Goldfish:基于矩阵分解的大规模 RDF 数据存储与查询系统

顾 荣 仇红剑 杨文家 胡 伟 袁春风 黄宜华

(南京大学计算机软件新技术国家重点实验室 南京 210093)

(江苏省软件新技术与产业化协同创新中心 南京 210093)

摘 要 随着互联网应用的迅猛发展和语义网技术研究的深入,语义数据呈现出爆炸性增长趋势.一方面,对于语义数据实现高效存储和查询是语义网应用的重要基础,越来越多的语义应用可以依赖于此以提供更好的服务;另一方面,语义数据的爆炸性增长,对大数据环境下的语义数据的存储与查询技术提出了新的挑战.传统的基于关系型数据库的语义数据与查询系统已难以满足大规模语义数据的存储与查询需求.该文针对大规模 RDF 数据的存储与查询问题,以 OpenRDF Sesame 框架为基础,采用分布式分层式存储架构,提出并实现了属性表存储结构来进行语义数据的存储.在此基础上,针对布尔矩阵分解算法在对大规模语义数据构造属性表较慢的问题,基于 Spark 分布式计算框架提出并实现了并行化频繁项集挖掘算法求解大规模矩阵分解,以加速属性表的构造过程.并且,在查询层增加了基于哈希转换等查询优化.最后,基于该文所提出的索引结构和优化方法设计实现了原型系统 Goldfish,并在大规模合成和真实数据集上进行了实验对比.结果表明,Goldfish 原型系统比 Rainbow 系统查询性能平均提升约 6 倍,比 Jena-HBase 查询性能平均提升约 500 倍,比基于 MapReduce 的 RDF 查询系统 SHARD 性能平均提升约 1200 倍.

关键词 大规模 RDF 存储;矩阵分解;分层式存储;大数据;语义网;Spark

中图法分类号 TP311 **DOI 号** 10.11897/SP.J.1016.2017.02212

Goldfish: A Large Scale Semantic Data Store and Query System Based on Boolean Matrix Factorization

GU Rong QIU Hong-Jian YANG Wen-Jia HU Wei YUAN Chun-Feng HUANG Yi-Hua

(*State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing* 210093)

(*Collaborative Innovation Center of Novel Software Technology and Industrialization, Nanjing* 210093)

Abstract With the rapid development of the Internet applications and the semantic web technology, the amount of the semantic data is exploding. On one hand, it is significant to store and query semantic data efficiently, as many applications can provide better services based on this. On the other hand, the rapid increase of the semantic data brings new challenges on efficient storing and querying semantic data in big data era. The traditional ways for semantic data management is to store and query the data in relational database management systems. As the data increases, the traditional ways can hardly handle big data. To address this problem, this paper proposed a distributed hierarchical storage architecture to store and query large-scale semantic data based on the OpenRDF Sesame

收稿日期:2016-02-22;在线出版日期:2016-10-19.本课题得到国家自然科学基金专项基金(61223003)、国家自然科学基金(61370019)和江苏省科技支撑计划项目(BE2014131)资助.**顾 荣**,男,1988 年生,博士研究生,中国计算机学会(CCF)学生会员,主要研究方向为大数据并行处理和云计算技术等.E-mail:gurongwalker@gmail.com.**仇红剑**,男,1990 年生,硕士研究生,主要研究方向为大数据并行处理与云计算技术.**杨文家**,男,1991 年生,硕士研究生,主要研究方向为大数据并行处理与云计算技术.**胡 伟**,男,1982 年生,副教授,中国计算机学会(CCF)会员,主要研究方向为本体工程、数据集成.**袁春风**,女,1963 年生,教授,中国计算机学会(CCF)会员,主要研究领域为体系结构、大数据处理、Web 信息检索与挖掘等.**黄宜华**(通信作者),男,1962 年生,教授,中国计算机学会(CCF)会员,主要研究领域为大数据并行处理与云计算技术、体系结构与并行计算.E-mail:yhuang@nju.edu.cn.

framework. The RDF storage mechanism is optimized by adopting the attribute table to replace the RDF triple store. Considering the big semantic data, a parallel frequent item set mining algorithm with Spark framework is proposed to generate the index of the attribute table. Moreover, a layer of optimized hash conversion is proposed to avoid wasting time in frequent hash table search during query stage. To evaluate the efficiency of the proposed approach in this paper, we implement a prototype system called Goldfish, and conduct a comparison use large-scale synthetic dataset and real dataset. Experiment results show that Goldfish is around 8 times faster than Rainbow, 500 times faster than Jena-HBase and 1200 times faster than the MapReduce based RDF querying system SHARD.

Keywords　large scale RDF store; matrix factorization; hierarchical storage; big data; semantic web; Spark

1　引　言

据 2008 年 7 月 Google 给出的数据显示，Web 上存在 1 TB 不同的 URI. IDC 统计显示全球数据总量 2011 年已有 1.8 ZB，并预测到 2020 年全球数据总量将超过 35 ZB，总体增长近 20 倍[1]. 这些 Web 网页承载着大规模的数据，网页之间也存在着各种超链接. Berners-Lee 等人于 1998 年提出了语义网[2]的概念，其目的是为了使计算机可以从语义的角度迅速准确理解和处理大规模网页. 通过对 Web 增加语义支持，使机器能够根据语义进行判断，实现高效的数据共享和人机智能协同. 语义数据通常采用 RDF①(Resource Description Framework)来表达，在形式上是一种高度稀疏的图数据.

语义数据的爆炸性增长对语义数据的存储与查询技术进一步提出了新的挑战. 截至 2012 年 3 月，LOD(Linked Open Data)项目②所收集的 RDF 数据集已经超过 325 亿条 RDF 三元组. 传统的基于关系型数据库的语义数据在处理语义数据的数据量和存储扩展性上存在一定的缺陷，难以满足迅猛增长的数据带来的需求. 与此同时，大数据技术在全球的学术界和工业界领域迅猛发展，为实现大规模语义数据高效存储与查询提供了技术支持. 目前出现了很多典型的分布式数据库和大数据处理平台，例如广为使用的分布式的、面向列的非关系型数据库 HBase. 此外，Redis 是一种基于内存的、可进行数据持久化的高性能 key-value 存储数据库. 本文设计实现了一套大规模 RDF 数据存储和查询系统，在存储层采用层次化结构，以 HBase 作为持久存储层存储全局 RDF 数据，分布式 Redis 存储经常被查询的热数据.

