计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2017,26(9):69−74 [doi: 10.15888/j.cnki.csa.005923]
©中国科学院软件研究所版权所有.
E-mail: csa@iscas.ac.cn
http://www.c-s-a.org.cn
Tel: +86-10-62661041

基于 Spark 和 Redis 的大规模 RDF 数据查询系统①

阳　杰, 王木涵, 徐九韵

(中国石油大学(华东) 计算机与通信工程学院, 青岛 266580)

摘　要: 随着语义 Web 技术的不断发展, RDF 数据量增长迅速, 单机 RDF 查询系统已经难以满足现实需要, 研究和构建分布式 RDF 查询系统已经成为学术界与工业界的研究热点之一. 现有的 RDF 查询系统主要是基于 Hadoop 或通用分布式技术. 前者磁盘 I/O 太高; 后者则可扩展性较差. 且两种系统在基本图模式查询时, 效率都较低. 针对上述问题, 本文设计了基于 Spark 和 Redis 的分布式系统架构, 并改进了查询计划生成算法, 最后实现了原型系统 RDF-SR. 该系统使用 Spark 减少了磁盘 I/O, 借助 Redis 提高了数据映射速率, 利用改进的算法减少了数据混洗次数. 实验表明, 相比于现有的其他系统, RDF-SR 既保持了较高可扩展性, 又在基本图模式查询时, 具有更高的性能.

关键词: 语义 Web; 大规模 RDF; Spark; Redis

引用格式: 阳杰,王木涵,徐九韵.基于 Spark 和 Redis 的大规模 RDF 数据查询系统.计算机系统应用,2017,26(9):69–74. http://www.c-s-a.org.cn/1003-3254/5923.html

Big RDF Graph Query System Based on Spark and Redis

YANG Jie, WANG Mu-Han, XU Jiu-Yun

(School of Computer & Communication Engineering, China University of Petroleum, Qingdao 266580, China)

Abstract: With the development of semantic web technology, RDF data grow rapidly. The single node RDF query system cannot meet the practical needs. Building distributed RDF query system has become one of the hotspots in the academia and industry. The existing RDF query system is based on Hadoop and general distributed technology. The disk I/O of the former is too high and the latter is less scalable. Besides, the two systems perform poorly in the basic pattern matching mode. In order to solve these problems, we design a distributed system architecture based on Spark and Redis, and improve the query plan generation algorithm. We call the prototype system RDF-SR. This system reduces the disk I/O by Spark, improves the data mapping rate by Redis and reduces the data shuffling process with improved algorithms. Our evaluation shows that RDF-SR performs better in the basic pattern matching mode compared with other systems.

Key words: semantic Web; big RDF graph; Spark; Redis

随着语义 Web 的不断发展, 基于语义网的应用也越来越多. RDF(资源描述框架)已被广泛地应用于各个领域的本体建模和推理中, 例如电子政务、生物医药、地理空间等领域. 以链接开放数据(Linked Open Data)项目为例, 该项目一共包含了约 520 亿个三元组, 这是典型的大规模 RDF 数据[1]. 传统单机 RDF 数据查询系统在数据集较小的情况下, 能够呈现出优异的性能, 但是随着 RDF 数据规模的增加, 其扩展能力和查询性能都会出现瓶颈, 海量 RDF 数据的存储和查询面临着巨大挑战[2]. 因此, 研究和构建分布式的 RDF 数据查询系统, 实现数据的快速查询具有十分重要的意义.

现有的分布式 RDF 查询系统可以分为两类: 基于通用分布式技术的 RDF 查询系统和基于 Hadoop 的 RDF 查询系统. 基于通用分布式技术的系统有 RDFPeers[3]、

