面向局域检索的时变图数据存储与查询模型

赵 萍[1] 寿黎但[1,2] 陈 珂[1,2] 陈 刚[1,2] 吴晓凡[3]
(浙江大学计算机科学与技术学院 杭州 310027)[1]
(浙江省大数据智能计算重点实验室(浙江大学) 杭州 310027)[2]
(网易(杭州)网络有限公司 杭州 310051)[3]

摘 要 时变图数据是实体间相互关联、实体属性和实体间关系会发生频繁变化的图结构数据,适用于电子商务的商品与用户关系表示、包含时间维度的知识图谱构建、企业组织架构管理等场景。针对建立时变图数据通用存储检索方案的挑战,文中提出了一种面向局域检索的模型方案,基于图数据库高效的关系检索能力以及分布式键值数据库在存储与查询方面的优势,实现了通用的可提供丰富表达能力的图数据历史存储检索系统。实验证明,所提方案在历史属性存储上具备显著的优势。

关键词 图数据库,时变数据,版本控制,数据查询
中图法分类号 TP311.132.2 **文献标识码** A **DOI** 10.11896/jsjkx.19100530C

Storage and Query Model for Localized Search on Temporal Graph Data

ZHAO Ping[1] SHOU Li-dan[1,2] CHEN Ke[1,2] CHEN Gang[1,2] WU Xiao-fan[3]
(College of Computer Science and Technology, Zhejiang University, Hangzhou 310027, China)[1]
(Key Laboratory of Big Data Intelligent Computing of Zhejiang Province (Zhejiang University), Hangzhou 310027, China)[2]
(Netease (Hangzhou) Network Co., Ltd, Hangzhou 310051, China)[3]

Abstract The temporal graph data is a graph structure data in which the entities are related to each other, and the entity attributes and the relationships between the entities frequently change. This model is applicable to product and user relationships representation in e-commerce, knowledge graphs that contains the history, and corporate organizational structure management. Aiming at the challenge of establishing a general storage scheme for time-varying graph data, this paper proposed a local-domain query based scheme for storing and retrieving time-varying graph data, which is based on the advantage of graph traversal on graph databases and the advantages of distributed key-value databases, achieving universal expression and provide rich expressions for storing graph data. Experiment results show that the system has significant advantages in the storage of historical attributes.

Keywords Graph database, Time-varying data, Version control, Data query

1 引言

时变图数据是实体间相互关联、实体的属性以及实体之间的关系会发生频繁变化的图结构数据。时变的特征为变化发生在图的局部,来自某条关系或实体上某项属性的变更,且变更发生的时间分散。时变是以单个实体、关系或属性为粒度的变更,具有局域性和分散性的特征。与此形成对比的是图的整体版本变更,如知识图谱的定期数据更新和版本维护,是以一定时间粒度下的全图差异作为单元的变更存储和查询。

时变图数据广泛适用于目前互联网的多个领域,如在电子商务领域可以将商品、类目、用户购买评论行为以及商品的评分变化等信息组织成动态图数据,从而实现用户个性化推荐、流量智能调度,提升电商平台的收益。社交网络领域通过组织在线用户之间的关注、交互关系,可以实现交友圈分析、新用户的关注推荐以及可能感兴趣的事件推送,从而实现社交网络应用智能化,提升用户的使用体验。具体以电子商务为例,用不同类型的节点表示商品和用户,用户与商品之间的收藏、购买和评论行为用连接在节点之间的边来表示。在繁

到稿日期:2018-07-15 返修日期:2018-09-25 本文受国家重点研发计划项目(2017YFB1201001),国家自然科学基金项目(61672455),浙江省自然科学基金(LY18F020005)资助。

赵 萍(1992—),女,硕士,主要研究领域为查询处理、数据库优化等;**寿黎但**(1974—),男,博士,教授,博士生导师,CCF 会员,主要研究领域为空间数据库、数据挖掘、数据可视化等;**陈 珂**(1977—),女,博士,副教授,硕士生导师,CCF 会员,主要研究领域为时空数据库、数据挖掘以及数据隐私保护等,E-mail:chenk@zju.edu.cn(通信作者);**陈 刚**(1973—),男,博士,教授,博士生导师,主要研究领域为大数据管理;**吴晓凡**(1984—),男,博士,主要研究领域为大数据智能计算。

忙的电子商城中,用户的参与行为使得图的结构和属性内容发生着持续不间断的变化。在这样的模型下,通常需要进行的查询包括:1)历史结构查询,查询历史上某个时间点下的图结构关系,如用户在过去一个月内的购买情况;2)历史内容查询,查询某个实体随着时间发生的属性变化,如商品在发布后一年内的销量变化。

目前,实际应用的这些时变数据通常采用传统数据库或特制数据库存储,存在应用开发和数据迁移的代价,未能通过建立通用的模型方案来封装实现细节以减少这一领域的成本。

时变图数据管理的主要难点在于变更信息的存储与检索的性能权衡。1)时变图数据的变更发生在离散时间点下的局部数据结构或属性上。局部的变更使得从全图角度而言变更之间的差异微小,难以基于每次变更做版本管理;变更时间分散的特性使得定期镜像方案下对时间窗口的选择成为问题。2)在时变图数据中查询的目标时间点分散,如果完全临时构建目标时间点下的数据会影响查询的响应时间,但若对每个查询时间点构建一份镜像可能造成严重的存储负担。

本文针对时变图数据库的常见需求场景,提出了在图的局域结构和变更查询场景下的时变图数据存储检索优化方案,实现了对历史属性的高效化存储和对历史关系结构的查询统一化。

本文的主要贡献可以概括为以下几点:

(1)提出面向时变图数据的存储方案。通过建立通用数据来存储模型,设计实现了跨数据库存储方案。该方案降低了用户在迁移数据和学习使用时的成本。

(2)设计时变图数据的查询,以支持关于图数据的变更写入和历史查询。在支持基本图数据查询的同时,扩展了图数据的应用场景,能够更高效地满足多领域的应用要求。

(3)基于不同种类的数据库和查询框架的现有优势,实现了上述存储和查询方案,并通过实验验证了方案的可行性。

本文第 2 节介绍了相关研究成果;第 3 节描述了模型需要解决的问题;第 4 节介了绍模型的设计和实现;第 5 节介绍了系统的实现方案和整体框架;第 6 节进行实验对比,以验证模型的可行性;最后总结全文并对未来工作进行展望。