本文的主要贡献可以归纳为以下三个方面：

(1) 针对 RDF 数据三元组表存储所表现出的查询效率和存储空间利用率低、可扩展性不佳等问题，本文设计了大规模属性表存储语义数据，强关联的属性会存储到一张表中，从而减少查询过程中的 join 操作. 进一步地，针对大规模数据集下构建属性表采用的 ASSO 算法存在求解速度较慢等问题，本文设计了基于 Spark 分布式框架的并行化频繁项集挖掘算法对其进行求解，以加速在大规模数据集下属性表的构造.

(2) 提出以 HBase 作为持久化存储层，Redis 作为分布式内存层的层次化语义数据存储架构，并进一步采用了基于哈希转换的查询优化. 本文采用分层式存储架构对语义数据进行分层存储，查询效率进一步提升. 并针对分层式 RDF 数据存储与查询系统在查询过程中出现的频繁查找哈希表的现象，提出增加哈希转换层. 通过增加哈希转化层，则仅需 SPARQL 查询的始末进行哈希转换，避免了频繁查找哈希表的开销.

(3) 基于以上优化方案的基础上，进一步设计并实现了大规模 RDF 数据存储与查询原型系统 Goldfish，并进行了相关的实验对比和可扩展性分析.

与现有系统相比，Goldfish 的优势体现在以下两点：

(1) 存储空间的高利用率和弹性高效的存储扩展性. 基于属性表存储结构优化，可以极大地提高存储空间的利用率，同时系统的存储容量可以随着集群节点的增加而呈现出近线性扩展.

① http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/
② http://linkeddata.org/

(2)高效的查询速度.采用基于属性表的分层式存储架构对语义数据进行存储、哈希转换的查询优化等措施提升查询的性能.

本文第 1 节简介本文相关的研究背景和现状,并阐明本文的主要内容;第 2 节介绍相关的背景知识;第 3 节简要阐述系统的整体框架;第 4 节重点阐述基于属性表的大规模 RDF 数据存储优化;第 5 节主要介绍基于哈希转换的查询优化;第 6 节介绍原型系统;第 7 节对本文提出的 Goldfish 系统性能进行实验分析;第 8 节讨论了相关工作;最后,在第 9 节我们对本篇论文进行总结,并指出下一步需要展开的工作.

2 背景知识

2.1 资源描述框架 RDF 和查询语言 SPARQL

RDF 是 W3C 提出的一种表达 WWW 资源信息的标准.一个 RDF 陈述(Statement)包括资源(主体)、属性(谓词)和属性值(客体).主体通常是指具体资源的 URI 引用(URI Reference),谓词指与资源描述有关的特征或关系,客体是指属性的值.这种三元组表达了信息资源间的对应关系.具体地,三元组的主体和客体间存在谓词表达的关系.Web 上的元数据进行可以通过 RDF 的标准进行语义编码、传递和重用.如图 1 所示,一个 RDF 模型结构由一组 RDF 三元组组成.其为一个有向标记图,其中主体或客体用节点表示,节点的内容可以为 URI 引用、字面量等,有向边从主体指向客体,为谓词的 URI 引用标记.RDF 数据是本文第 3 节提出的系统底层存储的数据,其中第 4 节中 RDF 数据作为原数据来构建属性表,另外 RDF 数据也是 7.3 节中对比系统底层存储的数据.

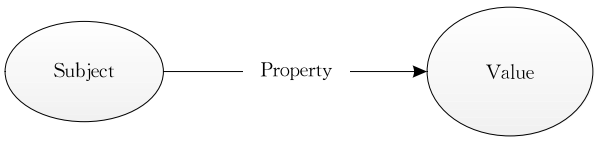

图 1 RDF 模型结构

在 2004 年,RDF 数据访问工组提出了针对 RDF 数据进行查询的语言 SPARQL.SPARQL 查询语句通常包括查询子句和 WHERE 子句.SPARQL 查询子句与 SQL 中的查询子句 SELECT 子句类似,用于指定出现在查询结果集的变量;而 WHERE 子句则通过基本图模式来同 RDF 数据匹配.SPARQL 语法支持四种类型的语义查询,分别为 SELECT、CONSTRUCT、DESCRIBE 和 ASK.经过几年的发展,SPARQL 已经成为 W3C 的推荐标准.SPARQL 语言与关系数据库中的 SQL 语言类似,这方便了用户对于 SPARQL 语言的使用.本文第 3 节提出的系统在查询层采用的 OpenRDF Sesame 框架中使用 SPARQL 作为 RDF 查询语言,7.3 节中的对比系统也使用 SPARQL 作为 RDF 查询语言.

2.2 分布式系统 Spark 介绍

Apache Spark 是由 UC Berkeley AMP 实验室于 2009 年发布的分布式内存计算框架,它是 Berkeley Data Analysis Stack(BDAS)中的核心项目.RDD[3](Resilient Distributed Dataset)是 Spark[4] 中的核心数据结构.它本质上是一种可以基于内存的弹性分布式数据集,允许用户将数据加载至内存后重复地使用,这样的设计适合计算密集型的机器学习算法和交互式查询算法.基于内存计算特点,与 Hadoop MapReduce 相比,Spark 在迭代计算和交互式应用上的性能快 10～100 倍.

Spark 和 Hadoop 类似,采用主从式结构,由 Master 和 Worker 两部分组成.Driver 程序是执行应用逻辑的入口点,Driver 启动后会连接 Spark 的 Master 节点,将对 RDD 的转换(transformation)和操作发送到各个 Worker 节点.Worker 是运行在工作节点上的守护进程 Executor,存储着数据分块,并享有集群内存,当 RDD 操作指令发送到各个节点上的 Executor 时,这些 Executor 对对应的数据分片进行本地化操作生成新的数据分片,并将生成的 RDD 返回或写入存储系统,具体见图 2 所示.数据的划分传输以及执行过程中的容错都不需要应用程序员关心.本文 4.2 节使用分布式内存计算框架 Spark 来并行化求解大规模属性表的构造问题.

