第47卷第10期 中国科学技术大学学报 Vol.47,No.10
2017年10月 JOURNAL OF UNIVERSITY OF SCIENCE AND TECHNOLOGY OF CHINA Oct. 2017

文章编号:0253-2778(2017)10-0823-14

分布式RDF关键词近似搜索方法

陈 远,汪璟玢

(福州大学数学与计算机科学学院,福建福州 350108)

摘要:现有的RDF关键词搜索方法主要是在大规模的RDF数据图上直接进行搜索,未能充分利用RDF本体中的语义信息,迭代次数过多造成搜索效率和效果不理想.针对这些问题,借助Redis内存数据库集群,提出分布式RDF关键词近似搜索算法(DKASR),即在分布式平台上实现大规模数据的并行搜索.算法结合RDF本体的语义信息构建本体子图,利用语义评分函数对本体子图进行排序,借助MapReduce计算模型实现并行搜索并返回Top-k结果;如果返回的结果没有达到Top-k,则对本体子图进行扩展生成近似本体子图,使用语义相似度函数对近似本体子图进行排序,再利用MapReduce计算模型实现并行搜索,直到返回Top-k结果.实验结果表明,DKASR算法能够高效正确地实现RDF关键词近似搜索并有效返回Top-k结果.

关键词:RDF;关键词;近似搜索;Redis;MapReduce

中图分类号:TP391 **文献标识码**:A doi:10.3969/j.issn.0253-2778.2017.10.004

引用格式:陈远,汪璟玢. 分布式RDF关键词近似搜索方法[J]. 中国科学技术大学学报,2017,47(10):823-836.
CHEN Yuan, WANG Jingbin. Distributed keyword approximate search method for RDF[J]. Journal of University of Science and Technology of China, 2017,47(10):823-836.

Distributed keyword approximate search method for RDF

CHEN Yuan, WANG Jingbin

(*College of Mathematics and Computer Science, Fuzhou University, Fuzhou* 350108, *China*)

Abstract: Existing RDF keyword search methods mainly search on the large-scale RDF data graph directly and do not make full use of the semantic information in the RDF ontology. Too many iterations lead to unfavorable search efficiency and unsatisfactory results. To solve these problems, a distributed keyword approximate search algorithm (DKASR) for RDF based on Redis memory database cluster was proposed and the parallel search of large-scale data on the distributed platform was realized. The algorithm constructs ontology sub-graphs by using the semantic information of RDF ontology, uses the semantic scoring function to sort ontology sub-graphs, and searches and returns the Top-k results concurrently with the aid of MapReduce computation model. If the results do not meet Top-k, ontology sub-graphs are extended to generate approximate ontology sub-graphs and the semantic similarity function is used to sort approximate ontology sub-graphs. Then, MapReduce computation model was used to realize the parallel search until the results meet Top-k. Finally, the results of experiments show that the DKASR algorithm

收稿日期:2016-08-28;**修回日期**:2016-12-08

基金项目:国家自然科学基金(61300104),福建省科技拥军基金(JG2014001),福建省自然科学基金(2012J01168),福州大学科技发展基金(2013-XQ-32)资助.

作者简介:陈远,男,1991年生,硕士生.研究方向:数据挖掘.E-mail:727947930@qq.com

通讯作者:汪景玢,硕士/副教授.E-mail:wjbcc@263.net

can realize the RDF keyword approximate search and return the Top-k results efficiently and accurately.
Key words: RDF; keyword; approximate search; Redis; MapReduce

0 引言

随着语义网的快速发展,资源描述框架(resource description framework, RDF)作为语义数据的描述标准被广泛应用.众多机构和项目均采用RDF来表达元数据,如Wikipedia、DBLP、IBM的"智慧地球"项目[1].面对爆发式增长的数据压力,如何存储和搜索大规模RDF数据成为亟待解决的问题[2].传统的存储技术在日益增长的数据面前暴露出许多不可克服的问题[3],然而结构简单的Redis数据库在存储海量RDF数据时却能具备优秀的性能,可满足大规模RDF数据储存的需求.

关键词搜索作为一种从RDF数据图检索信息的有效途径,普通用户在不需要熟悉任何标准的查询语言结构和底层数据模式的情况下就能快速有效地检索数据.根据查询处理方式的不同,RDF上的关键词查询大致可以分为两类[4]:第一类是由关键词构造出形式化查询语句再得到查询结果[5-7].该类方法通常包括关键词映射、构建查询和查询排序三个步骤.Gkirtzou等[7]结合数据图将包含用户查询关键词的子图映射并翻译成SPARQL查询语句,进行查询操作后返回结果.这类方法不但需要构建关键词索引和数据图的模式索引,而且要构建形式化查询语句,难以满足海量RDF数据存储和搜索的需求.第二类是由关键词直接构造查询的结果[8~10].这类方法通常需要借助有效的索引来快速定位子图并搜索结果,最常用的索引是倒排索引.文献[9]以实体三元组关联图为模型,封装文本信息到关联图顶点标签上,利用斯坦纳树问题的近似算法解决RDF数据的关键词查询问题.这类方法需要构建并维护索引,在处理海量数据的时候显得力不从心.

随着分布式思想的普及,要求关键词搜索不断演化为并行搜索分布式存储的大规模RDF数据[11].De Virgilio等[12]利用MapReduce计算模型将图并行问题转换为数据并行处理问题,实现了分布式RDF关键词的搜索.现存的分布式关键词搜索方法一般都是借助有效的索引直接在大规模的RDF数据图中搜索关键词匹配的节点,进行连接操作后返回结果,搜索效率不高;而且现存的方法中用到的存储媒介一般都是基于磁盘读取的,读取速度比较慢;同时由于不同用户对同一事物的描述可能存在差异,用户可能也不明确自己的搜索意图,在RDF数据搜索过程中可能返回空或者少量的结果[13].

本文利用Redis数据库集群来存储大规模的RDF数据,借助分布式Hadoop平台中的MapReduce计算框架,提出分布式RDF关键词近似搜索算法(distributed keyword approximate search method for RDF, DKASR),支持对实例、文本、类和属性的搜索.本文算法解决了海量数据无法快速搜索和搜索效果不理想的问题,并且支持返回用户可能感兴趣的结果,具有深远的理论和实际意义.

1 相关定义

问题定义:给定关键词集合 $Q=\{q_1,q_2,\cdots,q_i,\cdots,q_m\}$ 和RDF数据图 g,根据语义评分函数和语义相似度函数分别对本体子图和近似本体子图进行打分,分布式搜索返回与关键词集合匹配程度最高的Top-k结果.

定义 1.1 实例三元组.设 $t(s,p,o)$ 表示实例三元组,$s\in(I\cup B)$,$p\in(I\cup B)$,$o\in(I\cup B\cup L)$.其中 s 为主语,p 为谓语,o 为宾语,I 是 $IRIs$ 顶点集合,B 是空白顶点集合,L 是文本顶点集合,多个实例三元组组成一个RDF数据图.

定义 1.2 模式三元组.设 $T(S,P,O)$ 表示模式三元组,$S\in DS\in D$,$P\in A$,$O\in R$.其中 S 为主语,P 为谓语,O 为宾语,A 是RDF本体中定义的属性集合,属性有对象属性和数据属性两类,D 是RDF本体中定义的属性定义域集合,R 是RDF本体中定义的属性值域集合.

定义 1.3 本体扩展.模式三元组上的本体扩展是将本体扩展规则应用在本体的上下文中.设onto为RDF数据图的本体,closure(onto)为onto的闭包,给定模式三元组 T_1,并且 $T_1\notin$ closure(onto).如果用图1中三条规则的任一规则或者多条规则,可以由 T_1 得到 T_1',并且 $T_1'\notin$ closure(onto),则记为 $T_1\cup\text{onto}\cup\text{rules}\rightarrow T_1'$,称 T_1' 是 T_1 的一个近似模式三元组.图1中 sc 表示rdfs:subClassOf,sp 表示rdfs:subPropertyOf.

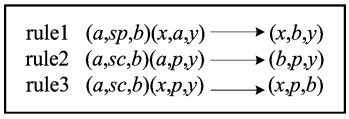

图 1　本体扩展规则 rules

Fig.1　Ontology extension rules

本体扩展包括属性扩展和类扩展，类扩展又可以分为对模式三元组的主语或者宾语进行的扩展.

(Ⅰ)模式三元组谓语的扩展.如果$(a,sp,b)\in$closure(onto)成立，则模式三元组(x,a,y)可扩展为(x,b,y).