① 收稿时间: 2016-12-13; 采用时间: 2017-01-09

© 中国科学院软件研究所 http://www.c-s-a.org.cn

YARS2[4]等. RDFPeers 建立在 P2P 系统 MAAN(Multi Attribute Addressable Network)上, 它把 RDF 数据的三个副本存储在不同节点上, 并建立了索引. 该系统在三元组匹配时能取得较好的性能, 但在基本图模式查询时查询代价过高. YARS2 在每个节点上用关系数据库存储 RDF 数据, 该系统在三元组匹配时性能较高, 但是在基本图模式下性能较低, 并且较难扩展. 基于 Hadoop 的系统主要有文献[5]、PigSPARQL[6]、文献[2]、Sempala[7]、HadoopRDF[8]等. 文献[5]基于 Hadoop 和 RDF-3X 实现, 其中 RDF-3X 是一种单机 RDF 数据库. 该文献还给出了一种图分区算法, 将数据分散存储到每个节点的 RDF-3X 数据库中, 并使联系紧密的三元组存储在同一节点上, 从而减少基本图模式查询时的网络 I/O. 由于该系统依赖于单机 RDF 数据库, 所以降低了存储系统的可扩展性. PigSPARQL采用平面文件组织 RDF 数据, 它将 Sparql 语句转换成 Pig Latin 实现并行查询, Pig Latin 会被解析成一系列的 MapReduce 作业来执行. 文献[2]采用了 Hadoop 和 HBase 构建 RDF 查询系统, 它利用 HBase 自身的索引机制提升三元组匹配速率, 并使用贪心算法生成查询计划. 该系统在处理三元组匹配和简单图匹配时都表现出了不错的性能, 系统的可扩展性也很好. 但是在较为复杂的基本图模式查询时, 性能较低. 主要原因是 Hadoop 在迭代执行任务时, 需要多次耗时的读盘和写盘操作, 严重影响了系统的查询性能. 除此之外, 该系统使用了贪心算法生成树状查询计划, 该算法每次迭代选择连接结果集最多的变量进行连接, 目的在于利用多路连接方法, 减少连接操作次数[2]. 该算法并有没有考虑查询中间结果集的规模信息, 然而如果合理利用这些信息, 调整连接的顺序, 可以进一步提升查询的效率.

基于此, 本文设计了一种基于 Spark 和 Redis 的分布式架构. Spark 是由加州大学伯克利分校的 AMP 实验室所开源的基于内存的通用并行框架[9]. 它将作业中间输出的结果保留在内存中, 不需要读写磁盘. 因此, Spark 可以有效减少迭代型作业的磁盘 I/O. Redis 是一个既可基于内存亦可持久化的键值型分布式数据库[10]. 它既保证了高效存取, 又保证了数据的完整性. 同时, Redis 非常容易扩展. 所以 Redis 可以显著提升并行读取数据映射表的速率. 本文还改进了查询计划生成算法, 该算法基于数据连接中间结果集的规模信息调整了结果集连接顺序, 并利用分布式平台广播数据的特性[11]减少了数据混洗操作次数, 进而提升查询速率.

本文的贡献点主要有如下两点:

(1) 设计出一种基于 Spark 和 Redis 的分布式架构, 并给出了原型系统 RDF-SR.

(2) 利用查询中间结果集的规模信息, 改进了查询计划生成算法.

本文安排如下: 第一部分是引言, 说明了本文的研究目的及意义、主要贡献点及组织结构. 第二部分是 RDF 查询基本算法, 介绍了 RDF 数据形式以及查询的基本算法. 第三部分是 RDF-SR 系统设计, 介绍了 RDF-SR 系统的特点和架构. 第四部分是查询计划生成算法, 描述了本文改进查询计划生成算法的原理和步骤. 第五部分是实验结果与分析, 主要测试了 RDF-SR 系统在基本图模式下的查询性能, 并与基于 Hadoop 的系统作了对比. 第六部分是小结, 总结本文的工作内容和成果, 指出了今后进一步进行研究的展望和设想.

1 RDF 查询基本算法

RDF 是以三元组(主语, 谓词, 宾语)的形式描述资源的. 一组三元组的集合被成为 RDF 图, RDF 图可以通过结点和弧线表示, 弧的两端是主语和宾语, 弧的方向总是由主语指向宾语. RDF 数据的基本图模式查询条件是一个带变量的子图, 因此 RDF 上的查询处理可以被转换为大图上的子图匹配问题[12]. 图 1 是 RDF 图的实例, 图 2 是查询子图的实例.

子图匹配 RDF 大图的基本算法可以分为如下几个步骤(假设子图有 k 个模式):

(1) 将 k 个模式分别与大图匹配, 得到 k 个匹配结果集(每个结果集只保留变量对应的列).

(2) 将 k 个结果集放入集合 S, 如果集合还存在两个结果集可以连接, 则重复执行步骤 3.

(3) 找出具有公共变量的两个结果集, 把它们从 S 中删除, 并将它们基于公共变量进行连接操作, 将连接产生的结果集加入 S.