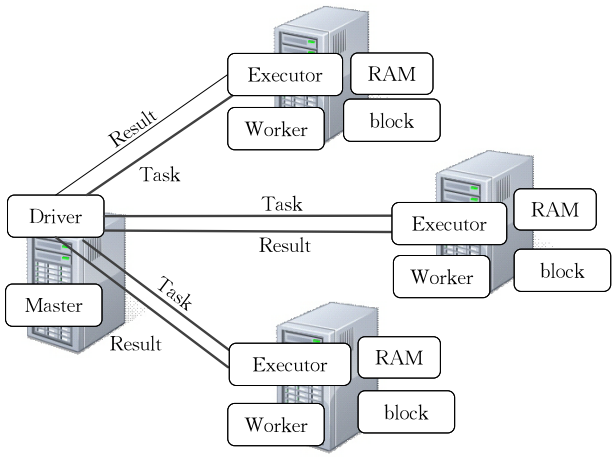

图 2 Spark 作业执行过程

2.3　属性表构建方法

有研究者[5]根据 Miettinen 等人提出的布尔矩阵分解 ASSO 算法[6]，在单机关系型数据库中以串行化的方式构造属性表. 其构造属性表的流程是首先将 RDF 语义数据表示成布尔矩阵的形式，然后通过布尔矩阵分解算法 ASSO 进行矩阵分解，并且引入最短描述距离 MDL[7](Minimum Description Length)来评估矩阵分解的结果，最后通过分解后的子矩阵得出语义数据之间的相似度来构造属性表.

通过布尔矩阵分解 ASSO 算法分析数据矩阵列与列之间的相关性是一种精度上有效的手段，其实质是借鉴关联挖掘的思想，将强关联的属性放到同一张表中，弱相关的属性放到不同的表中，因此也可以通过频繁项集挖掘算法来得到 RDF 数据属性间的分布来构建属性表.

用 $I=\{i_1,i_2,\cdots,i_m\}$ 表示 RDF 数据中所有属性的集合，$T=(tid,X)$ 表示一个事务，其中 tid 表示 RDF 的主体，X 表示该主体在 RDF 数据集中出现的属性集合，X 是 I 的子集，用 D 表示包含很多事务的一个集合. 属性子集 Y 的支持度计数为 D 中包含 Y 的事务个数，定义为式(1)

$$\sup(Y)=|\{tid|Y\subseteq X,(tid,X)\subseteq D\}| \quad (1)$$

相应地，属性子集 Y 的支持度则为 $\sup(Y)/N$，其中 N 为 D 中的事务个数.

若属性子集 Y 中属性间是强关联的则其支持度必须大于事先设定的最小支持度阈值. 存在这样一个事实：如果一个属性集合是强关联的，则其所有非空子集也是强关联的. 基于该事实，首先在 D 中找出所有支持度大于设定的最小支持度阈值的频繁 1-项集，之后基于频繁 $(k-1)$-项集来生成频繁 k-项集，其中 k-项集表示为含有 k 个属性的集合.

上述频繁项集挖掘算法中，所有 RDF 数据按照主体划分，每行表示一个事务 T. 若属性 A 和属性 B 是强关联的，则对应每一个事务 T，若属性 A 出现，则属性 B 也极有可能出现. 这里频繁项集挖掘算法的目标是找出属性间的关联，将强关联的属性放到同一张属性表中，属性 A,B 之间的关联度则通过 σ 来衡量，其中 σ 定义为式(2)

$$\sigma=|i|\text{若第 } i \text{ 行包含 } A \text{ 和 } B,i\in[1,N]|/N\}| \quad (2)$$

其中，N 表示总共的 RDF 数据行数，σ 介于 0 和 1 之间. 若 σ 大于实现设定的最小支持度阈值，则可以认为属性 A,B 是强关联的. 由此可见，频繁项集挖掘算法可以用来构建属性表.

3　系统框架

本文提出的分布式分层 RDF 数据查询系统 Goldfish 的系统总体结构如图 3 所示. 系统主要分为查询层与存储层两个部分. 查询层基于 OpenRDF Sesame 框架，主要包括查询解析器(Query Parser)、