2　相关工作

2.1　语义网络与版本管理

RDF(Resource Description Framework) 是 W3C 提出的一组语义网络表示的模型,丰富地描述和表达了网络资源的内容与结构。它与图的表示形式接近,且由于具有低耦合的特性,是普遍采用的发布、传输语义网络的数据格式。由于互联网上的关联数据随着时间的迁移会发生一系列关系和属性的变更,且往往存在多个创作者,因此如何跟踪变更、查询历史版本以及回滚到原先状态是对关联数据管理的一项重要需求[1-5]。

全部镜像(All Snapshot)方法将每一个版本都存成单独的三元组集合,支持跨多个版本的查询并且可以对比任意版本之间的差异。但如果版本之间的差异不大,那么该方案会造成冗余信息的空间存储代价[6]。

基于变更差异的方法 (Delta-based Approach)来自早期的传统版本管理方法[7]。该方法将版本间的差异作为基础数据存储,处理目标时间点下的版本查询时,通过既有镜像和存储的差异信息进行临时构建。随着构建的差异的增大,查询响应时间逐渐增长[8]。

另有部分学者提出根据版本之间的数据差异构建版本索引[9],以减少冗余信息的存储,并且优化构建临时版本的过程,但会存在版本间有依赖或离线构建索引耗时长等问题[10]。

在以上对于语义网络的版本管理的需求场景中,变更以版本为组织单位,一次版本包含多项变更。查询也以版本为粒度,即查询时间点下的版本数据,比较两个版本之间的变更差异。相应的版本管理方案都是对网络数据的整体变更做存储和索引,对变更的存储和查询相对于时变图数据更加聚合,但由此带来的查询限制也相对明显。而本文研究的时变图数据的场景有所不同,时变图数据的变更具有偶发不确定的特点,相应的查询时间点也没有限制,往往需要依据查询临时构建,不适用于当前场景。

2.2　图数据版本管理模型

除了使用 RDF 存储和检索语义网络外,文献[11-14]提出了带标签的属性图(Labeled Property Graph,LPG) 来对图数据进行建模。标签属性图提出用节点表示实体,用边表示实体之间的关系,而实体和关系都具有属性。RDF 创建的动机在于数据发布与交流,而标签属性图侧重于提升图数据的存储与检索,这样的模型在查询使用上更加简单易懂,在工业界有广泛应用。Neo4j 是基于这一模型的数据存储和查询的典型图数据库。在设计时变图数据存储和检索方向时,一些模型基于图数据库,利用图数据库对实体和关系进行存储,建立对时间信息存储的上层模型。因此,本文利用图数据库现有的互联数据存储检索优势,建立通用数据模型,避免了重造车轮。

TGraph 对属性按照变更进行划分,将结构与静态属性存储在 Neo4j 中,对于动态属性使用额外的物理文件存储管理。这一系统着重面向结构变化不频繁而属性变更频繁的数据场景,在结构变更管理上未深入探讨[15]。

DeltaGraph 方案使用镜像结合日志的方式,通过建立索引结构组织镜像与变更,以提供镜像查询与时间区间下的图差异查询。此类方法关注全图的镜像查询场景,对相近时间区间下的变更集中管理,在图规模大的场景下对镜像图内存存储与临时计算的需求较高[16]。

Ian Robinson 提出了一种模型方案,即在所有关系中存储内部属性字段 [from, to]来表示创建时间和终止时间(structural relationships),从而描述一段关系的存在时间。另外,其定义了身份节点(identity node)和状态(state)。身份节点表示所描述实体的概念,与其他实体概念相关联。状态表示不同的历史镜像,通过带时间属性的 STATE 关系与实体关联在一起。实体间的关系在身份节点之间表示。这一模型相对直观地索引了不同时间点下的历史状态,同时对关系

建立了时间区间的信息维护。如果需要检索特定时间点下的相关关系,可以从实体出发,逐层筛选过滤得到。但其存在的主要问题是不适用于属性变更频繁的场景,随着属性的变更,历史状态节点也会增加,对大量历史状态的存储和索引维护会为系统存储带来负担。因此,在时变图数据的存储应用上还需要考虑对历史镜像的存储优化[17]。

状态模型利用图数据库有效地组织了历史数据,通过关系的起始区间来表述时间信息[18]。文献[18]提出了关系节点(relationnode)的设计方案,以关系作为核心的图版本管理模型。状态模型还引入了网络节点的概念,一个网络节点代表一个关系图,关系图中的所有关系通过变更信息边与网络节点相关联。

状态模型方案通过使用变更边替代了区间的表达方案,使得对关系的表达更加灵活,可以定义关系的删除和重恢复的变更历史,另外也使得关系能够在不同网络之间重用。模型还可以将实体和历史状态以链表结构关联在一起,通过变更关联在网络节点上的边,以及使用时间戳代替版本号,以实现时间信息的管理维护。

3 问题描述与框架

3.1 问题描述

时变图数据由传统图数据与变更事件两部分组成。传统图数据以标签属性图为表示模型。图包括实体和关系,实体和关系有各自的属性值,属性值类型为字符串、数值以及数组。时变图指会随着时间发生改变的图,图的变更包括:1)图结构变更,图中的节点和边被插入和删除,造成图结构关系发生变化;2)图内容变更,图中节点和边所关联的属性发生变更。

定义 1(图结构变更) 已知初始数据图 $G=(V,E,L)$,$L(v)=\{(p_1,val_1),\cdots,(p_k,val_k)\}$,其中 V 表示节点集,E 表示边的集合,L 表示属性映射函数,将节点或边映射到属性对的集合上。对数据图进行结构更新的操作可用二元组 $gc_{\text{struct}}=(op,u)$,$u\in\{e,v\}$,$op\in\{I,D\}$表示。$op$ 表示操作类型,I 表示创建操作,D 表示删除操作,u 表示变更对象,e 表示某条边,v 表示某个节点。经图结构变更后,图 G 的节点集合 V、边集合 E 发生成员变化。

定义 2(图内容变更) 由三元组 $gc_{prop}=(u,p,val)$,$u\in\{e,v\}$表示关系或节点对象,p 表示变更对象下的某个属性,val 表示变更后的属性值。经过图内容变更后,图 G 的属性映射函数发生变化,$L'(u,p)=val$,$G'=(V,E,L')$。

定义 3(时变图数据) $GC_t=\{gc_1,\cdots,gc_k\}(k\geqslant 1)$ 表示 t 时刻下的变更操作,$gc\in\{gc_{struct},gc_{prop}\}$表示图结构变更或图内容变更。$GC:G->G'$表示数据图 G 基于更新操作 GC 得到新数据图 G'的过程。在时间域 $[0,T]$上的动态图 $G[0,T]$为数据图序列,定义如式(1)[19]所示:

$$G^{[0,r]}=\{G_0,G_1,\cdots G_r\} \tag{1}$$

$$G_0=G,GC_0=\varnothing,GC_t:G_{t-1}\rightarrow G_t,1\leqslant t\leqslant T \tag{2}$$