(Ⅱ)模式三元组主语的扩展.如果$(a,sc,b)\in$closure(onto)成立，则模式三元组(a,p,y)可扩展为(b,p,y).

(Ⅲ)模式三元组宾语的扩展.如果$(a,sc,b)\in$closure(onto)成立，则模式三元组(x,p,a)可扩展为(x,p,b).

定义 1.4　近似模式三元组.对于给定的模式三元组 T，通过应用定义 1.3 进行扩展得到T'，则称T'为 T 的一个近似模式三元组.

定义 1.5　RDF 数据图.设 $g=\{t_1,t_2,\cdots,t_i,\cdots,t_n\}$表示 RDF 数据图，由实例三元组组成.每个实例三元组 $t_i(s_i,p_i,o_i)$中的主语 s_i 和宾语 o_i 作为 g 中的节点，谓语 p_i 作为由主语节点指向宾语节点的一条有向边.

定义 1.6　本体图.设 $G=\{T_1,T_2,\cdots,T_i,\cdots,T_n\}$表示本体图，由模式三元组组成，是 RDF 本体的三元组表现形式.每个模式三元组 $T_i(S_i,P_i,O_i)$中的主语 S_i 和宾语 O_i 作为 G 中的节点，谓语 P_i 作为由主语节点指向宾语节点的一条有向边.

定义 1.7　本体子图.设 $G_S=\{T_1,T_2,\cdots,T_i,\cdots,T_m\}$表示本体子图，已知关键词集合 $Q=\{q_1,q_2,\cdots,q_i,\cdots,q_k\}$和本体图 $G=\{T_1,T_2,\cdots,T_i,\cdots,T_n\}$，对于 Q 中的每个关键词 $q_i(1\leqslant i\leqslant k)$，首先将 q_i 映射成对应的类 c_i 或者属性 p_i；然后在 G 中搜索包含 c_i 或者 p_i 的模式三元组，并将匹配的模式三元组加入集合 $\text{Set}=\{T_1,T_2,\cdots,T_i,\cdots,T_r\}$，对于集合 Set 中的模式三元组 $T_i(S_i,P_i,O_i)$和 $T_j(S_j,P_j,O_j)$，其中 $\exists i,j\in\{1,2,\cdots,r\}$，有 $S_i\neq S_j\&\&P_i\neq P_j\ \&\&O_i\neq O_j$；最后依次从 Set 中取出模式三元组进行连接，形成本体子图 G_S. G_S 以模式三元组集合的形式表示，两个模式三元组集合中的三元组不完全相同，则认为是不同的本体子图.

定义 1.8　近似本体子图.对于给定的本体子图 G_S，通过运用定义 1.3 对本体子图中至少一个模式三元组进行扩展得到G_S'，则称G_S'为 G_S 的一个近似本体子图.

定义 1.9　结果子图.设 $G_S=\{t_1,t_2,\cdots,t_i,\cdots,t_m\}$表示结果子图，结果子图是本体子图或者近似本体子图在 RDF 数据图中分布式搜索的结果，是由所有关键词匹配的实例三元组进行连接构成的子图，包含了关键词集合中所有的关键词.结果子图以实例三元组集合的形式表示，两个实例三元组集合中的三元组不完全相同，则认为是不同的结果子图.

定义 1.10　三元组连接.在构造本体子图或者结果子图时，对匹配的模式三元组或者实例三元组进行三元组连接操作，其中任意两个三元组通过主语、宾语或者其他三元组连接起来.三元组连接的形式化表示：对于模式三元组或者实例三元组集合 $\text{Set}=\{T_1,T_2,\cdots,T_i,\cdots,T_m\}$，给定 $T_i(S_i,P_i,O_i)$和 $T_j(S_j,P_j,O_j)$，其中 $\exists i,j\in\{1,2,\cdots,m\}$，如果$(S_i=S_j\&\&O_i\neq O_j)$或者$(S_i=O_j\&\&O_i\neq S_j)$或者$(O_i=S_j\&\&S_i\neq O_j)$或者$(O_i=O_j\&\&S_i\neq S_j)$，则称 T_i 与 T_j 相邻，可以进行三元组连接.

定义 1.11　语义评分函数.对得到的多个本体子图，本文利用语义评分函数进行打分，得分高的优先进行分布式搜索.设本体子图 $G_S=\{T_1,T_2,\cdots,T_i,\cdots,T_m\}$和类集合 $C=\{c_1,c_2,\cdots,c_i,\cdots,c_p\}$；包含属性集合 $P=\{p_1,p_2,\cdots,p_i,\cdots,p_q\}$，则该本体子图的语义评分函数可以表示为

$$\text{SSF}(G_S)=\alpha\frac{1}{\text{CCDis}(G_S)}+(1-\alpha)\frac{1}{\text{PPDis}(G_S)}\tag{1}$$

式中，$\text{CCDis}(G_S)=\sum_{i,j\in1,2,\cdots,p}\text{dis}(c_i,c_j)$，$\text{PPDis}(G_S)=\sum_{i,j\in1,2,\cdots,q}\text{dis}(p_i,p_j)$.

语义评分函数 $\text{SSF}(G_S)$由语义内容 $\text{CCDis}(G_S)$和语义结构 $\text{PPDis}(G_S)$两部分组成.α 是调节参数，当 $\alpha=0.5$ 时，表示两者的影响程度一样.$\text{dis}(c_i,c_j)$表示类 c_i 与类 c_j 之间的语义距离，且

$$\text{dis}(c_i,c_j)=\begin{cases}1,c_i\text{ 与 }c_j\text{ 在同一模式三元组中}\\ n,c_i\text{ 与 }c_j\text{ 不在同一模式三元组中}\end{cases}\tag{2}$$

式中，n 是这两个类之间最短路径上边的条数.类间距离之和越小，$\frac{1}{\mathrm{CCDis}(G_S)}$ 的值越大，说明该本体子图的语义内容越紧密.

$\mathrm{dis}(p_i,p_j)$ 表示属性 p_i 与属性 p_j 之间的语义距离，且

$$\mathrm{dis}(p_i,p_j)=\begin{cases}1, p_i \text{ 与 } p_j \text{ 所在模式三元组相邻}\\ d, p_i \text{ 与 } p_j \text{ 所在模式三元组不相邻}\end{cases} \tag{3}$$

式中，d 是这两个属性之间最短路径上模式三元组的个数.属性间距离之和越小，$\frac{1}{\mathrm{PPDis}(G_S)}$ 的值越大，说明该本体子图的语义结构与用户想要的搜索结果越相似.

利用语义评分函数 $\mathrm{SSF}(G_S)$，使得评分高的本体子图优先进行分布式搜索.

定义 1.12 语义相似度函数.本文用语义相似度函数来衡量初始本体子图与扩展后的近似本体子图之间的相似程度，相似度越大的越优先执行分布式搜索.语义相似度计算过程中涉及类节点之间、属性节点之间、模式三元组之间以及本体子图之间的语义相似度计算.本文借鉴文献[14]中定义的最小公共祖先(least common ancestor, LCA)的概念和语义相似度计算方法来完成本文的语义相似度计算.

(Ⅰ)类节点之间的语义相似度

模式三元组中的主语或宾语是一个类，在RDFs本体层次结构中可以看成一个节点，那么初始本体子图上的节点 c_1 和 c_1 扩展后对应的节点 c_1' 之间的语义相似度公式如下：

$$s(c_1,c_1')=d(c_1)+d(c_1')-2\times d(\mathrm{LCA}(c_1,c_1')) \tag{4}$$

式中，$d(c)$ 是指节点 c 在本体层次结构图中的深度.

(Ⅱ)属性节点之间的语义相似度

模式三元组中的谓语是一个属性，在RDFs本体层次结构中也可以看成一个节点，那么初始本体子图上的属性节点 p_1 和 p_1 扩展后对应的属性节点 p_1' 之间的语义相似度公式 $s(p_1,p_1')$ 与式(4)类似.

$$s(p_1,p_1')=d(p_1)+d(p_1')-2\times d(\mathrm{LCA}(p_1,p_1')) \tag{5}$$

式中，$d(p)$ 是指节点 p 在本体层次结构图中的深度.

