4.2 推迟笛卡尔积操作

在本文的基本算法中,初始星形匹配阶段在匹配星形 $T=(r,L)$,其中,$L=\{(p_1,l_1),\ldots,(p_t,l_t)\}$,$v\in V$ 时,首先根据叶子节点的标签在各个主语 v 的邻接链表 $N(v)$上得到各个叶子节点 l_i 的候选集合 $S(l_i)$,然后通过叶子节点候选集合的笛卡尔积$\{v\}\times S(l_1)\times\ldots\times S(l_t)$,得到星形 T 的所有匹配结果.但是大部分星形匹配结果无法形成最终的查询答案,从而导致在星形匹配阶段做了很多无用的笛卡尔积操作,在 Reduce 阶段连接代价过高.

本文的优化算法在星形匹配阶段仅仅得到各叶子节点的候选集合即可.在 Map 函数中计算局部查询图 P_{t-1} 和星形 T_t 的顶点交集 $V(P_{t-1})\cap V(T_t)$中的各个顶点的候选集,并通过这些顶点候选集的笛卡尔积得到 Reduce 函数连接的键.MapReduce 操作结束后,通过剩余顶点候选集的笛卡尔积得到最终的查询结果.将各个星形匹配过程中叶子节点候选集的笛卡尔积推迟到 MapReduce 操作结束的时候,做过滤部分无用的计算.

例 4:匹配查询图 Q_1 时,在前 3 轮的 Map 函数中,仅需要得到每个星形叶子节点的候选集.如图 6 所示,第 2 轮 MapReduce 中,通过 $V(P_1)\cap V(T_3)=\{?y,?g\}$中顶点的候选集合的笛卡尔积,得到星形 T_3 中$\{?y,?g\}$的匹配结果$\{?y\rightarrow House,?g\rightarrow HorrorFiction\}$,$\{?y\rightarrow TheStand,?g\rightarrow GothicFiction\}$ 和 $\{?y\rightarrow LordoftheFlies,?g\rightarrow Allegory\}$;$P_1$ 中$\{?y,?g\}$的匹配结果$\{?y\rightarrow TheStand,?g\rightarrow GothicFiction\}$和$\{?y\rightarrow TheShining,?g\rightarrow GothicFiction\}$.将这些匹配结果作为键,在 Reduce 过程中进行连接操作,可以观察到,仅当$\{?y\rightarrow TheStand,?g\rightarrow GothicFiction\}$可以得到局部查询图 P_2 的匹配结果.类似地,在第 3 轮 MapReduce 中,通过 $V(P_2)\cap V(T_1)=\{?x\}$中顶点候选集合的笛卡尔积,得到星形 T_1 和局部查询图 P_2 中$\{?x\}$的匹配结果集合,以这些匹配结果作为键值,在 Reduce 中进行连接操作.最后,通过剩余顶点集合$\{?a,?c,?w_2,?p,?w_1\}$中各顶点的候选集的笛卡尔积,得到查询 Q_1 的最终匹配结果$\{?y\rightarrow TheShining,?g\rightarrow GothicFiction,?x\rightarrow KingS,?a\rightarrow NationalBookA,?c\rightarrow Portland,?w_2\rightarrow GoldingW,?p\rightarrow Doubleday,?w_2\rightarrow DekkerT\}$.在基本算法 SDec 中,Map 函数在邻接链表 $N(GibsonW)$上匹配星形 T_2 时需要各个叶子节点的候选集的笛卡尔积 $S(?a)\times S(?c)\times S(?w_2)\times S(?g)\times S(?y)$,然而查询图中顶点$?x$ 匹配 RDF 数据图中顶点 $GibsonW$ 并不能形成最终答案,因此,基本算法做了无用的笛卡尔积,代价较高.相反,优化算法通过推迟笛卡尔积提前过滤掉在邻接链表 $N(GibsonW)$上的笛卡尔积操作.

