软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2019,30(7):2139−2174 [doi: 10.13328/j.cnki.jos.005841]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

知识图谱数据管理研究综述*

王 鑫[1,2], 邹 磊[3], 王朝坤[4], 彭 鹏[5], 冯志勇[1,2]

[1](天津大学 智能与计算学部,天津 300350)

[2](天津市认知计算与应用重点实验室,天津 300350)

[3](北京大学 计算机科学技术研究所,北京 100871)

[4](清华大学 软件学院,北京 100084)

[5](湖南大学 信息科学与工程学院,湖南 长沙 410082)

通讯作者: 王鑫, E-mail: wangx@tju.edu.cn

摘 要: 知识图谱是人工智能的重要基石.各领域大规模知识图谱的构建和发布对知识图谱数据管理提出了新的挑战.以数据模型的结构和操作要素为主线,对目前的知识图谱数据管理理论、方法、技术与系统进行研究综述.首先,介绍知识图谱数据模型,包括 RDF 图模型和属性图模型,介绍 5 种知识图谱查询语言,包括 SPARQL、Cypher、Gremlin、PGQL 和 G-CORE;然后,介绍知识图谱存储管理方案,包括基于关系的知识图谱存储管理和原生知识图谱存储管理;其次,探讨知识图谱上的图模式匹配、导航式和分析型 3 种查询操作.同时,介绍主流的知识图谱数据库管理系统,包括 RDF 三元组库和原生图数据库,描述目前面向知识图谱的分布式系统与框架,给出知识图谱评测基准.最后,展望知识图谱数据管理的未来研究方向.

关键词: 知识图谱;数据管理;数据模型;查询语言;存储管理;查询操作

中图法分类号: TP182

中文引用格式: 王鑫,邹磊,王朝坤,彭鹏,冯志勇.知识图谱数据管理研究综述.软件学报,2019,30(7):2139−2174. http://www.jos.org.cn/1000-9825/5841.htm

英文引用格式: Wang X, Zou L, Wang CK, Peng P, Feng ZY. Research on knowledge graph data management: A survey. Ruan Jian Xue Bao/Journal of Software, 2019,30(7):2139−2174 (in Chinese). http://www.jos.org.cn/1000-9825/5841.htm

Research on Knowledge Graph Data Management: A Survey

WANG Xin[1,2], ZOU Lei[3], WANG Chao-Kun[4], PENG Peng[5], FENG Zhi-Yong[1,2]

[1](College of Intelligence and Computing, Tianjin University, Tianjin 300350, China)

[2](Tianjin Key Laboratory of Cognitive Computing and Application, Tianjin 300350, China)

[3](Institute of Computer Science and Technology, Peking University, Beijing 100871, China)

[4](School of Software, Tsinghua University, Beijing 100084, China)

[5](College of Computer Science and Electronic Engineering, Hunan University, Changsha 410082, China)

Abstract: Knowledge graphs have become the cornerstone of artificial intelligence. The construction and publication of large-scale knowledge graphs in various domains have posed new challenges on the data management of knowledge graphs. In this paper, in accordance with the structural and operational elements of a data model, the current theories, methods, technologies, and systems of knowledge graph data management are surveyed. First, the paper introduces knowledge graph data models, including the RDF graph

* 基金项目: 国家自然科学基金(61572353); 天津市自然科学基金(17JCYBJC15400)

Foundation item: National Natural Science Foundation of China (61572353); Natural Science Foundation of Tianjin of China (17JCYBJC 15400)

收稿时间: 2018-09-18; 修改时间: 2019-02-20; 采用时间: 2019-03-25; jos 在线出版时间: 2019-04-10

CNKI 网络优先出版: 2019-04-09 17:32:35, http://kns.cnki.net/kcms/detail/11.2560.TP.20190409.1732.009.html

model and the property graph model, and also introduces 5 knowledge graph query languages, including SPARQL, Cypher, Gremlin, PGQL, and G-CORE. Second, the storage management schemes of knowledge graphs are presented, including relational-based and native approaches. Third, three kinds of query operations are discussed, which are graph pattern matching, navigational, and analytical queries. Fourth, the paper introduces mainstream knowledge graph database management systems, which are categorized into RDF triple stores and native graph databases. Meanwhile, the state-of-the-art distributed systems and frameworks that are used for processing knowledge graphs are also described, and benchmarks are presented for knowledge graphs. Finally, the future research directions of knowledge graph data management are put forward as well.

Key words: knowledge graph; data management; data model; query language; storage management; query operation

知识图谱作为符号主义发展的最新成果,是人工智能的重要基石.随着知识图谱规模的日益扩大,其数据管理问题愈加重要.一方面,以文件形式保存知识图谱无法满足用户的查询、检索、推理、分析及各种应用需求;另一方面,传统数据库的关系模型与知识图谱的图模型之间存在显著差异,关系数据库无法有效管理大规模知识图谱数据.为了更好地管理知识图谱,语义 Web 领域发展出专门存储 RDF 数据的三元组库;数据库领域发展出用于管理属性图的图数据库.但是目前还没有一种数据库系统被公认为是具有主导地位的知识图谱数据库.

目前,规模为百万顶点(10^6)和上亿条边(10^8)的知识图谱数据集已经常见.链接开放数据 2018 年 8 月发布的 LOD 云图中很多知识图谱数据集规模超过 10 亿条三元组.例如,维基百科知识图谱 DBpedia(>30 亿条)、地理信息知识图谱 LinkedGeoData(>30 亿条)和蛋白质知识图谱 UniProt(>130 亿条)等.各领域大规模知识图谱的构建和发布对知识图谱数据管理提出了新的挑战.本文将遵循数据管理领域的良好传统,以数据模型的结构和操作两大要素为主线,对目前的知识图谱数据管理理论、方法、技术与系统等方面的研究与实践进行综述.

数据模型是任何数据管理领域的基础与核心.众所周知,数据模型包括数据的结构、操作和约束.由于知识图谱数据管理尚处于起步阶段,知识图谱数据模型的结构和操作方面还未发展完善,约束方面仅有尚在制定中的 RDF Shapes 约束语言[1]等少量研究工作,故而本文仅综述知识图谱数据模型中的结构和操作要素.本文首先介绍目前知识图谱的两种主流数据模型:RDF 图模型和属性图模型;之后,作为知识图谱数据模型的操作,介绍 5 种知识图谱查询语言,包括 SPARQL、Cypher、Gremlin、PGQL 和 G-CORE;接着,介绍如何使用各种存储管理方案实现知识图谱逻辑模型的物理存储,包括基于关系的知识图谱存储管理和原生知识图谱存储管理;然后,探讨知识图谱上 3 种主要的查询操作类型,即图模式匹配、导航式和分析型查询;最后,介绍实现了知识图谱数据模型的主流数据库管理系统,包括 RDF 三元组库和原生图数据库,同时描述目前面向知识图谱的各种分布式系统与框架,并简要介绍知识图谱评测基准.本文最后对知识图谱数据管理的未来研究方向进行展望.作为阅读指导,图 1给出了本综述各部分内容之间的总体路线图.

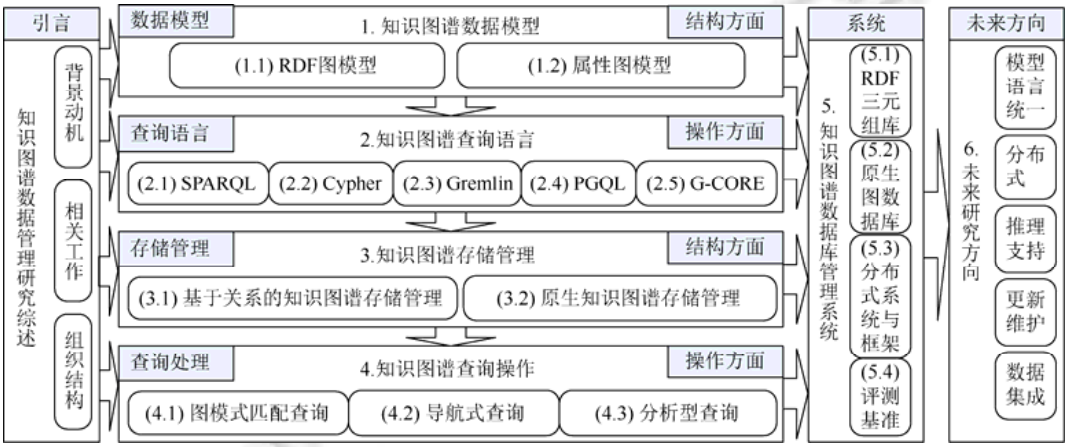

Fig.1 A roadmap of the contents of this survey

图1 本文各部分内容的总体路线图

相关工作.目前,尚未发现与本文相同使用数据模型要素作为主线对知识图谱数据管理进行综述的文献.文献[2]对 2002 年之前的早期图数据模型进行了综述,但这些图数据模型由于结构复杂均未成为后来知识图谱的表示基础.文献[3]对 2006 年之前的图模式匹配查询算法进行了综述,图模式匹配查询是目前知识图谱的查询操作类型之一.文献[4]对 RDF 图模式进行了形式化定义,对其上的基本图模式查询给出了若干理论结果.文献[5−7]是关于 RDF 图数据管理的综述,包括单机和分布式系统.文献[8]是关于图数据管理的较新综述,但其内容与本综述差别较大,且没有按照数据模型结构与操作要素展开介绍.文献[9]着重在图数据查询处理方面进行综述,包括 RDF 图和属性图上的图模式匹配和导航式查询,但内容上侧重理论结果,本文不仅涵盖了这些内容,而且还包括了知识图谱数据管理实现技术与系统,同时本文比文献[9]多介绍了 PGQL 和 G-CORE 两种查询语言.文献[10]综述了以顶点为中心的分布式图计算框架.文献[11]综述了分布式环境中的 RDF 存储和查询处理.文献[12]使用真实与合成数据集对主要的分布式 SPARQL 引擎进行了实验评估.文献[13]和文献[14]从分类体系和高层编程抽象角度综述了分布式图处理框架.文献[15]也是大规模图数据的分布式处理平台综述,但侧重于分析型处理任务.在中文文献中,文献[16]按照基于云计算平台、基于数据划分和联邦式系统的分类综述了 RDF 图分布式存储和查询方法,但没有涉及属性图数据管理.文献[17]从图存储、图索引、图分割、图计算模型、消息通信机制、图查询处理等方面综述了大规模图数据的分布式处理技术,但当时还未形成知识图谱数据模型.文献[18]仅就单机版本的图模式匹配查询进行了综述.最新的相关综述包括:文献[19]主要介绍了基于 Pregel 的分布式图处理框架的研究进展,而本文第 5.3 节会在更广的范围内介绍面向知识图谱数据管理的分布式系统与框架;文献[20]集中于讨论知识图谱上的推理方法与技术,而本文主题是知识图谱的数据管理,并在第 6 节中将知识图谱数据管理对于本体和知识推理的支持作为未来研究方向之一.由于篇幅所限,本文不涉及知识图谱在时间维度上的变化,关于动态图的图模式匹配查询研究综述可参见文献[21].

1 知识图谱数据模型

数据模型定义了数据的逻辑组织结构(structure)、其上的操作(operation)和约束(constraint),决定了数据管理所采取的有效方法与策略,对于存储管理、查询处理、查询语言设计均至关重要.关系数据库长盛不衰的一个重要原因是关系数据模型(relational data model)中简洁而通用的关系结构,以及用具有严格数学定义的关系代数表达关系上的操作和约束[22].使用图结构刻画数据最早要追溯到层次数据模型(hierarchical data model)和网状数据模型(network data model):层次数据模型使用树结构表示数据,树是一种特殊的图;网状数据模型虽然支持图结构,但与后来的图数据模型有较大差异,也始终未能成为主流数据模型.早期的若干图数据模型基本上是以图论中的图结构(G=(V,E),其中,V 是顶点集,E 是边集)或其扩展作为数据结构[2];可以认为,知识图谱数据模型是图数据模型的继承和发展.知识图谱数据模型基于图结构,用顶点表示实体,用边表示实体间的联系,这种一般和通用的数据表示恰好能够自然地刻画现实世界中事物的广泛联系.目前,主流的知识图谱数据模型有两种: RDF 图模型和属性图模型.下面分别进行介绍.

1.1 RDF图模型

RDF 全称为资源描述框架(resource description framework),是万维网联盟(World Wide Web consortium,简称 W3C)制定的在语义 Web(semantic Web)上表示和交换机器可理解(machine-understandable)信息的标准数据模型[23].在 RDF 图中,每个资源具有一个 HTTP URI 作为其唯一 id;RDF 图定义为三元组(s,p,o)的有限集合;每个三元组表示是一个事实陈述句,其中,s 是主语,p 是谓语,o 是宾语;(s,p,o)表示 s 与 o 之间具有联系 p,或表示 s 具有属性 p 且其取值为 o.下面给出 RDF 图的严格定义.

定义1(RDF 图). 设 U、B 和 L 为互不相交的无限集合,分别代表 URI、空顶点(blank node)和字面量(literal).一个三元组(s,p,o)$\in$($U\cup B$)$\times U\times$($U\cup B\cup L$)称为 RDF 三元组,其中,s 为主语,p 为谓语,o 为宾语.RDF 图 G 是 RDF 三元组的有限集合.

图 2所示的 RDF 图示例描述了一个电影知识图谱,其中包括电影(movie)Titanic、导演(director)James_Cameron、演员(actor)Leonardo_DiCaprio 和 Kate_Winslet 等资源以及这些资源上的若干属性及资源之间的执

导(direct)和出演(acts_in)联系.在 RDF 图示例中,用椭圆表示资源,用矩形表示字面量;一条有向边及其连接的两个顶点对应于一条三元组,尾顶点是主语,边标签是谓语,头顶点是宾语.资源所属的类型由 RDF 内置谓语 rdf:type 指定,如三元组(James_Cameron,rdf:type,Director)表示 James_Cameron 是导演.为简便起见,本文 RDF 图中资源和谓语 URI 名称省略了命名空间前缀(RDF 内置名称不省略,如 rdf:type).对于本例中字面量的类型,字符串放在双引号中,整数 195 是电影分钟数,定点数 1.79E9 和 2.0E8 分别是导演净资产和电影预算金额,日期 1954-08-16、1974-11-11 和 1975-10-05 分别是导演和两位演员的出生日期.由于空顶点的引入不会对 RDF 数据管理方法产生本质改变,本文将 RDF 图中的空顶点等同为 URI.

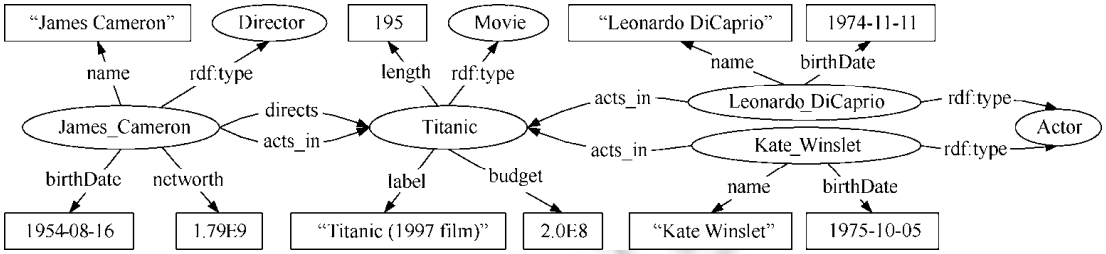

Fig.2 An RDF graph example: Movie knowledge graph

图2 RDF 图示例:电影知识图谱

例1:图 2 所示的 RDF 图的三元组集合形式为

G={(James_Cameron,rdf:type,Director),(James_Cameron,name,"James Cameron"),
(James_Cameron,birthDate,1954-08-16),(James_Cameron,networth,1.79E9),
(James_Cameron,directs,Titanic),(James_Cameron,acts_in,Titanic),
(Titanic,rdf:type,Movie),(Titanic,label,"Titanic(1997 film)"),
(Titanic,budget,2.0E8),(Titanic,length,195),
(Leonardo_DiCaprio,rdf:type,Actor),(Leonardo_DiCaprio,name,"Leonardo DiCaprio"),
(Leonardo_DiCaprio,birthDate,1974-11-11),(Leonardo_DiCaprio,acts_in,Titanic),
(Kate_Winslet,rdf:type,Actor),(Kate_Winslet,name,"Kate Winslet"),
(Kate_Winslet,birthDate,1975-10-05),(Kate_Winslet,acts_in,Titanic)}.

导演在此片中也出演了角色,但图 2 并没有包含出演(acts_in)的角色信息.实际上,RDF 图模型没有对于顶点和边上属性的内置支持.顶点上的属性可表示为宾语是字面量的三元组;边上属性的表示需要使用额外的机制,最常见的是利用"具体化(reification)"方法[24],引入额外的顶点表示整个三元组,将边属性表示为以该顶点为主语的三元组.图 3 给出了如何使用"具体化"为 acts_in 边添加角色(role)属性;通过引入资源 acts_in_1 来表示三元组(James_Cameron,acts_in,Titanic),并使用 RDF 内置谓语 rdf:subject、rdf:predicate 和 rdf:object 分别指明该条三元组的主语、谓语和宾语;三元组(acts_in_1,role,"Steerage Dancer")为 acts_in 边添加了 role 属性,即导演 James_Cameron 在影片 Titanic 中扮演了"Steerage Dancer"这一角色.为了简洁起见,图 3 中省略了三元组(acts_in_1,rdf:type,rdf:Statement),即指明新引入的资源 acts_in_1 的类型是三元组(RDF 内置类型 rdf:Statement 表示一条三元组).

从数据模型角度看,RDF 图是一种特殊的有向标签图(directed labeled graphs).与普通有向标签图相比,RDF 图的特殊性在于,其三元组集合的本质使得一个三元组中的谓语也可作为另一个三元组的主语或宾语.反映在有向标签图中,即边亦可作为顶点(如图 3 中的边 acts_in),顶点与边交集非空.在数学上,将这种图称为 3-均匀超图(3-uniform hypergraph)[25].文献[4]给出了关于 RDF 图更加丰富的形式化定义,包括 RDF 模式(RDF schema)[26]、基于模型论的语义和基本推理系统.需要指出的是,W3C 为 RDF 图定义了基于描述逻辑(一阶谓词逻辑的可判定子集)的本体语言 OWL[27],可描述概念之间更为复杂的逻辑关系,并给出了相应的逻辑推理规则.RDF 图上的

本体推理超出了本文的范围,感兴趣的读者可参见文献[28].

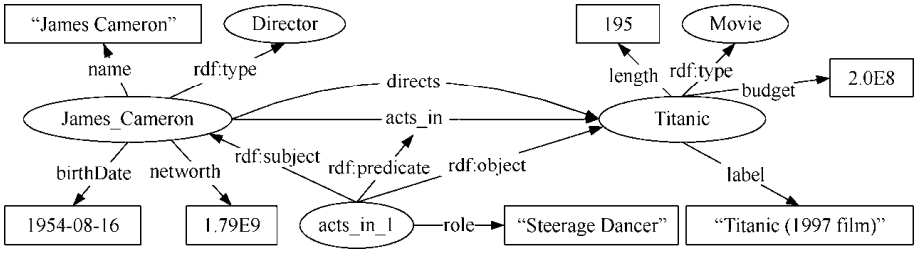

Fig.3 Representation of edge attributes in RDF graphs using reification

图3 RDF 图中通过"具体化"方法表示边上的属性

1.2 属性图模型

属性图是另一种管理知识图谱数据常用的数据模型.与 RDF 图模型相比,属性图模型对于顶点属性和边属性具备内置的支持.目前,属性图模型被图数据库业界广泛采用,包括著名的图数据库 Neo4j[29].最近,由图数据管理领域学术界和工业界成员共同组成的关联数据基准委员会(Linked Data Benchmark Council,简称 LDBC)也正在以属性图为基础对图数据模型和图查询语言进行标准化[30].下面给出属性图的形式化定义.

定义 2(属性图). 属性图 G 是 5 元组$(V,E,\rho,\lambda,\sigma)$,其中,(1) V 是顶点的有限集合;(2) E 是边的有限集合且 $V\cap E=\varnothing$;(3) 函数$\rho:E\to(V\times V)$将边关联到顶点对,如$\rho(e)=(v_1,v_2)$表示 e 是从顶点 v_1 到顶点 v_2 的有向边;(4) 设 Lab 是标签集合,函数$\lambda:(V\cup E)\to Lab$ 为顶点或边赋予标签,如 $v\in V$(或 $e\in E$)且$\lambda(v)=l$(或 $\lambda(e)=l$),则 l 为顶点 v(或边 e)的标签;(5) 设 $Prop$ 是属性集合,Val 是值集合,函数$\sigma:(V\cup E)\times Prop\to Val$ 为顶点或边关联属性,如 $v\in V$(或 $e\in E$)、$p\in Prop$ 且$\sigma(v,p)=val$(或$\sigma(e,p)=val$),则顶点 v(或边 e)上属性 p 的值为 val.