时变图数据为从图的创建至结束的整个时间维度上的所有历史变更的信息数据。时变图数据的历史查询按查询对象可以分为两个部分:结构查询和内容查询。结构查询在于查询图的节点连接关系,内容查询在于查询节点的历史状态。

由于在数据规模大的场景下处理全局查询,如获取时间点下的所有实体,因此对空间存储或计算性能的要求高,需要独立设计高性能的图计算系统。这里将优化场景限于局部查询进行讨论。以提升局域查询场景为主要目标,全局的查询基于各目标对象的局部查询实现。

定义 4(历史查询) 已知动态图 G,关于时变图数据的历史查询包括:

1)历史结构查询,给定时间 $t\in[0,T]$,指定目标节点 v,查询 v 的邻近节点集合 U。

$$U=\{v_1,\cdots,v_n\mid 1\leqslant i\leqslant n,(v,v_i)\in E_t\},G_t=(V_t,E_t,L_t) \tag{3}$$

2)历史内容查询,给定时间 t、节点 v、v 的属性 p,$t\in[0,T]$,$G_t=(V_t,E_t,L_t)$,$v\in V_t$,查询 Lt 下的映射属性值 $val=L_t(v,p)$,$v\in V_t$。

3.2 优化目标

系统对图的处理分为底层的物理存储和经过存储模型构建转换后的概念图。概念图是用户可理解层面的抽象图。系统以空图为初始数据,接收关于概念图的输入,输入分为两类:变更和查询。1)变更:向系统提交关于图的变更,系统执行并输出变更所代表的目标 id。2)查询:向系统提交基于时间维度的图查询,系统解析查询需求,计算返回概念层的查询结果。

框架描述与示例如图 1 所示。用户分别提交了变更和查询两类请求,左侧变更请求在图中创建节点,系统在创建节点后返回相应的节点 id。右侧查询请求查找时间区间下的节点变更,系统返回一系列包括时间区间下的创建节点的变更记录。系统处理用户请求,首先将概念图的描述转换为至底层存储的变更或查询,完成处理后将结果返回给用户。

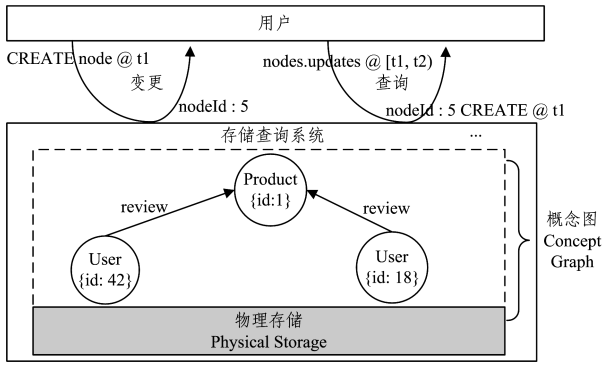


图 1 存储查询系统的输入和输出

Fig. 1 Input and output of query and storage system

系统依据用户的局域查询临时构建相应的局域概念图。在查询处理过程中,物理存储与概念图之间存在构建转换的时间代价。物理存储持久化变更数据需要一定的存储空间代价。为了达到系统的预期目标,需要解决时变图数据场景下的物理存储空间和概念图时间点镜像临时构建性能的权衡问题。形式化定义如式(4)所示,计算时变概念图在数据规模 S_{cg} 下系统的整体物理存储代价 S_{ps} 和图局部查询场景下的响应时长 Q,期望保证历史内容和结构的查询响应时长在 O(1) 内,即在局域查询效率不随数据规模增长而改变的场景下,尽

可能减小系统存储代价 S_{ps}。

$$\min S_{ps}(G^{[0,T]})$$
$$\text{s.t. } Q(S_{cg}) \sim \Theta(1) \quad (4)$$
$$|S_{cg}| \geqslant 1$$

4　存储与查询

4.1　存储模型

4.1.1　结构历史存储

时变图数据的结构信息包括：1）从图的创建至图的结束为止的所有节点和关系，这里称之为结构对象；2）结构对象的创建和删除的变更信息。一方面，结构对象的描述存在不同的时间维度下的状态，例如将商品的生存周期划分成概念发布，上架、下架两个时间纬度语义分别管理商品的创建和删除。另一方面，变更的描述需要具备灵活的事件表达能力，例如删除关系和实体后恢复的事件表达，需要包含变更事件的元信息描述，如创建者、编辑理由等。

经过对比发现 2.2 节中网络节点的模型适用于描述多时间维度事件的场景。模型创建内部的网络节点作为统筹管理节点的核心，利用网络节点到目标节点的边描述对于目标节点的变更事件。模型将用户在时间维度上定义的语义表示为一个网络节点，用于管理该维度上的图数据信息，同一实体可以复用存在于多个网络中。变更信息的边上属性包括：变更事件类型（创建或删除）以及事件发生时间。在变更边上可定义字段存储元变更信息。

原网络模型通过网络节点与关系的关联来管理网络，核心思想是对关系的版本管理。原模型应用在时变图数据的场景下，除了需要使用时间戳替代原先的版本号以外，还需加入对实体本身的事件管理。本文已将图的历史内容和结构存储分离，结构模型中不需要考虑历史状态的管理，对实体的事件管理可设计为与关系节点一致的组织方式。经过对网络模型的调整，得到的历史结构存储模型如图 2 所示。网络模型对关系节点的索引和对节点一样，通过从网络节点出发的边存储相关变更事件。目前出于设计的简化的目的，关系节点和实体节点连接的网络节点是同一个，变更关系也是一致的表达，从而统一了对其的查询封装过滤逻辑。