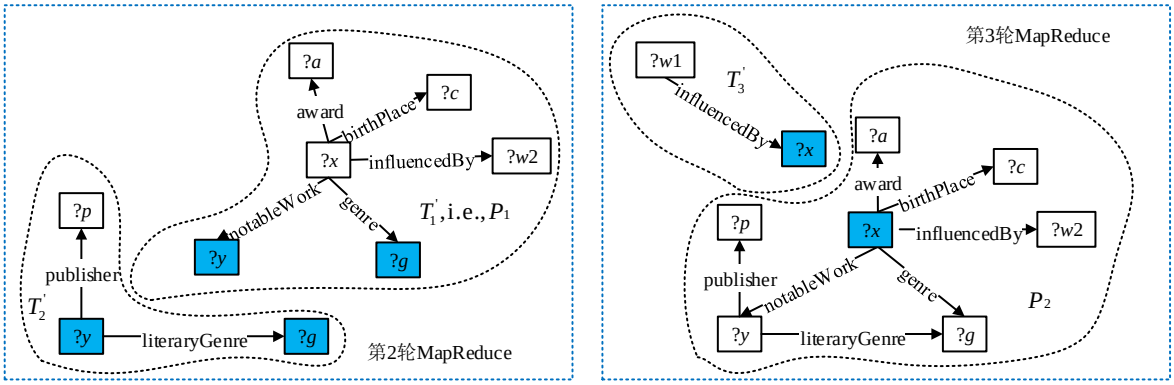

Fig.6 Matching process of the example SPARQL BGP query graph Q_1

图 6 SPARQL BGP 示例查询图 Q_1 查询匹配过程

5 实 验

本节在合成数据集和真实数据集上验证算法的有效性和可扩展性,并且与 SHARD,S2X 和 SDec 的优化版本的算法比较.不与 S2RDF 和 TriAD 方法作比较的原因如下.

(1) S2RDF 使用 ExtVP 索引提高查询效率,但是预处理时间代价过高,严重限制了该系统对大规模数据集的适用性,最新的文献[22]中通过实验也验证了这一点.例如:在相同的实验环境下,S2RDF 预处理 DBpedia 数据集的时间超过 2.5 个小时,远大于本文 SDec 系统的查询执行时间;

(2) 如文献[22]所述,TriAD 假设数据必须装载到集群的内存中,如果数据量超过了集群内存大小,TriAD

将不再适用.而本文没有这个限制条件,而且 TriAD 借助了大量的索引加快查询.

因此,与这两种方法进行比较实验是不公平的.

本文的 SDec 算法与图划分策略和分布式集群环境上的存储策略是独立无关的.实际上,本文使用分布式文件系统 HDFS(Hadoop distributed file system)的默认划分策略.

5.1 实验环境和数据集

本文实验平台使用的分布式集群包括 8 个相同的计算节点,每个节点使用主频为 3.60GHz 的 Intel(R) Core(TM) i7-7700 四核处理器,其内存大小为 16GB,硬盘大小为 500GB.节点间通信使用 1000Mbps 以太网.实验平台所用集群的所有节点均使用 Linux 64-bit CentOS 操作系统,其使用的 Hadoop 版本号为 2.7.4,Spark 版本号为 2.2.0.本文原型系统的所有代码均用 Scala 语言实现.

本实验使用的数据包括 WatDiv 标准合成数据集以及 DBpedia 真实数据集,这两个数据集都是 RDF 数据集.WatDiv 是一个标准的 RDF 合成数据集,允许用户定义自己的数据集并生成不同大小的数据集,本文使用了 3 个不同规模的 WatDiv 数据集(也就是 WatDiv1M,WatDiv10M 和 WatDiv100M)进行实验测试和比较;DBpedia 数据集是从维基百科中提取的一个真实数据集.本次实验中,所有数据集均为 N-Triple 格式,具体统计信息见表 2.本文根据 RDF 查询图的形状将它们分为 4 类,包括线性查询(linear queries,简称 L)、星形查询(star queries,简称 S)、雪花形查询(snowflake queries,简称 F)和复杂查询(complex queries,简称 C),见表 3.本文使用 WatDiv 数据集给定的 20 个基本查询模板进行实验测试.因为缺乏 DBpedia 数据集上的查询模板,本文设计 8 个查询覆盖上文提及的 4 个查询种类.