图 4 给出了图 2 中电影知识图谱的属性图示例.在属性图中,每个顶点和边都具有唯一 id(如顶点 v_1、边 e_2);顶点和边均可具有标签(如顶点 v_1 上的标签 Director、边 e_2 上的标签 acts_in),其作用基本相当于 RDF 图中的资源类型;顶点和边上均可具有一组属性,每个属性由属性名和属性值组成(如顶点 v_1 上的属性 name="James Cameron"、边 e_2 上的属性 role="Steerage Dancer").可以看出,利用边属性的定义,图 4 比图 2 增加了出演的角色信息,同时又没有改变属性图的整体结构(RDF 图的"具体化"方法增加了顶点和边,改变了图结构).

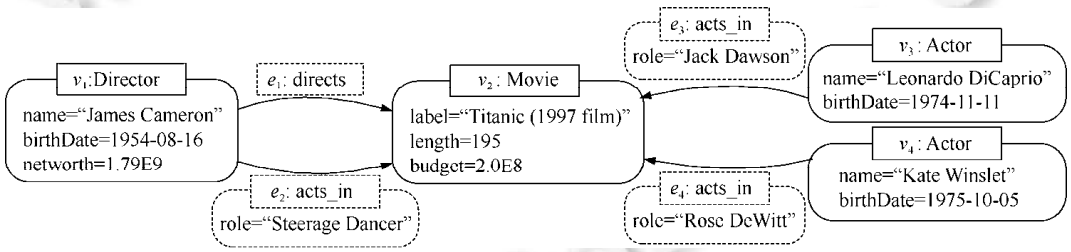

Fig.4 A property graph example: Movie knowledge graph

图4 属性图示例:电影知识图谱

例 2:图 4 所示的属性图 $G=(V,E,\rho,\lambda,\sigma)$,其中,

$V=\{v_1,v_2,v_3,v_4\}$,$E=\{e_1,e_2,e_3,e_4\}$,

$\rho(e_1)=(v_1,v_2)$,$\rho(e_2)=(v_1,v_2)$,$\rho(e_3)=(v_3,v_2)$,$\rho(e_4)=(v_4,v_2)$,

$\lambda(v_1)$=Director,$\lambda(v_2)$=Movie,$\lambda(v_3)$=Actor,$\lambda(v_4)$=Actor,

$\lambda(e_1)$=directs,$\lambda(e_2)$=acts_in,$\lambda(e_3)$=acts_in,$\lambda(e_4)$=acts_in,

$\sigma(v_1$,name)="James Cameron",$\sigma(v_1$,birthDate)=1954-08-16,$\sigma(v_1$,networth)=1.79E9,

$\sigma(v_2$,label)="Titanic(1997 film)",$\sigma(v_2$,length)=195,$\sigma(v_2$,budget)=2.0E8,

$\sigma(v_3$,name)="Leonardo DiCaprio",$\sigma(v_3$,birthDate)=1974-11-11,$\sigma(v_4$,name)="Kate Winslet",
$\sigma(v_4$,birthDate)=1975-10-05,$\sigma(e_2$,role)="Steerage Dancer",$\sigma(e_3$,role)="Jack Dawson",
$\sigma(e_4$,role)="Rose DeWitt".

在定义 2 中,每个顶点或边上只能有一个标签,每个属性也只能具有一个值;如果允许顶点或边上具有多个标签,可将定义中的函数λ改为$\lambda:(V\cup E)\rightarrow \mathbb{P}(Lab)$,如果允许属性对应多个值,可将函数$\sigma$改为$\sigma:(V\cup E)\times Prop\rightarrow \mathbb{P}(Val)$,其中,$\mathbb{P}(X)$表示集合 X 的幂集.在实际中,不同图数据库管理系统可能有不同规定,例如,在 Neo4j 实现的属性图模型中,顶点上可以有多个标签,边上只能有一个标签,每个属性只有一个值.

例 3:图 5 给出了一个关系更为复杂的社交网络知识图谱的属性图模型,其中,顶点标签有 3 种:Person、Post 和 Tag;边标签有 5 种:knows、likes、dislikes、hasTag 和 follower.Person 顶点上可有属性 firstName、lastName、gender 和 country;Post 顶点上可有属性 text 和 lang;Tag 顶点上可有属性 label;likes 和 dislikes 边上可有属性 date.注意到图中存在回路(cycle),如 $v_4e_1v_1e_5v_5e_8$ 和 $v_2e_3v_3e_4$.

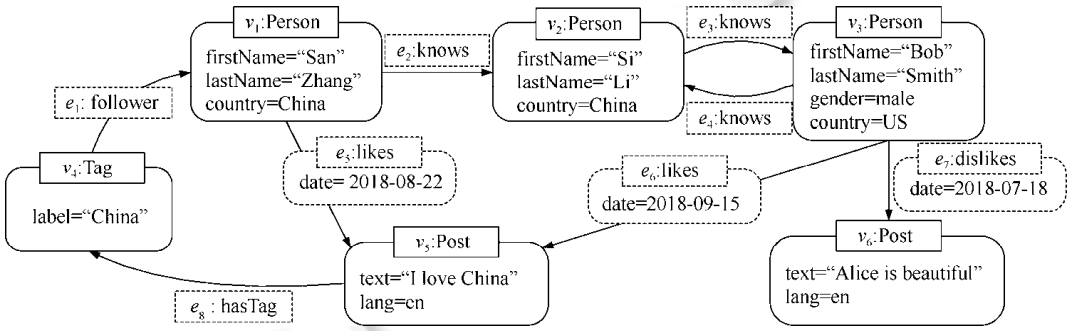

Fig.5 A property graph example: Social network knowledge graph

图5 属性图示例:社交网络知识图谱

1.3 两种数据模型比较

表 1 给出了 RDF 图模型和属性图模型这两种主流知识图谱数据模型的比较,包括数据模型的结构、操作和约束 3 个方面.RDF 图模型的表达力强于属性图模型,是因为 RDF 的超图本质,一条三元组的谓语可在另一条三元组中作主语或宾语,而属性图中顶点和边属性上却不能再定义属性[31].总体说来,由于 RDF 图具有加强的逻辑理论背景,加之语义 Web 多年的标准化工作,其数据模型特性相对完善,但也正是因为理论性过强而影响了其在工业界的推广;属性图虽然在理论基础方面还不够完善,但是随着 Neo4j 等图数据库的应用,其获得了较强的用户认可度.

Table 1 The comparison of the RDF graph model and the property graph model

表1 RDF 图模型和属性图模型的比较

数据模型特性		RDF 图模型	属性图模型
结构	标准化程度	已由 W3C 制定了标准化的语法和语义[23]	尚未形成工业标准
	数学模型	3-均匀有向标签超图	有向标签属性图
	表达力	RDF 图模型强于属性图模型	属性图模型弱于 RDF 图模型
	边属性表达	通过额外方法,如“具体化”	内置支持
	概念层本体定义	RDFS[26]、OWL[27]	不支持
	串行化格式	XML[32]、JSON[33]、N-Triples[34]、Turtle[35]等	CSV
操作	查询代数	SPARQL 代数[36]	无
	查询语言	SPARQL[36]	Cypher[37]、Gremlin[38]、PGQL[39]、G-CORE[30]
约束	约束语言	RDF Shapes 约束语言(SHACL)[1]	无

2 知识图谱查询语言

知识图谱查询语言主要实现了知识图谱数据模型的操作部分.目前,RDF 图的标准查询语言是 SPARQL;属

性图查询语言主要有 Cypher、Gremlin、PGQL 和 G-CORE.从查询语言的类型看,除了 Gremlin 属于过程式(procedural)语言外,SPARQL、Cypher、PGQL 和 G-CORE 均属于声明式(declarative)语言.下面分别加以介绍.关于各种知识图谱查询语言的比较请参见后文的表 5.

2.1 SPARQL

SPARQL 是 W3C 制定的 RDF 知识图谱标准查询语言[36].SPARQL 从语法上借鉴了 SQL.SPARQL 查询的基本单元是三元组模式(triple pattern),多个三元组模式可构成基本图模式(basic graph pattern).SPARQL 支持多种运算符,将基本图模式扩展为复杂图模式(complex graph pattern).SPARQL 1.1 版本引入了属性路径(property path)机制[40]以支持 RDF 图上的导航式查询.下面使用图 2 所示的电影知识图谱 RDF 图,通过示例介绍 SPARQL 语言的基本功能.

例 4:查询 1950 年之后出生的资产大于 1.0E9 的导演执导的电影的出演演员.

```
SELECT?x4?x5
WHERE {?x1 rdf:type Director.?x1 birthDate?x2.FILTER (?x2>=1950-01-01).
       ?x1 networth?x3.FILTER(?x3>1.0E9).?x1 directs?x4.?x5 acts_in?x4.}
```

查询结果:

?x4	?x5
Titanic	Leonardo_DiCaprio
Titanic	Kate_Winslet
Titanic	James_Cameron

SELECT 子句指明要返回的结果变量;WHERE 子句指明查询条件;“?x1 rdf:type Director.”是三元组模式,其中,rdf:type 和 Director 是常量,?x1 是变量(SPARQL 中的变量以?开头),句点表示一条三元组模式的结束.图 6 给出了例 4 对应的查询图,可以看出,每个三元组模式对应一条边,由 5 个三元组模式构成了基本图模式;关键字 FILTER 用于指明过滤条件,对变量匹配结果按条件进行筛选;加上了 FILTER 过滤后基本图模式,最后构成复杂图模式查询.

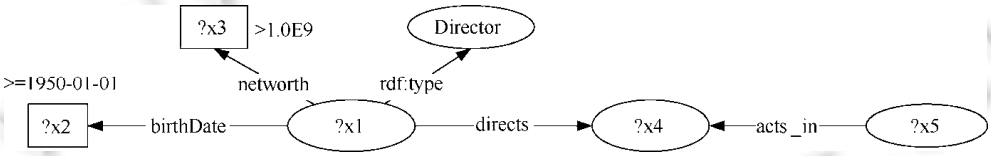

Fig.6 A complex graph pattern query: A SPARQL example

图6 复杂图模式查询:SPARQL 示例

例 5:查询具有有限多步“合作距离(collaborative distance)”的两名演员.

```
SELECT?x1?x2
WHERE{?x1 (acts_in/^acts_in)* ?x2.FILTER (?x1!=?x2).?x1 rdf:type Actor.?x2 rdf:type Actor.}
```

查询结果:

?x1	?x2
Leonardo_DiCaprio	Kate_Winslet
Kate_Winslet	Leonardo_DiCaprio

该查询用到了 SPARQL 1.1 中的属性路径功能实现导航式查询,实际上,(acts_in/^acts_in)*是正则表达式,匹配 0 步到多步“合作距离”,其中,运算符/表示路径连接(path concatenation),运算符^表示反向路径(inverse path),运算符*表示克林闭包(Kleene closure).使用 FILTER 将?x1 和?x2 匹配到同一演员的情况过滤掉.

SPARQL 经过 W3C 的标准化过程,具有精确定义的语法和语义.完整的 SPARQL 1.1 语法与语义请参见文献[36](W3C 推荐标准).文献[41]最早给出了 SPARQL 语义、复杂度和表达力相关的理论结果.W3C 推荐标准中图模式是包语义(bag semantics),属性路径中的闭包算子(即*和$^{+}$)是集合语义(set semantics).文献[42,43]的研究工作直接影响了 SPARQL 1.1 属性路径语义的确定,即用“存在式(existential)”的集合语义取代了之前草案中“数

路径式(counting paths)”的包语义,从而降低了 SPARQL 属性路径的计算复杂度.同时,SPARQL 1.1 中还包括专门用于 RDF 图数据更新和管理的子语言 SPARQL 1.1 Update[44].

2.2 Cypher

Cypher 最初是图数据库 Neo4j 中实现的属性图数据查询语言.2015 年,Neo4j 公司发起开源项目 open Cypher(https://www.opencypher.org),旨在对 Cypher 进行标准化工作,为其他实现者提供语法和语义的参考标准.虽然 Cypher 的发展目前仍由 Neo4j 主导,但包括 SAP HANA Graph[45]、Redis Graph[46]、AgensGraph[47]和 Memgraph[48]等在内的图数据库产品已经实现了 Cypher.Cypher 的一个主要特点是使用“ASCII 艺术(ASCII art)”语法表达图模式匹配.下面通过例子介绍 Cypher 语言的基本功能.使用的图数据是图 4 所示的电影知识图谱和图 5 所示的社交网络知识图谱.

例 6:查询 1950 年之后出生的资产大于 1.0E9 的导演执导的电影的出演演员.

```
MATCH (x1:Director)-[:directs]->(:Movie)<-[:acts_in]-(x2:Actor)
WHERE x1.networth>1.0E9 AND x1.birthDate>=date("1950-01-01")
RETURN x2
```

查询结果:

x2
{"name":"Kate Winslet","birthDate":"1975-10-05"}
{"name":"Leonardo DiCaprio","birthDate":"1974-11-11"}

本例与例4 的 SPARQL 查询问题相同.MATCH 子句用于指明要匹配的图模式,顶点信息写在圆括号“( )”中,边信息写在方括号“[ ]”中,属性信息写在花括号“{ }”中,用“:”分开顶点(或边)变量和标签.在本例中,“(x1:Director)”表示该图模式顶点要匹配的数据图顶点标签为 Director,且变量 x1 会绑定到匹配结果顶点;“-[:directs]->”表示要匹配标签为 directs 的有向边;MATCH 子句引导的图模式是一个以“(:Movie)”为中心、由两条边构成的星形图模式.这些语法说明了 Cypher 如何使用 ASCII 艺术形式表达图查询.WHERE 关键字与 MATCH 子句配合使用,用于指定图模式匹配的约束条件.内置函数 date 用于将字符串转化为日期类型.RETURN 子句用于返回结果变量.本例等价于 SPARQL 基本图模式查询.

例 7:查询 San Zhang 直接和间接认识的人.

```
MATCH (x1:Person)-[:knows*]->(x2:Person)
WHERE x1.firstName="San" AND x1.lastName="Zhang"
RETURN x1, x2
```

查询结果:

x1	x2
{"country":"China","firstName":"San","lastName":"Zhang"}	{"firstName":"Si","lastName":"Li","country":"China"}
{"country":"China","firstName":"San","lastName":"Zhang"}	{"firstName":"Bob","lastName":"Smith","country":"US","gender":"male"}
{"country":"China","firstName":"San","lastName":"Zhang"}	{"firstName":"Si","lastName":"Li","country":"China"}

本例使用图 5 作为知识图谱,展示了 Cypher 的导航式查询功能.Cypher 通过变长模式(variable-length pattern)匹配对导航式查询提供有限支持.不同于 SPARQL 属性路径,Cypher 变长模式仅实现了正则路径查询(regular path query)的一个子集,即传递闭包算子*只能作用在单独一个边标签上,如:knows*;对比例 5 中 SPARQL 属性路径对正则表达式的完整支持.

同时,Cypher 还提供了相应的语句进行属性图的数据更新操作,包括图结构更新和属性更新.Cypher 语言的官方文档请参见文献[29].最新的文献[37]给出了 Cypher 当前版本核心查询功能的形式语法和语义定义,并讨论了 Cypher 在下一版本将引入的新特性.

2.3 Gremlin

Gremlin 是 Apache TinkerPop 图计算框架[38]提供的属性图查询语言.Gremlin 是图遍历语言,其执行机制是在图中沿着有向边进行导航式的游走.这种执行方式决定了用户使用 Gremlin 需要指明具体的导航步骤,所以 Gremlin 是过程式语言.与受到 SQL 影响的声明式语言 SPARQL 和 Cypher 不同,Gremlin 更像一种函数式的编

程语言接口.下面通过例子来介绍 Gremlin 语言的基本功能.使用的属性图是图 4 所示的电影知识图谱.

例 8:查询 1950 年之后出生的资产大于 1.0E9 的导演执导的电影的出演演员名字.

```
g.V().hasLabel('Director').has('birthDate',gte('1950-01-01'))
.has('networth',gt(1.0E9)).out('directs').in('acts_in').hasLabel('Actor').values('name')
```

查询结果:

Leonardo DiCaprio

Kate Winslet

本例使用 Gremlin 表达基本图模式查询.g 是图遍历对象,即属性图.函数调用 g.V()返回图中所有顶点集;接着施加 3 个过滤条件,函数 hasLabel('Director')限定顶点标签为 Director,函数 has('birthDate',gte('1950-01-01'))限定顶点上属性 birthDate 值大于等于'1950-01-01',函数 has('networth',gt(1.0E9))限定顶点上属性 networth 值大于 1.0E9,其中,谓词 gte 和 gt 分别表示比较运算符≥和>;函数 out('directs')返回由满足上述条件的顶点集出发,沿有向边 directs 能够到达的顶点集,即 1950 年之后出生的资产大于 1.0E9 的导演执导的电影顶点;函数 in('acts_in')返回从这些电影顶点出发,沿着有向边 acts_in 的反方向能够到达的顶点集;函数 hasLabel('Actor')将不是 Actor 标签的顶点过滤掉,因此结果中不包括 James Cameron(虽然 v_1 到 v_2 也有 acts_in 边 e_2);最后,函数 values('name')返回这些顶点的 name 属性值.由本例可以看出 Gremlin 的图遍历、过程式和函数式风格.下面,我们来对比该查询的 SPARQL(例4)和 Cypher(例6)版本.

例 9:列出演员“Leonardo DiCaprio”与距其有限多步“合作距离”演员之间的全部路径.

```
g.V().hasLabel('Actor').has('name','Leonardo DiCaprio').repeat(
out('acts_in').hasLabel('Movie').in('acts_in').hasLabel('Actor')
).emit().path()
```

查询结果:

[v[3],v[2],v[3]]

[v[3],v[2],v[4]]

[v[3],v[2],v[3],v[2],v[3]]

[v[3],v[2],v[3],v[2],v[4]]

…

本例展示了 Gremlin 的导航式查询.使用函数 repeat 重复执行指定的导航操作;一步“合作距离”导航操作被执行了任意多次;使用 emit()输出每次重复执行的求值结果;使用 path()输出整条导航路径.可以看出操作后得到了无穷多个匹配路径,原因是 Gremlin 默认语义是“任意路径”语义,对于路径中顶点和边的重复出现没有限制.路径查询的不同语义将在第 4 节中给出讨论.

要了解 Gremlin 的全面语法功能,请参见文献[38].目前,还未见关于 Gremlin 形式语义方面的研究工作.

2.4 PGQL

PGQL 是 Oracle 在 2016 年提出的属性图查询语言[39],支持图模式匹配查询和导航式查询.PGQL 在语法结构上参照 SQL 设计,同时查询返回与 SQL 相同的结果集,可将其作为子查询嵌入到 SQL 查询中.PGQL 从 Cypher 借鉴了 ASCII 艺术语法表示图模式;与 Cypher 相比,PGQL 完整地支持正则路径查询语义;与 SPARQL 属性路径仅支持边标签构成的正则路径不同,PGQL 通过路径模式(path pattern)表达式定义,还支持正则路径中含有顶点标签条件以及顶点(或边)属性值比较,在提高了属性图上正则路径查询表达力(expressiveness)的同时保持求值复杂度(complexity)不变.下面给出一个 PGQL 查询示例,请对比例 5 和例 9.

例 10:列出与演员“Leonardo DiCaprio”距离有限多步“合作距离”的演员姓名和生日.

```
PATH collaborate AS ()-[:acts_in]->(:Movie)<-[:acts_in]-()
SELECT x2.name, x2.birthDate
FROM movie_graph
```

```
MATCH (x1:Actor)-/:collaborate*/->(x2:Actor)
WHERE x1.name='Leonardo DiCaprio' AND id(x1) <> id(x2)
```

查询结果:

x2.name	x2.birthDate
Kate Winslet	1975-10-05

查询开头使用 PATH 关键字指定名为“collaborate”路径模式“()-[:acts_in]->(:Movie)<-[:acts_in]-()”;与 SQL 一致,SELECT 子句用于返回查询结果变量,FROM 子句指明图数据,WHERE 子句给出过滤条件,其中,PGQL 内置函数 id(x)返回顶点或边 x 的标识 id;在 MATCH 子句中,“-/:collaborate*/->”表示匹配“collaborate”路径模式任意多次.通过这种方式可以表达出任意复杂的正则路径查询.