2 RDF-SR 系统设计

本系统的主要特点包含了以下三个方面:

(1) 系统能够支持 Sparql 查询中的三元组模式查询和基本图模式查询.

(2) 系统使用 Redis 存储 RDF 大图数据的 ID 映射表, 以及大图数据的统计信息.

(3) 系统使用 Spark 集群作为计算引擎.

© 中国科学院软件研究所 http://www.c-s-a.org.cn

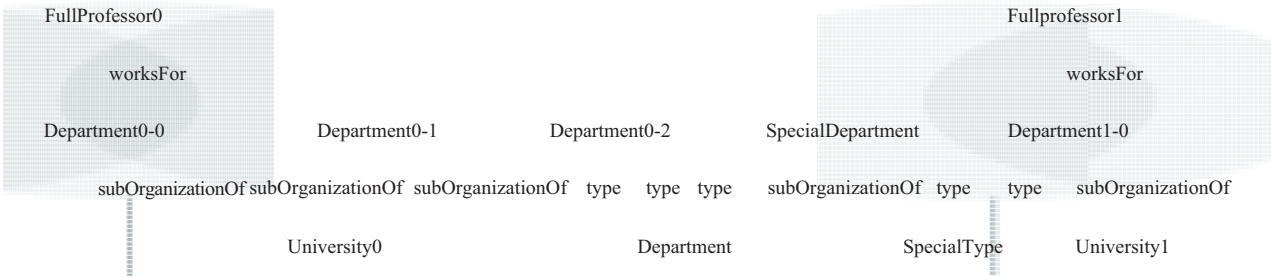


图 1 RDF 图

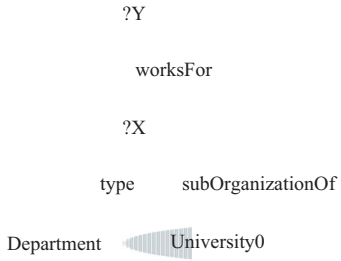


图 2 基本图模式查询条件

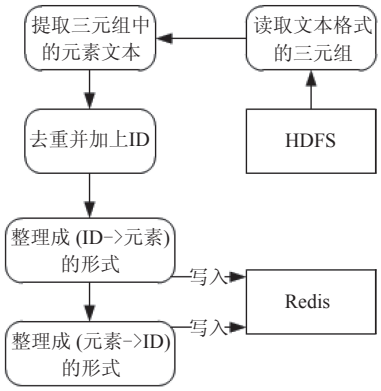
RDF-SR 使用 Sparql 作为用户操作接口, Sparql 是为 RDF 开发的一种查询语言和数据获取协议, 用户可以使用这种类似 SQL 的查询语言进行 RDF 数据的查询操作[13]. 整个系统可以分为两个部分, 数据预处理子系统和数据查询子系统.

数据预处理子系统又可以分为两个部分: 第一部分的功能是生成 ID 映射表, 并把 ID 映射表写入 Redis; 第二部分的功能是把三元组映射为 ID 形式, 并统计出元素在不同列出现的次数, 最后把统计信息写入到 Redis. 具体流程如图 3 所示.

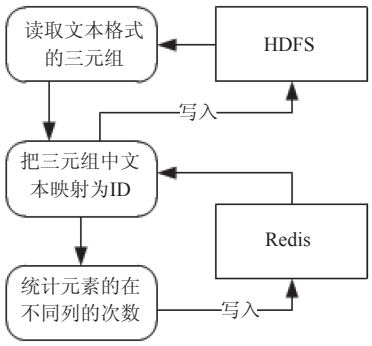
数据查询子系统主要功能是: 从用户那里接受 Sparql 语句, 然后用 Jena ARQ[14]解析出基本图模式的三元组; 把三元组内的文本映射为 ID, 然后读取大图统计信息, 最后用改进的算法生成查询计划; 把任务提交到 Spark 集群上运行, 然后选出用户关心的列, 并把 ID 映射为文本, 返回给用户. 具体如图 4 所示.

3 查询计划生成算法

查询计划生成算法改进的主要原理是通过近似计算结果集的大小, 优先选取数据量小的结果集当作连接的左表, 从而利用分布式平台广播数据的特性[11], 避免数据混洗操作, 提高连接的速度, 进而提高查询的整体速度. 采用近似计算而不采用实时计算的原因是, 在分布式环境中, 在计算阶段实时获取结果集的大小, 网络 I/O 开销较大, 反而会降低查询速度.