Table 2 Datasets

表 2 数据集

数据集	顶点个数	边个数
WatDiv1M	158 118	1 109 678
WatDiv10M	1 052 571	10 916 457
WatDiv100M	10 250 571	108 997 714
DBpedia	6 060 648	23 509 250

Table 3 Queries

表 3 查询

查询	WatDiv	DBpedia
L	L1, L2, L3, L4, L5	L1, L2
S	S1, S2, S3, S4, S5, S6, S7	S1, S2
F	F1, F2, F3, F4, F5	F1, F2
C	C1, C2, C3	C1, C2

5.2 实验结果

本节分 4 组实验进行详细介绍,包括改变优化算法中哈希函数个数 k、改变数据集规模及改变集群中节点个数来验证优化算法 $SDec_{opt}$、基本算法 Sdec,SHARD 和 S2X 这 4 种方法的效率和可伸缩性.以下实验结果中,每个查询测试 3 次取平均值,避免随机误差.

(1) 改变编码优化技术中参数 k 测试查询性能.

本组实验分别设置哈希函数个数 k=1,2,3,在 WatDiv100M 数据集上测试 20 个模板查询.随着 k 从 1 增加到 3,误判率 f 减小,可以在星形匹配阶段过滤更多的无用数据输入,加快数据查询;同时,随着 RDF 数据图中主语签名的位数组长度变长,占用内存空间越多,影响查询效率.

如图 7 所示,当 k=3 时,编码邻居信息增加查询运行时内存带来的影响占主要因素,导致查询时间增加;k=2 时,在绝大部分查询上可以取得最好的查询性能.随着哈希函数个数从 1 到 3,邻居信息 BF 编码的运行时内存变化.本文以下的所有实验中,优化方法中的哈希函数个数 k 取值为 2.

(2) SPARQL BGP 查询性能测试.

本组实验分别在合成数据集 WatDiv100M 和真实数据集 DBpedia 上验证集群节点个数为 8 时本文所提方法的查询效率.如图 8 所示,在合成数据集 WatDiv 上,$SDec_{opt}$ 在 20 个模板查询上都可以获得最高的查询效率;同时,SDec 也优于 SHARD 和 S2X.在本文中,运行时间超过 1×10^4s 用 INF 表示.如图 8 所示:对查询 F3 和 C2,S2X 不能在该时间限制内结束,而本文的优化算法仅需要 19.24s 和 36.92s 就可以得到查询结果.对剩余的 18 个查询,本文优化算法的查询时间与 S2X 相比提高了 7 倍~40 倍,平均查询时间提高 17 倍.究其原因:S2X 将 BGP 中所有三元组模式的匹配结果进行连接操作会产生大量的中间结果,代价较高;而本文的最小匹配结构为星形,同时考虑 RDF 图的丰富语义和结构信息,减少了大量无用的中间结果.与 SHARD 相比,$SDec_{opt}$ 的查询时间提高了 16 倍~99 倍,平均查询时间提高 40 倍.SHARD 采用三元组存储也将 BGP 查询分解为三元组模式进行匹配,在包含 3 列的数据表上进行连接操作,连接代价高.因此,$SDec_{opt}$ 算法的查询时间比 S2X 和 SHARD 平均提高了一个数量级.