目前,PGQL 仅有 Oracle PGX 一种实现版本[49].关于 PGQL 最新版本 1.1 的语法和语义请参见文献[50].

2.5 G-CORE

G-CORE 是由 LDBC 图查询语言工作组(LDBC Graph Query Language Task Force)设计的属性图查询语言.G-CORE 语言的设计目标是充分借鉴和融合各种已有图查询语言的优点,在查询表达力和求值复杂度之间寻求最佳平衡[30].G-CORE 与已有图查询语言相比:(1) 查询的输入输出均是图,彻底实现了图查询语言的可组合性(composability);(2) 将路径作为与顶点和边同等重要的图查询处理基本元素.为此,G-CORE 在属性图模型的基础上进行扩展,定义了路径属性图模型(path property graph model,简称 PPG).

定义 3(路径属性图). PPG 是 7 元组 $G=(V,E,P,\rho,\delta,\lambda,\sigma)$,其中,(1) V 是顶点的有限集合,E 是边的有限集合,P 是路径的有限集合,V,E,P 互不相交;(2) 函数 $\rho:E\to(V\times V)$ 将边关联到顶点对,如 $\rho(e)=(v_1,v_2)$ 表示 e 是从顶点 v_1 到顶点 v_2 的有向边;(3) 设 $Seq(X)$ 表示集合 X 中元素组成的序列,函数 $\delta:P\to Seq(V\cup E)$ 将路径映射到顶点和边交替组成的序列,如 $p\in P,\delta(p)=(v_1,e_1,v_2,\ldots,v_n,e_n,v_{n+1})$,其中,(i) $v_i\in V(1\leqslant i\leqslant n+1)$;(ii) $e_i\in E,\rho(e_i)=(v_i,v_{i+1})$ 或 $\rho(e_i)=(v_{i+1},v_i)(1\leqslant i\leqslant n)$;(4) 设 Lab 是标签集合,函数 $\lambda:(V\cup E\cup P)\to Lab$ 为顶点、边或路径赋予标签;(5) 设 $Prop$ 是属性集合,Val 是值集合,函数 $\sigma:(V\cup E\cup P)\times Prop\to Val$ 为顶点、边或路径关联属性.

从 PPG 的定义可以看出,路径已与顶点和边同为图数据模型中的“一等公民”.与顶点和边相同,路径也可以有标签和属性,路径属性可以描述属于路径的信息,如路径长度、导航开销等.下面给出一个 G-CORE 查询示例,请对比例 9 和例 10.

例 11:列出演员“Leonardo DiCaprio”与距其有限多步“合作距离”演员之间的最短路径及路径长度.

```
PATH collaborate=()-[:acts_in]->(:Movie)<-[:acts_in]-()
CONSTRUCT (x1)-/@p: collaborate_distance {distance:=c}/->(x2)
MATCH (x1:Actor)-/SHORTEST p⟨~collaborate*⟩ COST c/->(x2:Actor)
ON movie_graph
WHERE x1.name='Leonardo DiCaprio' AND x1 !=x2
```

查询结果:

@p
(v3)-[e3]->(v2)<-[e4]-(v4) {distance = 2}

在 G-CORE 中,每个查询均使用 CONSTRUCT 子句返回图作为查询结果,这保证了查询的可组合性,即一个查询的输出可以直接作为另一个查询的输入.PATH 关键字借鉴自 PGQL,用于定义路径模式,以构成任意复杂的正则路径查询,这里定义的路径模式 collaborate 表示两名演员之间的合作关系,即共同出演一部电影.G-CORE 中@前缀引导的变量 p 表示存储路径(stored path),即物化存储在图数据库中的路径;CONSTRUCT 子句构建的图由存储路径@p 组成,@p 的标签为 collaborate_distance,具有属性 distance,由顶点 x1 导航到顶点 x2.MATCH 子句匹配由演员 Leonardo DiCaprio 与其他演员(变量 x2,x1 !=x2 表示不能是同一演员)之间的所有有限多步“合作距离”中的最短路径,即 p⟨~collaborate*⟩,其中,p 是路径变量,~collaborate 是对 PATH 定义的路径模式的引用,*是克林闭包;COST c 表示最短路径代价为 c,默认代价即路径长度,c 作为属性值保存到存储路径@p 的属性 distance 中.可见,查询结果是一条完整的路径信息.

目前,G-CORE 仅有一个开源的语法解析器[51],还没有数据库系统实现.关于 G-CORE 的详细语法和语义请参见文献[52].

3 知识图谱存储管理

首先介绍基于关系的知识图谱存储机制,然后给出两种典型的原生知识图谱数据库的底层存储.

3.1 基于关系的知识图谱存储管理

关系数据库目前仍是使用最多的数据库管理系统.基于关系数据库的存储方案是目前知识图谱数据的一种主要存储方法.本小节将按照时间发展顺序依次介绍各种基于关系的知识图谱存储方案,包括:三元组表、水平表、属性表、垂直划分、六重索引和 DB2RDF.

3.1.1 三元组表

三元组表(triple table)是将知识图谱存储到关系数据库的最简单、最直接的办法,就是在关系数据库中建立一张具有 3 列的表,该表的模式为

triple_table(subject,predicate,object)

subject、predicate 和 object 这 3 列分别表示主语、谓语和宾语;将知识图谱中的每条三元组存储为三元组表 triple_table 中的一行记录.

例 12:图 7是图 2所示电影知识图谱对应的三元组表,一共有 16 行,限于篇幅,仅列出前 7 行.

triple_table

subject	predicate	object
James_Cameron	type	Director
James_Cameron	name	"James Cameron"
James_Cameron	birthDate	1954-08-16
James_Cameron	networth	1.79E9
James_Cameron	directs	Titanic
James_Cameron	acts_in	Titanic
Titanic	type	Movie
…	…	…

Fig.7 Triple table: An example

图7 三元组表示例

三元组表存储方案虽然简单明了,但三元组表的行数与知识图谱的边数相等,其最大问题在于将知识图谱查询翻译为 SQL 查询后会产生三元组表的大量自连接操作.例如,例 4 的 SPARQL 查询翻译为等价的 SQL 查询后如例 13 所示.一般自连接的数量与 SPARQL 中三元组模式数量相当.当三元组表规模较大时,多个自连接操作将影响 SQL 查询性能.

采用三元组表存储方案的代表是 RDF 数据库系统 3store[53].

例 13:在三元组表存储方案中,将例 4 的 SPARQL 查询转化为等价的 SQL 查询.三元组表的表名为 t.

```
SELECT t4.object, t5.subject
FROM t AS t1, t AS t2, t AS t3, t AS t4, t AS t5
WHERE t1.subject=t2.subject AND t2.subject=t3.subject AND t3.subject=t4.subject
  AND t4.object=t5.object AND t1.predicate='rdf:type' AND t1.object='Director'
  AND t2.predicate='birthDate' AND t2.object>='1950-01-01' AND t3.predicate='networth'
  AND t3.object>1.0E9 AND t4.predicate='directs' AND t5.predicate='acts_in'
```

3.1.2 水平表

水平表(horizontal table)存储方案同样非常简单.水平表的每行记录存储知识图谱中一个主语的所有谓语和宾语.实际上,水平表相当于知识图谱的邻接表.水平表的列数是知识图谱中不同谓语的数量,行数是知识图谱中不同主语的数量.

例 14:图 8 是图 2 中电影知识图谱对应的水平表,共有 4 行、10 列.

horizontal_table

subject	type	name	birthDate	networth	label	budget	length	directs	acts_in
James_Cameron	Director	James Cameron	1954-08-16	1.79E9				Titanic	Titanic
Titanic	Movie				Titanic (1997 film)	2.0E8	195		
Leonardo_DiCaprio	Actor	Leonardo DiCaprio	1974-11-11						Titanic
Kate_Winslet	Actor	Kate Winslet	1975-10-05						Titanic

Fig.8 Horizontal table: An example

图8 水平表示例

在水平表存储方案中,例 4 中的 SPARQL 查询可以等价地翻译为例 15 中的 SQL 查询.可见,与三元组表相比,水平表存储方案使得查询大为简化,自连接操作由 4 个减少到 2 个.

例 15:在水平表存储方案中,将例 4 的 SPARQL 查询转化为等价的 SQL 查询.水平表的表名为 t.

```
SELECT t2.subject,t3.subject
FROM t AS t1,t AS t2,t AS t3
WHERE t1.type='Director' AND t1.birthDate>='1950-01-01' AND t1.networth>1.0E9
  AND t1.directs=t2.subject AND t3.acts_in=t2.subject
```

但是水平表的缺点在于:(1) 所需列的数目等于知识图谱中不同谓语数量,在真实知识图谱数据集中,不同谓语数量可能为几千个到上万个,很可能超出关系数据库所允许的表中列数目上限;(2) 对于一行来说,仅在极少数列上具有值,表中存在大量空值,空值过多会影响表的存储、索引和查询性能;(3) 在知识图谱中,同一主语和谓语可能具有多个不同宾语,即一对多联系或多值属性,而水平表的一行一列上只能存储一个值,无法应对这种情况(可以将多个值用分隔符连接存储为一个值,但这违反了关系数据库设计的第一范式);(4) 知识图谱的更新往往会引起谓语的增加、修改或删除,即水平表中列的增加、修改或删除,这是对于表结构的改变,成本很高.

采用水平表存储方案的代表是早期的 RDF 数据库系统 DLDB[54].

3.1.3 属性表

属性表(property table)存储方案是对水平表的细分,将同类主语存到一个表中,解决了表中列数目过多的问题.例 16 给出了图 2 所示电影知识图谱对应的属性表存储方案,分为 director(导演)、movie(电影)和 actor(演员)3 个表.

例 16:图 9 是图 2 中电影知识图谱对应的属性表存储方案,共由 3 个表组成.

director

subject	type	name	birthDate	networth
James_Cameron	Director	James Cameron	1954-08-16	1.79E9

movie

subject	type	label	budget	length
Titanic	Movie	Titanic (1997 film)	2.0E8	195

actor

subject	type	birthDate
Leonardo_DiCaprio	Actor	1974-11-11
Kate_Winslet	Actor	1975-10-05

Fig.9 Property table: An example

图9 属性表存储方案示例

在属性表存储方案中,例 4 中的 SPARQL 查询可以等价地翻译为例 17 中的 SQL 查询.该查询与水平表上的等价查询(例 15)相比,t1 变为了 director,t2 变为了 movie,t3 变为了 actor,由自连接转变为多表连接,而且每个表对应于一个类型,省去了类型(type 列)的判断,提高了查询的可读性.

例 17:在属性表存储方案中,将例 4 的 SPARQL 查询转化为等价的 SQL 查询.

```
SELECT movie.subject,actor.subject
FROM director,movie,actor
WHERE director.birthDate>='1950-01-01' AND director.networth>1.0E9
```

AND director.directs=movie.subject AND actor.acts_in=movie.subject

属性表既克服了三元组表的自连接问题,又解决了水平表中列数目过多的问题.实际上,水平表就是属性表的一种极端情况,即水平表是将所有主语划归为一类,因此属性表中的空值问题得到很大的缓解.但属性表仍存在如下一些缺点:(1) 对于规模稍大的真实知识图谱数据,主语的类别可能有几千到上万个,需要建立几千到上万个表,这往往超过了关系数据库的限制;(2) 即使在同一类型中,不同主语具有的谓语集合也可能差异较大,会造成与水平表中类似的空值问题;(3) 水平表中存在的一对多联系或多值属性存储问题在属性表中仍然存在.

采用属性表存储方案的代表系统是 RDF 三元组库 Jena[55].

3.1.4 垂直划分

垂直划分(vertical partitioning)存储方案是由美国麻省理工学院 Abadi 等人在 2007 年提出的 RDF 数据存储方法[56].该方法为每种谓语建立一张两列的表(subject,object),表中存放知识图谱中由该谓语连接的主语和宾语,表的总数量即知识图谱中不同谓语的数量.例 18 给出了图 2 所示电影知识图谱对应的垂直划分存储方案,9 种谓语对应着 9 张表,每张表都只有主语和宾语列.

例 18:图 10 是图 2 所示电影知识图谱对应的垂直划分存储方案,共由 9 个表组成.

type

subject	object
James_Cameron	Director
Titanic	Movie
Leonardo_DiCaprio	Actor
Kate_Winslet	Actor

name

subject	object
James_Cameron	James Cameron

birthDate

subject	object
James_Cameron	1954-08-16
Leonardo_DiCaprio	1974-11-11
Kate_Winslet	1975-10-05

networth

subject	object
James_Cameron	1.79E9

label

subject	object
Titanic	Titanic (1997 film)

budget

subject	object
Titanic	2.0E8

length

subject	object
Titanic	195

directs

subject	object
James_Cameron	Titanic

acts_in

subject	object
James_Cameron	Titanic
Leonardo_DiCaprio	Titanic
Kate_Winslet	Titanic

Fig.10 Vertical partitioning: An example

图10 垂直划分存储方案示例

在垂直划分存储方案中,例 4 中的 SPARQL 查询可以等价地翻译为例 1 中的 SQL 查询.该查询涉及到 5 张谓语表的连接操作,其中有 3 个“subject-subject”等值连接.由于表中的行都按 subject 列进行排序,可快速执行这种垂直划分方案中最常用的连接操作.

例 19:在垂直划分存储方案中,将例 4 的 SPARQL 查询转化为等价的 SQL 查询.

```
SELECT directs.object,acts_in.subject
FROM type,birthDate,networth,directs, acts_in
WHERE type.subject=birthDate.subject AND birthDate.subject=networth.subject
  AND networth.subject=directs.subject AND acts_in.object=directs.object
  AND type.object='Director' AND birthDate.object>='1950-01-01' AND networth.object>1.0E9
```

与之前基于关系数据库的知识图谱存储方案相比,垂直划分有一些突出的优点:(1) 谓语表仅存储出现在知识图谱中的三元组,解决了空值问题;(2) 一个主语的一对多联系或多值属性存储在谓语表的多行中,解决了多值问题;(3) 每个谓语表都按主语列的值进行排序,能够使用归并排序连接(merge-sort join)快速执行不同谓语表的连接查询操作.

不过,垂直划分存储方案依然存在如下几个缺点:(1) 需要创建的表的数目与知识图谱中不同谓语数目相等,而大规模的真实知识图谱(如,DBpedia、YAGO、WikiData 等)中谓语数目可能超过几千个,在关系数据库中维护如此规模的表需要花费很大开销;(2) 越是复杂的知识图谱查询操作,需要执行的表连接操作数量越多,而

对于未指定谓语的三元组查询,将发生需要连接全部谓语表进行查询的极端情况;(3) 谓语表的数量越多,数据更新维护代价越大,对于一个主语的更新将涉及多张表,产生很高的更新时 I/O 开销.

采用垂直划分存储方案的代表数据库是 SW-Store[57].

3.1.5 六重索引

六重索引(sextuple indexing)存储方案是对三元组表的扩展,是一种典型的"空间换时间"策略,其将三元组全部 6 种排列对应地建立为 6 张表,即 spo(主语,谓语,宾语)、pos(谓语,宾语,主语)、osp(宾语,主语,谓语)、sop(主语,宾语,谓语)、pso(谓语,主语,宾语)和 ops(宾语,谓语,主语).不难看出,其中 spo 表就是原来的三元组表.六重索引通过 6 张表的连接操作不仅缓解了三元组表的单表自连接问题,而且提高了某些典型知识图谱查询的效率.

六重索引方案的优点有:(1) 知识图谱查询中的每种三元组模式查询都可以直接使用相应的索引进行快速前缀范围查找,表 2 给出了全部 8 种三元组模式查询能够使用的索引;(2) 可以通过不同索引表之间的连接操作直接加速知识图谱上的连接查询.

Table 2 Triple pattern queries and usable indexes in sextuple indexing

表2 六重索引方案下三元组模式查询和可使用的索引表

序号	三元组模式查询	可用索引
1	(s,p,o)	spo,pos,osp,sop,pso,ops
2	(s,p,?x)	spo,pso
3	(s,?x,o)	sop,osp
4	(?x,p,o)	pos,ops
5	(s,?x,?y)	spo,sop
6	(?x,?y,o)	osp,ops
7	(?x,p,?y)	pos,pso
8	(?x,?y,?z)	spo,pos,osp,sop,pso,ops

六重索引存储方案存在的问题包括:(1) 虽然部分缓解了三元组表的单表自连接问题,但需要花费 6 倍的存储空间开销、索引维护代价和数据更新时的一致性维护代价,随着知识图谱规模的增大,该问题会愈加突出;(2) 当知识图谱查询变得复杂时,会产生大量的连接索引表查询操作,依然不可避免索引表的自连接.

使用六重索引方法的典型系统有 RDF-3X[58]和 Hexastore[59].

3.1.6 DB2RDF

DB2RDF 是由 IBM 于 2013 年提出的一种面向实体的 RDF 知识图谱存储方案[60,61],该方案是以往 RDF 关系存储方案的一种权衡与折中,既具备了三元组表、属性表和垂直划分方案的部分优点,又克服了这些方案的部分缺点.三元组表的优势在于"行维度"上的灵活性,即存储模式不会随行的增加而变化;DB2RDF 方案将这种灵活性扩展到"列维度"上,即将表的列作为谓语和宾语的存储位置,而不将列与谓语进行绑定.插入数据时,将谓语动态映射存储到某列;方案能够确保将相同谓语映射到同一组列上.

DB2RDF 存储方案由 4 张表组成,即:dph 表、rph 表、ds 表和 rs 表;例20 给出了图 2所示电影知识图谱对应的 DB2RDF 存储方案.dph(direct primary hash)是存储方案的主表,该表中一行存储一个主语(subject 列)及其全部谓语(pred$i$ 列)和宾语(val$i$ 列),$0\leqslant i\leqslant k$,k 一般是关系数据库支持的表中最大列数目;如果一个主语的谓语数量大于 k,则一行不足以容纳下一个实体,将在下一行存储第 k+1 到 2k 个谓语和宾语,以此类推,这种情况叫作溢出(spill);spill 列是溢出标志,即对于一行能存储下的实体,该行 spill 列为 0,对于溢出的实体,该实体所有行的 spill 列为 1.例如,在例20 的 dph 表中,实体 James_Cameron 溢出,其余实体均未溢出.

对于多值谓语的处理,引入 ds(direct secondary hash)表.当 dph 表中遇到一个多值谓语时,则在相应的宾语处生成一个唯一的 id 值;将该 id 值和每个对应的宾语存储为 ds 表的一行.例如,在图 2 的基础上添加三元组(James_Cameron,directs,Avatar),这时,directs 就成为多值谓语,在例 20 的 dph 表中,在其宾语列 val2 中存储 id 值 lid:1;在 ds 表中存储 lid:1 关联的两个宾语 Titanic 和 Avatar.

实际上,dph 表实现了列的共享:一方面,不同实体的相同谓语总是会被分配到相同列上;另一方面,同一列中可以存储多个不同的谓语.比如,主语 Leonardo_DiCaprio 和 Kate_Winslet 的谓语 acts_in 都被分配到 pred4 列,同时,该列还存储了主语 Titanic 的谓语 length.正是由于 DB2RDF 方案具备"列共享"机制,才使得在关系表中最

大列数目上限的情况下可以存储远超出该上限的谓语数目,也能够有效地解决水平表方案中存在的谓语稀疏性空值问题.在真实的知识图谱中,不同主语往往具有不同的谓语集合,例如,谓语 birthDate 只有人(person)才具有,谓语 budget 只有电影(movie)才具有,这也是能够实现列共享的原因所在.

