marcus A, Madden S R, et al. Scalable semantic web data management using vertical partitioning //Proc of VLDB'07. Trondheim, Norway: VLDB Endowment, 2007: 411-422

[7] Sidirourgos L, Goncalves R, Kersten M, et al. Column-store support for RDF data management: not all swans are white // Proc of VLDB'08. Trondheim, Norway: VLDB Endowment, 2008: 1553-1563

[8] Abadi D J, Marcus A, Madden S, et al. SW-Store: A vertically partitioned DBMS for Semantic Web data management. The VLDB Journal, 2009, 18(2): 385-406

[9] Franke C, Morin S, Chebotko A, et al. Distributed semantic web data management in HBase and MySQL cluster //Proc of Cloud'11. Los Alamitos, CA: IEEE Computer Society, 2011: 105-112

[10] Husain M F, Doshi P, Khan L. Storage and retrieval of large RDF graph using hadoop and MapReduce //Proc of CloudCom'09. Berlin: Springer, 2009: 680-686

[11] Rohloff K, Schantz R. High-performance, massively scalable distributed systems using the mapreduce software framework: The shard triple-store //Proc of PSI EtA'10. New York: ACM, 2010: 1-4

[12] Sun J, Jin Q. Scalable RDF store based on HBase and MapReduce //Proc of Advanced Computer Theory and Engineering. New York: ACM, 2010: 633-636

[13] Abraham J, Brazier P, Chebotko A, et al. Distributed storage and querying techniques for a semantic Web of scientific workflow provenance //Proc of Services Computing (SCC'10). Los Alamitos, CA: IEEE Computer Society, 2010: 178-185

[14] Punnoose R, Crainiceanu A, Rapp D. Rya: A scalable RDF triple store for the clouds //Proc of Cloud Intelligence (Cloud-I'12). New York: ACM, 2012: 1-4

[15] Chang F, Dean J, Ghemawat S, et al. Bigtable: A distributed storage system for structured data. Journal of ACM Trans on Computer Systems (TOCS), 2008, 2(26):

[16] W3C Recommendation. OWL 2 Web ontology language profiles. (2012-12-11). http://www.w3.org/TR/owl2-profiles/

[17] W3C Recommendation. SPARQL 1.1 query language. (2013-03-21). http://www.w3.org/TR/sparql11-query/

[18] Guo Y, Pan Z, Heflin J. LUBM: A benchmark for OWL knowledge base systems. Journal of Semantic Web, 2005, 3(2): 158-182

朱 敏 女,1987 年生,硕士研究生,主要研究方向为 RDF 数据管理.

程 佳 男,1987 年生,硕士研究生,主要研究方向为数据库技术.

柏文阳 男,1967 年生,副教授,中国计算机学会高级会员,主要研究方向为数据库、数据挖掘、数据库安全.