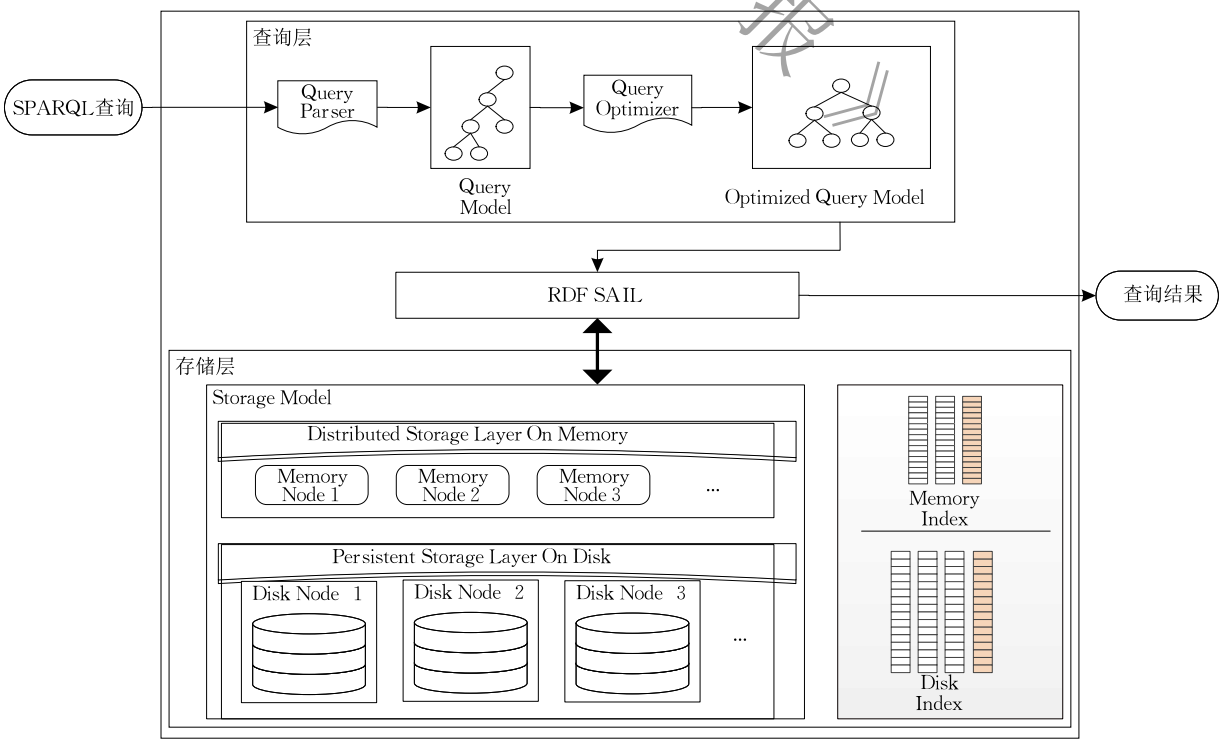

图 3　Goldfish 系统的总体架构(查询层基于 Sesame 框架，存储层基于分层式存储架构)

查询优化器(Query Optimizer)和查询执行器(Query Evaluator),其主要功能是对查询语句进行解析和优化,并将其转化为对应的查询模型(Query Model).存储架构以 HBase 为持久化存储层,Redis 为分布式内存层构成分布式层次化存储.

持久存储层使用 HBase 存储大规模语义数据,提供持久存储服务. HBase 能够保证数据存储的可靠性,当存储的语义数据规模增大或者动态增加节点时,HBase 能自动完成负载均衡操作. 分布式内存层使用 Redis 存储常用的 SP* 和 *PO 查询模式,保证查询的高效性. 这样通过持久化存储层来存储属性表做容错,分布式内存层存放热数据,使得更多的查询发生在分布式内存层,而不是持久化存储层.同时由于采用属性表存储结构,将强关联的属性放在一张表中,使得一般不会进行连接操作.

如图 3 所示,一个 SPARQL 查询首先会传入查询解析器进行查询语句的解析. 经过查询解析器之后,查询语句被构造成查询解析树. 接下来,查询优化器会对得到的查询解析树进行优化,调整查询语句之间的连接顺序,构造查询优化树. 然后,查询执行器会对查询优化树进行对应的查询. 查询执行器会在底层的数据存储层进行查找,并将查询的对应结果进行输出.

4 基于属性表的 RDF 数据存储优化

4.1 基于布尔矩阵分解的 RDF 数据存储布局

属性表的目标是减少查询过程中涉及的自连接操作、提高存储空间的利用效率以及提升系统的可扩展性. 基于 RDF 数据稀疏图的特性,本文采用了一种基于布尔矩阵分解的 RDF 数据存储布局.

通过下面的简单例子来阐述如何通过布尔矩阵分解 ASSO 算法在 RDF 数据中构造属性表的过程. 下面是一系列的 RDF 数据,每一行代表一条 RDF 数据,以主语-属性-宾语的顺序排列.

```
author1 rdf:hasName Alice
author1 rdf:hasPub pub1
pub1 rdf:hasType Journal
pub1 rdf:year 2004
author2 rdf:hasName Bob
author2 rdf:hasPub pub2
pub2 rdf:hasType Article
pub2 rdf:hasTitle Why SPARQL?
pub2 rdf:year 2004
author3 rdf:hasPub pub3
pub3 rdf:hasType Journal
pub3 rdf:year 2008
author3 rdf:hasName Cindy
```

(1) 根据 RDF 数据构建成布尔矩阵

将上面例子中的 RDF 数据按照矩阵表的方式构建布尔矩阵,即以 RDF 数据中全部的主体作为行,以 RDF 数据中全部的属性作为列,行列的交叉单元表示对应三元组的 RDF 数据是否存在,1 表示数据集中存在以该主语和属性组合的三元组,反之,0 表示不存在. 比如,author1 存在 hasName 的属性,故 author1 和 hasName 行列交叉处的值在布尔矩阵中为 1. 上述 RDF 数据集对应如下的数据矩阵(此处将该矩阵记为 $\boldsymbol{C}$).

$$\begin{array}{c} \\ author1 \\ author2 \\ author3 \\ pub1 \\ pub2 \\ pub3 \end{array} \begin{array}{c} \begin{array}{ccccc} hasName & hasPub & hasType & hasYear & hasTitle \end{array} \\ \begin{bmatrix} 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 \end{bmatrix} \end{array}$$

可以看到上面的布尔矩阵包含着很多的列,同时矩阵非常稀疏,这也就是 RDF 数据稀疏性与多样性的体现. 为了能够有效地对 RDF 数据进行存储与查询,必须将矩阵 $\boldsymbol{C}$ 进行规则化,即:减少列名集合的大小和减少矩阵的稀疏性;否则,会浪费大量的存储空间,同时降低查询处理的效率.

(2) 布尔矩阵分解

对由上面例子中构造的布尔矩阵 $\boldsymbol{C}$ 利用 ASSO 算法进行布尔矩阵分解. 利用布尔矩阵分解后,原 RDF 数据布尔矩阵将分解成以下的两个子矩阵 $\boldsymbol{S}$ 和 $\boldsymbol{B}$,见式(3)

$$\boldsymbol{C} = \boldsymbol{S} \times \boldsymbol{B} \tag{3}$$

因此,示例中的布尔矩阵可以按照频繁项集挖掘的方式求得属性列之间的关系,将强关联的属性列放在一张表中,弱相关的属性放在不同的表中. 其中,矩阵 $\boldsymbol{C}$ 经过分解之后,会得到两个子矩阵 $\boldsymbol{S}$ 和子矩阵 $\boldsymbol{B}$.

$$\boldsymbol{S} = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \\ 0 & 1 & 0 \end{bmatrix}, \boldsymbol{B} = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}.$$