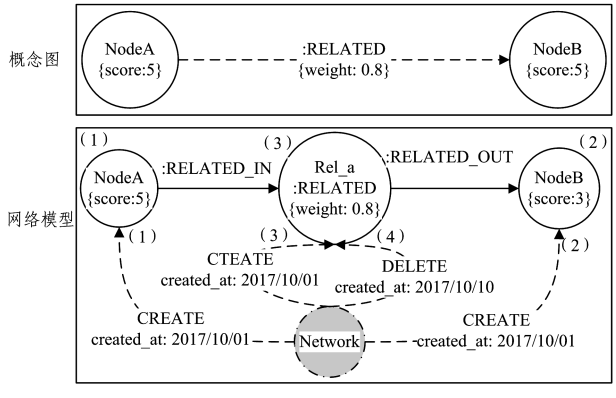


图 2　网络模型与关系节点应用举例

Fig. 2　Example of network model and relation node

概念图中首先分别创建了实体 NodeA（1）与 NodeB（2），而后建立关系 RELATED（3），之后将关系 10/01 删除（4）。在节点创建的同时系统建立变更边连接 Network 节点，如图 2 中的流程（1）和（2）所示。关系建立时系统创立相应标签的关系节点连接两个目标节点，并连接网络节点，如图 2 中的流程（3）所示。当用户需要删除目标关系时，系统并不在物理存储中直接抹除目标结构，而是建立相应的类型为删除的变更边连接目标结构与网络节点，如图 2 中的流程（4）所示。

4.1.2　属性历史存储

时变图数据的实体关系属性内容是时变图数据的重要部分，记录存储时变图的属性变更。高效检索图中属性的历史状态是时变图数据的重要需求。

时变图数据作为实体和关系属性内容的重要载体，其中的实体往往具备多方面的属性描述，且图中属性值的变化具有偶发性、局部性和独立性的特点。独立性表现在属性变更之间相互独立，例如同一实体中属性 a 的变化独立于属性 b。属性可分为静态属性和动态属性，静态属性变更较少、相对稳定，如商品名称，而动态属性受到社交行为影响变更频繁，如商品评价、销售量等。在状态模型下，随着动态属性的变更，静态属性的数据内容需要被复制，造成了数据的冗余。如能将属性之间的变更存储解耦，则能达到降低内容存储的效果。

通用的时变图数据管理与查询系统无法预测图数据属性的变更频繁程度，但通过分离实体下属性的变更事件存储，可以有效地减少动态属性带来的状态历史膨胀。考虑到关于历史的数据检索往往以单个属性为主要查询目标而非某个实体的所有属性状态，且关于实体的单次属性变更也以局部属性变更为主，因此在时变图数据的场景下将历史属性值以属性的粒度进行划分管理是适用的。

本文提出将实体中的历史镜像细化到属性粒度进行存储，从而提升存储效率。对每一个实体的每一项属性建立日志存储结构，存储属性每一次发生变更后的状态。模型通过解耦实体属性，避免了由于属性变化的组合而带来的历史状态的组合。故而通过减小镜像的存储粒度至属性级别，减少了局部属性发生变化时的存储代价[24]。

图 3 给出了存储历史内容的方案。首先创建了 NodeA，$score=2$，创建相应属性字典与属性日志结构（见图 2 中的流程（1）），其中修改了属性 score 为 5，在相应的日志中添加记录（见图 2 中的流程（2）），然后创建了节点 NodeB 和相应日志（见图 2 中的流程（3）），最后 NodeB 发生变更，$score=3$，并将相应描述添加入日志（见图 2 中的流程（4））。

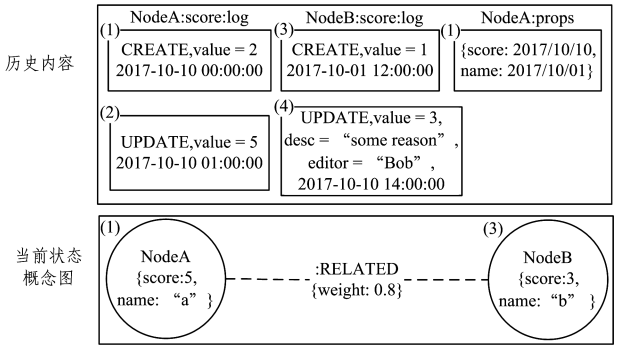


图 3　历史内容存储模型

Fig. 3　History storage model

4.1.3　查询模型

基于前两节中的存储模型描述，图 4 给出了对图的变更

示例。结构创建的同时需要向日志中添加相应的记录以及为网络节点添加相应的变更关系。查询时,根据日志变更检索时间点下的最近变更状态,从而得到历史状态。

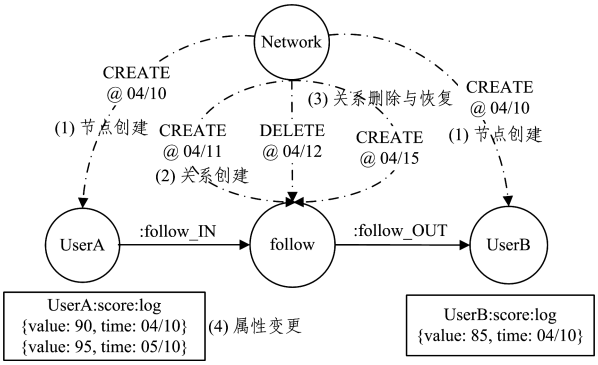


图 4 基于存储模型的变更实例

Fig. 4 Example of data updating process in storage model

时变图数据的变更流程如图 5 所示。系统将用户给定的变更操作分为属性内容和结构两部分进行记录,这两部分的变更可以并行执行,执行完成后检查一致性。如果在执行过程中出现问题,则执行回滚,重新执行该步骤;如果顺利完成记录变更,则返回相应的数据结构。

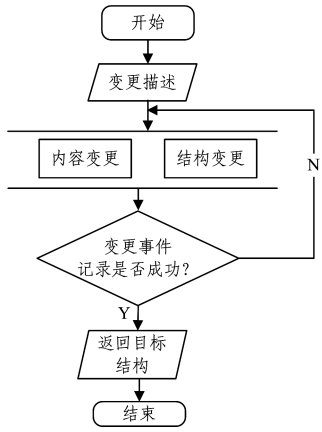


图 5 时变图数据变更流程图

Fig. 5 Flowchart of update of time-varying graph data

时变图数据的查询流程如图 6 所示。由于变更存储的划分,系统对用户的查询也进行了相应的分类,将对结构和对历史属性的查询交由不同的底层来存储执行,最终返回相应的结果。

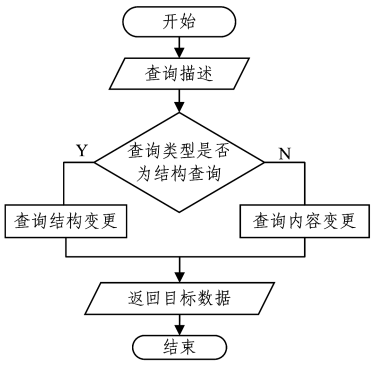


图 6 时变图数据查询流程图

Fig. 6 Query flowchart of time-varying graph data

下文将通过对变更和查询的分类,来分别描述相应的示例和执行流程。在例子中,以 Tinkerpop 框架的查询语言 Gremlin 说明执行流程,但在一些场景下接口上有差异。Gremlin 基于 Groovy 实现。作为同样基于 JVM 的语言,Groovy 支持对 Java 语言的直接调用。在时变图数据查询中传入 Java 的时间类型 Date,Date 的构造参数为毫秒级别的时间戳。