例 20:图 11 是图 2 所示电影知识图谱对应的 DB2RDF 存储方案(rs 表示省略).

dph

subject	spill	pred1	val1	pred2	val2	pred3	val3	pred4	val4
James_Cameron	1	type	Director	name	James Cameron	birthDate	1954-08-16	networth	1.79E9
James_Cameron	1			directs	lid:1			acts_in	Titanic
Titanic	0	type	Movie	label	Titanic (1997 film)	budget	2.0E8	length	195
Leonardo_DiCaprio	0	type	Actor	name	Leonardo DiCaprio	birthDate	1974-11-11	acts_in	Titanic
Kate_Winslet	0	type	Actor	name	Kate Winslet	birthDate	1975-10-05	acts_in	Titanic

rph

subject	spill	pred1	val1	…	predk	valk
Director	0	rdf:type	James_Cameron	…		
1954-08-16	0	birthDate	James_Cameron	…		
…	…	…	…	…	…	…
Titanic	0	acts_in	lid:2	…	directs	James_Cameron

ds

lid	elm
lid:1	Titanic
lid:1	Avatar

rs

lid	elm
lid:2	Leonardo_DiCaprio
lid:2	Kate_Winslet
lid:2	James_Cameron

Fig.11 DB2RDF: An example

图11 DB2RDF 存储方案示例

从知识图谱数据模型的角度来看,dph 表和 ds 表实际上存储了实体顶点(主语)的出边信息(从主语经谓语到宾语);为了提高查询处理效率,还需要存储实体顶点的入边信息(从宾语经谓语到主语).为此,DB2RDF 方案提供了 rph(reverse primary hash)表和 rs(reverse secondary hash)表.

例 21:在 DB2RDF 存储方案中,将例 4 的 SPARQL 查询转化为等价的 SQL 查询.

```
SELECT t4.subject,t5.elm
FROM dph AS t1,dph AS t2,ds AS t3,rph AS t4,ds AS t5
WHERE t1.subject=t2.subject AND t2.val2=t3.lid AND t3.elm=t4.subject AND t4.val1=t5.lid
  AND t1.pred1='type' AND t1.val1='Director' AND t1.pred3='birthDate' AND t1.val3>='1950-01-01'
  AND t1.pred4='networth' AND t1.val4>1.0E9 AND t2.pred2='directs' AND t4.pred1='acts_in'
```

在 DB2RDF 方案中,谓语到列的映射是需要重点考虑的问题.因为关系表中最大列的数目是固定的,该映射的两个优化目标是:(1) 使用的列的数目不要超过某个值 m;(2) 尽量减少将同一主语的两个不同谓语分配到同一列的情况,从而减少溢出现象,因为溢出会导致查询时自连接的发生.

谓语到列映射的一种方法是使用一组哈希函数,将谓语映射到一组列编号,并将谓语及其宾语存储到这组列中的第 1 个空列上;在一个主语对应的一行中,如果存储某谓语(及其宾语)时,哈希函数计算得出的这组列中的所有列都被之前存储的该主语的谓语占用,则产生溢出,到下一行存储该谓语.例如,表 3 给出了谓语到列映射的哈希函数表,其中包括 h_1 和 h_2 两个哈希函数,映射了 5 个谓语到列编号组.比如,当存储到三元组(James_Cameron,directs,Titanic)时,谓语 directs 被 h_1 映射到列 pred2,被 h_2 映射到列 pred3,但这两列都已被占用,这时产生溢出,将谓语 directs 溢出到下一行的列 pred2 中存储,如图 11的 dph 表所示.

Table 3 Predicate-to-column hash function table

表3 谓语到列映射的哈希函数表

谓语	h_1	h_2
type	1	3
name	2	1
birthDate	3	4
networth	4	2
directs	2	3

如果可以事先获取知识图谱的一个子集,则可以利用知识图谱的内在结构优化谓语到列的映射.方法是将谓语到列的映射转化为图着色(graph coloring)问题.将一个主语上出现的不同谓语称为共现谓语(co-occurrence predicates),目标是让共现谓语着上不同颜色(映射到不同列中),非共现谓语可以着上相同颜色(映射到同一列中),并使所用颜色数最少.需要指出的是,虽然图着色是经典的 NP 难问题,但即使是真实知识图谱,其不同谓语总数也是相对较少的(一般在几千个规模上).

如果在大规模真实知识图谱中(如 DBpedia),图着色所需颜色数量超过了关系数据表的列数上限 m,则根据某种策略(如最频繁使用的前 k 个谓语)选取一个谓语子集,使得该谓语子集到列的映射满足图着色要求;对于不在该子集中的谓语,再使用前面提到的哈希函数组策略进行映射.

DB2RDF 存储方案已经实现到了最新的 IBM DB2 数据库系统中[61].

3.2 原生知识图谱存储管理

不同于基于关系的存储方案,原生知识图谱存储是指专门为知识图谱而设计的底层存储管理方案.不同原生知识图谱数据库的物理和逻辑存储层设计与实现也各有差异.本小节选取两种具有代表性的原生知识图谱存储管理方案进行介绍:一种是面向属性图的 Neo4j 存储;另一种是面向 RDF 图的 gStore 存储.

3.2.1 Neo4j

Neo4j 是目前最流行的属性图数据库,其原生图存储层的最大特点是具有“无索引邻接(index-free adjacency)”特性.所谓“无索引邻接”是指,每个顶点维护着指向其邻接顶点的直接引用,相当于每个顶点都可看作是其邻接顶点的一个“局部索引”,用其查找邻接顶点比使用“全局索引”节省大量时间.这就意味着图导航操作代价与图大小无关,仅与图的遍历范围成正比.

为了实现“无索引邻接”,Neo4j 将边也作为数据库的“一等公民”(即数据库中最基本、最核心的概念,如关系数据库中的“关系”),属性图的顶点、边、标签和属性被分开存储在不同文件中.正是这种将图结构与图上标签和属性分开存储的策略,使得 Neo4j 具有高效率的图遍历能力.图 12 给出了 Neo4j 2.2 版本中顶点和边记录的物理存储结构(其他版本可能有变化),其中每个顶点记录占用 15 字节,每个边记录占用 34 字节.

顶点记录的第 0 字节 inUse 是记录使用标志字节,表示该记录是正在使用中还是已经删除并可回收用来装载新记录;第 1 字节~第 4 字节 nextRelId 是与顶点相连的第 1 条边的 id;第 5 字节~第 8 字节 nextPropId 是顶点的第 1 个属性的 id;第 9 字节~第 13 字节 labels 是指向顶点标签存储的指针,若标签较少会直接存储在此处;第 14 字节 extra 用于存储一些内部使用的标志信息.

边记录第 0 字节 inUse 的含义与顶点记录相同,是表示是否正被数据库使用的标志;第 1 字节~第 4 字节 firstNode 和第 5 字节~第 8 字节 secondNode 分别是该边的起始顶点 id 和终止顶点 id;第 9 字节~第 12 字节 relType 是指向该边的关系类型的指针;第 13 字节~第 16 字节 firstPrevRelId 和第 17 字节~第 20 字节 firstNext RelId 分别为指向起始顶点上前一个和后一个边记录的指针;第 21 字节~第 24 字节 secPrevRelId 和第 25 字节~第 28 字节 secNextRelId 分别为指向终止顶点上前一个和后一个边记录的指针;指向前后边记录的 4 个指针形成了两个“关系双向链”;第 29 字节~第 32 字节 nextPropId 是边上的第 1 个属性的 id;第 33 字节 firstInChain Marker 是表示该边记录是否是“关系链”中第 1 条记录的标志.

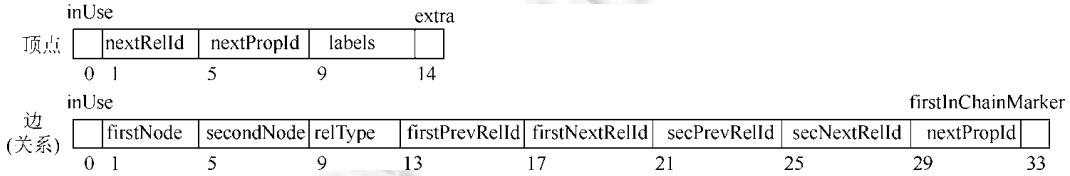

Fig.12 Physical storage structures of vertex and edge records in Neo4j

图12 Neo4j 中顶点和边记录的物理存储结构

Neo4j 能够实现顶点和边快速定位的关键是“定长记录(fixed-size record)”存储方案,以及将具有定长记录的图结构与具有变长记录的属性数据分开存储.例如,一个顶点记录长度是 15 字节,如果要查找 id 为 99 的顶点

记录所在位置(id 从 0 开始),则可直接到顶点存储文件第 1485(15×99)个字节处访问(存储文件从第 0 个字节开始).边记录也是“定长记录”,长度为 34 字节.这样,数据库已知记录 id 可以 $O(1)$的代价直接计算其存储地址,从而避免了全局索引中 $O(\log n)$的查找代价.

图 13是图 5所示顶点 v_2 和 v_3 及其出边 e_3、e_4、e_6 和 e_7 的 Neo4j 物理存储结构,展示了 Neo4j 中各种存储文件之间是如何交互的.存储在顶点文件中的顶点 v_2 和 v_3 均有指针(nextPropId)指向存储在属性文件中各自的第 1 个属性记录;也有指针(nextRelId)指向存储在边文件中各自的第 1 条边,分别为边 e_3 和 e_4.若要查找顶点属性,可由顶点找到其第 1 个属性记录,再沿着属性记录的单向链表进行查找;若要查找一个顶点上的边,可由顶点找到其第 1 条边,再沿着边记录的双向链表进行查找;当找到了所需的边记录后,可由该边进一步找到边上的属性;还可由边记录出发访问该边连接的两个顶点记录(图 13所示虚线箭头).需要注意的是,每个边记录实际上维护着两个双向链表,一个是起始顶点上的边,一个是终止顶点上的边,可以将边记录想象为被起始顶点和终止顶点共同拥有,用双向链表的优势在于,不仅可在查找顶点上的边时进行双向扫描,而且支持在两个顶点间高效率地添加和删除边.这些操作除了记录字段的读取,就是定长记录地址的计算,均是 $O(1)$时间的高效率操作.可见,正是由于将边作为“一等公民”、将图结构实现为定长记录的存储方案,赋予了 Neo4j 作为原生图数据库的“无索引邻接”特性.

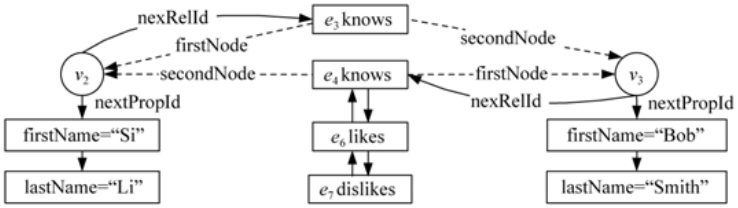

Fig.13 Physical storage structures of property graphs in Neo4j

图13 Neo4j 中属性图的物理存储

3.2.2 gStore

gStore 将 RDF 数据图中每个资源的所有属性和属性值映射到一个二进制位串上.具体而言,对于每个属性或属性值,gStore 都定义一个固定长度的位串并将位串中所有位置为 0.然后,利用若干个预先定义的字符串哈希函数将属性或属性值按照标识符映射到若干个小于位串长度的整数值,进而将位串上这些值所对应的位置置为 1.图 14 给出了图 2 所示 James_Cameron 在 gStore 中所对应编码的示例,每个属性都对应一个 8 位的位串,每个属性值都对应一个 12 位的位串.每个属性都按照其标识符由 2 个字符串哈希函数映射到 2 个小于 8 的正整数.例如,type 通过 2 个哈希函数分别被映射到 3 和 8,然后 ntype 对应的位串第 3 位和第 8 位就被置为 1.同理,每个属性值都按照其标识符由 3 个字符串哈希函数映射到 3 个小于 12 的正整数.如 Director 通过 3 个哈希函数分别被映射到 3、8 和 10,然后 Director 所对应的位串相应位置就被置为 1.

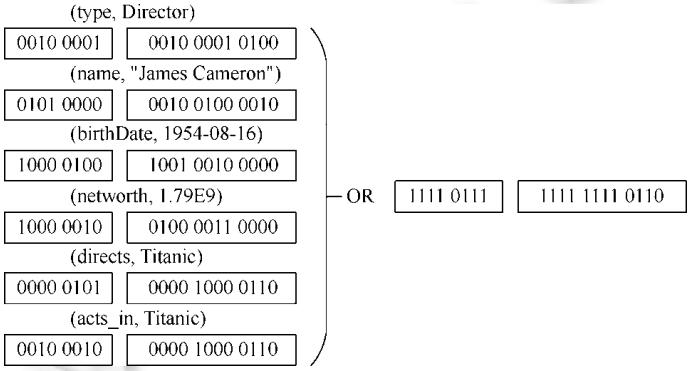

Fig.14 The encoding technique of gStore

图14 gStore 的编码技术

gStore 将所有位串按照 RDF 图结构组织成一棵签章树(signature tree).在签章树的基础上,如果 RDF 知识图谱中两个实体之间有一条边,那么这两个实体所对应的签章树上的点也连上一条边,且这条边被赋上属性的编码.如此,gStore 中所有实体的编码就被组织成一种新的树形索引——VS*树.VS*树被分为若干层,每一层都是 RDF 数据图的摘要.图 15 显示了一个 VS*树的示例.基于 VS*树,gStore 可以完成高效率的数据存储、更新与查询操作.当进行 SPARQL 查询处理时,将每个查询中的变量在这个 VS*树上进行检索,找到相应的候选解,然后再将这些候选解通过连接操作拼接起来.

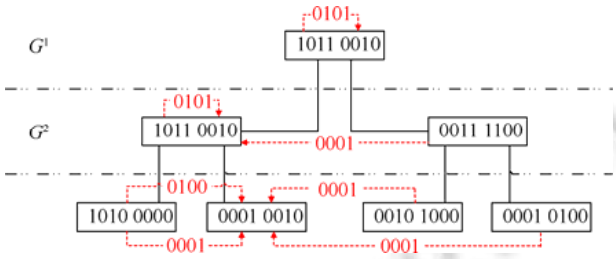

Fig.15 VS*-tree

图15 VS*树

3.3 知识图谱存储方案比较

表 4 给出了本节前面介绍的 8 种知识图谱存储方案的比较.

Table 4 The comparison of knowledge graph storage schemes

表4 知识图谱存储方案的比较

存储方案		优点	缺点	代表性系统	文献
基于关系	三元组表	① 存储结构简单	① 大量自连接操作开销巨大	3store	[53]
	水平表	① 知识图谱的邻接表,存储方案简单	① 可能超出所允许的表中列数目的上限 ② 表中可能存在大量空值 ③ 无法表示一对多联系或多值属性 ④ 谓语的增加、修改或删除成本高	DLDB	[54]
	属性表	① 克服了三元组表的自连接问题 ② 解决了水平表中列数目过多的问题	① 真实知识图谱需建立的关系表数量可能超过上限 ② 由于知识图谱的灵活性,表中可能存在大量空值 ③ 无法表示一对多联系或多值属性	Jena	[55]
	垂直划分	① 解决了空值问题 ② 解决了多值问题 ③ 能够快速执行不同谓语表的连接查询	① 真实知识图谱需维护大量谓语表 ② 复杂知识图谱查询需执行的表连接操作 ③ 数据更新维护代价大	SW-Store	[56]
	六重索引	① 每种三元组模式查询均可直接使用对应索引快速查找 ② 通过不同索引表之间的连接操作直接加速知识图谱上的连接查询	① 需要花费 6 倍的存储空间开销和数据更新维护代价 ② 复杂知识图谱查询会产生大量索引表连接查询操作	RDF-3X Hexastore	[58,59]
	DB2RDF	① 既具备了三元组表、属性表和垂直划分方案的部分优点,又克服了这些方案的部分缺点 ② 实现列维度上的灵活度,为谓语动态分配所在列	① 真实知识图谱可能存在较多溢出情况	IBM DB2	[60,61]
原生	Neo4j	① 具备“无索引邻接”特性 ② 边作为“一等公民” ③ “定长记录”存储方案	① 成熟度不如基于关系的方案	Neo4j	[62]
	gStore	① 基于位串的存储方案 ② “VS 树”索引加快查询	① 成熟度不如基于关系的方案	gStore	[63]

4 知识图谱查询操作

虽然第 2 节中介绍的各种知识图谱查询语言在语法和语义上有所差异,甚至风格迥然,但这些查询语言均具备两种基本的查询机制:图模式匹配(graph pattern matching)和图导航(graph navigation).此外,知识图谱的分析型查询是目前若干图处理框架的内置操作.

4.1 图模式匹配查询

图模式匹配查询是图查询语言中最基本、最重要的一类图查询操作;基本图模式(basic graph pattern,简称BGP)匹配又是图模式匹配查询的核心.下面给出基本图模式的定义,这里仅考虑RDF图和属性图的公共图结构部分,即知识图谱 $G=(V,E)$.

定义 4(基本图模式(BGP)). 给定知识图谱 G,其上的基本图模式 Q 被定义为

$$\wedge_{1\leqslant i\leqslant m}(s_i,p_i,o_i),$$

其中,(1) s_i、p_i 和 o_i 是常量或变量($1\leqslant i\leqslant m$);(2) (s_i,p_i,o_i)是三元组模式($1\leqslant i\leqslant m$);(3) $\wedge$表示逻辑合取.

BGP 实际上就是图上三元组模式的合取,因此也称为合取查询(conjunctive query).BGP 匹配查询存在两种语义定义,即子图同态(subgraph homomorphism)和子图同构(subgraph isomorphism).

定义 5(BGP 匹配的子图同态语义). 给定知识图谱 $G=(V,E)$和其上的 BGP Q $\wedge_{1\leqslant i\leqslant m}(s_i,p_i,o_i)$,设 $\overline{s}=(s_i,\ldots,s_m)$, $\overline{p}=(p_i,\ldots,p_m)$, $\overline{o}=(o_i,\ldots,o_m)$,BGP Q 的子图同态语义定义为:(1) 设σ是从 $\overline{s}$、$\overline{p}$ 和 $\overline{o}$ 到 G 中顶点集 V 的映射;(2) $(G,\sigma)\models Q$ 当且仅当$\forall i\in[1,m]$有$(\sigma(s_i),\sigma(p_i),\sigma(o_i))\in E$;(3) $Q(G)$是所有使$(G,\sigma)\models Q$ 成立的σ集合.

定义 5 给出的是 BGP 匹配的子图同态语义,Q 中的不同变量可以映射到 G 中的同一顶点.另外,还有要求更为严格的子图同构语义.现将这两种语义归纳如下.

(1) 子图同态语义.该语义即是定义 5 对应的语义,找出 G 中与 Q 存在同构关系的全部子图,允许 Q 中多个不同变量映射到 G 中的同一个顶点上.BGP 子图同态语义是定义其他更严格语义的基础,在数据库理论领域已有若干研究工作[64−68].目前,SPARQL 语言中的 BGP 采用的即为子图同态语义[36].例如,在例 5 中,如果去掉 FILTER (?x1 !=?x2),会新增两条匹配,即?x1 和?x2 都映射到顶点 Leonardo_DiCaprio 和 Kate_Winslet.

(2) 子图同构语义.该语义在子图同态语义上增加限制,目的是使得 G 中的映射保持 Q 的结构.按照不同的严格程度,子图同构语义又分为全无重复语义(no-repeated-anything semantics)、顶点无重复语义(no-repeated-vertex semantics)和边无重复语义(no-repeated-edge semantics).

a) 全无重复语义.该语义是在定义5 子图同构语义基础上加上σ是单射(injective)映射的限制,即 BGP Q 中的全部变量都映射到知识图谱的不同顶点或边上.虽然 SPARQL 默认采用子图同态语义,但在例5 中使用额外的条件 FILTER (?x1 !=?x2)获得子图同构全无重复语义.

b) 顶点无重复语义.该语义只限制$\sigma(s_i)$和$\sigma(o_i)$为单射映射,即 BGP Q 中的不同顶点变量映射到知识图谱的不同顶点上,允许不同边变量映射到知识图谱的相同边上.

c) 边无重复语义.该语义需要将定义5 中的σ对顶点和边的映射加以区别对待,对于 p_i 重新定义$\sigma(p_i)$为单射映射,即 BGP Q 中的不同边变量映射到知识图谱的不同边上,允许不同顶点变量映射到知识图谱的相同顶点上.目前,Cypher 语言采用该语义[29].

同态和同构语义的区别是考察一个匹配之内是否允许顶点或边的重复匹配,还需要考虑另一个维度上的语义区别,即知识图谱 G 的所有匹配结果 $Q(G)$中是否允许重复.

(1) 集合语义(set semantics).$Q(G)$定义为匹配的集合,即 $Q(G)$中不能包含重复的匹配.

(2) 包语义(bag semantics).$Q(G)$定义为匹配的包,即一个匹配重复出现的次数等于与该匹配对应的不同映射的数量.