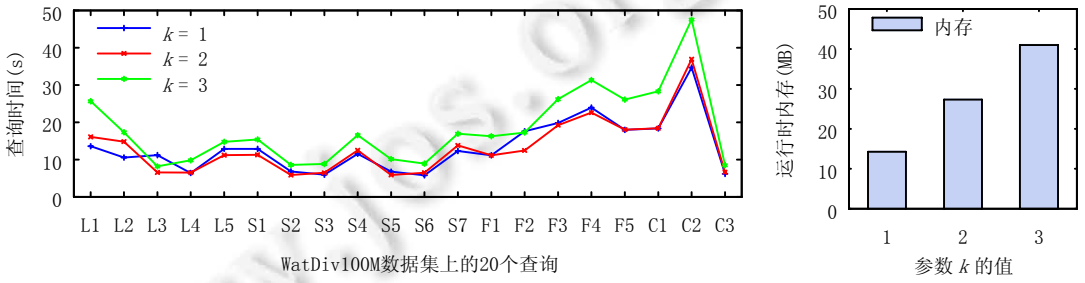

Fig.7 Experiments of changing the number of hash functions k and the run memory

图 7 改变哈希函数个数 k 的实验结果及运行时内存变化

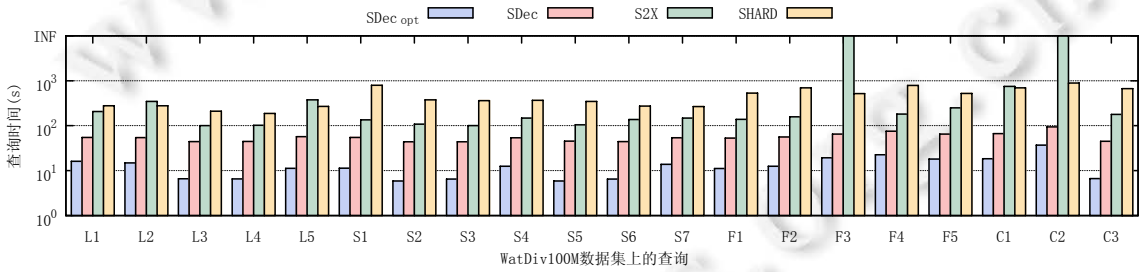

Fig.8 Query efficiency of different SPARQL processors over synthetic dataset

图 8 不同 SPARQL 处理器在合成数据集上的查询效率

此外,与基本算法 SDec 相比,优化算法时间减少了 60.68%~87.00%,即,$SDec_{opt}$ 可以更有效地评估 BGP 查询.该实验结果说明,本文的优化技术效果非常显著.原因包含以下两点:(1) 优化算法对邻居信息进行 BF 编码,在星形匹配阶段,如果星形结构的签名与 RDF 图中主语 s 的签名不匹配,直接将 $N(s)$上的星形匹配剪掉,以此过滤掉大量的无用计算;(2) 基本算法在星形匹配阶段进行了大量的无用笛卡尔积,而优化算法通过推迟笛卡尔积操作减少很多无用的笛卡尔积操作,提高了查询性能.

如图 9 所示,在真实数据集 DBpedia 上,$SDec_{opt}$ 在所有查询上的效率也是最高的.回答查询 C2 时 S2X 报错,导致时间缺失.这说明 S2X 无法有效回答涉及大量中间结果的复杂查询.对于剩余的 7 个查询,优化算法 $SDec_{opt}$ 查询时间比 S2X 快 7 倍~497 倍,平均查询时间比 S2X 快 120 倍;同时,与 SHARD 相比,优化算法 $SDec_{opt}$ 查询时间快 7 倍~26 倍,平均查询时间快 15 倍.即:在真实数据集上,本文算法的平均时间也比 SHARD 和 S2X 快一个数量级.另外,由图 9 观察到:优化算法的查询时间明显提高,比基本算法 SDec 查询时间减少 49.63%~78.71%.

(3) 改变数据集规模的查询性能测试.

本组实验通过改变 WatDiv 数据集规模从 1M~100M 在 8 个节点的集群上测试 18 个查询.本文将这 18 个

查询分为 4 个查询种类,分别计算 SDec,SDec$_{opt}$,S2X 和 SHARD 的 4 类平均查询时间.如图 10 所示,随着数据规模增大,所有 SPARQL 查询处理器的查询时间都增加,SHARD 和 S2X 的查询时间增加幅度远大于本文方法.观察图 10 可知:数据集规模从 10M 个三元组增加到 100M 个三元组,S2X 和 SHARD 的查询性能严重下降.如:对雪花形查询,SHARD 和 S2X 在 WatDiv100M 上需要将近 1000s 的时间才可以得到查询结果,而本文的优化算法仅需 10s 左右.