历史结构查询:给定时间点 t,查询该时间点下存在的实体,根据条件过滤返回满足条件的节点。举例如下:

```
graph.when(new Date()).V().has(id,1)
  => V[1]
```

1. 依据 id 查找目标节点 v;
2. 获取 v 与网络节点 n 之间的变更关系 $R=\{(v,n)\}$;
3. 条件过滤 $R'=\{r \mid r\in R, r.created\leqslant t\}$;
4. 在 R' 中取创建时间最大的记录 r

$$\underset{r\in R}{\operatorname{argmax}}\ r.created$$

5. 若 r 的类型为 create,表示时间点下目标节点存在,则返回 v,否则表示节点已删除。

历史内容查询:给定时间点下查询某节点的属性值。举例如下:

```
graph.when(new Date(1530316800000))
  .V(42).value("name")
  => "Bob"
```

1. 检索目标节点 v,获取 v 的 id;
2. 在属性内容中查找 v 所属 id 下属性 p 的日志结构 log;
3. 查找 log 不超过 t 的最近日志记录 r

$$\underset{r\in R}{\operatorname{argmax}}\ r.time \mid r.time\leqslant t$$

4. 返回 r 中的属性值。

5 查询系统实现方案

由于时变图数据具有变更频繁的特性,在对时变图数据进行存储时,需要保证变更信息写入及时、不冲突,以及稳定可持久化的存储。作为时变图数据的底层,基本查询的提供系统需要保证查询响应足够快,这决定了系统的设计方向是提供联线事务处理平台(OLTP)。Neo4j 是 OLTP 的典型图数据库,通过对节点和关系的原生存储提升了对连接关系的查询检索效率。时变图数据是以图为基础结构的包含频繁变更信息的历史数据,目标系统在既有的完善的图数据库设计的基础之上扩展,以实现对历史变更的管理,是在有限的时间精力下避免重造车轮、站在巨人肩膀之上的优化方案[21]。

然而,由于图数据领域的飞速发展,图数据库的更新也日新月异,各类支持不同功能和场景的图数据库不断涌现和更新。而目前时变图数据的存储和检索是另外一项未标准化的而现实中普遍存在的需求。本文提出的方案并不是基于文件层级的存储方案,而是以不同数据库范式为基础的模型选择和设计方案。本文提出的基于图数据存储建立统一的模型方案,具备跨越底层实现,不拘泥于既有框架的功能,具有更高的通用性和可移植性。

图数据库擅长图结构的存储,但对于冗长的历史镜像存储和检索无法发挥优势。另外,图数据库存在对图数据做分片存储的问题[22]。因此,不管是从功能支持还是存储限制的

角度,图数据库都难以实现关于历史内容的管理检索。鉴于此,不如考虑分离出历史数据,转而使用其他存储和检索媒介。

对比其他存储媒介存储图的历史数据的方案,如将各类实体的所有历史状态存储在关系型数据库的关系表中,表中的一行代表一类实体中某一实体在某时间点上的历史状态。但关系型图数据库要求预先设置数据类型,在实际应用时不如 NoSQL 灵活,因此我们考虑采用其他方案。其中一种选择是擅长日志检索的存储体系如 Elastic[23]。由于在数据规模较小的实验环境下,Elastic 框架过于笨重,我们在测试时选用了具备日志流时间查询的轻量内存数据库 Redis 替代文档检索引擎。Redis 是通过 C 语言实现的短小精悍的内存数据库,能实现高效的分布式读写性能,被广泛用于查询缓存、通用键值读写等领域[24]。流日志功能作为 Redis 新提出的数据结构,支持日志流类型,可存储和检索时间点下的键值对数据,可以提供高性能的日志检索功能[25]。

通过利用图数据库和日志检索系统的既有优势,将时变图数据整体上分为两部分:结构数据与内容数据。结构数据存储在图数据库中,内容数据存储在 Redis 中。通过查询封装供外部调用,时变图数据存储查询框架如图 7 所示。

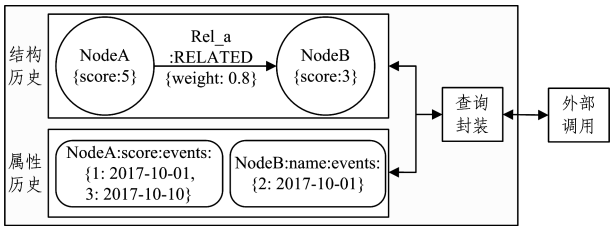


图 7　时变图数据存储查询框架

Fig. 7　Query framework of time-varying graph data storage

结构数据指包括从图的创建到终结为止,图中网络、网络下的所有节点、节点的相互关系的所有创建和删除历史。在任意时间点的有关图的关系遍历以及时间区间下图的结构变化,都是对结构数据的查询。这些结构历史数据通过模型构建存储在 Neo4j 图数据库中,通过 Neo4j 的 API 调用提供存储和检索功能。

内容数据指所有图中节点和关系的历史属性信息。在时间区间下的节点的历史属性查询,以及已知历史属性值查询曾经具有该值的节点,都是对内容数据的查询。内容数据以日志的形式存储在擅长日志存储检索的数据库中,根据查询需求及时返回存储结果。

一方面,结构数据与内容数据的分离存在存储时两方的数据一致性问题;另一方面,对于查询重于写入的场景而言,数据分离可以化解查询的负担,通过对查询的分类将查询分配到不同的存储检索引擎上,以提升系统的查询处理吞吐量。

在实际应用中,出于通用性的考虑,需要在存储模型上层建立查询封装,从而免去查询时对内部细节的学习了解。Tinkerpop 是由 Apache 提出的开源查询框架,为图数据的处理系统提供了统一的查询语法,提升了图查询的通用性[26]。借助 Tinkerpop 的查询封装,可以融合上层既有的聚合运算框架,从而支持更多、更丰富的查询计算功能;同时经过外部程序可以通过 HTTP 接口调用查询,从而支持查询服务化,使得框架易用[26-27]。

6　实验

6.1　实验环境

本文的模型与对比模型搭建在同一服务器上,以 Neo4j 为模型进行存储,上层以 Tinkerpop 框架的 gremlin-server 服务提供查询调用和数据返回。具体服务器硬件配置如表 1 所列。模型对比实验场景下相关的软件环境配置如表 2 所列。