实际上,单就 BGP 本身来讲,不会出现重复的匹配,集合和包语义是等价的.但当 BGP 扩展为复杂图模式后,重复匹配就会出现.下面讨论复杂图模式.

从关系数据管理角度来看,BGP 对应于自然连接.在 BGP 的基础上扩展投影、过滤(选择)、连接、交、差

和可选(左外连接)等其他关系操作,形成复杂图模式(complex graph pattern,简称 CGP).

(1) 投影(projection).$Q(G)$返回的变量集合及其匹配值称为 Q 的输出变量.BGP 默认输出变量为 Q 中所有变量.投影操作返回 BGP 中变量的指定子集作为输出变量.相当于关系代数中的投影操作.

(2) 过滤(filter).指定变量参与的条件表达式,过滤 CGP 匹配结果数量.相当于关系代数中的选择操作.

(3) 连接(join).CGP Q_1 和 Q_2 通过连接操作组成更复杂的 CGP Q_3,Q_3 的输出变量是 Q_1 和 Q_2 输出变量的并集,Q_3 的匹配结果通过连接 Q_1 和 Q_2 的匹配结果获得,即当 Q_1 和 Q_2 输出变量的交集映射到相同常量时,Q_1 和 Q_2 的匹配结果能够进行连接.相当于关系代数中的连接操作.

(4) 并(union)和差(difference).CGP Q_1 和 Q_2 的并(差)构成 CGP Q_3,Q_3 的匹配结果是 Q_1 和 Q_2 的匹配结果的并(差)集.相当于关系代数中的并和差操作.

(5) 可选(optional).CGP Q_1 和 Q_2 的可选操作构成 CGP Q_3,Q_3 的匹配结果中不仅包括 Q_1 和 Q_2 的连接结果,而且包括 Q_1 中与 Q_2 连接不上的结果,即 Q_3 中包括 Q_1 的全部结果.相当于关系代数中的左外连接操作.

已经证明 BGP 求值在各种语义下均为 NP 完全问题.文献[69]通过将图同态(graph homomorphism)问题规约到 BGP 子图同态语义证明了 NP 难;文献[70]通过将子图同构(subgraph isomorphism)问题规约到 BGP 子图同构语义证明了 NP 难.按照查询语言求值复杂度分析传统[71],虽然 BGP 的组合复杂度(combined complexity)是 NP 完全的,但如果将 BGP Q 固定,仅将知识图谱 G 作为输入,其数据复杂度(data complexity)不仅是多项式的,而且还能在对数空间复杂度内完成求值[72].

CGP 求值的组合复杂度在 SPARQL 语言上有大量研究工作.在集合语义下,包含投影、过滤、连接和并操作的 SPARQL CGP 的组合复杂度仍然是 NP 完全的;如果 CGP 还包含差和可选操作,则其相当于关系代数的表达力,组合复杂度升到 PSPACE 完全[71];差操作(SPARQL 中的 MINUS 关键字)已被证明可用可选、过滤和连接操作模拟,所以没有差操作的 CGP 仍然是 PSPACE 完全的[73];甚至仅包含连接和可选操作的 CGP 也是 PSPACE 完全的[74].可见,可选操作是造成 CGP 高复杂度的根源.对于包语义下的 SPARQL,其求值问题的组合复杂度同样是 PSPACE 完全的[74].目前,尚缺少关于其他知识图谱查询语言 CGP 复杂度相关的理论研究工作.

在图模式匹配查询的求值算法方面,数据库领域对于 BGP 有着较长的研究历史,积累了大量工作.求值的经典方法有基于图遍历的 Ullmann[70]、Nauty[75]和 VF2[76],但这些方法的执行效率并不适用于大规模图数据.一般地,大规模图上 BGP 求值的策略有如下 3 种:(1) 采用基于相似度的不精确匹配(inexact matching)语义代替子图同态或同构语义,典型方法包括文献[77−79];(2) 采用近似解(approximate solutions)代替最优解(optimal solutions),典型方法包括文献[80−82];(3) 保持子图同态或同构的精确语义不变,基于“空间换时间”策略构建形式多样的图索引,采用索引削减子图匹配搜索空间,将该类方法按时间顺序列出,主要包括:GraphGrep[83]、gIndex[84]、Closure-tree[85]、FG-Index[86]、Tree+Delta[87]、TreePi[88]、GDI[89]、GCoding[90]、QuickSI[91]、GraphQL[92]、GADDI[93]、SPath[94]、SING[95]、GraphGrepSX[96]、Lindex[97]、PathIndex[98]和 SQBC[99].

策略(1)舍弃了精确子图匹配,策略(2)不能找到最优解,均无法满足知识图谱上对精确查询的需求.策略(3)虽然保持了查询语义,但现有方法存在 3 个方面的问题:(1) 大多数方法针对由若干小图组成的图集合,每个图的顶点和边数为几十到几百,集合中小图的数量为几千到几万(如实验中普遍使用的 AIDS 数据集),这与知识图谱具有百万规模节点和上亿规模边的情形大不相同;(2) 即使是针对单个大图提出的 GraphQL、GADDI 和 SPath 方法,其能够处理的数据规模仍无法满足知识图谱大图数据的要求(如文献[93]中规模最大的实验数据顶点和边数分别为 6 410 和 53 844,文献[92,94]中使用的实验数据顶点和边数分别为 3 112 和 12 519);(3) 构建索引的时间和空间复杂度均为超线性的(如最坏情况下 SPath 方法构建索引的时间和空间复杂度分别为 $O(nm)$和 $O(n^2)$,n 和 m 分别为顶点和边数),使用这些方法对知识图谱构建索引并不现实.值得注意的是,Fan 等人近年来提出了一系列 BGP 查询新方法[100−103].这些方法的核心思想是使用比子图同态或同构约束更弱的图模拟(graph simulation)和有界模拟(bounded simulation)语义定义 BGP 求值,从而使所提算法的时间复杂度降为立方级.实际上,该类方法改变了精确 BGP 语义且没有使用任何索引结构,所以仍可归类为策略(1).

面向 RDF 图的图模式匹配查询方法集中于策略(3).(1) 以 RDF-3X[58]和 Hexastore[59]为代表的六重索引方

法会创建 6 种不同排列的三元组索引.其问题在于,需将图模式匹配拆分为较多的连接操作,大量中间结果的连接影响性能.(2) 以 GRIN[104]和 DOGMA[105]为代表的结构索引方法以结构相似度作为度量指标,将 RDF 图进行划分并建立结构索引,之后在结构索引而非原始 RDF 图上执行图模式匹配,但索引维护代价较高.(3) 以 BitMat[106]和 TirpleBit[107]为代表的位图存储方法虽然节省了存储空间,但数据更新维护代价较大,且进行查询优化的自由度在一定程度上受到了制约.(4) 以 gStore[63]和 chameleon-db[108]为代表的图索引与匹配方法,将 SPARQL BGP 查询分为过滤和匹配两个步骤,过滤基于图索引,匹配采用子图同构算法,避免使用连接操作.

4.2 导航式查询

在图模式匹配查询中,匹配结果是知识图谱的子图,其中的路径长度是固定的.知识图谱上另一种重要的查询方式是导航式查询,其匹配的路径结果不能事先确定,需要按照图的拓扑结构进行导航.例如,第 3 节中例 **5** 查询“具有有限多步合作距离”的两名演员,例 7 查询“San Zhang 直接和间接认识的人”,例 9 查询“演员 Leonardo DiCaprio 与距其有限多步合作距离演员之间的全部路径”等均为导航式查询.

最简单的导航式查询是判断两个顶点之间是否存在一条路径,即可达性查询(reachability query).已有大量研究工作求解可达性查询,相关综述请参见文献[109].在实际应用中,往往进一步要求结果路径满足某种约束,其中最常用的是正则路径查询(regular path query,简称 RPQ).

定义2(正则路径查询(RPQ)). 给定知识图谱 $G=(V,E)$,其边上标签集合为Σ;在 G 中从顶点 v_0 到 v_m 的路径ρ为序列 $v_0a_0v_1a_1v_2\ldots v_{m-1}a_{m-1}v_m$,其中,$v_i\in V,a_i\in\Sigma,(v_i,a_i,v_{i+1})\in E$.路径$\rho$的标签为字符串$\lambda\rho=a_0\ldots a_{m-1}\in\Sigma^*$.$G$ 上的 RPQ Q 被定义为

$$(x,r,y)$$

其中,(1) x 和 y 是常量或变量;(2) r 是Σ上的正则表达式,其递归定义为 $r::=\varepsilon|a|r/r|r|r|r^*,a\in\Sigma$,/、|和*分别是串连接、并和克林闭包运算.

定义 7(RPQ 的任意路径语义). 给定知识图谱 $G=(V,E)$和其上的 *RPQ* $Q(x,r,y)$,Q 的任意路径语义定义为:$Q(G)$是 G 中所有标签满足正则表达式 r 的路径集合,即 $Q(G)=\{\rho|\lambda(\rho)\in L(r)\}$.

由于知识图谱 G 中可以有环,$Q(G)$中的匹配路径集合可能是无穷的,这种情况下,RPQ 的任意路径语义是不可计算的.在实际应用中,需要对该语义做出限制,使 RPQ 求值是可实现的.

(1) 任意路径语义(arbitrary path semantics).该语义即定义 7 给出的 $Q(G)$返回全部满足正则表达式 r 的路径.当然,在该语义下,$Q(G)$可能包含无穷多条路径;即使是有限多条路径(没有环),枚举所有路径也是不可行的(复杂度已被证明是#P 完全[42]).但基于该语义定义的“存在式”语义只关注两个顶点对之间是否存在一条满足条件的路径,即 $Q(G)=\{(x,y)|$存在从 x 到 y 的路径ρ使得$\lambda(\rho)\in L(r)\}$,避免了引起高复杂度的“数路径”问题[42,43],在实际实现中是可行的.例如,SPARQL 属性路径采取“存在式”语义[36].

(2) 最短路径语义(shortest path semantics).在该语义下,$Q(G)$返回满足正则表达式 r 的最短路径(或长度相等的若干最短路径).例如,G-CORE 中的正则路径查询采取的是该语义[30].

(3) 无重复顶点语义(no-repeated-node semantics).在该语义下,$Q(G)$返回满足正则表达式 r 的路径且每条路径中无重复出现的顶点(每个顶点仅出现 1 次).无重复顶点的路径称为简单路径(simple path).在一些实际场景中需要该语义,例如,在规划游览路线时,一般不希望访问一个地点多于 1 次.但该语义的求值复杂度早已被证明是 NP 完全的[110].

(4) 无重复边语义(no-repeated-edge semantics).在该语义下,$Q(G)$返回满足正则表达式 r 的路径且每条路径中无重复出现的边(每条边仅出现 1 次).这时,前面的例子变为允许访问一个地点多于 1 次,但是不能走重复的路线.目前,Cypher 语言采用这种语义[29].

根据不同需求,RPQ 的输出可以分为以下几种情况.

(1) 布尔值.判断在 $Q(G)$中两个给定顶点之间是否存在一条满足正则表达式 r 的路径;(2) 顶点.返回 $Q(G)$中路径的起点和终点对集合;(3) 路径.返回 $Q(G)$中的一条、多条或全部路径;(4) 图.将 $Q(G)$中的全部或部分路径整合为 G 的子图返回.

集合语义和包语义.对于 RPQ 输出为顶点的情况,集合语义与包语义不同.在集合语义下,顶点对(*u*,*v*)仅输出 1 次,无论 *Q*(*G*)中有多少条 *u* 到 *v* 的路径.在包语义下,顶点对(*u*,*v*)返回的次数等于 *Q*(*G*)中 *u* 到 *v* 的路径条数.基于前面的讨论,包语义与任意路径语义的结合实际上也是不可行的,因为包语义蕴含了“数路径”,其复杂度已被证明是#P 完全的[42].

RPQ 与 BGP 可结合在一起形成 CRPQ(conjunctive RPQ),其定义是 BGP 的扩展,即允许 BGP 中的边是 RPQ.关于 CRPQ 的理论研究请参见文献[67,111,112].文献[113]已证明 CRPQ 的组合复杂度是 NP 完全的.实际上,由第 2 节已经看出,CRPQ 是一般知识图谱语言的核心.在一些情况下,需要在查询中指定路径之间的某些关系(如比较路径标签是否相同),为此,文献[113]还提出了 ECRPQ(extended CRPQ)作为 CRPQ 的扩展,并且证明了 ECRPQ 的组合复杂度是 PSPACE 完全的.在 RPQ 基础上扩展的表达力更丰富的其他导航式查询形式包括:嵌套正则表达式 NRE(nested regular expression)[114]、正则数据路径查询 RDPQ(regular data path query)[115]和 Datalog 扩展[68].

RPQ 的求值理论上可看作正则表达式 *r* 对应的自动机与知识图谱 *G* 对应的自动机的相乘操作,其复杂度为 PTIME[110].Koschmieder 等人[116]提出使用“罕见标签(rare-label)”方法对 RPQ 进行分解,然后进行分段求值.该方法实际上采取了分治策略,但需要通过预处理事先确定“罕见标签”,查询处理采用了导航式遍历方法.Fan 等人[117]在基于模拟语义的子图匹配查询基础上引入了 RPQ 查询的一个子集,该工作对所提查询模型进行了完善的理论分析,并给出了相应的执行算法,其复杂度为立方级.Gubichev 等人[118]基于 RDF-3X 中的 FERRARI 索引在可达性查询的基础上扩展实现了 RPQ 查询处理.Dey 等人[119]基于关系数据库实现了具有起源保障(provenance-aware)的 RPQ 查询.Wang 等人[120]提出了大规模 RDF 图上的高效率起源保障 RPQ 求值算法.

4.3 分析型查询

不同于图模式匹配和导航式查询,分析型查询并不关心满足条件的图结构局部实例,而是面向于度量整个知识图谱的全局聚合信息.简单的分析型查询包括求知识图谱统计聚合信息(如顶点和边计数)、顶点的度、图中的最大/最小/平均度、图的直径等.较复杂的分析型查询主要是图上计算密集型的一些分析和挖掘算法,包括:

(1) 特征路径长度(characteristic path length).图中所有顶点对之间最短路径长度的平均值.刻画一个图中顶点之间的关联程度.

(2) 连通分量(connected components).返回图中所有顶点子集,这些集合中的顶点能够通过边相互达到.

(3) 社区发现(community detection).返回图中所有顶点子集,这些集合中的顶点以某种定义在同一社区中.

(4) 聚集系数(clustering coefficient).顶点的聚集系数是该顶点的邻居顶点相互连接的概率.图的聚集系数是图中所有顶点的平均聚集系数.

(5) PageRank.刻画了一个 Web 浏览者的随机游走行为.顶点的 PageRank 值表示 Web 浏览者访问到该顶点的概率.PageRank 可作为图中顶点相对重要程度的度量指标.

有关图数据上分析与挖掘算法的详细介绍请参见文献[121].对于大规模知识图谱,分析型查询往往计算量巨大,需要使用分布式图处理框架实现并行计算,详见第 5.3 节.

4.4 知识图谱查询语言比较

表 5 给出了 5 种知识图谱查询语言的语法、语义及相关特性的详细比较信息,包括:图模式匹配查询和导航式查询的语法和语义、分析型查询的支持程度、查询可组合性、数据更新语言 DML 和数据定义语言 DDL 的支持程度以及实现系统等.关于 SPARQL、Cypher 和 Gremlin 的比较可进一步参见文献[9];关于 Cypher、PGQL 和 G-CORE 的比较可进一步参见文献[122].

5 知识图谱数据库管理系统

本节首先给出目前主要知识图谱数据库管理系统的简要介绍,包括:RDF 三元组库和原生图数据库;然后,综述分布式图数据处理系统与框架;最后,介绍图数据管理系统的评测基准.

Table 5 The comparison of knowledge graph query languages

表5 知识图谱查询语言比较

语法/语义/特性		SPARQL	Cypher	Gremlin	PGQL	G-CORE
图模式匹配查询	语法	CGP	CGP	CGP(无可选)[1]	CGP	CGP
	语义	子图同态、包[2]	无重复边、包[2]	子图同态、包[2]	子图同构[3]、包[2]	子图同态、包[2]
导航式查询	语法	RPQ 超集(增加反向边和属性集上的否定)	RPQ 子集(*只能作用在单边)	RPQ 超集(增加通过表达式比较属性值)	RPQ 超集(增加比较路径上的顶点和边)	RPQ 超集(增加复杂路径表达式)
	语义	任意路径、集合[4]	无重复边[5]、包[2]	任意路径[6]、包[2]	最短路径[7]、包[8]	最短路径[9]、包[2]
分析型查询		聚合函数	聚合函数	聚合函数、PageRank、PeerPressure 聚类	聚合函数	聚合函数
查询可组合性		否	是	是	否	是
数据更新语言 DML		CRUD[10]	CRUD	无	无	CR
数据定义语言 DDL		无	有	无	无	无
实现系统		Jena、RDF4J、gStore、Virtuoso 等	Neo4j、AgensGraph 等	TinkerTop 等	Oracle PGX	无

注:1. Gremlin 不显式支持可选(optional)操作,但可以通过其他语法特性等价模拟.2. 可通过 DISTINCT 关键字支持集合语义.3. PGQL 默认的图模式匹配查询语义是子图同构,可使用 ALL 关键字改为子图同态.4. SPARQL 中只有当使用*运算使得属性路径查询无法等价写为 CGP 时才使用集合语义.5. Cypher 可通过 shortestPath 函数支持最短路径语义.6. Gremlin 中其他语义可以被模拟出来.7. PGQL 路径查询可通过用户定义函数实现其他语义.8. PGQL 路径查询返回单条最短路径,集合和包语义相同.9. G-CORE 路径查询可通过 ALL 关键字改为任意路径语义.10. CRUD 分别代表 CREATE 创建、READ 读取、UPDATE 更新和 DELETE 删除

5.1 RDF三元组库

主要的开源 RDF 三元组数据库包括:Apache Jena、Eclipse RDF4J 以及学术界的 RDF-3X 和 gStore;主要的商业 RDF 三元组数据库包括:Virtuoso、AllegroGraph、GraphDB、BlazeGraph 和 Stardog[123].

1. 开源 RDF 三元组数据库:Jena

Jena[55]是 Apache 顶级项目,其前身为惠普实验室开发的 Jena 工具包.Jena 是语义 Web 领域主要的开源框架和 RDF 三元组库,较好地遵循了 W3C 标准,其功能包括:RDF 数据管理、RDFS 和 OWL 本体管理、SPARQL 查询处理等.Jena 具备一套原生存储引擎,可对 RDF 三元组进行基于磁盘或内存的存储管理.同时,具有一套基于规则的推理引擎,用以执行 RDFS 和 OWL 本体推理任务.

2. 开源 RDF 三元组数据库:RDF4J

RDF4J[124]目前是 Eclipse 基金会旗下的开源孵化项目,其前身是荷兰软件公司 Aduna 开发的 Sesame 框架,功能包括:RDF 数据的解析、存储、推理和查询等.RDF4J 提供内存和磁盘两种 RDF 存储机制,支持 SPARQL 1.1 查询和更新语言.

3. 开源 RDF 三元组数据库:RDF-3X

RDF-3X 是由德国马克斯-普朗克计算机科学研究所研发的 RDF 三元组数据库系统,其最初成果发表于 2008 年的数据库国际会议 VLDB[58],后经功能扩展和完善,最新版本是 GH-RDF3X,源代码可以从 GitHub 上下载 https://github.com/gh-rdf3x/gh-rdf3x.RDF-3X 的最大特点在于其为 RDF 数据专门设计的压缩物理存储方案、查询处理和查询优化技术.

4. 开源 RDF 三元组数据库:gStore

gStore[63]是由北京大学、加拿大滑铁卢大学和香港科技大学联合研究项目开发的基于图的 RDF 三元组数据库.gStore 的存储层使用 RDF 图对应的签章图(signature graph)并建立“VS*树”索引以加速查找.将 RDF 图 G 中的每个实体顶点及其邻居属性和属性值编码成一个二进制位串,由这些位串作为顶点组成一张与 RDF 图 G 对应的签章图 G^*;在执行 SPARQL 查询时,将查询图 Q 也转化为一张查询的签章图 Q^*.为了支持在 G^*上快速查找到 Q^*的匹配位置,gStore 系统建立了“VS*树”索引,其基本思想是为签章图 G^*建立不同详细程度的摘要图(summary graph);利用“VS*”树索引提供的摘要图,gStore 系统可以大幅度缩小 SPARQL 查询的搜索空间.