(Ⅲ)模式三元组之间的语义相似度

设初始本体子图中的模式三元组 $T_1(S_1,P_1,O_1)$ 和近似本体子图中对应的模式三元组 $T_1'(S_1',P_1',O_1')$，综合公式(4)和(5)，本文 T_1 和 T_1' 的语义相似度公式如下：

$$s(T_1,T_1')=s(S_1,S_1')+s(P_1,P_1')+s(O_1,O_1') \tag{6}$$

(Ⅳ)本体子图之间的语义相似度

设初始本体子图 $G_1=\{T_1,T_2,\cdots,T_i,\cdots,T_m\}$ 和近似本体子图 $G_1'=\{T_1',T_2',\cdots,T_i',\cdots,T_m'\}$，则 G_1 和 G_1' 的语义相似度公式如下：

$$s(G_1,G_1')=1/\sum_{i=1}^{m}s(T_i,T_i') \tag{7}$$

初始本体子图与近似本体子图之间的语义相似度越大，说明该近似本体子图与初始本体子图越相似，那么该近似本体子图越优先进行分布式搜索，这样就能保证在返回结果没有达到 Top-k 的情况下进行近似分布式搜索，也能返回用户最想要的结果.

2 DKASR 算法

DKASR算法为了避免直接在大规模RDF数据图上进行耗时的迭代搜索，利用RDF本体的特点构造输入关键词集合对应的本体子图，并且综合考虑语义内容和语义结构对本体子图进行评分排序，评分高的优先进行分布式搜索；然后利用MapReduce计算框架并行搜索返回Top-k结果；如果得到的结果没有达到Top-k，则对本体子图进行扩展，生成近似本体子图，利用语义相似度函数对生成的近似本体子图进行语义相似度评分，评分高的优先进行分布式搜索，直到返回Top-k结果为止.由于RDF本体涵盖了资源和属性的分类及关联，而且RDF本体通常是确定的规模为kB级别的数据，因此构造本体子图或近似本体子图都是非常高效的.

DKASR算法的总体框架如图2所示，DKASR算法主要包括以下几个步骤：

Step1 对RDF本体和RDF实例数据进行预处理，并生成相应的文件；

Step2 结合Redis分布式数据库集群的特点，将Step1中生成的文件内容分布式地存储在Redis数据库中；

Step3 根据RDF本体信息和输入的关键词集合 Q，构建关键词集合对应的本体子图；

Step4　利用语义评分函数对构建的本体子图进行打分并排序，得分高的优先进行分布式搜索，跳到 Step5；

Step5　Map 阶段搜索本体子图中各模式三元组对应的实例三元组；

Step6　Reduce 阶段则将接收到的实例三元组进行连接，得到结果子图，并返回结果子图；

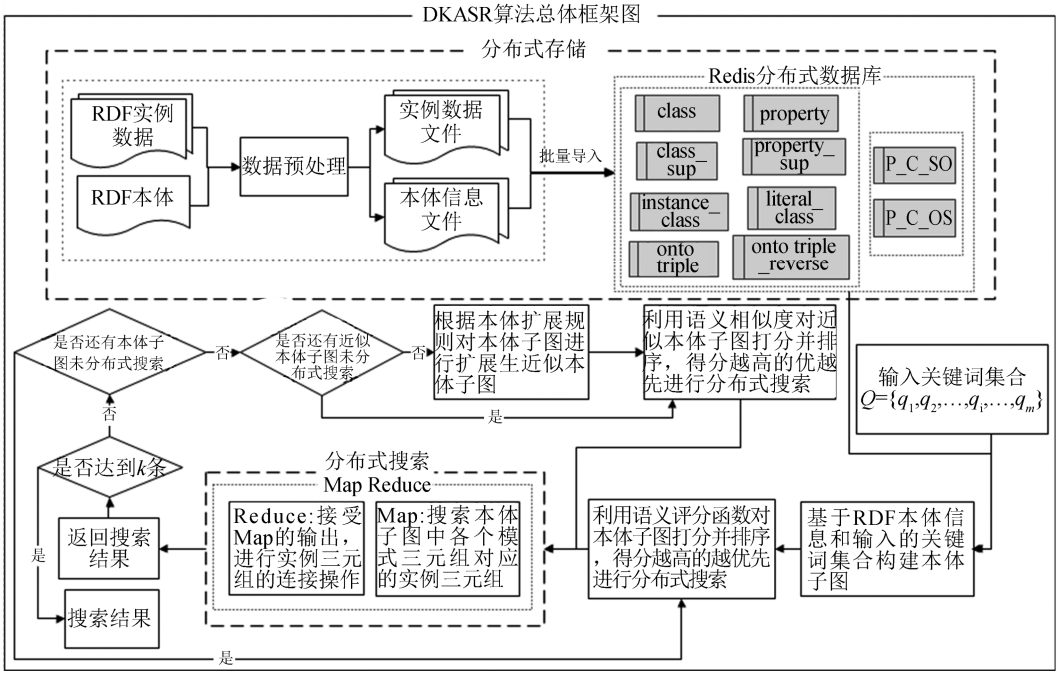

图 2　DKASR 算法总体框架图

Fig.2　The general framework of DKASR algorithm

Step7　判断结果子图中的实例三元组是否达到 k 条，如果已经达到，则结束搜索，跳到 Step12，否则跳到 Step8；

Step8　判断是否还有本体子图没有进行分布式搜索，如果有，则跳到 Step4，否则跳到 Step9；

Step9　判断是否有扩展后的近似本体子图未进行分布式搜索，如果有，则跳到 Step11，否则跳到 Step10；

Step10　根据本体扩展规则按 Step4 中的排序对本体子图进行扩展并生成近似本体子图，跳到 Step11；

Step11　利用语义相似度函数对生成的近似本体子图进行打分并排序，得分高的优先进行分布式搜索，跳到 Step5；

Step12　算法结束.

2.1　分布式存储方案

DKASR 算法利用 Redis 内存数据库集群作为数据存储的媒介，集群中 Redis 内存数据库的个数可根据需求动态增减.由于 Redis 中对 Set 集合的添加、删除和查找的复杂度都是 $O(1)$，本文将 RDF 本体信息和大规模的 RDF 实例数据进行预处理，分别存储在 Redis 内存数据库集群的 Set 集合中.具体的存储方案如表 1 所示.

表 1 中，Class、Property、OntoTriple 和 OntoTriple_Reverse 集合用来存储 RDF 本体的信息.根据存储的本体信息、Instance_Class 和 Literal_Class 集合中的信息可以快速判断输入的关键词是类、属性、实例还是文本，并且可以快速定位到每个关键词匹配的模式三元组，为构建本体子图做好准备.Class_Sup 用来存储类的父类信息，Property_Sup 用来存储属性的父属性信息，在使用定义 1.3 进行本体扩展时，根据 Class_Sup 和 Property_Sup 集合可以将本体子图扩展为近似本体子图.P_C_SO 和 P_C_OS 用来存储 RDF 实例数据，在进行分布式搜索时，根据本体子图中模式三元组的信息，可以大大缩小搜索范围并且能够快速搜索到每个模式三元组对应的实例三元组，实现高效的分布式并行搜索.