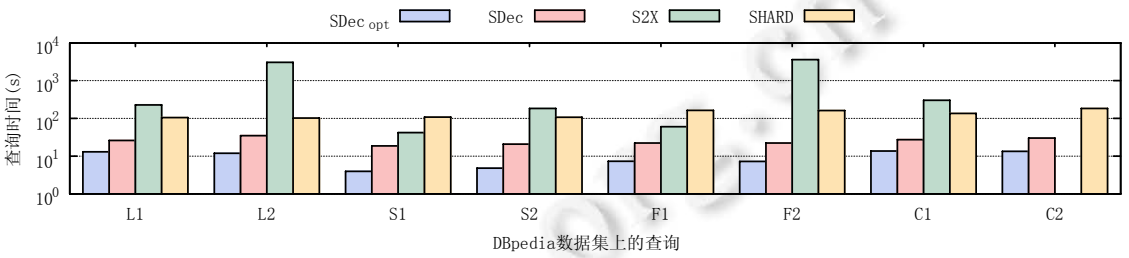

Fig.9 Query efficiency of different SPARQL processors over real-world dataset

图 9 不同 SPARQL 查询处理器在真实数据集上的查询效率

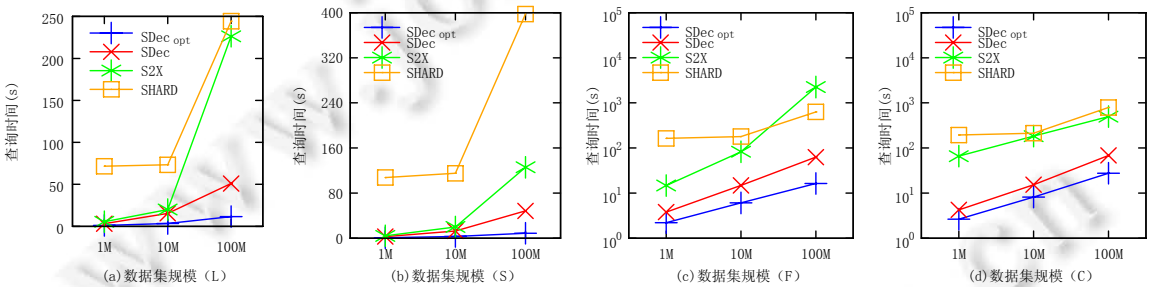

Fig.10 Experiment results of changing the size of datasets

图 10 改变数据集规模的实验结果

(4) 算法可伸缩性测试.

本组实验在 WatDiv100M 和 DBpedia 数据集上进行,改变集群节点个数,从 4~8.从每类查询中随机选取 1 个查询,也就是 L4,S4,F2,C3.如图 11 所示:随着集群节点个数的增加,在合成数据集 WatDiv 上,这 4 个 SPARQL 查询处理器的查询时间减少.原因是随着集群节点个数的增加,CPU 核数增加,查询过程中计算的并行度增加,查询所需的总时间减少.由图 11 可以看出,SDec$_{opt}$ 的查询性能始终是最高的.

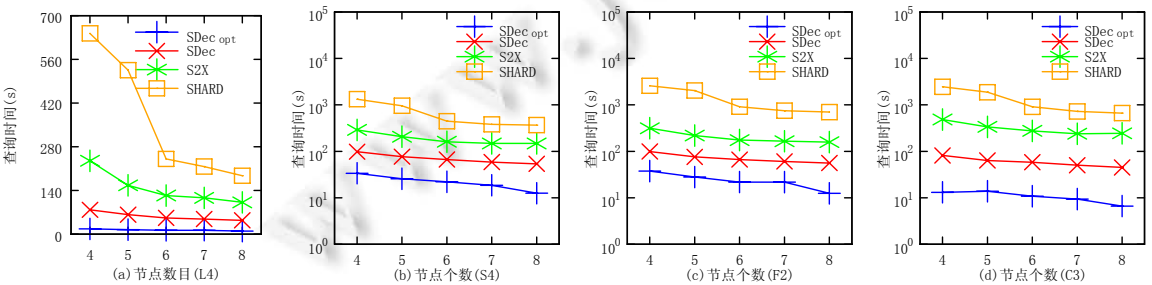

Fig.11 Experiment results of changing the number of sites on synthetic dataset WatDiv100M

图 11 改变集群节点个数在合成数据集 WatDiv100M 上的实验结果

在真实数据集 DBpedia 上,改变集群节点的个数验证算法的可伸缩性.本次实验选取 L1,S1,F1,C1 查询,即,

从每类查询中随机选取 1 个查询进行实验.如图 12 所示:随着集群节点个数从 4 增加到 8,上述 4 个 SPARQL 查询处理方法的查询时间逐渐减少.