表 1　硬件及系统配置

Table 1　Hardware and system configuration

名称	型号
操作系统	Ubuntu 14.04 LTS Server
CPU	2GHz * 8 Core
内存	64GB

表 2　软件配置

Table 2　Software configuration

名称	版本
JRE	1.8.0_161-b12
Neo4j	3.3.1
Tinkerpop	3.3.2-SNAPSHOT
Redis	2.8.4

本文选用的数据集为亚马逊的商品原信息数据[28],描述了部分亚马逊商品的分类、销售排名、相关商品以及用户评论历史。为了丰富场景中的动态历史属性,实验基于该数据[28]构建出商品和用户之间的评论关系图,为商品创建平均历史评分的属性,存储时间点下的整体平均评分,评分随着用户评价发生历史变化。实验中,取评论历史中第一条评论的时间作为上架时间,最末一条评论的时间作为下架时间来模拟真实情况。

为了分析数据集中的关系与实体增长趋势,在测试环境中将商品数量以每 1 000 条为批次,对比不同商品数量下商品相关的评论和评论用户的数据数量,结果如图 8 所示。通过对比发现,评论和评论用户的数据数量是商品数量的十倍以上。在商品数量为 100000 的情况下,模型中评论数量和评论用户数量已过万级。这种数据内容比重分布一定程度上影响了查询效率,使得关系的数据数量远高于实体的数据数量。

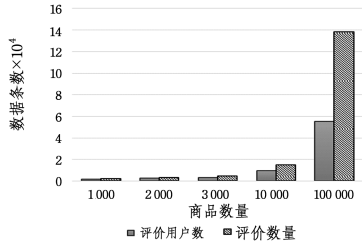


图 8　数据集实体关系数量增长趋势图

Fig. 8　Graph of entity relations quantity growth in dataset

6.2　模型评估

对于存储模型,在对比实验中需要测试存储等量的信息数据所需的空间大小,以及同样的查询所需的返回时间。存储空间的大小决定了模型的部署应用门槛,而查询响应时间决定了模型的应用场景领域。

为了构建模型对比,实验使用上层抽象框架调用数据库来实现,根据不同的模型方案来实现同一查询需求,从而实现

同一语义的数据图在不同模型下的存储和查询。模型描述如图 9 所示。图 9 中左侧为实验模型，右侧为对比模型即状态模型。状态模型的所有数据都存储在图数据库中，而实验模型中的一部分数据存储在内存数据库中。

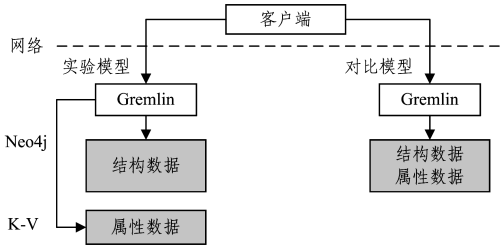


图 9 实验架构

Fig. 9 Experiment structure

本文模型以 Neo4j 存储图结构数据，存储代价为 S_{struct}，以 Redis 存储属性数据，存储消耗为 S_{prop}。如果两者的存储代价之和远小于对比模型的存储代价 S'，则可认为在存储方面本文提出的模型具有优势。在具体物理存储代价的评价方面，Neo4j 图数据库以存储文件在磁盘上的占用量衡量；Redis 作为内存数据库以，INFO 命令查询服务信息中的数据内存占用指标来衡量。

在查询方面，实验以基础查询为评测对象，比较本文模型与状态模型方案的性能。查询分为结构查询和历史查询。方案中包括底层存储和统一查询框架两层，由于接口调用，层级之间的数据传输会存在明显的性能差异，出于公平性的考虑，实验在不同的场景下都需要在同一层进行对比查询。由于两类模型都将结构历史存储在 Neo4j 中，因此关于结构的查询以基于 Neo4j 的图查询语言 Cypher 执行；而涉及到属性内容查询时，由于本文模型封装了 Redis 调用，两者统一通过查询框架 Tinkerpop 的查询语言 Gremlin 执行。

查询性能的评测以返回时间为主要衡量标准，而数据库(如 Neo4j，MySQL)存在冷启动的时间，且查询响应速度会受到运行环境的影响。测试过程中，对同一条查询进行 $1+N$ 次调用，抛去第一次查询的响应时间，将后 N 次的查询响应时间作平均后返回得到去抖动后的结果，计时精确度以纳秒为单位。为避免网络的影响，将查询前后的计时命令传输到服务器端解析执行，将最后统计结果返回给客户端。N 在查询响应快的场景下的取值为 1 000 次，在耗时长的查询下的取值为 10。

6.3 结果分析

关于查询性能的对比，实验将查询分为数据变更和查找两部分。变更主要包括实体创建、关系创建和属性变更，是时变图数据的主要变更方式。查找方式主要包括基于时间点的查找和基于时间区间的查找，这里对比基于时间点的实体、关系、属性值查找。基于时间区间的查找与此相似，故不再对比基于单个实体的时间区间查找，而是对比基于时间区间的全局变更，以展示在不同数据规模下的性能差异。

为了对比两个模型对数据集的存储资源的占用情况，通过对商品数量以 1 000 条为单位，分批次将商品相关的用户和评论创建在两个模型中，计算它们的存储消耗。由于用户评论在数据集中所占的比重较大，且包含的属性信息少，这里将其作为结构数据统计，记为 S_{struct}。而将包含众多历史属性的商品用户实体信息作为属性数据，其存储代价记为 S_{prop}。经过统计得到的存储占用分布如表 3 所列。

表 3 时变图数据存储代价的结果对比

Table 3 Storage cost comparison between models

(单位：MB)

商品规模	模型	S_{prop}	S_{struct}	合计
$N=1000$	实验模型	5.7	50.3	66.72
	对比模型	27	28	56
$N=2000$	实验模型	9.6	77.4	103.64
	对比模型	41	59	100
$N=3000$	实验模型	14	122	159.81
	对比模型	75	76	151
$N=10000$	实验模型	51	398	519.26
	对比模型	238	203	441
$N=100000$	实验模型	483	3408.2	4408.99
	对比模型	2252.8	1740.8	3993.6

实验结果表明，在数据规模小的场景下，如 $N=1000$ 时，对比模型优于实验模型。随着商品规模的增大，属性数据在当前模型下的存储代价显著低于状态模型。在 N 为 100 000 时，实验模型的属性数据存储不足结构存储的 1/4，而结构数据存储为对比模型的两倍。