表 1 存储方案

Tab.1 Storage scheme

Set 名称	存储内容	Set 存储结构
Class	储存 RDF 本体中定义的类信息	Class=$\{c_1,c_2,\cdots,c_i,\cdots,c_n\}$,其中 c_i 表示类
Property	储存 RDF 本体中定义的属性、属性的定义域以及值域信息	Property=$\{p_1,p_2,\cdots,p_i,\cdots,p_n\}$,其中 $p_i=\{dr_1,dr_2,\cdots,dr_j,\cdots,dr_m\}$,$dr_j=(d_j,r_j)$,$p_i$ 表示属性,d_j 表示 p_i 的一个定义域,r_j 表示d_j 对应的值域
Class_Sup	存储类的父类信息	Class_Sup=$\{CS_1,CS_2,\cdots,CS_i,\cdots,CS_n\}$,其中$CS_i=\{S_1,S_2,\cdots,S_j,\cdots,S_m\}$,$CS_i$ 表示类,S_j 表示CS_i 的一个父类
Property_Sup	存储属性的父属性信息	Property_Sup=$\{PS_1,PS_2,\cdots,PS_i,\cdots,PS_n\}$,其中$PS_i=\{S_1,S_2,\cdots,S_j,\cdots,S_m\}$,$PS_i$ 表示属性,S_j 表示PS_i 的一个父属性
OntoTriple	存储所有的模式三元组信息	OntoTriple=$\{S_1,S_2,\cdots,S_i,\cdots,S_n\}$,其中 $S_i=\{PO_1,PO_2,\cdots,PO_j,\cdots,PO_m\}$,$PO_j=(P_j,O_j)$,$S_i$ 表示模式三元组的主语,P_j 表示 S_i 的一个谓语,O_j 表示 P_j 对应的宾语
OntoTriple_Reverse	存储所有模式三元组的反转备份	OntoTriple_Reverse=$\{O_1,O_2,\cdots,O_i,\cdots,O_n\}$,其中 $O_i=\{PS_1,PS_2,\cdots,PS_j,\cdots,PS_m\}$,$PS_j=(P_j,S_j)$,$O_i$ 表示模式三元组的宾语,P_j 表示 O_i 的一个谓语,S_j 表示 P_j 对应的主语
Instance_Class	储存 RDF 实例数据中的实例与实例所属类的映射关系	Instance_Class=$\{IC_1,Ic_2,\cdots,IC_i,\cdots,IC_n\}$,其中 $IC_i=\{I_1,I_2,\cdots,I_j,\cdots,I_m\}$,$IC_i$ 表示类,I_j 表示IC_i 的一个实例
Literal_Class	储存 RDF 数据图中的文本与包含该文本的实例三元组主语所属类的映射关系	Literal_Class=$\{LC_1,LC_2,\cdots,LC_i,\cdots,LC_n\}$,其中$LC_i=\{L_1,L_2,\cdots,L_j,\cdots,L_m\}$,$L_j$ 表示一个文本,LC_i 表示L_j 所在实例三元组主语所属的类
P_C_SO	具有相同谓语且主语所属类相同的实例三元组,存储在同一个 Set 中,以 S,O 的形式存储	P_C_SO=$\{P_iC_j\_SO\}$,$1\leqslant i\leqslant n$,$1\leqslant j\leqslant m$,n 表示属性的个数,m 表示属性 P_i 的定义域中类的个数,其中 $P_iC_j\_SO=\{SO_1,SO_2,\cdots,SO_k,\cdots,SO_q\}$,$SO_k=(S_k,O_k)$,$S_k$ 表示谓语为 P_i 且主语所属类为 C_j 的实例三元组的主语,O_k 表示 S_k 对应的宾语
P_C_OS	存储 P_C_SO 的反转备份,具有相同谓语且宾语所属类相同的实例三元组,存储在同一个 Set 中,以 O,S 的形式存储	P_C_OS=$\{P_iC_j\_OS\}$,$1\leqslant i\leqslant n$,$1\leqslant j\leqslant m$,n 表示属性的个数,m 表示属性 P_i 的值域中类的个数,其中 $P_iC_j\_OS=\{OS_1,OS_2,\cdots,OS_k,\cdots,OS_q\}$,$OS_k=(O_k,S_k)$,$O_k$ 表示谓语为 P_i 且宾语所属类为 C_j 的实例三元组的宾语,S_k 表示 O_k 对应的主语

2.2 构建本体子图

本文利用 RDF 本体的语义结构特征,先把关键词映射成对应的类或者属性,然后在本体图上找到类或者属性匹配的模式三元组,接着对模式三元组根据定义 1.10 进行连接,生成本体子图.由于关键词映射对应的类可能存在多个,并且类和属性可能存在于多个不同的模式三元组中,因此会生成多个本体子图,此时利用定义 1.11 对每个本体子图进行打分,得分高的优先在大规模的 RDF 数据图中搜索本体子图匹配的结果子图.RDF 本体图是 RDF 数据图的浓缩摘要,涵盖了资源和属性的分类及关联,而且规模大小一般都为 kB 级别(RDF 本体中定义的类和属性个数一般是几十到几百级别),通过本体可以推导出任意两个类或者属性的关联关系,并且可以快速构建关键词集合对应的本体子图,确定关键词之间的关系.在 RDF 本体图上先进行搜索和连接操作,会大大减少耗时,提高搜索效率.本文构建本体子图的过程如算法 2.1 所示.

算法 2.1 build-Onto-SubGraph(Q,OntoInfo)

输入:关键词集合 Q,RDF 本体信息 OntoInfo

输出:排好序的本体子图大堆

开始

tripleSet ← Q 中关键词映射后对应的类或属性匹配的模式三元组集合

G_1 ← ∅; //初始化本体子图为空

$H_1 \leftarrow \varnothing$; //本体子图大堆,大堆中的本体子图根据语义评分排序

For $T_i(S_i,P_i,O_i) \in$ tripleSet && i = 1,2,…,m // m 为 tripleSet 集合大小

G_1.add($T_i(S_i,P_i,O_i)$);

For $T_j(S_j,P_j,O_j) \in$ tripleSet && j = i+1 && j = 2,3,…,m

If check(G_1)==true //如果 G_1 中包含了全部映射后对应的类或属性

If contain(H_1,G_1)==false //如果 H_1 中不存在 G_1

SSF(G_1); //计算 G_1 的语义评分

H_1.push(G_1); //将 G_1 加入到 H_1 中

$G_1 \leftarrow \varnothing$;

G_1.add($T_i(S_i,P_i,O_i)$);

continue;

End if

Else

For $T_k(S_k,P_k,O_k) \in G_1$ && k = 1,2,…,n //n 为 G_1 中三元组条数

If (($S_k=S_j$ && $O_k \neq O_j$)||($S_k=O_j$ && $O_k \neq S_j$)||($O_k=S_j$ && $S_k \neq O_j$)||($O_k=O_j$ && $S_k \neq S_j$))

G_1.add($T_j(S_j,P_j,O_j)$);

break;

End If

End For

End If

End For

End For

Return H_1;

结束

举例说明,以 Q_2 为输入的搜索关键词集合.Q_2 构建本体子图的具体过程如图 3 所示.

1.判断关键词类型

关键词	所属类型
under graduate degree from	属性
research assistant	类
http://university982.ed	实例
graduate student121@department7.university0.edu	文本

2.映射关键词

关键词	映射后对应的类或属性
under graduate degree from	under graduate degree from
resear assistant	resear assistant
http://www.university982.ed	university
graduate student121@department1.university0.edu	resear assistant

3. 匹配模式三元组

映射后对应的类或属性	匹配的模式三元组
under graduate degree from	T_1(research assistant,undergraduate degree from,university) T_2(graduate student,undergraduate degree from,university)
research assistant	T_1(research assistant,undergraduate degree from,university) T_2(research assistant,degree from,university)
university	T_1(research assistant,undergraduate degree from,university) T_2(research assistant,degree from,university)
research assistant	T_2(research assistant,email address,"literal")

4. 生成本体子图

本体子图
T_1(research assistant,undergraduate degree from,university) T_2(research assistant,email address,"literal")
T_1(graduate student,undergraduate degree from,university) T_2(research assistant,degree from,university) T_2(research assistant,email address,"literal")

图 3　本体子图构建过程

Fig.3　Construction process of ontology sub-graphs

(Ⅰ)根据 2.1 节存储的信息可以得知各个关键词所属的类型;

(Ⅱ)根据 2.1 节中 Instance_Class 和 Literal_Class 集合的信息分别将 http://www.University982.ed 和 GraduateStudent121@Department7.University0.edu 映射成对应的类 University 和 ResearchAssistant;

(Ⅲ)在完成对关键词的映射之后,根据 2.1 节中 OntoTriple 和 OntoTriple_Reverse 集合的信息找到类或属性匹配的模式三元组,每个类或属性匹配的模式三元组可能很多,这里最多只列出其中的两个.可以看到映射后对应的类或属性中有两个 ResearchAssistant,但是由于关键词所属的类型不同,所以匹配到的模式三元组不同;

(Ⅳ)完成上述步骤之后,利用定义 1.10 对模式三元组进行连接,最后生成本体子图,可能会生成多个本体子图,这里只列出其中的两个.构建完成的两个本体子图的图结构如图 4 所示.