通过布尔矩阵分解 ASSO 算法,可以得到如上面矩阵 $\boldsymbol{S}$ 和矩阵 $\boldsymbol{B}$ 的两个子矩阵. 根据矩阵 $\boldsymbol{B}$ 构造属性表,矩阵 $\boldsymbol{B}$ 每一行代表一个属性表,每行为 1 的列集合即是一个属性表中要存储的属性列名集

合，可以放在同一张属性表中，如下所示. 矩阵的行数代表了属性表的个数，构造得到的属性表可以在非关系型数据库中作为一个独立的表存储.

$$\begin{array}{c} \\ Table1 \\ Table2 \\ Table3 \end{array}\begin{array}{c} \begin{array}{ccccc} hasName & hasPub & hasType & hasYear & hasTitle \end{array} \\ \begin{bmatrix} 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \end{array}$$

(3) 基于属性表存储结构存储 RDF 数据

通过对 RDF 数据以属性表的方式进行存储，将属性强关联的 RDF 数据放在同一张表中，属性弱相关的属性放在不同的表中，既可以兼顾查询效率，又减少了 RDF 数据冗余的存储. 因此，通过矩阵分解 ASSO 算法后得到的属性表如表 1 所示.

表 1　ASSO 算法构造的属性表

(a) 子表 1

Subject	hasName	hasPub
author1	Alice	pub1
author2	Bob	pub2
author3	Cindy	pub3

(b) 子表 2

Subject	hasType	hasYear
pub1	Journal	2004
pub2	Article	2004
pub3	Journal	2008

(c) 剩余表

Subject	hasTitle
pub2	Why SPARQL?

最后对 RDF 数据，按照上面步骤构造的属性表，将数据插入到对应属性的属性表中.

4.2　基于 YAFIM 算法的属性间关联规则生成

由 2.3 节和 4.1 节中的例子可知，布尔矩阵分解构造属性表的实质是将强关联的属性放到一张表中，弱相关的属性放到不同的表中，该算法在单机上面对大规模 RDF 数据构造属性表的串行处理方式比较低效，可以使用易于并行化的频繁项集挖掘算法对其进行大规模求解. 本节采用基于 Spark 分布式计算框架的并行化频繁项集挖掘算法 YAFIM[8]，以加速构造大规模 RDF 数据属性表.

在 YAFIM 中关联度阈值 σ 设定为与具体实验数据集相关的经验值. 实验中，若 σ 值设定过高则会存在很多属性是弱相关的，造成属性表的数量增多；若 σ 值设定过低，就会造成每个属性间都是强关联的. 本文中我们采用的是调试得出的 0.6～0.8 的经验值，频繁项集挖掘算法的并行化求解速度很快. 另外，由于我们引入了剩余表来存储与其他 RDF 数据的属性关联性较弱的语义数据，因此无论 σ 值设定多少，总可以保证整个系统的正确性.

本节依然沿用 4.1 节中用到的例子，利用 Spark 框架并行化来处理，包括 RDF 数据格式转换、利用 YAFIM 构造属性表结构并在此基础上存储 RDF 数据.

4.2.1　RDF 数据格式转换

利用分布式内存计算框架 Spark 来并行化处理 RDF 数据格式转换，整个过程如图 4 所示，其中每个圆角矩形表示 RDD 的一个分区. 首先 Spark 从分布式文件系统 HDFS 将 RDF 数据加载至 RDD，利用 RDD 算子，将所有的 RDF 数据按照主体重新划分：RDD 中的每条记录中第一列为不同的主体，第二列为对应主体在 RDF 数据集中出现的属性，把同一主体的属性放在同一条 RDD 记录中. 上面的 RDF 数据经过处理之后 RDD 中每条记录的格式就变为如表 2 所示的形式.

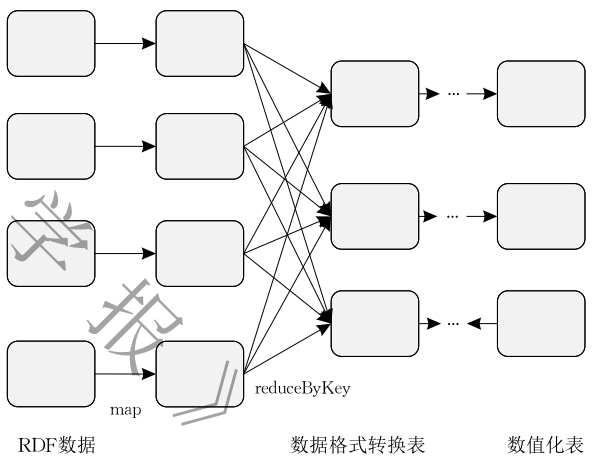

图 4　RDF 数据格式转换过程

表 2　RDF 数据格式转换表

Subject	*Property*
*author*1	*rdf*:*hasName*,*rdf*:*hasPub*
*author*2	*rdf*:*hasName*,*rdf*:*hasPub*
*author*3	*rdf*:*hasName*,*rdf*:*hasPub*
*pub*1	*rdf*:*hasType*,*rdf*:*year*
*pub*2	*rdf*:*hasType*,*rdf*:*hasTitle*,*rdf*:*year*
*pub*3	*rdf*:*hasType*,*rdf*:*year*

接下来，对上面格式的语义数据进行数值化处理. 假定用数值 1～5 分别代表 *hasName*，*hasPub*，*hasType*，*hasYear* 和 *hasTitle* 这五个属性，上面格式的 RDF 数据就会以数值形式表示为以下内容. 其中行号 1～6 分别表示不同的主体 *author*1 到 *pub*3. 如表 3 所示，数值化表即事务数据集，每行表示一个事务，每个事务第一列为主体的编码，第二列为该主

体在 RDF 数据集中出现的属性编码.

表 3 数值化表

Subject	*Property*	*Subject*	*Property*
1	1,2	4	3,4
2	1,2	5	3,4,5
3	1,2	6	3,4

4.2.2 利用频繁项集挖掘算法构造属性表结构

基于 Spark 框架,本文实现了并行频繁项集挖掘算法,并且利用该算法处理大规模 RDF 数据,得到 RDF 数据在属性表中的分布.