5. 商业 RDF 三元组数据库:Virtuoso

Virtuoso[125]是 OpenLink 公司开发的商业混合数据库产品,支持关系数据、对象-关系数据、RDF 数据、XML 数据和文本数据的统一管理,其同时发布商业版本 Virtuoso Universal Server(Virtuoso 统一服务器)和开源版本

OpenLink Virtuoso.Virtuoso 虽然是可以支持多种数据模型的混合数据库管理系统,但其基础源自开发了多年的传统关系型数据库管理系统,因此具备较为完善的事务管理、并发控制和完整性机制.因为 Virtuoso 可以较为完善地支持 W3C 的 Linked Data 系列协议,包括 DBpedia 在内的很多开放 RDF 知识图谱选择其作为后台存储系统.

6. 商业 RDF 三元组数据库:AllegroGraph

AllegroGraph[126]是 Franz 公司开发的 RDF 三元组数据库.AllegroGraph 对语义推理功能具有较为完善的支持.除了三元组数据库的基本功能外,AllegroGraph 还支持动态物化的 RDFS++推理机、OWL2 RL 推理机、Prolog 规则推理系统、时空推理机制、社会网络分析库、可视化 RDF 图浏览器等.

7. 商业 RDF 三元组数据库:GraphDB

GraphDB[127]是由 Ontotext 软件公司开发的 RDF 三元组数据库.GraphDB 实现了 RDF4J 框架的 SAIL 层,可以使用 RDF4J 的 RDF 模型、解析器和查询引擎直接访问 GraphDB.GraphDB 的特色是对于 RDF 推理功能的良好支持,其使用内置的基于规则的“前向链(forward-chaining)”推理机,由显式(explicit)知识经过推理得到导出(inferred)知识,对这些导出知识进行优化存储;导出知识会在知识库更新后相应地同步更新.

8. 商业 RDF 三元组数据库:BlazeGraph

Blazegraph[128]是一个基于 RDF 三元组库的图数据库管理系统,在用户接口层同时支持 RDF 三元组和属性图模型,既实现了 SPARQL 语言也实现了 Blueprints 标准及 Gremlin 语言.Blazegraph 的内部实现技术是面向 RDF 三元组和 SPARQL 的,是“基于 RDF 三元组库的图数据库”.

9. 商业 RDF 三元组数据库:Stardog

Stardog[129]是由 Stardog Union 公司开发的 RDF 三元组数据库,其支持 RDF 图数据模型、SPARQL 查询语言、属性图模型、Gremlin 图遍历语言、OWL2 标准、用户自定义的推理与数据分析规则、虚拟图、地理空间查询以及多用编程语言与网络接口支持.Stardog 对 OWL2 推理机制具有良好的支持,同时具备全文搜索、GraphQL 查询、路径查询、融合机器学习任务等功能,能够支持多种不同编程语言和 Web 访问接口.

5.2 原生图数据库

目前主要的原生图数据库有 Neo4j、JanusGraph、OrientDB 和 Cayley.

1. 最流行的图数据库:Neo4j

Neo4j[29]是由 Neo 技术公司开发的图数据库.可以说,Neo4j 是目前流行程度最高的图数据库产品.Neo4j 基于属性图模型,其存储管理层为属性图的节点、节点属性、边、边属性等元素设计了专门的存储方案.这使得 Neo4j 在存储层对于图数据的存取效率优于关系数据库.

2. 分布式图数据库:JanusGraph

JanusGraph[130]是在原有 Titan 系统[131]基础上继续开发的开源分布式图数据库.JanusGraph 的存储后端与查询引擎是分离的,可使用分布式 Bigtable 存储库 Cassandra 或 HBase 作为存储后端.JanusGraph 借助第三方分布式索引库 ElasticSearch、Solr 和 Lucene 实现各类型数据的快速检索功能,包括地理信息数据、数值数据和全文搜索.JanusGraph 还具备基于 MapReduce 的图分析引擎,可将 Gremlin 导航查询转化为 MapReduce 任务.

3. 图数据库:OrientDB

OrientDB[132]最初是由 OrientDB 公司开发的多模型数据库管理系统.OrientDB 虽然支持图、文档、键值、对象、关系等多种数据模型,但其底层实现主要面向图和文档数据存储管理的需求设计.其存储层中数据记录之间的联系并不是像关系数据库那样通过主外键的引用,而是通过记录之前直接的物理指针.OrientDB 对于数据模式的支持相对灵活,可以管理无模式数据(schema-less),也可以像关系数据库那样定义完整的模式(schema-full),还可以适应介于两者之间的混合模式(schema-mixed)数据.在查询语言方面,OrientDB 支持扩展的 SQL 和 Gremlin 用于图上的导航式查询;OrientDB 的 MATCH 语句实现了声明式的模式匹配,这类似于 Cypher 语言查询模式.

4. 图数据库:Cayley

Cayley[133]是由 Google 公司工程师开发的一款轻量级开源图数据库.Cayley 的开发受到了 Freebase 知识图

谱和 Google 知识图谱背后的图数据存储的影响.Cayley 使用 Go 语言开发,可以作为 Go 类库使用;对外提供 REST API;具有内置的查询编辑器和可视化界面;支持多种查询语言,包括:基于 Gremlin 的 Gizmo、GraphQL 和 MQL;支持多种存储后端,包括:键值数据库 Bolt、LevelDB,NoSQL 数据库 MongoDB、CouchDB、PouchDB、ElasticSearch,关系数据库 PostgreSQL、MySQL 等.

其他原生图数据库还包括:构造在 Amazon 云平台上的 Amazon Neptune[134]、多模型图数据库 ArangoDB[135]、微软的 Azure CosmosDB[136]、DataStax 的 Enterprise Graph[137]、Sparsity 的 Sparksee[138]以及 TigerGraph[139]等.

5.3 分布式图数据处理系统与框架

在大数据时代,分布式/并行技术已成为大规模知识图谱数据管理不可或缺的工具.目前,规模为百万顶点(10^6)和上亿条边(10^8)的知识图谱数据集已不在少数[140].截至 2019 年 1 月 LOD(linked open data,链接开放数据)知识图谱发布的 RDF 图数据集共计 1 234 个,其总规模目前虽没有准确的统计数据,但保守估计达上千亿条三元组[141].LOD 中很多单个数据集的规模已超过 10 亿条三元组,例如,维基百科数据集 DBpedia 2014 为 30 亿条[142]、蛋白质数据集 UniProt 为 90.2 亿条[143]、地理信息数据集 LinkedGeoData 为 200 亿条[144].

近年来提出的面向大规模图数据的分布式系统与框架包括:基于分布式文件系统(如 GFS[145])或基于 Bigtable 模型[146]的图数据存储层和基于 MapReduce[147]、Pregel[148]及 GraphLab 框架[149]的图数据处理层.现有分布式/并行图数据管理系统大多基于这些框架进行扩展,其中包括:(1) 基于 MapReduce 的系统:YARS2[150]、SPIDER[151]、SHARD[152];(2) 基于 Bigtable 的系统:Titan[131]、CumulusRDF[153];(3) 基于服务总线的系统:Blazegraph[128];(4) 基于内存存储的系统:Trinity[154]、Trinity.RDF[155].

大规模 RDF 图上的分布式查询处理方法引入了图分割和查询分解策略.文献[156]将 RDF 图划分为若干分片,每个分片的边界节点扩展到“*n* 跳(*n*-hop)”邻居,同时将 SPARQL 查询划分为若干子查询进行并行求值.文献[157]将顶点及其邻居定义为“顶点块”,采用启发式规则对顶点块进行分布式存储,同时将查询进行分解,达到增大并行度且减少通信开销的目的.文献[158]提出的 EAGRE 方法基于边上的谓语信息对图和查询进行分解.文献[159]提出的 TriAD 方法采用 METIS 方法将 RDF 图分割为若干片段,每个片段在多个机器上存储,同时维护一个包含划分信息的摘要图,通过 MPI 框架的异步消息传递进行系统通信.文献[160]提出的 DREAM 系统只对查询进行分解并不分解 RDF 图,能够根据分解后子查询的复杂度自适应地分配执行查询所需的机器数量.

在分布式图查询处理上,基于部分求值(partial evaluation)技术[161]已提出了一系列有效方法.最近,Fan 等人利用部分求值提出了图数据的可达性查询分布式版本[162];Ma 和 Fan 等人利用部分求值提出了基于模拟语义的子图匹配的分布式版本[163,164];Peng 等人利用部分求值提出了 SPARQL 查询的分布式版本[165];Wang 等人提出了基于部分求值的 RDF 图上 RPQ 分布式算法[166].

此外,目前已有若干基于现有分布式计算框架的知识图谱查询处理工作.文献[167]展示的 H_2RDF+系统基于 HBase 分布式 Bigtable 存储库构建了三元组的六重索引.文献[168]给出的 Sempala 是基于分布式 SQL-on-Hadoop 数据库 Impala 和 Parquet 分布式文件格式的 RDF 图数据查询引擎.Lai 等人提出了基于 TwinTwig 结构分解的 MapReduce 分布式高效子图枚举算法,但该算法仅用于无向无标签图[169,170].Bi 等人进一步给出了通过尽可能推迟笛卡尔积执行来提高子图匹配效率的技术[171].Schätzle 等人提出的 S2RDF 系统将 SPARQL 查询转换为 Spark 分布式计算框架上的 RDD 操作,其离线建立了大量索引,用于加速在线查询,但对于大规模知识图谱,索引构建时间开销可能很高[172].文献[173]利用 RDF 图的语义和结构作为启发信息将查询图进行星形分解,设计了一种基于 MapReduce 的分布式 SPARQL BGP 匹配算法.He 等人提出的 Stylus 是一种利用强类型信息构建优化存储方案和查询处理的分布式 RDF 图存储库,其底层基于键值库[174].Peng 等人在文献[175]中给出了一种能够根据查询负载优化图划分和存储的 RDF 图存储方案.Xin 等人给出了基于 Pregel 图计算框架的起源保障 RPQ 分布式求值算法[176].开源项目 Apache Rya 是基于分布式列存储系统 Accumulo 开发的 RDF 三元组库[177].开源项目 Cypher for Apache Spark[178]是 Neo4j 公司开发的用于将 Cypher 查询转换为 Spark 并行操作的模块.关于基于 Pregel 的分布式图处理框架的最新研究进展可参见文献[19].

5.4 知识图谱数据库管理系统比较

表 6 比较了常见的 25 种知识图谱数据库管理系统,包括许可证、数据模型、存储方案、查询语言、特点描述、最新版本和是否活跃.

Table 6 The comparison of knowledge graph database management systems

表6 常见知识图谱数据库管理系统的比较

类型	名称	许可证	数据模型/存储方案	查询语言	特点描述	最新版本	是否活跃
基于关系	3store	开源	RDF 图/三元组表	SPARQL	早期系统,三元组表存储方案的代表性系统	3.0.17 (2006-7-17)	否
	DLDB	研究原型	RDF 图/水平表	SPARQL	早期系统,水平表存储方案的代表性系统	已不维护	否
	Jena	开源	RDF 图/属性表	SPARQL	主流的语义 Web 工具库、RDF 数据库和 OWL 推理工具	3.10.0 (2018-12-30)	是
	SW-Store	研究原型	RDF 图/垂直划分	SPARQL	科研原型系统,垂直划分存储方案的代表性系统	已不维护	否
	IBM DB2	商业	RDF 图/DB2RDF	SPARQL/SQL	支持 RDF 的主流商业数据库	11.1 (2016-4-12)	是
	Oracle 18*c*	商业	RDF 图/关系存储	SPARQL/PGQL	支持 RDF 的主流商业数据库	18*c* (2018-2-5)	是
RDF 三元组库	RDF4J	开源	RDF 图/SAIL API	SPARQL	主流的语义 Web 工具库、RDF 数据库、提供 SAIL 接口	2.5.0 (2019-3-7)	是
	RDF-3X	开源	RDF 图/六重索引	SPARQL	科研原型系统,六重索引存储方案的代表性系统	0.3.8 (2013-11-22)	否
	gStore	开源 研究原型	RDF 图/VS*树	SPARQL	科研原型系统,原生图存储,使用了基于位串图存储技术	0.7.2 (2018-11-4)	是
	Virtuoso	商业/开源	RDF 图/多模型混合	SPARQL/SQL	语义 Web 项目常用的 RDF 数据库,基于成熟的 SQL 引擎	8.2 (2018-10-22)	是
	AllegroGraph	商业	RDF 图/三元组索引	SPARQL	对语义推理功能具有较为完善的支持	6.5.0 (2019-3-4)	是
	GraphDB	商业	RDF 图/三元组索引	SPARQL	支持语义 Web 标准的主流产品,支持 SAIL 层推理功能	8.8.1 (2019-1-30)	是
	BlazeGraph	商业	RDF 图/三元组索引	SPARQL/Gremlin	基于 RDF 三元组库的图数据库,实现了 SPARQL 和 Gremlin	2.1.4 (2016-8-30)	否
	StarDog	商业	RDF 图/三元组索引	SPARQL	对 OWL2 推理机制具有良好的支持	6.1.2 (2019-3-7)	是
原生图数据库	Neo4j	商业/开源	属性图/原生图存储	Cypher	最流行的图数据库,基于属性图模型,实现了原生优化存储	3.5.3 (2019-2-11)	是
	JanusGraph	开源	属性图 分布式存储	Gremlin	分布式图数据库,存储后端与查询引擎分离,实现了 Gremlin	0.2.2 (2018-10-9)	是
	OrientDB	商业	属性图/原生图存储	SQL/Gremlin	支持多模型的原生图数据管理系统,对数据模式的灵活支持	3.0.17 (2019-3-7)	是
	Cayley	开源	RDF 图/外部存储	Gremlin/GraphQL	轻量级开源图数据库,易于扩展对新语言和存储后端的支持	0.7.5 (2018-11-27)	是
分布式系统与框架	Sempala	开源 研究原型	RDF 图/分布式存储	SPARQL	基于 HDFS 存储,使用 Impala SQL 引擎的 RDF 三元组库	2.1 (2017-7-7)	否
	TriAD	开源 研究原型	RDF 图/ 分布式存储六重索引	SPARQL	基于 MPI 框架的异步通信协议	GitHub 源码未发布	否
	H_2RDF+	开源 研究原型	RDF 图/ 分布式存储六重索引	SPARQL	基于 HBase 构建六重索引	GitHub 源码未发布	否
	S2RDF	开源 研究原型	RDF 图/ 分布式存储垂直划分	SPARQL	基于 Spark 框架建立大量索引	1.1 (2016-4-4)	否
	Stylus	开源 研究原型	RDF 图/ 分布式存储属性表优化	SPARQL	基于分布式内存键值库的 RDF 三元组库	GitHub 源码未发布	否
	Apache Rya	开源	RDF 图/ 分布式存储三元组索引	SPARQL	基于列存储 Accumulo 的 RDF 三元组库	3.2.12 (2018-03-04)	是
	Cypher for Apache Spark	开源	属性图/ 分布式存储 DataFrame	Cypher	基于 Spark 框架的 Cypher 引擎	0.3.0 (2019-3-8)	是

表 6 不仅包括本节介绍的 RDF 三元组库、原生图数据库和分布式图数据处理系统与框架,还纳入了第 3.1 节中介绍的基于关系的知识图谱存储方案的代表性系统.

5.5 评测基准

评测基准(benchmark)是客观评价数据库管理系统性能的标准工具.关系数据库有著名的 TPC 评测基准.知识图谱数据库管理系统的评测基准仍处于发展完善阶段.目前,链接数据评测基准委员会(Linked Data Benchmark Council,简称 LDBC)是知识图谱数据管理系统评测基准开发的主要组织,其成员包括了主要的图数据库公司和研究机构[179].LDBC 目前开发了两个评测基准:社会网络评测基准(social network benchmark,简称 SNB)和语义出版评测基准(semantic publishing benchmark,简称 SPB).SNB 设计用于评测知识图谱数据库管理系统的事务查询负载、分析查询负载和图分析算法;SPB 设计用于评测查询和更新混合负载并兼具一定的语义推理评测功能.不过,LDBC 评测基准中的部分功能尚处于开发阶段.

此外,在 LDBC 成立之前,学术界已经开发了若干 RDF 数据库评测基准,包括:LUBM、SP^{2}Bench、BSBM 和 DBPSB.LUBM[180]是最早的 RDF 评测基准,不支持 SPARQL 1.1 新特性,也不支持数据更新,具备一定的推理评测能力;SP^{2}Bench[181]是基于 DBLP 数据集构建的 SPARQL 评测基准,不支持更新;BSBM[182]基于电子商务数据模式,是功能相对完善的 SPARQL 评测基准,同时包括事务和分析型负载,但是,BSBM 数据集过于规整,以致于基于关系数据库的系统的评测结果均较优.DBPSB[183]使用基于 DBpedia 的真实数据集,但其负载过于简单,只包括基本查询,同样不支持更新.WatDiv 评测基准[184]的特点是能够支持用户自定义生成测试数据的模式,其查询负载根据数据模式图上的随机游走产生,分为线性查询、星形查询、雪花形查询和复杂查询 4 类.Link Bench[185]是基于 Facebook 社交网络真实数据和负载开发的评测基准,能够模拟 Facebook 真实生产情景下的社交网络图数据库查询和更新操作,但该评测基准已不再被维护.

表 7 给出了本节介绍的 8 种知识图谱数据管理评测基准的比较,包括发布机构、生成数据特点、查询负载特点和活跃程度.

Table 7 The comparison of knowledge graph data management benchmarks

表7 知识图谱数据管理评测基准的比较

评测基准	发布机构	生成数据特点	查询负载特点	是否活跃	文献
SNB	LDBC	用 DBpedia 真实数据作为引用数据,生成一段时间内的社会网络图	54 个语言中立的查询负载,面向知识图谱数据管理系统的事务查询、分析查询和图分析	是	[186]
SPB	LDBC	用英国广播公司(BBC)和 DBpedia 等真实数据作为引用数据	36 个 SPARQL 查询更新混合负载,兼具语义推理评测功能	是	[187]
LUBM	美国理海大学	生成任意大小的基于大学本体的合成 RDF 数据集	14 个 SPARQL 查询,覆盖 SPARQL 基本查询特性	是	[180]
SP^{2}Bench	德国弗莱堡大学	基于 DBLP 真实数据集结构进行规模扩展,生成任意大小数据集	17 个 SPARQL 查询,覆盖 SPARQL 的典型算子组合	否	[181]
BSBM	德国曼海姆大学	基于电子商务数据模式进行规模扩展,包括产品、厂商、顾客评价等	12 个 SPARQL 查询、2 个更新、8 个分析型查询	否	[182]
DBPSB	德国莱比锡大学	基于 DBpedia 真实数据集,采样法生成较小数据,重复生成较大数据	25 个 SPARQL 查询,根据 DBpedia 真实查询选出	否	[183]
WatDiv	加拿大滑铁卢大学	支持用户自定义数据模式,控制 RDF 结构化程度、谓语分布等	20 个 SPARQL 查询模板,通过数据模式上随机游走生成	否	[184]
LinkBench	Facebook	基于 Facebook 社交网络真实数据进行规模扩展,生成任意大小数据集	模拟 Facebook 真实生产情景的 SQL 查询和更新操作	否	[185]

6 未来研究方向

目前,知识图谱数据管理的理论、方法、技术与系统处于快速发展和开发完善阶段.数据库学术和产业界对知识图谱数据管理研发投入正在不断增加.本节将未来的研究方向归纳如下.

(1) 知识图谱数据模型与查询语言的统一

目前,知识图谱数据模型和查询语言尚不统一.考虑到关系数据库的兴起与发展流行,其中主要因素是具有精确定义的关系数据模型和统一的查询语言 SQL.统一的数据模型和查询语言不仅减轻了数据库管理系统的研发成本,而且降低了用户设计、构建、管理和维护数据库的代价,同时降低了新用户的学习难度.