对于生成的多个本体子图,利用定义 1.11 计算每个本体子图的语义评分,评分高的优先进行分布式搜索,图 4 中本体子图的语义评分计算过程如下.其中 01,02,03,04,05 和 06 分别表示 ResearchAs-

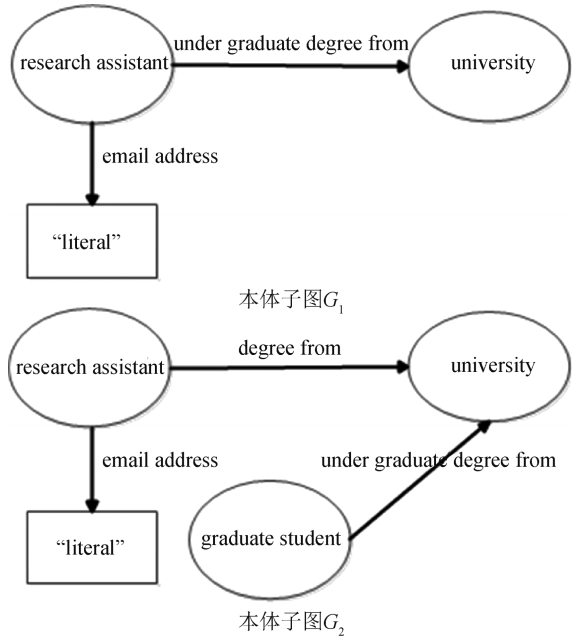

图 4 本体子图示例

Fig.4 Ontology sub-graphs samples

sistant, University, undergraduateDegreeFrom, emailAddress, GraduateStudent 和 degreeFrom,语义评分函数公式(1)中的参数 α 设置为 0.5.

$$SSF(G_1) = 0.5\frac{1}{dis(01,02)}+0.5\frac{1}{dis(03,04)} = 0.5+0.5 = 1;$$

$$SSF(G_2)= 0.5\frac{1}{dis(01,02)+dis(01,05)+dis(02,05)}+ 0.5\frac{1}{dis(03,04)+dis(03,06)+dis(04,06)}= 0.5\frac{1}{1+2+1}+0.5\frac{1}{2+1+1}= 0.25.$$

由计算结果可见,$SSF(G_1) > SSF(G_2)$,所以本体子图 G_1 先进行分布式搜索.

2.3 构建近似本体子图

目前的关键词搜索中是假定用户明确自己的搜索意图,实际上用户对本体的结构和内容有可能并不了解,不同的用户对同一事物的描述可能存在差异,用户可能也不明确自己的搜索意图.在这种情况下即使用户使用了明确的搜索关键词,搜索中仍然有可能返回空或少量的搜索结果.同时,在大多数情况下用户很难通过几个简单的关键词准确真实地表达自己的搜索需求,因此这可能导致搜索的结果和用户需求之间存在一定的差异.

对于初始本体子图中的一个模式三元组,该模式三元组中的主语或宾语表示的是一个类,谓语表示的是一个属性.本文在进行分布式搜索时,如果分布式搜索得到的结果没有达到 k 条,则要对初始本体子图中的模式三元组按照定义 1.3 进行扩展,那么类可以扩展为其对应的超类,属性可以扩展为其对应的超属性,相应地就得到了该模式三元组的近似模式三元组.由于扩展方式的多样性,初始本体子图可以通过扩展得到多个近似本体子图;然后通过定义 1.12 对近似本体子图进行评分,得分高的优先进行分布式搜索,这样就能有效地返回与用户意图相近、可能感兴趣的结果.

定义 1.12 是用来衡量初始本体子图与近似本体子图之间的语义相似程度.两个本体子图的语义相似度越大,说明二者越相似,即在语义上具有越强的相似性.语义相似度的计算主要考虑 RDFs 所体现的本体层次结构,如图 5 所示.RDF 本体中定义了类与属性、值域与定义域在属性上的约束以及子类与子属性的包蕴关系,通过对 RDF 本体的分析,可以推导得到类与类、类与属性以及属性与属性之间的语义关联.从图 5 可以看出,不管是类还是属性通过定义 1.3 进行扩展都可以一直扩展到本体层次结构的最顶端节点,但是随着扩展层数的增加,搜索结果的正确率会随之降低,为了确保具有较高的正确率,本文采取只扩展一层的策略.

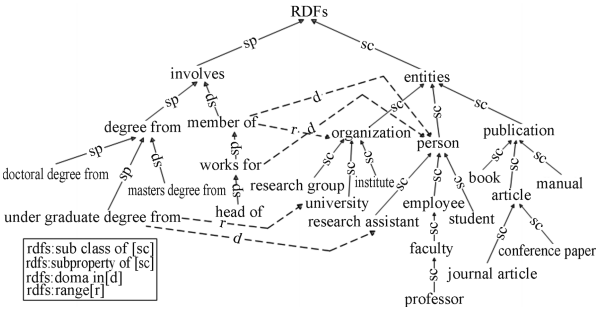

图 5 本体层次结构图

Fig.5 Ontology hierarchy structure

本文构建近似本体子图的过程如算法 2.2 所示.

算法 2.2 build-ApproOnto-SubGraph(H_1)

输入:算法 2.1 的输出 H_1

输出:排好序的近似本体子图大堆集合,集合中每个大堆存储 H_1 中本体子图对应的近似本体子图

开始

GSet ← ∅; //近似本体子图集合,初始化为空

HSet ← ∅; //大堆集合,集合中每个元素是一个大堆,大堆

中的近似本体子图根据语义相似度排序
For $G_i(T_1,T_2,\cdots,T_p,\cdots,T_m)\in H_1$ && i = 1,2,…,n //n 为 H_1 中本体子图的个数
  For $T_j(S_j,P_j,O_j)\in G_i$ && j = 1,2,…,m //m 为 G_i 中模式三元组条数
    GSet = extend(T_j); //T_j 按照定义 1.3 进行扩展,得到近似本体子图集合
  For $G_k\in$GSet && k = 1,2,…,q //q 为 GSet 集合大小
    If contain(HSet[i], G_k)==false //如果 HSet[i]中不存在 G_k
      s(G_k, G_i); //计算 G_k 与 G_i 的语义相似度
      HSet[i].add(G_k); //加入到大堆集合中对应的大堆中
    End If
  End for
  GSet ← ∅;
 End for
End for
Return HSet;
结束

以 2.2 节例子中构建的本体子图 G_1 为例来构建近似本体子图,利用定义 1.3 中的一条或者多条本体扩展规则对 G_1 中的模式三元组进行扩展,由于扩展方式的多样性,可以得到多个近似模式三元组,这里只列出其中的两种扩展策略,具体的扩展过程如图 6 所示.

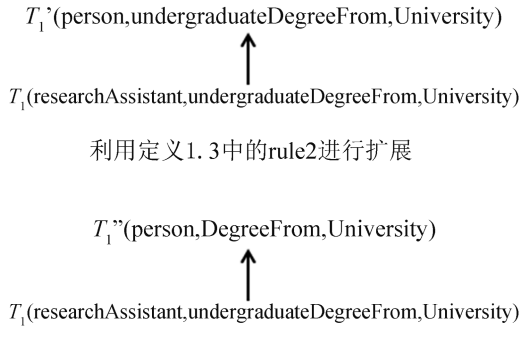

图 6　模式三元组扩展

Fig.6　Extension of pattern triples

经过图 6 的两种扩展策略后,G_1 生成的两个近似本体子图如图 7 所示.

对于生成的多个近似本体子图,利用定义 1.12 计算每个近似本体子图与 G_1 之间的语义相似度,语义相似度高的优先进行分布式搜索,图 7 中近似本体子图的语义相似度计算过程如下,其中 T_1,T_1' 和 T_1"如图 6 中所示.

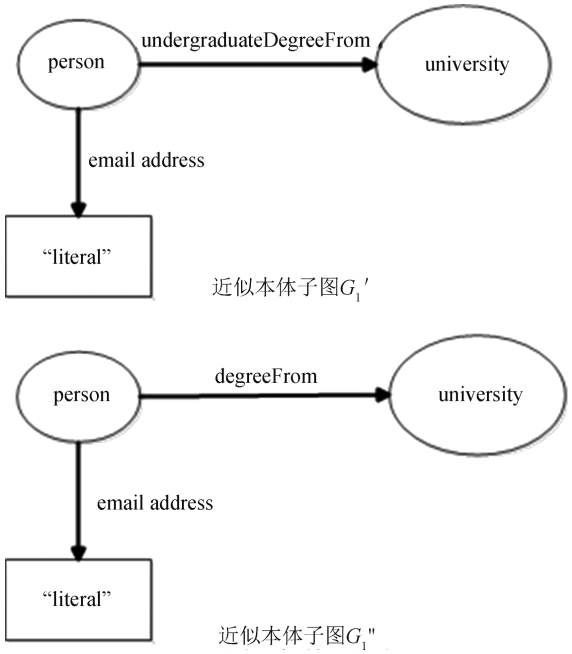

图 7　近似本体子图示例

Fig.7　Approximate ontology sub-graphs samples

$$s(G_1,G_1')=\frac{1}{s(T_1,T_1')}=\frac{1}{2+3-2*2}=1;$$

$$s(G_1,G_1'')=\frac{1}{s(T_1,T_1'')}=\frac{1}{(2+3-2*2)+(2+3-2*2)}=0.5.$$

由计算结果可见,$s(G_1,G_1')>s(G_1,G_1'')$,所以近似本体子图 G_1'先进行分布式搜索.