图 5 为 YAFIM 并行化流程示例,首先 Spark 从分布式文件系统 HDFS 读入上一步生成的 RDF 数据数值化表,利用 RDD 算子①操作生成频繁 1-项集,然后主节点将频繁 1-项集取回本地,生成候选 2-项集并将其广播到每个从节点,接下来经过一系列 RDD 算子操作生成频繁 2-项集,如此反复利用上一步生成的频繁项集便可求得最大频繁项集. YAFIM 算法主要包括两个阶段:生成频繁 1-项集,利用频繁 k-项集迭代生成频繁$(k+1)$-项集.

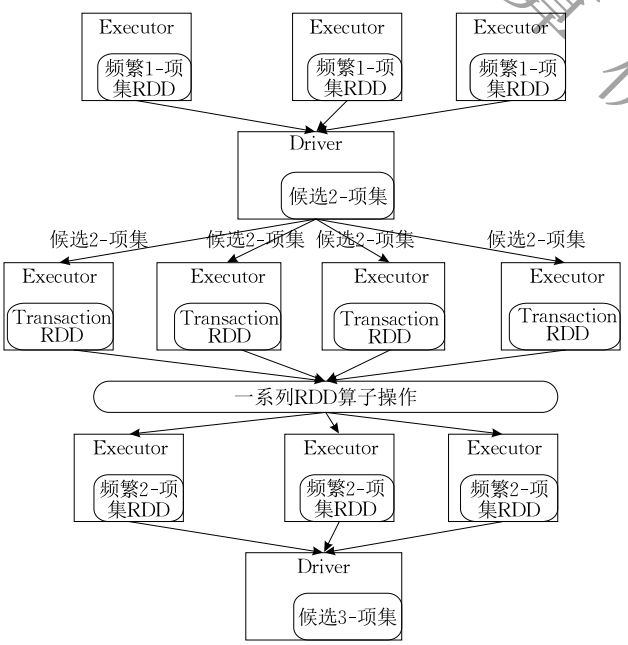

图 5 并行化频繁项集挖掘算法构造属性表流程示例

(1)第一阶段:从分布式文件系统 HDFS 上获取 RDF 数据的数值化表,加载到分布式内存 RDD 中.

如图 6 所示,原始事务数据集由操作 flatMap 操作(由 workers 执行)去读取事务数据集(RDF 数据的数值化表),并且将所有的事务数据转化为 Spark RDD. 接下来每一个事务(*transactions*)中执行 *flatMap* 操作,从中得到所有的 *items* 项集. 执行完 *flapMap* 操作后,执行 *map* 操作,将所有项集 *items* 转化为〈*Item*,1〉的 *key*/*value* 形式. 最后,*reduceByKey* 函数统计每个项集 *items* 的支持度,并对支持度小于最小支持度 σ 的项集进行剪枝. 所有超过最小支持度 σ 的项集将会生成频繁 1-项集.

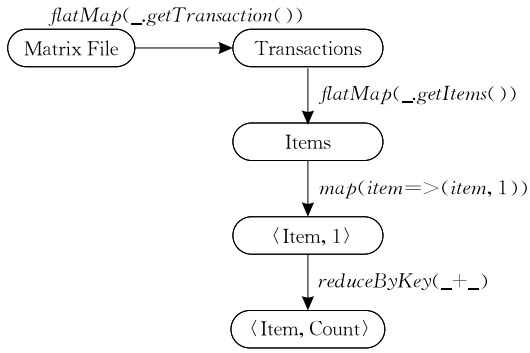

图 6 YAFIM 算法第一阶段的 Lineage 图

(2)第二阶段:不断迭代以使用 k-频繁项集生成$(k+1)$-频繁项集.

如图 7 所示,首先,读取 k-频繁项集 L_k 并且以〈*itemset*,*count*〉的形式将其存储为 Spark RDD. 然后从 k-频繁项集中得到候选$(k+1)$-项集 C_{k+1}. 为了加速从候选项集中查找$(k+1)$-频繁项集的过程,将候选$(k+1)$-项集 C_{k+1} 存放在哈希树中. 接下来,通过 *flatMap* 操作从原始事务库 RDD 中去获得每个候选项集的支持度. 进一步,对每个频繁项集使用 *map* 函数来输出〈*itemset*,1〉的 *key*/*value* 对的形式. 最后,通过 reduceByKey 操作来收集所有的候选项集的支持度,并且对支持度大于最小阈值 σ 的频繁项集以〈*itemset*,*count*〉的 *key*/*value* 对的形式进行输出作为$(k+1)$-频繁项集.

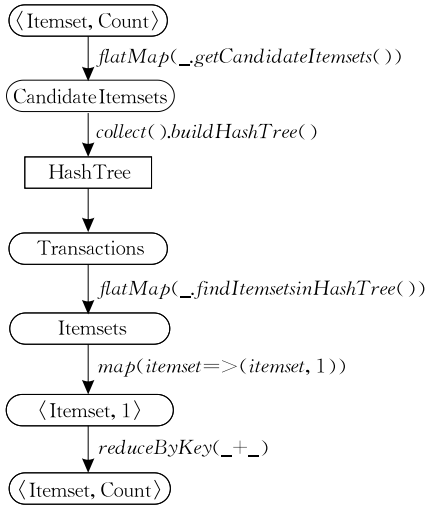

图 7 YAFIM 算法第二阶段的 Lineage 图

① http://spark.apache.org/docs/latest/programming-guide.html#rdd-operations

在上面的例子中，通过 YAFIM 算法，可以发现属性 hasName 和 hasPub 具有相关性，其关联度大于给定的最小支持度阈值，并且属性 hasType 和 hasTitle 具有相关性. 因此，将属性 hasName 和 hasPub 放在一张属性表中；将属性 hasType 和 hasYear 放在一张属性表中；剩下的属性 hasTitle 与其他属性的关联度小于给定的最小阈值，将其单独放在剩余表(left-over table)中. 表 4 为示例的属性表结构.