(a) 数据处理1



(b) 数据处理2

图 3 数据预处理子系统

近似计算连接结果集的大小, 是数据库领域的一个重要问题, 已有较为丰富的研究成果. 利用选择率来预估连接结果集的大小程度, 是一种较为有效的途径[17]. 为了充分利用在 RDF 数据预处理阶段得到的统计信息, 本文定义了一种选择率计算方法. 假设 RDF 图 G 中有 N 个三元组. 列号 $k \in \{0,1,2\}$, 其中 0 是主语列,

© 中国科学院软件研究所 http://www.c-s-a.org.cn

1 是谓语列, 2 是宾语列. e_k 是位于位置 k 的元素, 模式 TP 也就可以表示为(e_0, e_1, e_2). 元素选择率的计算公式为

$$sel(e_k)=\begin{cases}\frac{count(e_k)}{N}, & e_k\text{不是变量}\\ 1, & e_k\text{是变量}\end{cases} \tag{1}$$

其中 $count(e_k)$是指元素 e_k 在图 G 中出现的次数. 模式 TP 选择率的计算公式为:

$$sel(TP)=\prod_{k=0}^{2} sel(e_k) \tag{2}$$

连接结果集大小上界的计算公式为:

$$S_{TP}=match(TP) \tag{3}$$

$$S_{join}=S_{left}\times S_{right} \tag{4}$$

其中 $match(TP)$是指模式 TP 匹配图 G 后的结果集大小. 连接结果选择率的计算公式为:

$$sel_{join}=\frac{S_{join}}{N^2}\times sel\left(TP_{left}\right) \tag{5}$$

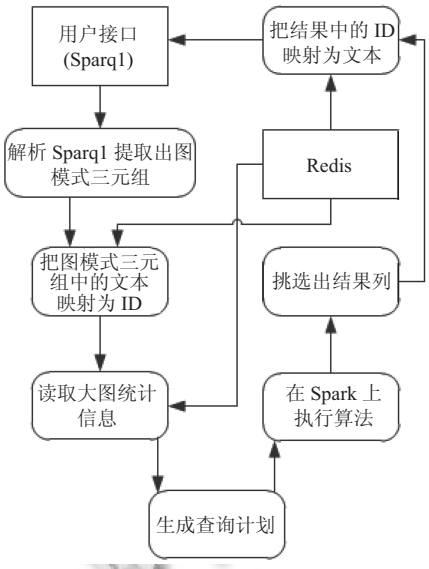


图 4 数据查询子系统

该算法可以分成如下几个步骤(假设有 k 个模式):

(1) 预处理大图数据, 分别遍历大图的主语列、谓词列、宾语列的所有元素, 统计不同元素在不同列出现的次数.

(2) 将每个模式与大图匹配得到 k 个结果集.

(3) 从步骤 1 统计的数据中, 得到每个结果集的大小, 把所有结果集加入集合 S, 一直循环执行步骤 4, 直到集合 S 中不存在任何两个结果集拥有公共变量.

(4) 选出最小的两个可连接的结果集, 从 S 中删除, 同时小表在左大表在右进行连接, 连接后产生结果集 A, 近似计算 A 的大小, 并加入 S.

本文使用一个的简单例子说明该算法的有效性. 假设例子中的 3 个模式匹配大图数据后得到 3 个结果集, 如表 1 所示.

表 1 匹配结果集

编号	A	B	C
结果集大小	10	100	10000

对于这 3 个结果集, 使用贪心算法和本文提出的算法分别生成查询计划并进行连接操作, 具体如图 5 所示.