2.4　MapReduce 分布式搜索

本文的 Map 阶段搜索本体子图中每个模式三元组匹配的实例三元组,Reduce 阶段则完成实例三元组的连接操作并且返回结果子图.Map 阶段:本体子图以模式三元组集合的形式表示,针对本体子图中的每个模式三元组,并行搜索 P_C_SO 和 P_C_OS 中与该模式三元组匹配的实例三元组,并将得到的结果传给 Reduce 阶段.Map 阶段的具体过程如算法 2.3 所示.

算法 2.3　Map 阶段

输入:key 为行号,value 为本体子图的标记与模式三元组的组合
输出:key 为本体子图的标记,value 为实例三元组集合
开始
instTripleSet ← ∅; //初始化实例三元组集合为空
flag← 分割输入中的 value,得到本体子图的标记
ontoTriple ← 分割输入中的 value,得到模式三元组
instTripleSet = serach(ontoTriple); //搜索该模式三元组

匹配的实例三元组
write(flag,instTripleSet);
结束

Reduce阶段:接收Map阶段传过来的实例三元组集合,根据本体子图中已有的连接关系,对实例三元组进行连接,得到结果子图并返回.Reduce阶段的具体过程如算法2.4所示.

算法2.4 Reduce阶段

输入:算法2.3的输出
输出:key为结果子图,value为任意值
开始
InstTripleSet ← 同一本体子图Map阶段输出的实例三元组集合
G_s ← ∅; //初始化结果子图为空
GsSet ← ∅; //初始化结果子图集合为空
For $t_i(s_i,p_i,o_i)$ ∈ InstTripleSet && i = 1,2,…,n //n为InstTripleSet集合大小
  G_s.add(ti(s_i,p_i,o_i));
  For $t_j(s_j,p_j,o_j)$ ∈ InstTripleSet && j=i+1 && j = 2,3,…,n
    If check(G_s)==true //如果G_s中包含了全部关键词
      If contain(G_sSet, G_s)==false //如果G_sSet中不存在G_s
write(G_s,"");//将G_s输出
        G_sSet.push(G_s); //将G_s加到G_sSet中
          G_s ← ∅;
          G_s.add($t_i(s_i,p_i,o_i)$);
      continue;
End if
    Else
For $t_k(s_k,p_k,o_k)$ ∈ Gs && k = 1,2,…,n //n为Gs中实例三元组条数
        If((s_k=s_j&&o_k≠o_j)||(s_k=o_j&&o_k≠s_j)||
          (o_k=s_j&&s_k≠o_j)||(o_k=o_j&&s_k≠s_j))
Gs.add($t_j(s_j,p_j,o_j)$);
          break;
End if
    End for
End if
  End for
End for
结束

2.5 算法正确性和复杂性分析

2.5.1 算法正确性分析

现有的RDF关键词搜索算法主要是将关键词直接在大规模的RDF数据图中进行搜索,然后再将关键词搜索的结果进行连接形成最终的结果子图.本文算法充分利用RDF本体信息,针对不同类型的关键词进行了正确的映射并且快速找到匹配的模式三元组.同时运用定义1.10中的连接方案连接模式三元组,将原本分散的关键词通过本体子图关联起来,在分布式搜索的时候同样能够精确返回与各关键词匹配的结果子图.虽然生成的本体子图可能存在多个,但本文借助语义评分函数使得语义内容和语义结构接近用户意图的本体子图优先进行分布式搜索.在所有本体子图都完成搜索但结果还未达到k条时,本文优先对语义评分高的本体子图进行扩展,生成对应的近似本体子图.为了确保结果的正确性,本文只进行一层扩展.同时针对生成的多个近似本体子图利用语义相似度函数来评分,评分高的优先进行分布式搜索.这样就能优先返回用户最想要的结果,从而保证算法的正确性.相比于现有的RDF关键词搜索算法,本文算法具有与之相当甚至更高的正确性.

2.5.2 算法复杂性分析

本文算法的时间复杂性主要包括构建本体子图、构建近似本体子图和MapReduce分布式搜索阶段的时间复杂性.假设关键词映射后对应的类或属性匹配的模式三元组条数为N,输入的搜索关键词个数为k,生成的本体子图个数为M,每个本体子图平均生成的近似本体子图个数为Q,每个模式三元组平均对应的$p_ic_j_SO$集合大小为p,关键词匹配的实例三元组条数为n.下面列出这三个阶段最坏情况下的时间复杂度.

(Ⅰ)构建本体子图阶段的复杂度:$O(N^2*k^2)$.

(Ⅱ)构建近似本体子图阶段的复杂度:$O(M*Q*k^3)$.

(Ⅲ)MapReduce分布式搜索阶段的复杂度:

Map阶段的时间复杂度:$O(p)$;

Reduce阶段的时间复杂度:$O(n^2*k^2)$.

综上可知,DKASR算法最坏情况下搜索一次的时间复杂度为$O(N^2*k^2)+O(M*Q*k^3)+O(p)+O(n^2*k^2)$.由于本文将数据进行预处理并分门别类地存储在Redis内存数据库的Set集合中,算法中匹配的数据都是小规模的数据;同时充分利用了RDF本体信息构建了本体子图和近似本体子图,避免了直接在大规模的RDF数据图上直接进

行搜索，从而降低了算法整体的时间复杂度.相比于 RDF 数据图规模的大小，N，kM，Q，p 和 n 都非常小，因此可以认为本文算法的最坏时间复杂性是常量级别.

3　实验与结果分析

3.1　实验设置

本文采用完全分布式 Hadoop 集群环境，由 8 台工作站构成，其中 1 台作为 HDFS 的名称节点兼 MapReduce 的主节点，7 台作为 HDFS 的数据节点兼 MapReduce 的从节点.本文实验所使用的工作站硬件环境为 Intel(R) Core(TM) i5-3570，CPU 3.40 GHz 双核双线程，内存 8 GB，磁盘 500 GB.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.同时，在 Hadoop 集群上部署了 Redis 集群，每台上都有 Redis 集群的一个主节点和从节点.

本文采用 LUBM (Lehigh University Benchmark)数据集对 DKASR 算法进行测试.利用 LUBM 数据生成器分别生成了 50、100、300、500、1 000所大学的数据，测试了本文算法在这 4 组数据集下的搜索效率和效果，并与现有的关键词搜索算法进行了对比.数据集的基本参数说明如表 2 所示.

表 2　数据集的基本参数说明

Tab.2　The basic parameters of data sets

数据集名称	三元组个数/万	属性数/个	类数/个	文件大小/GB
LUBM50	689	51	43	0.54
LUBM100	1 377	51	43	1.06
LUBM300	4 135	51	43	3.19
LUBM500	6 887	51	43	5.34
LUBM1000	13 778	51	43	10.71

本文实验所用的搜索示例如表 3 所示，总共有 6 个关键词集合 Q_1～Q_6，每个关键词集合分别包含 2～5 个关键词.将 Q_1～Q_6 分为两组，其中第一组 Q_1～Q_3 包含了实例、文本、类和属性 4 种类型的关键词，第二组 Q_4～Q_6 只包含部分类型的关键词.

本文将 DKASR 算法与 KREAG[9]、DKSRM[12] 在相同的实验环境下针对不同的数据集进行对比实验.为了便于进行对比实验，语义评分函数公式(1)中的参数 α 设置为 0.5；Top-k 的 k 值设置为 10；对比实验中的数据都是 10 次搜索的平均值.

表 3　搜索示例

Tab.3　Search samples

搜索	关键词集合
Q_1	Research12, teacherOf, Course, FullProfessor6, http://www.Department12.University0.edu/UndergraduateStudent95
Q_2	undergraduateDegreeFrom, ResearchAssistant, http://www.University982.edu, GraduateStudent121@Department7.University0.edu
Q_3	teacherOf, Research15, Course, http://www.Department8.University0.edu/AssociateProfessor9
Q_4	http://www.University0.edu, officeNumber, orgPublication
Q_5	Lecturer4, UndergraduateStudent, takesCourse
Q_6	researchInterest, http://www.Department10.University0.edu/AssistantProfessor6

3.2　实验结果

3.2.1　搜索响应时间的分析比较

为了测试本文算法的搜索效率，在不同的数据集下利用本文算法对 Q_1～Q_6 的搜索响应时间进行了测试，如表 4 所示.为了明确 2.1 节分布式存储方案的作用，实验测试了有按 2.1 节方案存储和没有按 2.1 节方案存储 Q_1～Q_6 构建本体子图的响应时间，如图 8 所示；同时测试了 LUBM1000 数据集下有按 2.1 节方案存储和没有按 2.1 节方案存储的搜索响应时间，如图 9 所示.