但是,由于知识图谱的发展原因,其数据模型与查询语言的统一存在着一定难度.其一,RDF 图源于语义 Web 的发展而产生,其在标准制定之初即面向 Web 上全局资源的表示、发布和集成,其上还基于描述逻辑定义了 RDF 模式(RDF schema)语言和 Web 本体语言(OWL),形成了一整套高级语义表示和推理机制;DBpedia、YAGO、WikiData 等著名知识图谱实际上均是 RDF 格式.另一方面,属性图来自于图数据库领域,其顶点和边属性的方便表示机制弥补了 RDF 图的不足,但目前仍未形成一致公认的严格数学定义,比如,G-CORE 语言为了提高路径的地位还定义了路径属性图.当前,亟需定义统一的知识图谱数据模型,既具有 RDF 图面向 Web 的优点(如使用 URI 唯一标识资源),又具备属性图上便于数据存储的优势,同时将已有 RDF 知识图谱数据映射转换为新数据模型格式,作为真实的大规模知识图谱数据集,提升统一数据模型的认可度.其二,统一现有知识图谱查询语言迫在眉睫.第 2 节列出的主要查询语言就有 5 种之多,SPARQL 面向 RDF 图,其余 4 种均面向属性图.知识图谱数据模型统一后,必然需要制定统一的查询语言.目前,openCypher 组织已经发出了“GQL 宣言”[188],准备将 Cypher、PGQL 和 G-CORE 融合为属性图标准查询语言 GQL.但是,其中并未考虑 RDF 图的 SPARQL 语言.面向上述统一知识图谱数据模型,研制统一知识图谱查询语言,定义精确语法和语义,是未来的一个重要研究方向.

(2) 大规模知识图谱数据的分布式存储方案

第 3 节讨论的知识图谱存储管理方案,无论是基于关系的还是原生的,均是在单机系统上实现的.大规模知识图谱数据的分布式存储的研发目前尚处于起步阶段.知识图谱数据的分布式存储面临的第一个问题是大规模图数据的划分.图划分问题本身是一个经典的 NP 完全问题.即使使用公认最优的 METIS 图划分算法,对于大规模图数据在单机上执行划分也几乎是不可行的.所以,首先需要研究面向大规模知识图谱数据的分布式图划分算法,该算法既要考虑按照知识图谱的图结构和知识语义信息作为图划分标准,尽可能地有利于支持知识图谱查询的快速执行,又要避免算法复杂度过高.其次,在知识图谱划分的基础上,提出分布式存储方案.需要考虑:是面向 OLTP 和 OLAP 设计两种不同存储方案,还是设计可以平衡不同类型查询的统一存储;可选的物理层实现框架包括分布式关系数据库存储层、分布式文件系统、分布式 Bigtable 系统和分布式键值存储库;扩展单机版的 RDF 图或属性图存储方案,使其适应分布式物理存储底层是一种可选思路.再次,还需要面向知识图谱查询处理设计不同的索引方案,比如,面向图模式匹配查询的索引、面向导航式路径查询的索引和面向分析型查询的索引.

(3) 大规模知识图谱数据的分布式查询处理

虽然第 5.3 节介绍了现有的知识图谱分布式查询处理方法、框架与系统,但按照数据库系统的观点,目前还没有形成基于底层存储方案支撑的分布式查询处理完善机制.大部分已有方法均是为了解决某种特定查询问题而设计的专用算法,或者是基于 MapReduce、Pregel、MPI 等已有分布式处理框架而设计的算法.从分布式数据库管理系统的角度出发,需要形成具备一整套逻辑和物理算子的分布式查询处理高效算法,充分考虑存储和索引结构,同时考虑分布式通信开销,进而形成一套适合知识图谱分布式查询的代价模型.在此基础上,研究知识图谱分布式查询优化方法.

(4) 知识图谱数据管理对于本体和知识推理的支持

知识图谱数据与传统关系数据的一个最大区别是对本体的表示和内在的知识推理能力.例如,在 RDF 图数据之上,还有 RDF 模式(RDFS)和 Web 本体语言(OWL)的定义,可用于表示丰富的高级语义知识,同时定义了不同层面的推理功能,即从已有知识推导出隐含知识.目前的知识图谱数据管理还没有充分考虑到对于本体和知识推理的支持.如何在存储层和查询处理层支持知识图谱高层本体的有效管理和高效率的推理功能是非常有意义的研究方向.关于面向知识图谱的知识推理最新研究进展可参见文献[187].

(5) 大规模知识图谱的更新维护

真实的知识图谱数据随时间的推移会发生不断的更新变化.目前的知识图谱数据管理系统有很多假设知识是只读的和单向追加的,对于大规模知识图谱的更新维护基本没有考虑.首先,需要在知识图谱统一查询语言中专门为知识图谱更新设计数据更新子语言;其次,在知识更新过程中,往往会涉及到多版本控制、一致性约束和不一致消解等多种问题.这些问题在知识图谱数据管理系统中如何解决需要深入研究.

(6) 大规模知识图谱的数据集成

对于多个单独维护的知识图谱数据库或者历史遗留的知识图谱存在数据集成需求.目前,Linked Data技术[189]是 RDF 图上进行数据集成的规范方法,在多个符合 Linked Data 的 RDF 图上可以构建 SPARQL 联邦查询系统,进行跨知识图谱的集成查询.但在新型分布式知识图谱数据管理背景下,仍然存在着若干需要研究的问题,比如,分布式知识图谱数据库的不同集成方式、联邦查询的性能优化和效率提升、面向不一致知识图谱的数据集成等.

7 总 结

本文以数据模型的结构和操作要素为主线,对目前的知识图谱数据管理的理论、方法、技术与系统进行了研究综述:包括两种知识图谱数据模型和 5 种知识图谱查询语言;介绍了基于关系的和原生的知识图谱存储管理;讨论了知识图谱上的图模式匹配、导航式和分析型查询操作;介绍了 RDF 三元组库和原生图数据库这两类知识图谱数据库管理系统,描述了目前面向知识图谱的分布式系统与框架的现状,给出了知识图谱评测基准的情况.最后,展望了知识图谱数据管理的未来研究方向.

References:

[1] Knublauch H, Kontokostas D. Shapes constraint language (SHACL). W3C Editor's Draft, https://w3c. github.io/data-shapes/shacl/

[2] Angles R, Gutierrez C. Survey of graph database models. ACM Computing Surveys, 2008,40(1):1–39.

[3] Gallagher B. Matching structure and semantics: A survey on graph-based pattern matching. AAAI FS, 2006,6(2):45–53.

[4] Gutierrez C, Hurtado CA, Mendelzon AO. Foundations of semantic Web databases. Journal of Computer and System Sciences, 2011,77(3):520–541.

[5] Özsu MT. A survey of RDF data management systems. Frontiers of Computer Science, 2016,10(3):418–432.

[6] Zou L, Zsu MT. Graph-based RDF data management. Data Science and Engineering, 2017,2(1):56–70.

[7] Wylot M, Hauswirth M, Cudré-Mauroux P. RDF data storage and query processing schemes: A survey. ACM Computing Surveys (CSUR), 2018,51(4):84.

[8] Angles R, Gutierrez C. An Introduction to Graph Data Management. Cham: Springer-Verlag, 2018. 1–32.

[9] Angles R, Arenas M, Barcelo P. Foundations of modern query languages for graph databases. ACM Computing Surveys, 2016, 50(5):68.

[10] McCune RR, Weninger T, Madey G. Thinking like a vertex: A survey of vertex-centric frameworks for large-scale distributed graph processing. ACM Computing Surveys (CSUR), 2015,48(2):25.

[11] Kaoudi Z, Manolescu I. RDF in the clouds: A survey. The VLDB Journal, 2015,24(1):67–91.

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

[13] Heidari S, Simmhan Y, Calheiros RN. Scalable graph processing frameworks: A taxonomy and open challenges. ACM Computing Surveys (CSUR), 2018,51(3):60.

[14] Kalavri V, Vlassov V, Haridi S. High-level programming abstractions for distributed graph processing. IEEE Trans. on Knowledge and Data Engineering, 2018,30(2):305–324.

[15] Yan D, Bu Y, Tian Y. Big graph analytics platforms. Foundations and Trends in Databases, 2017,7(1-2):1–195.

[16] 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).

[17] Yu G, Gu Y, Bao YB. Large scale graph data processing on cloud computing environments. Chinese Journal of Computers, 2011, 34(10):1753–1767 (in Chinese with English abstract).

[18] Yu J, Liu YB, Zhang Y, *et al*. Survey on large-scale graph pattern matching. Journal of Computer Research and Development, 2015,52(2):391–409 (in Chinese with English abstract).

[19] Wang TT, Rong CT, LU W, Du XY. Survey on technologies of distributed graph processing systems. Ruan Jian Xue Bao/Journal of Software, 2018,29(3):569−586 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5450.htm [doi: 10.13328/j.cnki.jos.005450]

[20] Guan SP, Jin XL, Jia YT, Wang YZ, Cheng XQ. Knowledge reasoning over knowledge graph: A survey. Ruan Jian Xue Bao/Journal of Software, 2018,29(10):74–102 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5551.htm [doi: 10.13328/j.cnki.jos.005551]

[21] Xu J, Zhang QZ, Zhao X, Lü P, Li TS. Survey on dynamic graph pattern matching technologies. Ruan Jian Xue Bao/Journal of Software, 2018,29(3):663–688 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5444.htm [doi: 10.13328/j.cnki.jos.005444]

[22] Codd EF. A relational model of data for large shared data banks. Communications of the ACM, 1970,13(6):377–387.

[23] Klyne G, Carroll JJ, McBride B. RDF 1.1 concepts and abstract syntax. W3C Recommendation, 2018. https://www.w3.org/TR/rdf11-concepts/

[24] Lee TB. Reifying RDF (properly), and N3. 2018. https://www.w3.org/DesignIssues/Reify.html

[25] Wikipedia. The free encyclopedia. Hypergraph, 2018. http://en.wikipedia.org/wiki/Hypergraph

[26] Brickley D, Guha RV. RDF Schema 1.1. W3C Recommendation, 2018. https://www.w3.org/TR/rdf-schema/

[27] W3C OWL Working Group. OWL 2 Web ontology language document overview. 2nd ed., 2018. https://www.w3.org/TR/owl2-overview/

[28] Pascal H, Krotzsch M, Rudolph S. Foundations of Semantic Web Technologies. London: Chapman and Hall/CRC, 2009. 28–36.

[29] The Neo4j Team. The Neo4j Manual v3.4. 2018. https://neo4j.com/docs/developer-manual/current/

[30] Angles R, Arenas M, Barceló P. G-CORE: A core for future graph query languages. In: Das G, ed. Proc. of the 2018 Int'l Conf. on Management of Data. Houston: ACM, 2018. 1421–1432.

[31] Hartig O. Reconciliation of RDF* and property graphs. Computer Science Databases, 2014, 1–18.

[32] Gandon F, Schreiber G. RDF 1.1 XML syntax. 2018. https://www.w3.org/TR/rdf-syntax-grammar/

[33] Sporny M, Longley D, Kellogg G, Lanthaler M, Lindström N. A JSON-based serialization for linked data. 2018. https://www.w3.org/TR/json-ld/

[34] Beckett D. RDF 1.1 N-Triples. A line-based syntax for an RDF graph. 2018. https://www.w3.org/TR/n-triples/

[35] Beckett D, Lee TB, Prud'hommeaux E, Carothers G. RDF 1.1 Turtle. Terse RDF Triple Language. 2018. https://www.w3.org/TR/turtle/

[36] Harris S, Seaborne A, Prud'hommeaux E. SPARQL 1.1 query language. W3C Recommendation, 2013,21(10):778.

[37] Francis N, Green A, Guagliardo P. Cypher: An evolving query language for property graphs. In: Das G, ed. Proc. of the 2018 Int'l Conf. on Management of Data. New York: ACM, 2018. 1433–1445.

[38] Apache TinkerPop. TinkerPop3 Documentation v.3.3.3. 2018. http://tinkerpop.apache.org/docs/3.3.3/reference/

[39] Van Rest O, Hong S, Kim J. PGQL: A property graph query language. In: Das G, ed. Proc. of the 4th Int'l Workshop on Graph Data Management Experiences and Systems. New York: ACM, 2016. 7.

[40] Kostylev EV, Reutter JL, Romero M, Vrgoč D. SPARQL with property paths. In: Arenas M, ed. Lecture Notes in Computer Science. Cham: Springer-Verlag, 2015. 3–18.

[41] Pérez J, Arenas M, Gutierrez C. Semantics and complexity of SPARQL. In: Cruz I, ed. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer-Verlag, 2006. 30–43.

[42] Arenas M, Conca S, Pérez J. Counting beyond a Yottabyte, or how SPARQL 1.1 property paths will prevent adoption of the standard. In: Misselis J, ed. Proc. of the World Wide Web. New York: ACM, 2012. 629–638.

[43] Losemann K, Martens W. The complexity of evaluating path expressions in SPARQL. In: Benedikt M, ed. Proc. of the 31st ACM SIGMOD-SIGACT-SIGAI Symp. on Principles of Database Systems. New York: ACM, 2012. 101–112.

[44] Paul G, Passant A, Polleres A. SPARQL 1.1 Update. 2018. https://www.w3.org/TR/sparql11-update/

[45] Rudolf M, Paradies M, Bornhövd C, Lehner W. The graph story of the SAP HANA database. Datenbanksysteme für Business, Technologie und Web (BTW), 2013,20(13):403–420.

[46] MkDocs and Material for MkDocs. RedisGraph. 2018. https://oss.redislabs.com/redisgraph/

[47] Bitnine Co, Ltd. AgensGraph—The performance-driven graph database. 2018. https://bitnine.net/

[48] Memgraph Ltd. Memgraph. 2018. https://memgraph.com/

[49] Oracle Corporation. Oracle Parallel Graph Analytics (PGX). 2018. http://www.oracle.com/technetwork/oracle-labs/parallel-graph-analytics

[50] Oracle and/or its affiliates. Oracle PGQL 1.1 Specification. 2018. http://pgql-lang.org/spec/1.1/

[51] van Rest O. G-CORE grammar and parser. 2018. https://github.com/ldbc/ldbc_gcore_parser

[52] Angles A, Arenas M, Barceló P, Boncz PA, Fletcher GHL, Gutierrez C, Lindaaker T, Paradies M, Plantikow S, Sequeda JF, van Rest O, Voigt H. G-CORE: A core for future graph query languages. In: Sahni S, ed. Proc. of the ACM Computing Surveys. New York: ACM, 2017. 1–33.

[53] Harris S, Gibbins N. 3store: Efficient bulk RDF storage. In: Volz R, ed. Proc. of the 1st Int'l Workshop on Practical and Scalable Semantic Systems. Sanibel Island: CEUR-WS.org, 2004. 81–95.

[54] Pan Z, Heflin J. DLDB: Extending relational databases to support semantic Web queries. In: Volz R, ed. Proc. of the 1st Int'l Workshop on Practical and Scalable Semantic Systems. Sanibel Island: CEUR-WS.org, 2004. 109–113.

[55] Wilkinson K. Jena property table implementation. In: Smart PR, ed. Proc. of the 2nd Int'l Workshop on Scalable Semantic Web Knowledge Base Systems. Athens, 2006. 35–46.

[56] Abadi DJ, Marcus A, Madden SR. Scalable semantic Web data management using vertical partitioning. In: Klas W, ed. Proc. of the 33rd Int'l Conf. on Very Large Data Bases. Vienna: VLDB Endowment, 2007. 411–422.

[57] Abadi DJ, Marcus A, Madden SR. SW-store: A vertically partitioned DBMS for semantic Web data management. VLDB Journal, 2009,18(2):385–406.

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

[59] Weiss C, Karras P, Bernstein A. Hexastore: Sextuple indexing for semantic Web data management. Proc. of the VLDB Endowment, 2008,1(1):1008–1019.

[60] Bornea MA, Dolby J, Kementsietsidis A. Building an efficient RDF store over a relational database. In: Ross K, ed. Proc. of the 2013 ACM SIGMOD Int'l Conf. on Management of Data. New York: ACM, 2013. 121–132.

[61] Sun W, Fokoue A, Srinivas K. Sqlgraph: An efficient relational-based property graph store. In: Sellis T, ed. Proc. of the 2015 ACM SIGMOD Int'l Conf. on Management of Data. New York: ACM, 2015. 1887–1901.

[62] Ian R, Webber J, Eifrem E. Graph Databases. 2nd ed., San Francisco: O'Reilly Media, 2015. 42–58.

[63] Zou L, Özsu M T, Chen L. gStore: A graph-based SPARQL query engine. The VLDB Journal, 2014,23(4):565–590.

[64] Calvanese D, De Giacomo G, Lenzerini M. Containment of conjunctive regular path queries with inverse. ACM Trans. on Computational Logic, 2000,9(3):176–185.

[65] Wood PT. Query languages for graph databases. ACM SIGMOD Record, 2012,41(1):50–60.

[66] Barceló P. Querying graph databases. In: Hull R, ed. Proc. of the 32nd ACM SIGMOD-SIGACT-SIGAI Symp. on Principles of Database Systems. New York: ACM, 2013. 175–188.

[67] Barceló P, Libkin L, Reutter JL. Querying regular graph patterns. Journal of the ACM (JACM), 2014,61(1):8–45.

[68] Reutter JL, Romero M, Vardi MY. Regular queries on graph databases. Theory of Computing Systems, 2017,61(1):31–83.

[69] Hell P, Nesetril J. Graphs and Homomorphisms. 2nd ed., Hattiesburg: Oxford University Press, 2004. 155–171.

[70] Ullmann JR. An algorithm for subgraph isomorphism. Journal of the ACM (JACM), 1976,1(23):31–42.

[71] Vardi MY. The complexity of relational query languages. In: Burkhard WA, ed. Proc. of the 14th Annual ACM Symp. on Theory of Computing. New York: ACM, 1982. 137–146.

[72] Abiteboul S, Hull R, Vianu V. Foundations of Databases. Boston: Addison-Wesley Longman Publishing Co., 1995.

[73] Angles R, Gutierrez C. The expressive power of SPARQL. In: Sheth A, ed. Lecture Notes in Computer Science. Berlin: Springer-Verlag, 2008. 114–129.

[74] Schmidt M, Meier M, Lausen G. Foundations of SPARQL query optimization. In: Segoufin L, ed. Proc. of the 13th Int'l Conf. on Database Theory. New York: ACM, 2010. 4–33.

[75] McKay BD. Practical Graph Isomorphism. Nashville: Tennessee, 1981. 45–87.

[76] Cordella LP, Foggia P, Sansone C. A (sub) graph isomorphism algorithm for matching large graphs. IEEE Trans. on Pattern Analysis and Machine Intelligence, 2004,26(10):1367–1372.

[77] Shapiro LG, Haralick RM. Structural descriptions and inexact matching. IEEE Trans. on Pattern Analysis and Machine Intelligence, 1981,(5):504–519.

[78] Tsai WH, Fu KS. Subgraph error-correcting isomorphisms for syntactic pattern recognition. IEEE Trans. on Systems, Man and Cybernetics, 1983,(1):48–62.

[79] Cordella LP, Foggia P, Sansone C, Vento M. Subgraph transformations for the inexact matching of attributed relational graphs. In: Jolion JM, ed. Proc. of the Graph Based Representations in Pattern Recognition. Vienna: Springer-Verlag, 1998. 43–52.

[80] Zou L, Chen L, Özsu MT. Distance-join: Pattern match query in a large graph database. Proc. of the VLDB Endowment, 2009, 2(1):886–897.

[81] Umeyama S. An eigendecomposition approach to weighted graph matching problems. IEEE Trans. on Pattern Analysis and Machine Intelligence, 1988,10(5):695–703.

[82] Christmas WJ, Kittler J, Petrou M. Structural matching in computer vision using probabilistic relaxation. IEEE Trans. on Pattern Analysis and Machine Intelligence, 1995,17(8):749–764.

[83] Giugno R, Shasha D. Graphgrep: A fast and universal method for querying graphs. Object Recognition Supported by User Interaction for Service Robots, 2002,(8)2:112–115.

[84] Yan X, Yu PS, Han J. Graph indexing: A frequent structure-based approach. In: Valduriez P, ed. Proc. of the 2004 ACM SIGMOD Int'l Conf. on Management of Data. New York: ACM, 2004. 335–346.

[85] He H, Singh AK. Closure-tree: An index structure for graph queries. In: Weikum G, ed. Proc. of the 22nd Int'l Conf. on Data Engineering, ICDE 2006. Washington: IEEE Computer Society, 2006. 38.

[86] Cheng J, Ke Y, Ng W. FG-index: Towards verification-free query processing on graph databases. In: Ooi BC, ed. Proc. of the 2007 ACM SIGMOD Int'l Conf. on Management of Data. New York: ACM, 2007. 857–872.