表 4　示例属性表结构

(a) 子表 1

Subject	hasName	hasPub

(b) 子表 2

Subject	hasType	hasYear

(c) 剩余表

Subject	hasTitle

4.2.3　利用属性表存储结构进行 RDF 数据存储

通过上一步骤在 RDF 数据中生成的属性表分布，在存储层创建对应的属性表，并将 RDF 数据转换存储到对应的属性表中. 在上面的例子中，插入 RDF 数据后的属性表如表 5 所示.

通过以上过程，在分层式 RDF 数据查询系统中利用属性表存储结构存储 RDF 数据. 在利用属性表存储结构实现分层式 RDF 数据查询系统存储层的过程中，这里通过剩余表(left-over table)存储与其他 RDF 数据的属性关联性较弱的语义数据. 所以，当新的 RDF 数据插入到分层式 RDF 数据存储查询系统时，如果新插入的 RDF 数据具有新的属性值，

表 5　YAFIM 构造的属性表

(a) 子表 1

Subject	hasName	hasPub
author1	Alice	pub1
author2	Bob	pub2
author3	Cindy	pub3

(b) 子表 2

Subject	hasType	hasYear
pub1	Journal	2004
pub2	Article	2004
pub3	Journal	2008

(c) 剩余表

Subject	hasTitle
pub2	Why SPARQL?

会直接存放到剩余表中；否则，便插入到对应的属性表中. 并且，随着新 RDF 数据的插入，会定期更新属性表.

5　基于哈希转换的查询优化

SPARQL 查询引擎是 RDF 数据查询系统的重要组成部分，如图 8 所示，SPARQL 查询引擎主要包括查询解析器、查询优化器和查询执行器三个部分. 当 SPARQL 查询引擎获取用户提交的查询请求后，首先查询解析器会对查询语句执行语法和语义解析操作，并生成查询模型，提交给查询优化器. 然后查询优化器根据输入的查询模型，其会在编译过程中生成许多查询结果等价的方案模型，并计算每个模型具体的查询代价，选取性能最优的查询模型提交给查询执行器. 最后查询执行器根据优化后的查询模型，生成查询策略并执行查询操作.

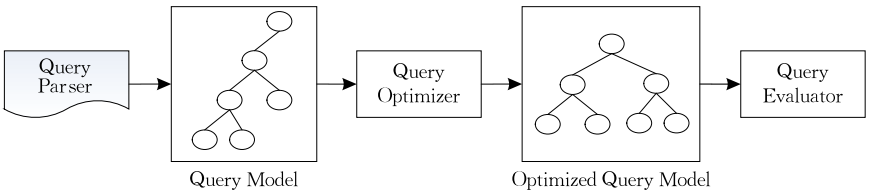

图 8　SPARQL 查询引擎主要模块

本节首先对查询过程中的时间消耗进行详细的测试与分析，以定位需要重点优化的模块. 图 9 展示了具有代表性的 LUBM-1 语义数据集的 14 条测试查询语句的执行过程中各个阶段的时间占比. 如图 9 所示，横轴代表不同的测试查询语句，纵轴代表每条查询语句在查询解析器、查询优化器和查询执行器上的时间消耗比例，其中“查询解析器”、“查询优化器”和“查询执行器”代表查询层的三个部分.

从图 9 可以看出在查询执行器的时间消耗比例平均是最高的，查询执行器则是根据优化后的查询模型，在数据库中执行查询策略. 进一步分析可知，查询执行器的时间开销可以分为两个部分：查询存

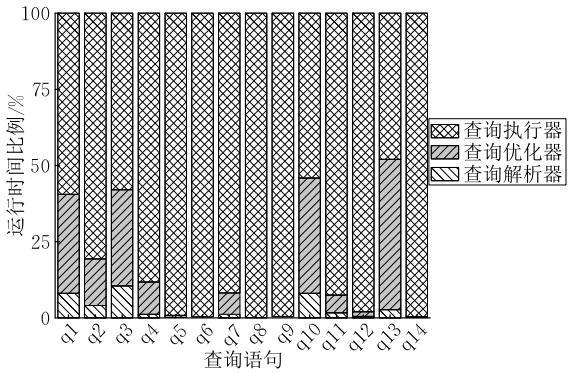

图 9 LUBM-1 规模的查询时间耗时过程分析

储层数据的时间和查找哈希表的时间. 如图 10 所示为 Query1 中查询执行器的时间分析,坐标横轴代表查询返回的 RDF 结果集的规模,坐标纵轴代表查询执行时间. "查找数据库"代表查询执行器在存储层数据库查找结果消耗的时间,"查找哈希表"代表将 RDF 数据和其对应的哈希值转换的时间开销. 从图 10 可以发现,随着 RDF 查询结果集规模的增长,

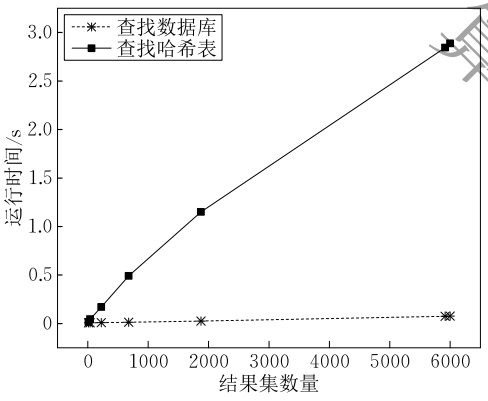

图 10 查询执行器查询过程分析

在底层数据库查找的时间增加平缓,但是进行哈希转换的时间近线性增长. 这表明了查询执行器的时间主要消耗在频繁查找哈希表的操作上.

由于 SPARQL 查询引擎只能识别 RDF 数据,而在系统存储层存储的是 RDF 数据的哈希值,存储层与查询层的异构性导致 Goldfish 系统在查询过程中需要频繁查询哈希表. 因此,查询返回的 RDF 数据结果集的数量越大,查询所消耗的时间越多,查询执行器中查找哈希表在 RDF 数据查找的总时间中所占的权重越大. 因此,下面的工作主要集中在如何减小查询执行器中的查找哈希表开销.