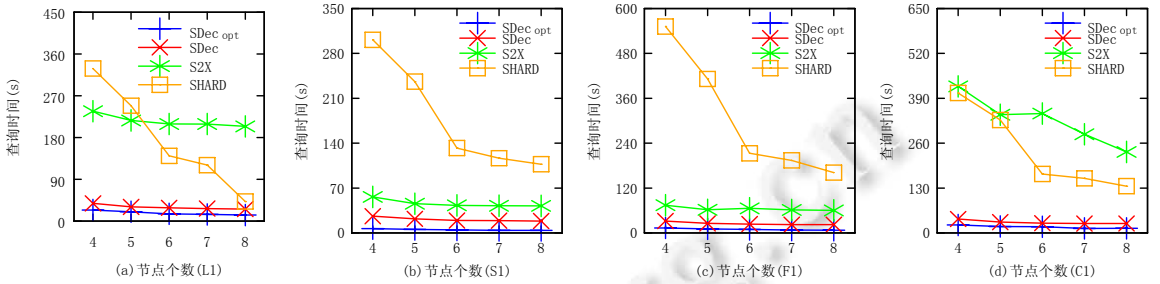

Fig.12 Experiment results of changing the number of sites on real-world dataset DBpedia

图 12 改变集群节点个数在真实数据集 DBpedia 上的实验结果

6 总 结

本文提出了基于 MapReduce 框架的查询处理器 SDec,有效回答大规模 RDF 智能图数据上的 SPARQL BGP 查询.本文的 SDec 算法利用 RDF 图内嵌的丰富语义和 RDF 图结构分解查询图为星形集合,并确定这些星形的匹配顺序,以此减少星形匹配过程中的中间结果.此外,为了进一步提高查询算法的效率,本文设计了两个优化技术,包括对邻居信息编码过滤无用的数据输入和推迟笛卡尔积操作改进基本算法的性能.合成数据集和真实数据集上的大量实验验证了本文所提算法的有效性、高效性和伸缩性,并且实验结果显示,本文的算法优于 SHARD 和 S2X.

References:

[1] Hammoud M, Rabbou DA, Nouri R, Beheshti SMR, Sakr S. DREAM: Distributed RDF engine with adaptive query planner and minimal communication. Proc. of the VLDB Endowment, 2015,8(6):654−665.

[2] Pérez J, Arenas M, Gutierrez C. Semantics and complexity of SPARQL. In: Cruz I, ed. Proc. of the Semantic Web (ISWC 2006). Berlin: Springer-Verlag, 2006. 30−43.

[3] Dyer M, Greenhill C. The complexity of counting graph homomorphisms. Random Structures & Algorithms, 2000,17(3-4): 260−289.

[4] Du F, Chen YG, Du XY. Survey of RDF query processing techniques. Ruan Jian Xue Bao/Journal of Software, 2013,24(6): 1222−1242 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/4387.htm [doi: 10.3724/SP.J.1001.2013.04387]

[5] Neumann T, Weikum G. RDF-3X: A RISC-style engine for RDF. Proc. of the VLDB Endowment, 2008,1(1):647−659.

[6] Zou L, Chen L, Shen X, Huang R, Zhao D. gStore: A graph-based SPARQL query engine. VLDB Journal—The Int'l Journal on Very Large Data Bases, 2014,23(4):565−590.

[7] Rohloff K, Schantz RE. High-performance, massively scalable distributed systems using the MapReduce software framework: The SHARD triple-store. In: Proc. of the Programming Support Innovations for Emerging Distributed Applications. New York: ACM Press, 2010. Article No.4.