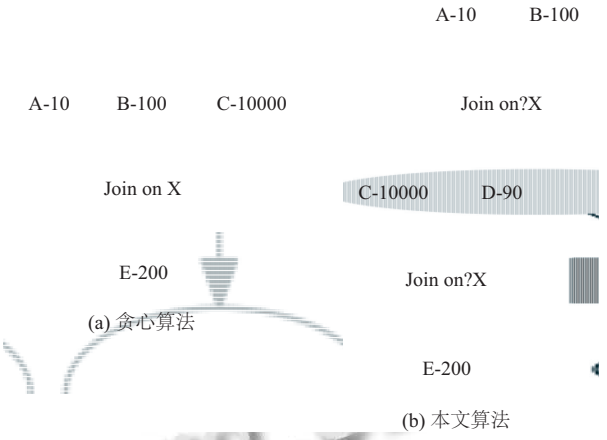


图 5 两种方法生成的查询计划

图 5 中, 左图是使用贪心算法生成的查询计划, 使用了多路连接, 右图是使用本文的算法生成的查询计划, 图中的数字代表结果集的大小. 分析上面的例子可以发现: 使用贪心算法生成的查询计划, 虽然只需要一次多路连接就能得到最终结果, 但是却由于较大结果集 C 的影响, 触发了数据混洗操作, 产生大量网络 I/O, 导致查询效率较低; 使用本文提出的算法生成的查询计划, 虽然有两次连接, 但是可以将较小结果集当作左表, 利用分布式平台广播数据的特性, 避免了数据混洗操作, 反而使得整体查询效率提高.

4 实验结果与分析

4.1 测试环境和数据来源

本文基于青云大数据平台[15]搭建了具有 5 个节点

© 中国科学院软件研究所 http://www.c-s-a.org.cn

的 Spark 集群和一个 Redis 节点, 节点配置情况如表 2 所示.

表 2 节点配置情况

节点类型	CPU	内存	硬盘	操作系统	数量
Spark-Master	2*2.5 GHz	4 G	10 G	CentOS	1
Spark-Worker	4*2.5 GHz	16 G	10 G	CentOS	4
Redis	4*2.5 GHz	16 G	10 G	CentOS	1

为了测试系统的性能, 需要做一些基准测试. 本文采用当前广泛使用的基准测试集 LUBM(Lehigh University Benchmark, LUBM)[16]. 本文采用 LUBM 生成器, 生成了三组大学的语义数据集, 数据集规模如表 3 所示.

表 3 数据集规模

大学(所)	100	200	400
三元组	14235689	26695778	51992568
大小(G)	1.09	2.13	4.07

4.2 基本图模式查询性能测试

为了测试系统在基本图模式下的查询性能, 本文选用了 Q1, Q2, Q3 这三条 Sparql 语句进行了测试, 并与基于 Hadoop 的系统(文献[2])做了对比. Q1,、Q2、Q3 的细节如表 4 所示.

表 4 Sparql 语句细节

Sparql	变量	模式
Q1	?X ?Y	?Y rdf:type ub:Department ?X ub:worksFor ?Y ?Y ub:subOrganizationOf <http://www.University0.edu >
Q2	?X ?Y ?Z	?X rdf:type ub:GraduateStudent ?Y rdf:type ub:University ?Z rdf:type ub:Department ?X ub:memberOf ?Z ?Z ub:subOrganizationOf ?Y
Q3	?X ?Y ?Z	?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 6 和图 7 展示了实验的结果. 从图 6 得出的结论是 RDF-SR 在 3 个基本图模式查询时的性能, 比基于 Hadoop 的系统快 3-6 倍. 从图 7 中可以看出, 随着数据规模的增加, RDF-SR 查询时间的增长速度比基于 Hadoop 的系统更加缓慢. 这些主要得益于本系统的 3 个特点: 使用了基于内存的分布式计算引擎 Spark 进行运算, 显著减少了磁盘 I/O; 使用了基于内存的数据库 Redis, 提升了读取 ID 映射表和大图数据统计信息的速率; 使用了改进了的查询计划生成算法, 避免了大量数据混洗操作.