实验模型对历史数据的组织避免了状态中重复属性值的存储冗余，而在整体上当前方案的存储代价与状态模型方案持平甚至略高。这是由于本文模型中关系的结构存储代价高于状态模型，而在数据集中关系的数量规模相比商品数量差距较大造成的。然而，即使关系结构复杂导致存储代价上升，由于模型将历史内容和历史结构分离，历史内容可以存储在其他节点中甚至分布式存储，因此能相应地缓和原模型在单节点上的存储压力，故在多节点存储的场景下对于关系数据的存储代价是可以接受的。由此可以得出：在图结构复杂的场景下，随着关系数量的增长，存储的空间代价会有明显上升，而在历史属性复杂的场景下，实验模型的存储方案有着显著的优势。

为了比对模型关于图数据变更的响应能力，实验对比了两个模型下实现创建实体、创建关系和更改属性的查询的平均响应时间，结果如图 10 所示。

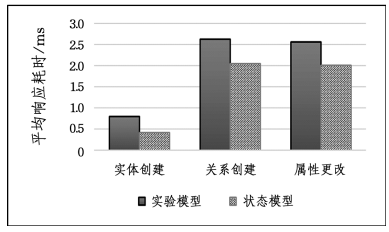


图 10 不同模型变更响应时间的对比

Fig. 10 Response time comparison for updates in different models

从实验结果可以看出，实验模型在变更上的效率不如状态模型，实体创建与属性变更由于涉及到对不同数据库的数据调用和信息同步存在代价消耗，导致变更效率受到影响。而在关系创建时由于模型提升了关系表示的复杂度，因此产生了性能代价。

对基于时间点的查询进行实验，给定时间和实体 ID，两种模型分别执行时间点查询、相邻的关系节点查询和历史属

性值查询,得到的结果如表 4 所列。

表 4　时间点下查询响应时长的对比结果

Table 4　Response time comparison of query at time point

(单位:ms)

商品规模	模型对象	目标实体状态	目标关系状态	目标实体历史属性值
$N=1000$	实验模型	0.2034	0.2826	0.1688
	对比模型	0.3727	0.4475	0.3409
$N=2000$	实验模型	0.1480	0.3022	0.2178
	对比模型	0.3336	0.4099	0.3694
$N=3000$	实验模型	0.2598	0.2734	0.2353
	对比模型	0.4257	0.3094	0.3416
$N=10000$	实验模型	0.2777	0.3114	0.1259
	对比模型	0.6628	0.4819	0.6850
$N=100000$	实验模型	0.2283	0.3303	0.6160
	对比模型	0.3778	0.3141	0.7477

实验结果显示,在检查目标实体状态和历史属性时,实验方案存在一定优势,这是由于模型对实体的状态描述结构不同,状态模型将对实体结构的变更信息和对实体属性的变更信息进行统一表达,导致需要综合所有历史状态并通过过滤分析来检查实体的状态,影响了检索的性能。而目标模型将两者分离,使得查询目标明确,避免了结果运算的过程。

而在实体相邻关系上,实验方案存在显著不足,这是由于关系节点的方案提升了图结构的复杂度,决定了关系遍历时当前关系的存在状态检查比关系的时间区间起始检查更加复杂。但由于目标模型在对关系的表达和管理能力上优于对比方案,因此在关系检索计算相对次要的环境下,这样的性能代价是可以接受的。

与此同时,在不同规模数据集上,随着数据集的增长,各项查询的复杂度并没有明显提升,举例目标实体状态的比较如图 11 所示。商品数量、关系数量的增长,不影响对单个实体状态、关系遍历和历史属性的查询性能,即使随着数据量的增大,在不超过存储和内存负载的情况下,对于局部图的检索查询性能有稳定保证,该优势源自由图数据库存储和查询的特性。

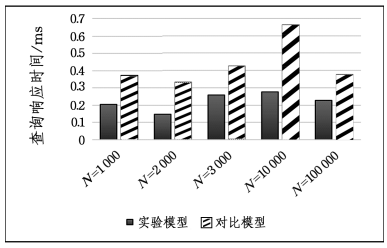


图 11　时间点下不同数据规模单节点查询响应时间的对比

Fig. 11　Comparison of query response time of single node with different data sizes at time points

从单个实体出发的基于时间区间的历史查询与时间点的查询的流程相似,结果差异不明显,因此不再进行实验对比。对于时变图数据的查询不仅限于从局部点出发的查询,从全图的角度基于时间区间的历史变更也可以反映图的变化趋势,这也是时变图数据在实际应用中的重要查询之一。实验中,在给定时间区间时,我们执行了如下查询:查询时间区间的起始时间和终止时间下的图状态之间的差异、查询时间区间内新建的节点和关系,结果如表 5 所列。

表 5　时间区间下创建节点历史查询对比结果

Table 5　Comparison of creating node history queries in time intervals

商品规模	模型对象	新创建的节点	
		变更数量	响应时长/ms
$N=1000$	实验方案	183	10.93
	对比方案		32.11
$N=2000$	实验方案	384	14.08
	对比方案		47.05
$N=3000$	实验方案	558	26.67
	对比方案		77.69
$N=10000$	实验方案	1903	88.39
	对比方案		323.27
$N=100000$	实验方案	18524	1747.50
	对比方案		3174.08

表 6　时间区间下创建关系历史查询对比结果

Table 6　Comparison of creating relational history queries in time intervals

商品规模	模型对象	新创建的节点	
		变更数量	响应时长/ms
$N=1000$	实验方案	7115	61.22
	对比方案		63.16
$N=2000$	实验方案	12888	110.17
	对比方案		119.43
$N=3000$	实验方案	19595	174.75
	对比方案		242.69
$N=10000$	实验方案	71038	249.95
	对比方案		1122.69
$N=100000$	实验方案	678150	2187.14
	对比方案		15723.00

图 12 给出了时间区间下新建实体的返回效率随着变更差异的影响。由图 12 可知,在数据规模增大时所有方案的性能都受到了影响,而状态模型受到的影响较大,这与状态模型的状态检查能力相关。

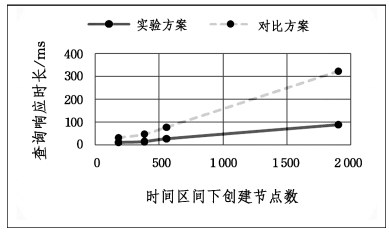


图 12　时间区间下实体变更查询效率与变更数量的关系

Fig. 12　Relation between entity change query efficiency and change number in time interval