[8] Husain M, Mcglothlin J, Masud MM, Khan LR, Thuraisingham B. Heuristics-based query processing for large RDF graphs using cloud computing. IEEE Trans. on Knowledge & Data Engineering, 2011,23(9):1312−1327.

[9] Schätzle A, Przyjaciel-Zablocki M, Berberich T, Lausen G. S2X: Graph-parallel querying of RDF with GraphX. In: Wang F, ed. Proc. of the Biomedical Data Management and Graph Online Querying. Switzerland: Springer-Verlag, 2016. 155−168.

[10] Erling O, Mikhailov I. Virtuoso: RDF Support in a Native RDBMS. Berlin: Springer-Verlag, 2009. 501−519.

[11] Schatzle A, Przyjacielzablocki M, Skilevic S, Lausen G. S2RDF: RDF querying with SPARQL on spark. Proc. of the VLDB Endowment, 2016,9(10):804−815.

[12] Zeng K, Yang J, Wang H, Shao B, Wang Z. A distributed graph engine for Web scale RDF data. Proc. of the VLDB Endowment, 2013,6(4):265−276.

[13] Peng P, Zou L, Chen L, Zhao D. Processing SPARQL queries over distributed RDF graphs. The VLDB Journal—The Int'l Journal on Very Large Data Bases, 2016,25(2):243−268.

[14] Gurajada S, Seufert S, Miliaraki I, Theobald M. TriAD: A distributed shared-nothing RDF engine based on asynchronous message passing. In: Dyresson C, ed. Proc. of the ACM Conf. on Management of Data. New York: ACM Press, 2014. 289−300.

[15] Lai L, Qin L, Lin X, Chang L. Scalable subgraph enumeration in MapReduce. Proc. of the VLDB Endowment, 2015,8(10): 974−985.

[16] Sun Z, Wang H, Wang H, Shao B, Li J. Efficient subgraph matching on billion node graphs. Proc. of the VLDB Endowment, 2012, 5(9):788−799.

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

[18] Aluc G, Hartig O, Ozsu M, Daudjee K. Diversied stress testing of RDF data management systems. In: Mika P, ed. Proc. of the Semantic Web (ISWC 2014). Berlin: Springer-Verlag, 2014. 197−212.

[19] Zou L, Peng P. A survey of distributed RDF data management. Journal of Computer Research and Development, 2017,54(6): 1213−1224 (in Chinese with English abstract).

[20] Gonzalez JE, Xin RS, Dave A, Crankshaw D, Franklin MJ, Stoica I. GraphX: Graph processing in a distributed dataflow framework. OSDI, 2014,14(2):599−613.

[21] Zaharia M, Chowdhury M, Franklin MJ, Shenker S, Stoica I. Spark: Cluster computing with working sets. HotCloud, 2010, 10(10-10):95.

[22] Abdelaziz I, Harbi R, Khayyat Z, Kalnis P. A survey and experimental comparison of distributed SPARQL engines for very large RDF data. Proc. of the VLDB Endowment, 2017,10(13):2049−2060.

附中文参考文献:

[4] 杜方,陈跃国,杜小勇.RDF 数据查询处理技术综述.软件学报,2013(6):1222−1242. http://www.jos.org.cn/1000-9825/4387.htm [doi: 10.3724/SP.J.1001.2013.04387]

[19] 邹磊,彭鹏.分布式 RDF 数据管理综述.计算机研究与发展,2017,54(6):1213−1224.

王鑫(1981－),男,天津人,博士,副教授,CCF 高级会员,主要研究领域为知识图谱数据管理,图数据库,大规模知识处理.

徐强(1993－),女,硕士生,主要研究领域为知识图谱数据管理,图数据库.

柴乐乐(1995－),女,硕士生,主要研究领域为知识图谱表示学习.

杨雅君(1983－),男,博士,讲师,CCF 专业会员,主要研究领域为图数据管理,图挖掘.

柴云鹏(1983－),男,博士,副教授,博士生导师,CCF 专业会员,主要研究领域为分布式系统,云计算,存储系统.