表 4　DKASR 算法不同数据集下的搜索响应时间(单位：s)

Tab.4　Response time of search process with DKASR algorithm on different data sets(Unit: s)

	Q_1	Q_2	Q_3	Q_4	Q_5	Q_6
LUBM50	16	20	17	14	25	11
LUBM100	17	22	18	16	30	13
LUBM300	20	26	21	17	38	15
LUBM500	23	30	23	19	46	18
LUBM1000	26	34	26	24	57	21

从表 4 可以看出，本文算法在不同的数据集下，Q_1～Q_6 的搜索响应时间保持在同一数量级上，没有明显的变化.其中 Q_6 由于包含实例型关键词并且只拥有两个关键词，在分布式搜索过程中匹配到的数据和连接的次数都最少，因此搜索效率最快；Q_5 中没有包含实例型关键词，这样会匹配到较多的数

据,在进行数据传输和连接的时候会耗费较多的时间.

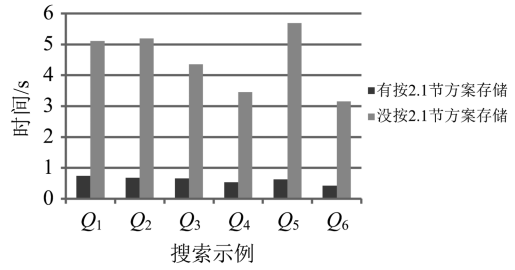

图 8 有没按 2.1 节方案存储的构建本体子图响应时间对比

Fig.8 The Comparison of Response Time for Constructing Ontology Sub-graphs between with Storage Scheme in 2.1 and without Storage Scheme in 2.1

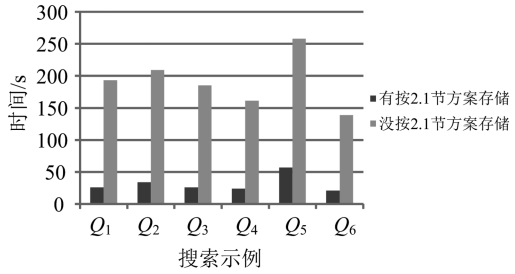

图 9 LUBM1000 数据集下有没按 2.1 节方案存储的搜索响应时间对比

Fig.9 The Comparison of Response Time on LUBM1000 between with Storage Scheme in 2.1 and without Storage Scheme in 2.1

从图 8 和图 9 可看出,不论是第一组还是第二组的搜索示例,有按 2.1 节方案存储的方法在效率上都有很大的提升.例如对于 Q_1,按 2.1 节方案存储的方法在构建本体子图上比没有按 2.1 节方案存储的方法效率提高了近 6 倍,在分布式搜索效率上提高了 6 倍多;对于 Q_4,在构建本体子图上效率提高了 5 倍多,在分布式搜索效率上提高了近 6 倍.由于将本体信息和实例数据先进行预处理,再把预处理后的数据按 2.1 节分布式存储方案存储在 Redis 集群中,在构建本体子图以及分布式搜索时可以大大缩小搜索的范围,快速生成本体子图和结果子图.

为了体现本文算法在搜索效率上的优势,在 LUBM1000 数据集下测试了 DKASR 算法以及 KREAG、DKSRM 对 $Q_1 \sim Q_6$ 的搜索响应时间,如图 10 所示.同时图 11(a)和图 11(b)分别显示了在不同数据集下三者算法对第一组和第二组搜索的平均响应时间.

从图 10 和图 11 可以看出,本文算法的搜索效率远高于对比算法.与 KREAG 以及 DKSRM 相比,

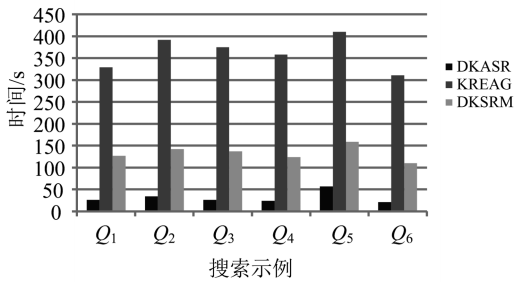

图 10 LUBM1000 数据集下各算法的搜索响应时间对比

Fig.10 The comparison of search response time among algorithms on LUBM1000

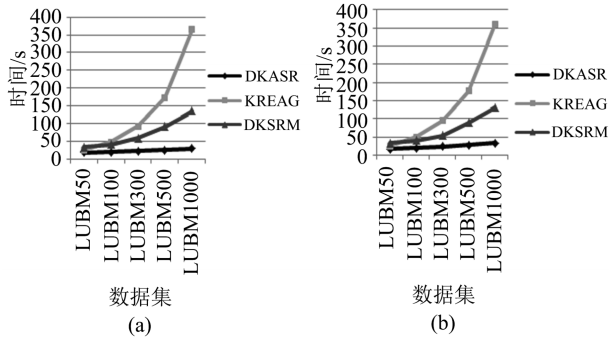

图 11 不同数据集各算法两组搜索的平均响应时间对比

Fig.11 The comparison of average response time between two groups of search on different data sets

本文算法在 LUBM1000 数据集下 $Q_1 \sim Q_6$ 的搜索响应时间均远远少于两者,同时第一组和第二组的平均响应时间随数据集的增大变化不大,几乎保持在同一水平线上,而另外两者算法都有不同程度的增加.如在 LUBM1000 数据集下,DKASR 算法第一组搜索的平均搜索效率相对于 KREAG 提高了 11 倍多,比 DKSRM 快了近 4 倍;DKASR 算法第二组搜索的平搜索效率也比 KREAG 快了 9 倍多,比 DKSRM 快了近 3 倍.由于本文算法在分布式的环境下利用了 Redis 内存数据库和 MapReduce 计算框架,将数据进行预处理并分门别类地存储,在搜索时可以快速定位到与关键词匹配的数据,同时将大规模 RDF 数据图上的连接问题转换为小规模 RDF 本体上的连接问题,大大地降低了搜索的计算复杂度;而 KREAG 算法由于要构建并维护关键词索引和最短通路索引,在数据量小的情况下还可以较快地搜索出结果,但是随着数据集规模的增大,构建并维护索引的耗时随之急剧增加,最终导致搜索效率的快速下降;DKSRM 算法利用为 RDF 数据图构建路径索引来完成搜索,随着数据量的增大路径索引的规模也会变得很大,并且大量的路径索引是分布式地存储在集群的不同节点上,在进行连接操作返

回结果的时候网络传输开销很大，造成搜索响应时间不理想.

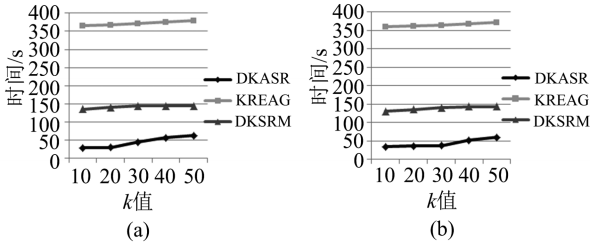

图 12　Top-k 的 k 值影响

Fig.12　The effect of the value of k

图 12 显示了 Top-k 的 k 值的变化对搜索响应时间的影响.其中图 12(a)是不同 k 值下第一组搜索在 LUBM1000 数据集中的平均响应时间，图 12(b)是不同 k 值下第二组搜索在 LUBM1000 数据集中的平均响应时间.图 12(a)和图 12(b)分别测试了 3 种算法.从图 12 可以看出，不论是第一组还是第二组搜索，k 值的增大使得 DKASR 算法的平均响应时间都随之增加.这是因为当用户要求返回的结果越多，搜索的结果可能没有达到 Top-k，这时就要对本体子图进行扩展，构建近似本体子图，然后再进行分布式搜索，直到返回 Top-k 结果，由于 RDF 本体数据规模很小并且存储在 Redis 集群中，构建近似本体子图的效率很高，如图 13 所示，$Q_1 \sim Q_6$ 对应的近似本体子图可以在 1.4 s 内构建完成，因此在进行近似搜索的时候也能快速返回搜索结果；KREAG 算法由于没有进行二度的搜索，可以一次性完成近似搜索，所以随着 k 值的增大平均响应时间变化不大；而 DKSRM 算法也是一次性完成搜索，但是该算法不支持近似搜索，只能进行精确搜索，因此当 k 达到一定数值的时候，返回的结果个数不会发生变化，对应的平均响应时间也保持不变.