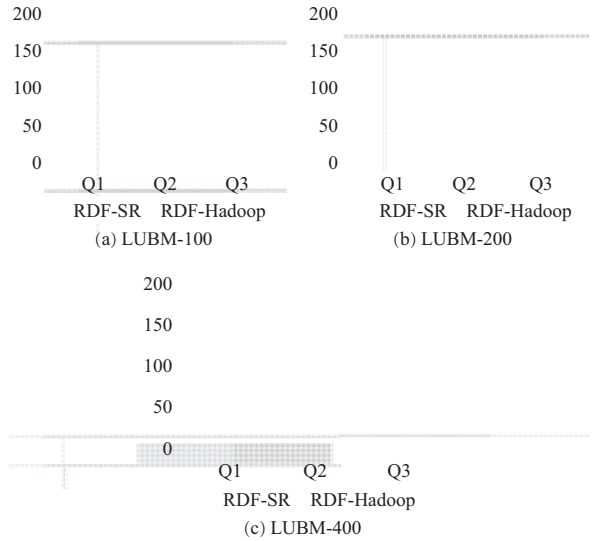


图 6 三组数据集上的查询时间对比

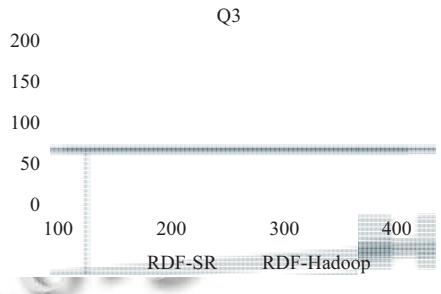


图 7 不同规模数据集下, 查询时间的走势

5 小结

本文的主要工作是针对现有分布式系统在基本图模式下查询大规模 RDF 效率较低的问题, 设计了一种基于 Spark 和 Redis 的分布式系统架构; 利用查询中间结果集的规模信息, 改进了查询计划生成算法; 实现了原型系统 RDF-SR. 实验证明 RDF-SR 相比于基于 Hadoop 的 RDF 查询系统, 在基本图模式下的查询效率显著提高.

本文的不足之处在于, RDF-SR 并没有使用索引结构来提升三元组匹配的速率, 同时也没有考虑 RDF 数据动态变化的场景. 所以未来的工作在于加入索引机制进一步提升查询效率, 并考虑在 RDF 数据动态变化的场景下, 如何保证数据的查询效率.

© 中国科学院软件研究所 http://www.c-s-a.org.cn

参考文献

1 http://lod-cloud.net/versions/2011-09-19/lod-cloud.html.

2 宋纪成. 海量 RDF 数据存储与查询技术的研究与实现[硕士学位论文]. 北京: 北京工业大学, 2013.

3 Cai M, Frank M. RDFPeers: A scalable distributed RDF repository based on a structured peer-to-peer network. Proc. of the 13th International Conference on World Wide Web. New York, NY, USA. 2004. 650–657.

4 Harth A, Umbrich J, Hogan A, *et al*. YARS2: A federated repository for querying graph structured data from the web. The Semantic Web. Lecture Notes in Computer Science. Berlin Heidelberg. 2007. 211–224.

5 Huang JW, Abadi DJ, Ren K. Scalable SPARQL querying of large RDF graphs. Proc. of the VLDB Endowment, 2011, 4(11): 1123–1134.

6 Schätzle A, Przyjaciel-Zablocki M, Hornung T, *et al*. PigSPARQL: A SPARQL query processing baseline for big data. Proc. of the 2013 International Conference on Posters & Demonstrations Track. Sydney, Australia. 2013. 241–244.

7 Schätzle A, Przyjaciel-Zablocki M, Neu A, *et al*. Sempala: Interactive SPARQL query processing on hadoop. The Semantic Web-ISWC 2014. Cham. 2014. 164–179.

8 Du JH, Wang HF, Ni Y, *et al*. HadoopRDF: A scalable semantic data analytical engine. Intelligent Computing Theories and Applications. Berlin Heidelberg. 2012. 633–641.

9 https://spark.apache.org/docs/latest/index.html.

10 Carlson JL. Redis in Action. Greenwich, CT, USA: Manning Publications, 2013.

11 Apache Spark. Spark programming guide. https://spark.apache.org/docs/latest/programming-guide.html#broadcast-variables.

12 杜方, 陈跃国, 杜小勇. RDF 数据查询处理技术综述. 软件学报, 2013, 24(6): 1222–1242.

13 Prud'hommeaux E, Seaborne A. SPARQL query language for RDF. W3C Recommendation, 2008: 15.

14 ARQ-A SPARQL processor for jena. http://jena.apache.org/documentation/query/index.html.

15 https://www.qingcloud.com.

16 LUBMft -the RDF fulltext benchmark. http://www.l3s.de/~minack/rdf-fulltext-benchmark.

17 Stocker M, Seaborne A, Bernstein A, *et al*. SPARQL basic graph pattern optimization using selectivity estimation. Proc. of the 17th International Conference on World Wide Web. Beijing, China. 2008. 595–604.

© 中国科学院软件研究所 http://www.c-s-a.org.cn