[87] Zhao P, Yu JX, Yu PS. Graph indexing: Tree+delta>=graph. In: Klas W, ed. Proc. of the 33rd Int'l Conf. on Very Large Data Bases. Vienna: VLDB Endowment, 2007. 938–949.

[88] Zhang S, Hu M, Yang J. Treepi: A novel graph indexing method. In: Ioannidis Y, ed. Proc. of the 23rd IEEE Int'l Conf. on Data Engineering. Washington: IEEE, 2007. 966–975.

[89] Williams DW, Huan J, Wang W. Graph database indexing using structured graph decomposition. In: Damien K, ed. Proc. of the 23rd IEEE Int'l Conf. on Data Engineering. Washington: IEEE, 2007. 976–985.

[90] Zou L, Chen L, Yu JX. A novel spectral coding in a large graph database. In: Teubner J, ed. Proc. of the 11th Int'l Conf. on Extending Database Technology: Advances in Database Technology. New York: ACM, 2008. 181–192.

[91] Shang H, Zhang Y, Lin X. Taming verification hardness: An efficient algorithm for testing subgraph isomorphism. Proc. of the VLDB Endowment, 2008,1(1):364–375.

[92] He H, Singh AK. Graphs-at-a-time: Query language and access methods for graph databases. In: Shasha D, ed. Proc. of the 2008 ACM SIGMOD Int'l Conf. on Management of Data. New York: ACM, 2008. 405–418.

[93] Zhang S, Li S, Yang J. GADDI: Distance index based subgraph matching in biological networks. In: Kersten M, Novikov B, Teubner J, eds. Proc. of the 12th Int'l Conf. on Extending Database Technology: Advances in Database Technology. New York: ACM, 2009. 192–203.

[94] Zhao P, Han J. On graph query optimization in large networks. Proc. of the VLDB Endowment, 2010,3(1-2):340–351.

[95] Di Natale R, Ferro A, Giugno R. Sing: Subgraph search in non-homogeneous graphs. BMC Bioinformatics, 2010,11(1):96.

[96] Bonnici V, Ferro A, Giugno R, Pulvirenti A, Shasha D. Enhancing graph database indexing by suffix tree structure. In: Dijkstra TMH, Tsivtsivadze E, Marchiori E, Heskes T, eds. Proc. of the Pattern Recognition in Bioinformatics, PRIB 2010. Lecture Notes in Computer Science, Heidelberg: Springer-Verlag, 2010. 195–203.

[97] Yuan D, Mitra P. Lindex: A lattice-based index for graph databases. The VLDB Journal, 2013,22(2):229–252.

[98] Gouda K, Hassaan M. Compressed feature-based filtering and verification approach for subgraph search. In: Guerrini G, ed. Proc. of the 16th Int'l Conf. on Extending Database Technology. New York: ACM, 2013. 287–298.

[99] Zheng W, Zou L, Lian X. SQBC: An efficient subgraph matching method over large and dense graphs. Information Sciences, 2014,261:116–131.

[100] Fan W, Li J, Ma S. Graph pattern matching: From intractable to polynomial time. Proc. of the VLDB Endowment, 2010,3(1-2):264–275.

[101] Fan W, Wang X, Wu Y. Incremental graph pattern matching. ACM Trans. on Database Systems (TODS), 2013,38(3):18.

[102] Fan W. Graph pattern matching revised for social network analysis. In: Deutsch A, ed. Proc. of the 15th Int'l Conf. on Database Theory. New York: ACM, 2012. 8–21.

[103] Fan W, Wang X, Wu Y. Diversified top-*k* graph pattern matching. Proc. of the VLDB Endowment, 2013,6(13):1510–1521.

[104] Udrea O, Pugliese A, Subrahmanian VS. GRIN: A graph based RDF index. In: Holte RC, Howe A, eds. Proc. of the 22nd AAAI Conf. on Artificial Intelligence. Menlo Park: AAAI Press, 2007. 1465–1470.

[105] Bröcheler M, Pugliese A, Subrahmanian VS. DOGMA: A disk-oriented graph matching algorithm for RDF databases. In: Alani H, Kagal L, Fokoue A, Groth P, Biemann C, Parreira JX, Aroyo L, Noy N, Welty C, Janowicz K, eds. Proc. of the Int'l Semantic Web Conf. Heidelberg: Springer-Verlag, 2009. 97–113.

[106] Atre M, Chaoji V, Zaki MJ. Matrix Bit loaded: A scalable lightweight join query processor for RDF data. In: Rappa M, ed. Proc. of the 19th Int'l Conf. on World Wide Web. New York: ACM, 2010. 41–50.

[107] Yuan P, Liu P, Wu B, *et al*. TripleBit: A fast and compact system for large scale RDF data. Proc. of the VLDB Endowment, 2013, 6(7):517–528.

[108] Aluç G, Ozsu MT, Daudjee K. Chameleon-DB: A workload-aware robust RDF data management system. In: Rappa M, ed. Proc. of the 19th Int'l Conf. on World Wide Web. New York: ACM, 2010. 41–50.

[109] Yu JXJ, Cheng J. Graph reachability queries: A survey. In: Aggarwal C, Wang H, eds. Advances in Database Systems-Managing and Mining Graph Data. Boston: Springer-Verlag, 2010. 181–215.

[110] Mendelzon AO, Wood PT. Finding regular simple paths in graph databases. SIAM Journal on Computing, 1995,24(6):1235−1258.

[111] Florescu D, Levy AY, Suciu D. Query containment for conjunctive queries with regular expressions. In: Mendelson A, Paredaens J, eds. Proc. of the 17th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems. New York: ACM, 1998. 139–148.

[112] Calvanese D, De Giacomo G, Lenzerini M, Vardi MY. Reasoning on regular path queries. ACM SIGMOD Record, 2003, 83–92.

[113] Barceló P, Libkin L, Lin AW. Expressive languages for path queries over graph-structured data. ACM Trans. on Database Systems (TODS), 2012,37(4):31.

[114] Pérez J, Arenas M, Gutierrez C. nSPARQL: A navigational language for RDF. In: Sheth A, eds. Lecture Notes in Computer Science. Heidelberg: Springer-Verlag, 2008. 66–81.

[115] Libkin L, Vrgoč D. Regular path queries on graphs with data. In: Deutsch A, ed. Proc. of the 15th Int'l Conf. on Database Theory. New York, 2012. 74–85.

[116] Koschmieder A, Leser U. Regular path queries on large graphs. In: Ailamaki A, Bowers S, eds. Lecture Notes in Computer Science. Heidelberg: Springer-Verlag, 2012. 177–194.

[117] Fan WF, Li JZ, Ma S, Tang N, Wu YH. Adding regular expressions to graph reachability and pattern queries. In: Proc. of the 27th IEEE Int'l Conf. on Data Engineering. Washington: IEEE Computer Society, 2011. 39–50.

[118] Gubichev A, Bedathur SJ, Seufert S. Sparqling kleene: Fast property paths in RDF-3X. In: Boncz P, ed. Proc. of the 1st Int'l Workshop on Graph Data Management Experiences and Systems. New York: ACM, 2013. 14–21.

[119] Dey S, Cuevas-Vicenttín V, Köhler S. On implementing provenance-aware regular path queries with relational query engines. In: Guerrini G, ed. Proc. of the Joint EDBT/ICDT 2013 Workshops. New York: ACM, 2013. 214–223.

[120] Wang X, Ling J, Wang JH, Wang KW, Feng ZY. Answering provenance-aware regular path queries on RDF graphs using an automata-based algorithm. In: Chung CW, Border A, Shim K, eds. Proc. of the 23rd Int'l Conf. on World Wide Web. New York: ACM, 2014. 395–396.

[121] Aggarwal CC, Wang H. Managing and Mining Graph Data (Advances in Database Systems). New York: Springer Science-Business Media, 2010. 1–620.

[122] Plantikow S. Summary chart of cypher, PGQL, and G-core. Neo4j. 2018. https://gql.today/wp-content/uploads/2018/05/ytz-030r1-Summary-Chart-of-Cypher-PGQL-GCore-1.pdf

[123] Wikipedia. The free encyclopedia. LargeTripleStores. 2018. https://www.w3.org/wiki/LargeTripleStores

[124] Eclipse RDF4J. RDF4J. 2018. http://rdf4j.org/

[125] OpenLink Software. OpenLink Virtuoso. 2018. https://virtuoso.openlinksw.com/

[126] Franz Inc. AllegroGraph. 2018. https://franz.com/agraph/allegrograph/

[127] Ontotext. GraphDB. 2018. http://graphdb.ontotext.com/

[128] Blazegraph by Systap, LLC. Blazegraph. 2018. https://www.blazegraph.com/

[129] Stardog Union. Stardog—The knowledge graph platform for the enterprise. 2018. http://www.stardog.com/

[130] JanusGraph Authors. JanusGraph—Distributed graph database. 2018. http://janusgraph.org/

[131] Spmallette. Titan—Distributed graph database. 2018. http://titan.thinkaurelius.com/

[132] Callidus Software Inc. OrientDB-multi-model database. 2018. http://orientdb.com/

[133] CayleyGraph. Cayley. 2018. https://cayley.io/

[134] Amazon Web Services, Inc. Amazon Neptune—Fast, reliable graph database build for cloud. 2018. https://aws.amazon.com/neptune/

[135] Star ArangoDB. ArangoDB—Native multimodel database. 2018. https://arangodb.com/

[136] Microsoft Azure. Microsoft Azure Cosmos DB. 2018. https://docs.microsoft.com/en-us/azure/cosmos-db/introduction

[137] DataStax. DataStax enterprise graph. 2018. https://www.datastax.com/products/datastax-enterprise-graph

[138] Sparsity Technologies. Sparksee—Scalable high-performance graph database. 2018. http://www.sparsity-technologies.com/#sparksee

[139] TigerGraph. TigerGraph—The first native parallel graph. 2018. https://www.tigergraph.com/

[140] Stanford University. Stanford large network dataset collection. 2018. http://snap.stanford.edu/data/

[141] Insight Centre for Data Analytics. The linked open data cloud. 2018. https://lod-cloud.net/

[142] University of Mannheim. DBpedia. 2018. http://wiki.dbpedia.org/About

[143] UniProt Consortium. UniProt. 2018. http://www.uniprot.org/

[144] AKSW. LinedGeoData. 2018. http://linkedgeodata.org/About

[145] Ghemawat S, Gobioff H, Leung ST. The Google file system. In: Scott ML, Peterson L, eds. Proc. of the 19th ACM Symp. on Operating Systems Principles. New York: ACM, 2003. 29–43.

[146] Chang F, Dean J, Ghemawat S. Bigtable: A distributed storage system for structured data. ACM Trans. on Computer Systems (TOCS), 2008,26(2):4.

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

[148] Malewicz G, Austern MH, Bik AJC. Pregel: A system for large-scale graph processing. In: Elmagarmid A, ed. Proc. of the 2010 ACM SIGMOD Int'l Conf. on Management of Data. New York: ACM, 2010. 135–146.

[149] Low Y, Bickson D, Gonzalez J. Distributed GraphLab: A framework for machine learning and data mining in the cloud. Proc. of the VLDB Endowment, 2012,5(8):716–727.

[150] Harth A, Umbrich J, Hogan A, Decker S. YARS2: A federated repository for querying graph structured data from the Web. In: Aberer K, eds. Lecture Notes in Computer Science. Heidelberg: Springer-Verlag, 2007. 211–224.

[151] Choi H, Son J, Cho YH. SPIDER: A system for scalable, parallel/distributed evaluation of large-scale RDF data. In: Cheung D, ed. Proc. of the 18th ACM Conf. on Information and Knowledge Management. New York: ACM, 2009. 2087–2088.

[152] Rohloff K, Schantz RE. High-performance, massively scalable distributed systems using the MapReduce software framework: The SHARD triple-store. In: Tilevich E, ed. Proc. of the Programming Support Innovations for Emerging Distributed Applications. New York: ACM, 2010. 4.

[153] Ladwig G, Harth A. CumulusRDF: Linked data management on nested key-value stores. In: Grau BC, ed. Proc. of the 7th Int'l Workshop on Scalable Semantic Web Knowledge Base Systems. Bonn, 2011. 30.

[154] Shao B, Wang H, Li Y. Trinity: A distributed graph engine on a memory cloud. In: Ross K, ed. Proc. of the 2013 ACM SIGMOD Int'l Conf. on Management of Data. New York: ACM, 2013. 505–516.

[155] Zeng K, Yang J, Wang H. A distributed graph engine for Web scale RDF data. PVLDB, 2013,6(4):265–276.

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

[157] Lee K, Liu L. Scaling queries over big RDF graphs with semantic hash partitioning. Proc. of the VLDB Endowment, 2013,6(14): 1894–1905.

[158] Zhang X, Chen L, Tong Y. EAGRE: Towards scalable I/O efficient SPARQL query evaluation on the cloud. In: Proc. of the ICDE 2013 IEEE Int'l Conf. on Data Engineering. Washington: IEEE, 2013. 565–576.

[159] Gurajada S, Seufert S, Miliaraki I. TriAD: A distributed shared-nothing RDF engine based on asynchronous message passing. In: Dyreson C, ed. Proc. of the 2014 ACM SIGMOD Int'l Conf. on Management of Data. New York: ACM, 2014. 289–300.

[160] Hammoud M, Rabbou DA, Nouri R. DREAM: Distributed RDF engine with adaptive query planner and minimal communication. Proc. of the VLDB Endowment, 2015,8(6).

[161] Jones ND. An introduction to partial evaluation. ACM Computing Surveys (CSUR), 1996,28(3):480–503.

[162] Fan W, Wang X, Wu Y. Performance guarantees for distributed reachability queries. Proc. of the VLDB Endowment, 2012,5(11): 1304–1316.

[163] Ma S, Cao Y, Huai J. Distributed graph pattern matching. In: Mille A, ed. Proc. of the 21st Int'l Conf. on World Wide Web. New York: ACM, 2012. 949–958.

[164] Fan W, Wang X, Wu Y. Distributed graph simulation: Impossibility and possibility. Proc. of the VLDB Endowment, 2014,7(12).

[165] Peng P, Zou L, Özsu MT, 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.

[166] Wang X, Wang JH, Zhang XW. Efficient distributed regular path queries on RDF graphs using partial evaluation. In: Mukhopadhyay S, ed. Proc. of the 25th ACM Int'l Conf. on Information and Knowledge Management. New York: ACM, 2016. 1933–1936.

[167] Papailiou N, Tsoumakos D, Konstantinou I. H2RDF+: An efficient data management system for big RDF graphs. In: Dyreson C, ed. Proc. of the 2014 ACM SIGMOD Int'l Conf. on Management of Data. New York: ACM, 2014. 909–912.

[168] Schätzle A, Przyjaciel-Zablocki M, Neu A, Lausen G. Sempala: Interactive SPARQL query processing on hadoop. In: Mika P, *et al*., eds. Lecture Notes in Computer Science. Cham: Springer-Verlag, 2014. 164–179.

[169] Lai L, Qin L, Lin X. Scalable subgraph enumeration in mapreduce. Proc. of the VLDB Endowment, 2015,8(10):974–985.

[170] Lai L, Qin L, Lin X. Scalable distributed subgraph enumeration. Proc. of the VLDB Endowment, 2016,10(3):217–228.

[171] Bi F, Chang LJ, Lin XM, Qin L, Zhang WJ. Efficient subgraph matching by postponing Cartesian products. In: Proc. of the SIGMOD Conf. 2016. 1199–1214.

[172] Schätzle A, Przyjaciel-Zablocki M, Skilevic S. S2RDF: RDF querying with SPARQL on spark. Proc. of the VLDB Endowment, 2016,9(10):804–815.

[173] Wang X, Xu Q, Chai LL, Yang YJ, Chai YP. Efficient distributed query processing on large scale RDF graph data. Ruan Jian Xue Bao/Journal of Software, 2019,30(3):498−514 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5696.htm [doi: 10.13328/j.cnki.jos.005696]

[174] He L, Shao B, Li Y. Stylus: A strongly-typed store for serving massive RDF data. Proc. of the VLDB Endowment, 2017,11(2): 203–216.

[175] Peng P, Zou L, Chen L. Adaptive distributed RDF graph fragmentation and allocation based on query workload. IEEE Trans. on Knowledge and Data Engineering, 2019,31(4):670–685.

[176] Xin YQ, Wang X, Jin D, Wang S. Distributed efficient provenance-aware regular path queries on large RDF graphs. In: Liu CF, Zou L, Li JX, eds. Proc. of the Database Systems for Advanced Applications. London: Springer-Verlag, 2018. 766–782.

[177] Punnoose R, Crainiceanu A, Rapp D. SPARQL in the cloud using Rya. Information Systems, 2015,48:181–195.

[178] S1CK. Cypher for Apache Spark. 2019. https://github.com/opencypher/cypher-for-apache-spark

[179] LDBC. LDBC Task force: Property graphs data model. 2019. http://www.ldbcouncil.org

[180] Guo Y, Pan Z, Heflin J. LUBM: A benchmark for OWL knowledge base systems. Web Semantics: Science, Services and Agents on the World Wide Web, 2005,3(2-3):158–182.

[181] Schmidt M, Hornung T, Lausen G. SP2Bench: A SPARQL performance benchmark. In: Proc. of the 25th IEEE Int'l Conf. on Data Engineering. Washington: IEEE, 2009. 222–233.

[182] Bizer C, Schultz A. The Berlin sparql benchmark. Int'l Journal on Semantic Web and Information Systems (IJSWIS), 2009,5(2): 1–24.

[183] Morsey M, Lehmann J, Auer S, Ngomo ACN. DBpedia SPARQL benchmark—Performance assessment with real queries on real data. In: Aroyo L, *et al*., eds. Lecture Notes in Computer Science. Heidelberg: Springer-Verlag, 2011. 454–469.

[184] Aluç G, Hartig O, Özsu MT, Daudjee K. Diversified stress testing of RDF data management systems. In: Mika P, *et al*., eds. Lecture Notes in Computer Science. Cham: Springer-Verlag, 2014. 197–212.

[185] Armstrong TG, Ponnekanti V, Borthakur D. LinkBench: A database benchmark based on the Facebook social graph. In: Ross K, ed. Proc. of the 2013 ACM SIGMOD Int'l Conf. on Management of Data. New York: ACM, 2013. 1185–1196.

[186] LDBC. Social network benchmark (SNB). 2019. http://ldbcouncil.org/benchmarks/snb

[187] LDBC. Semantic publishing benchmark (SPB) v2.0. 2019. http://ldbcouncil.org/benchmarks/spb

[188] The GQL Manifesto team. The GQL Manifesto. 2018. https://gql.today/

[189] Speicher S, Arwe J, Malhotra A. Linked data platform 1.0. W3C Recommendation, 2015,26.

附中文参考文献:

[16] 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.

[17] 于戈,谷峪,鲍玉斌.云计算环境下的大规模图数据处理技术.计算机学报,2011,34(10):1753–1767.

[18] 于静,刘燕兵,张宇.大规模图数据匹配技术综述.计算机研究与发展,2015,52(2):391–409.

[19] 王童童,荣垂田,卢卫.分布式图处理系统技术综述.软件学报,2018,29(3):569–586. http://www.jos.org.cn/1000-9825/5450.htm [doi: 10.13328/j.cnki.jos.005450]

[20] 官赛萍,靳小龙,贾岩涛.面向知识图谱的知识推理研究进展.软件学报,2018,29(10):74–102. http://www.jos.org.cn/1000-9825/5551.htm [doi: 10.13328/j.cnki.jos.005551]

[21] 许嘉,张千桢,赵翔.动态图模式匹配技术综述.软件学报,2018,29(3):663–688. http://www.jos.org.cn/1000-9825/5444.htm [doi: 10.13328/j.cnki.jos.005444]

[173] 王鑫,徐强,柴乐乐,杨雅君,柴云鹏.大规模 RDF 图数据上高效率分布式查询处理.软件学报,2019,30(3):498–514. http://www.jos.org.cn/1000-9825/5696.htm [doi: 10.13328/j.cnki.jos.005696]

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.

彭鹏(1987－),男,博士,助理教授,CCF 专业会员,主要研究领域为分布式知识图谱管理.

邹磊(1981－),男,博士,教授,博士生导师,CCF 高级会员,主要研究领域为图数据库,知识图谱,大数据系统.

冯志勇(1965－),男,博士,教授,博士生导师,CCF 杰出会员,主要研究领域为知识工程,服务计算.

王朝坤(1976－),男,博士,副教授,博士生导师,CCF 专业会员,主要研究领域为图和社交数据管理,大数据系统.