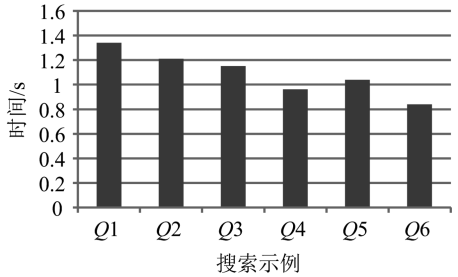

图 13　构建近似本体子图的响应时间

Fig.13　The response time of constructing approximate ontology sub-graphs

三种算法在不同 k 值下返回结果数的对比如图 14 所示.其中图 14(a)和图 14(b)分别是不同 k 值下第一组搜索和第二组搜索在 LUBM1000 数据集中

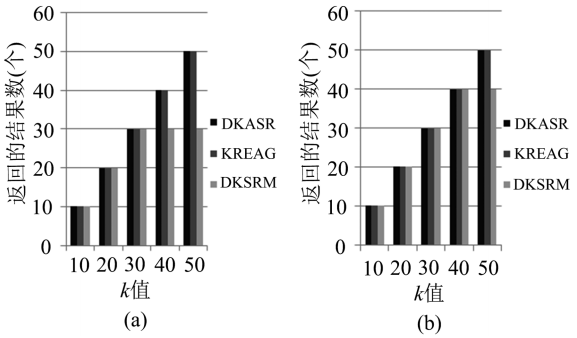

图 14　不同 k 值下各算法返回结果数对比

Fig.14　The comparison of the results among algorithms under different values of k

的平均结果数.由图 14 可以看出，DKASR 和 KREAG 算法在不同 k 值下都能返回对应的 Top-k 结果，而 DKSRM 算法随着 k 值的增大，返回的结果没有达到 Top-k.原因是 DKASR 和 KREAG 算法都支持近似搜索，在精确结果不能达到 Top-k 时，近似搜索直到返回 Top-k 结果为止，DKSRM 算法只能返回精确结果，而且精确结果的个数是确定的.

3.2.2　搜索效果的分析比较

为了验证本文算法的搜索效果，主要从查准率和查全率两个方面进行评价.查准率(Precision)用来衡量 Top-k 搜索结果的准确率，是搜索结果中正确的实例三元组条数与搜索结果中实例三元组总条数的比值.查全率(Recall)是搜索结果中正确的实例三元组条数与数据集中相关的实例三元组总条数的比值.针对不同的数据集，为每个搜索示例创建一个集合 Set，用来存储该搜索示例相关的全部搜索结果.查准率和查全率的定义如下所示.

$$\text{查准率}=\frac{\text{搜索结果中正确的实例三元组条数}}{\text{搜索结果中实例三元组总条数}},$$

$$\text{查全率}=\frac{\text{搜索结果中正确的实例三元组条数}}{\text{Set 中实例三元组条数}}.$$

图 15 给出了 3 种算法在 LUBM1000 数据集下查准率和查全率的对比，由图 15 可以看出，本文算法的查准率和查全率均高于另外两种算法.从图 15(a)可以看出，本文算法第一组搜索的查准率相比于第二组搜索偏低，原因是第一组搜索中包含了全部类型的关键词，搜索结果比较少，除了返回正确的搜索结果外，在进行近似搜索的时候会返回其他与关键词有间接关系的结果；第二组搜索的查准率基本达到 100%.从图 15(b)可以看出，本文算法第一组搜索的查全率几乎接近 100%，第二组的查全率比第一组的偏低，这是因为第二组搜索没能返回全部正确的实例三元组就已经满足 Top-k 结果，遗漏了部分正确的实例三元组，最终造成查全率偏低.

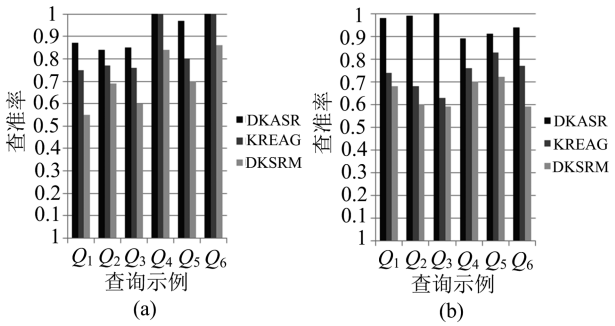

图15 LUBM1000数据集下各算法查准率和查全率对比

Fig.15 The comparison of precision and recall among algorithms on LUBM1000

4 结论

本文提出的DKASR算法通过利用RDF本体信息构建本体子图,将数据进行预处理并分布式存储在Redis集群中,运用Hadoop平台实现分布式RDF关键词近似搜索.实验表明,DKASR算法能够快速有效地搜索出所有关键词(包括实例、文本、类和属性)匹配的数据,同时可以近似搜索返回用户感兴趣的结果,无论在搜索响应时间和搜索效果方面都明显优于KREAG算法和DKSRM算法.

参考文献(References)

[1] 刘博. 基于关键词的RDF数据图查询模型研究[D]. 郑州:郑州大学, 2015.

[2] 姜旭,张波.采用RDF的查询扩展研究[J]. 计算机应用与软件, 2011, 28(12):210-212.
JIANG Xu, ZHANG Bo. On query extension using RDF[J]. Computer Applications and Software, 2011, 28(12):210-212.

[3] 董书睐,汪璟玢. HMSST:一种高效的SPARQL查询优化算法[J]. 计算机科学, 2014, S2: 323-326, 336.
DONG Shujian, WANG Jingbin. HMSST: An efficient algorithm for SPARQL query[J]. Computer Science, 2014, S2: 323-326, 336.

[4] 杜方,陈跃国,杜小勇. RDF数据查询处理技术综述[J]. 软件学报, 2013, 24(6): 1222-1242.
DU Fang, CHEN Yueguo, DU Xiaoyong. Survey of RDF query processing techniques[J]. Journal of Software, 2013, 24(6): 1222-1242.

[5] TRAN T, WANG H, RUDOLPH S, et al. Top-*k* exploration of query candidates for efficient keyword search on graph-shaped (RDF) data[C]// Proceedings of the 25th International Conference on Data Engineering. Shanghai, China: IEEE, 2009: 405-416.

[6] ZENZ G, ZHOU X, MINACK E, et al. From keywords to semantic queries-Incremental query construction on the semantic Web[J]. Journal of Web Semantics, 2009, 7(3): 166-176.

[7] GKIRTZOU K, PAPASTEFANATOS G, DALAMAGAS T. RDF keyword search based on keywords-to-sparql translation[C]// Proceedings of the First International Workshop on Novel Web Search Interfaces and Systems. Melbourne, Australia: ACM, 2015: 3-5.

[8] LE W C, LI F F, KEMENTSIETSIDIS A, et al. Scalable keyword search on large RDF data[J]. IEEE Transactions on Knowledge and Data Engineering, 2014, 26(11): 2774-2788.

[9] 李慧颖,瞿裕忠. KREAG:基于实体三元组关联图的RDF数据关键词查询方法[J]. 计算机学报, 2011, 34(5): 825-835.
LI Huiying, QU Yuzhong. KREAG: Keyword query approach over RDF data based on entity-triple association graph[J]. Chinese Journal of Computers, 2011, 34(5): 825-835.

[10] ELBASSUONI S. Effective searching of RDF knowledge bases[D]. Munchen: Max-Planck-Institut Für Informatik, 2011.

[11] KAOUDI Z, MANOLESCU I. RDF in the clouds: A survey[J]. VLDB Journal, 2015, 24(1): 67-91.

[12] DE VIRGILIO R, MACCIONI A. Distributed Keyword Search Over RDF Via Mapreduce[M]// The Semantic Web: Trends and Challenges. Springer, 2014: 208-223.

[13] 周文健. 基于语义距离的RDF本体查询方法研究[D]. 沈阳: 东北大学, 2011.

[14] 章登义,吴文李,欧阳黜霏. 基于语义度量的RDF图近似查询[J].电子学报, 2015, 43(7): 1320-1328.
ZHANG Dengyi, WU Wenli, OUYANG Chufei. Approximating query with semantic-based measure on RDF graphs[J]. Acta Electronica Sinica, 2015, 43(7): 1320-1328.