由于响应时长受到硬件性能和运行时环境等多方面的影响,实验统计了在 $N=100000$ 的数据场景下各类查询的 IO 次数作为对比标准,结果如表 7 所列。

表 7　$N=100000$ 场景下各查询 IO 次数的统计对比

Table 7　Comparison results of IO count when $N=100000$

查询类型	实验模型	对比模型
目标实体状态	9	360
目标关系状态	177	37
目标实体历史属性值	7	652
新创建的节点	60177	1028868
新创建的关系	89166	10689692

局部查询的内容为:查询在 2005/06/16 时间点下的目标实体为 ID=0790747324 的商品,查询目标商品与 ID 为 A22IK3I6U76GX0 的用户的评论关系。全局的查询对象为从 2000-2003 年之间时间区间下图的所有变更。

实验结果显示在对图的新建实体的查询上实验模型存在显著优势,而在关系查询上存在劣势。这是因为关系查询受数据场景和状态模型表达能力的限制两方面的影响,实验数据中不存在实体间的关系发生反复变更的场景。数据中同一用户对同一商品的多条评论属于多条关系,在这样的场景与状态模型中以起始时间表达关系的方式相契合,表达简洁,故而在查询时效率也更高。

结束语 本文主要研究了实体属性、关系结构发生长期连续变更的图数据的存储和检索方案,提出了对时变图数据分离存储和建模的方案,最大化地利用了现有数据库研究的优势,便于实现具有存储和查询变更历史、丰富表达能力的图数据库系统。通过实验证明,本文提出的模型相比传统的状态模型可优化解决历史数据的存储问题,在实体状态和历史属性检索以及全图的实体变更查询上具备性能优势。后续工作将研究在关系上的模型优化以提升存储和查询性能,以及开发全局的历史属性检索等功能。

参 考 文 献

[1] ARENAS M, GUTIERREZ C, PÉREZ J. Foundations of RDF databases[C]// Reasoning Web International Summer School. Berlin: Springer, 2009: 158-204.

[2] BERNERS-LEE T, HENDLER J, LASSILA O. The Semantic Web[J]. Scientific American, 2001, 284(5): 28-37.

[3] W3C. SPARQL query language for RDF [EB/OL]. https://www.w3.org/TR/rdf-sparql-query/.

[4] BOLLACKER K, EVANS C, PARITOSH P, et al. Freebase: a Collaboratively Created Graph Database for Structuring Human Knowledge[C]// Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data. New York: ACM, 2008: 1247-1250.

[5] MAX V, GROZA T. SemVersion: An RDF-based ontology versioning system[C]// Proceedings of the IADIS International Conference WWW/Internet. 2006.

[6] TICHY, WALTER F. RCS-a system for version control[J]. Software: Practice and Experience, 1985, 15(7): 637-654.

[7] CLAUDIUS H, BROCCO M, WÖRNDL W. Scalable Semantic Version Control for Linked Data Management[C]// LDQ@ ESWC. 2015.

[8] YANNIS T, THEOHARIS Y, ANDREOU D. European Semantic Web Conference[M]. Berlin: Springer, 2008.

[9] IM D H, ZONG N, KIM E H, et al. A Hypergraph-based Storage Policy for RDF Version Management System[C]// Proceedings of the 6th International Conference on Ubiquitous Information Management and Communication. New York: ACM, 2012: 74-78.

[10] BARRASA J. RDF Triple Stores vs. Labeled Property Graphs: What's the Difference"[R]. Neo4j, 2017.

[11] RENZO A, GUTIERREZ C. Survey of graph database models [J]. ACM Computing Surveys (CSUR), 2008, 40(1): 1.

[12] MILLER J J. Graph Database Applications and Concepts with Neo4j[C]// Proceedings of the Southern Association for Information Systems Conference. Atlanta: AIS, 2013, 2324(S36): 141-147.

[13] IAN R, WEBBER J, EIFREM E. Graph databases [M]. O'Reilly Media, Inc., 2013.

[14] MIAO Y S, et al. Immortalgraph: A system for storage and analysis of temporal graphs [J]. ACM Transactions on Storage (TOS), 2015, 11(3): 14.

[15] HUANG H X, et al. Tgraph: a temporal graph data management system[C]// Proceedings of the 25th ACM International on Conference on Information and Knowledge Management. ACM, 2016.

[16] UDAYAN K, DESHPANDE A. Efficient snapshot retrieval over historical graph data[C]// 2013 IEEE 29th International Conference on Data Engineering (ICDE). IEEE, 2013.

[17] Neo Technology. Time-Based Versioned Graphs [EB/OL]. iansrobinson.com/2014/05/13/time-based-versioned-graphs.

[18] Neo Technology. GraphGist: Network versions [EB/OL]. neo4j.com/graphgist/network-versions.

[19] XU J, ZHANG Q Z, ZHAO X, et al. Survey on Dynamic Graph Pattern Matching Technologies[J]. Journal of Software, 2018, 29(3): 663-688. (in Chinese)
许嘉,张千桢,赵翔,等. 动态图模式匹配技术综述[J]. 软件学报, 2018, 29(3): 663-688.

[20] VIJITBENJARONK W D, LEE J, SUZUMURA T, et al. Scalable Time-Versioning Support for Property Graph Databases [C]// 2017 IEEE International Conference on Big Data. New York: IEEE, 2017: 1580-1589.

[21] SALIM J, VANSTEENBERGHE V. An empirical comparison of graph databases[C]// 2013 International Conference on Social Computing (SocialCom). IEEE, 2013.

[22] FLORIAN H, PEINL R. Performance of graph query languages: comparison of cypher, gremlin and native access in Neo4j [C]// Proceedings of the Joint EDBT/ICDT 2013 Workshops. ACM, 2013.

[23] CLINTON G, TONG Z. Elasticsearch: The Definitive Guide: A Distributed Real-Time Search and Analytics Engine [M]. O'Reilly Media, Inc., 2015.

[24] CARLSON, JOSIAH L. Redis in action[M]. Manning Publications Co., 2013.

[25] ANTIREZ. Streams: a new general purpose data structure in Redis[EB/OL]. http://antirez.com/news/114.

[26] RODRIGUEZ, MARKO A. The gremlin graph traversal machine and language (invited talk)[C]// Proceedings of the 15th Symposium on Database Programming Languages. ACM, 2015.

[27] MARCUS P, LEHNER W, RUDOLF M. Extending graph traversals with application logic [P]. U. S. Patent Application No. 14/820, 357.

[28] LESKOVEC J, ADAMIC L A, HUBERMAN B A. The dynamics of viral marketing [J]. ACM Transactions on the Web (TWEB), 2007, 1(1): 5.