针对频繁查找哈希表所带来的开销,本节提出在查询执行层增加哈希转换部分的方案,将 SPARQL 查询在查询执行层直接转换成哈希值在存储层的数据库中进行查找. 基于此,优化的具体思路是:当查询语句经过查询解析器和查询优化器之后,在向底层存储的 RDF 数据查询结果之前,增加一层查询转换层,查询执行器直接以哈希值的形式在底层的 RDF 数据中进行查找,在最后返回查询结果时,只需对返回查询的结果查找哈希表即可,这样就避免了查询时频繁查找哈希表带来的开销. 通过在底层采用属性表存储结构,在查询层增加哈希转换优化后,可以良好地支持单表范围查询、包含 Union 操作的查询以及多个表的 join 查询等.

图 11 表示了优化后的查询层. 通过在查询引擎层增加哈希语义转换,使得查询语句在查询引擎层就转换为哈希值进行 RDF 数据的查找,减少了哈希语义转换的开销. 本文性能评估部分的第 7.2.3 节也验证了查询层哈希转换优化后效果.

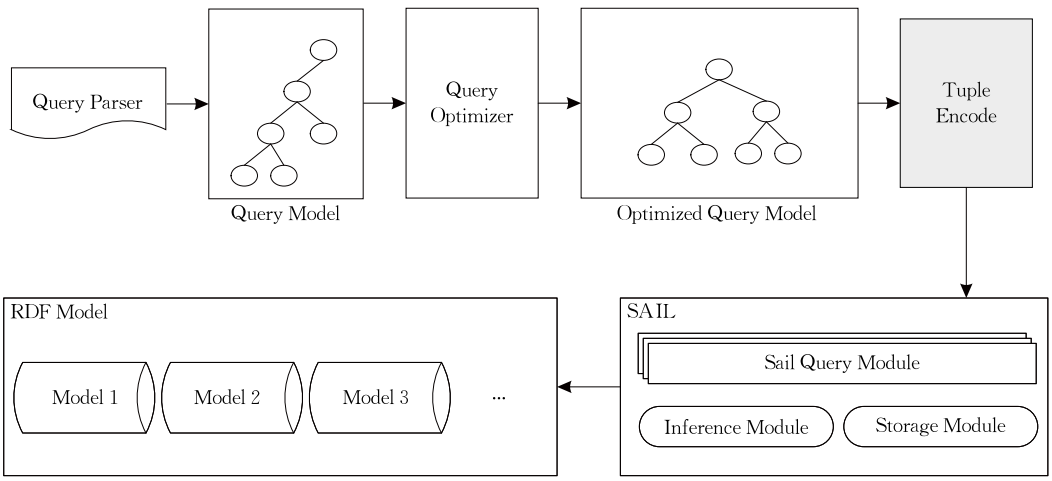

图 11 增加哈希转换的查询层

6　原型系统实现

本章我们简要介绍 Goldfish 原型系统的实现. 本系统查询层基于 OpenRDF Sesame 框架，RDF 管理模块和 RDF 输入输出模块沿用了 Sesame 框架提供的处理方式. RDF 查询模块、SAIL 层等模块则重新设计，并确保重新设计后的模块符合 Sesame 框架制定的接口规范. 系统的体系结构如图 12 所示. 图 12 中 SAIL 模块针对基于属性表存储结构的存储层需要重新设计，Repository 模块根据基于查询层哈希转换的优化重新设计. 其余的模块则依然沿用分层式 RDF 数据查询系统的模块.

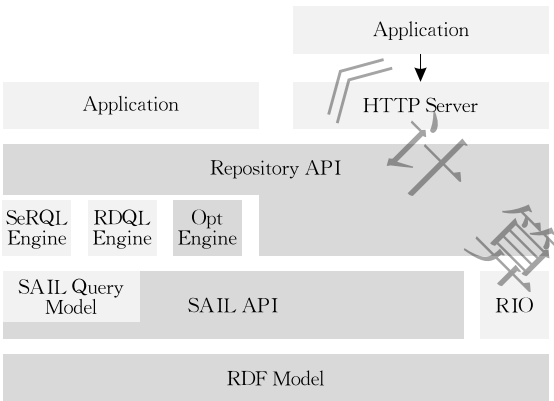

图 12　原型系统内部模块图

为了使 Goldfish 系统的用户体验更好，便于用户交互，原型系统采用前端＋后台处理的方式. 前端采用 Tomcat 服务器 Web 服务，后台采用优化后的分布式分层 RDF 数据查询系统，并采用基于 HBase＋Redis＋Sesame 的环境开发和部署，通过 Sesame 框架提供的统一接口.

图 13 给出了 Goldfish 系统的前端设计框架. 系统的前端采用 JSP 页面来展示查询界面和搜索结果，Web 服务部署在 Tomcat 之上. 如图 13 所示，JSP 页面通过查询接口调用后台优化后的分层式 RDF 数据存储查询系统，并获得查询结果，最后返回给用户页面.

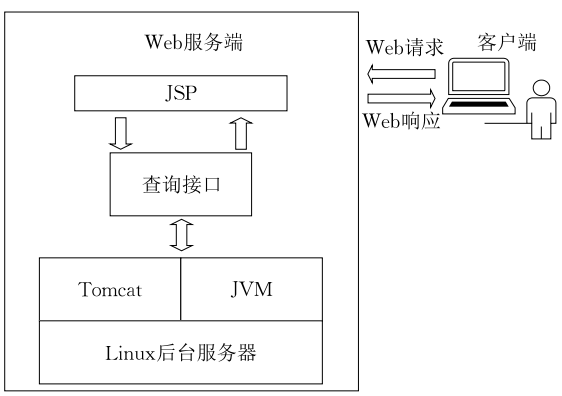

图 13　服务交互请求处理图

7　实验结果与分析

7.1　实验数据及平台

实验数据集选用了人工合成数据集和真实数据集. 其中人工合成数据集采用的是当前公认的面向大规模语义数据应用环境的知识库系统测试基准 LUBM(Lehigh University Benchmark)[9] 大学语义数据；真实语义数据集采用的是 WordNet 3.1[10] 和 DBLP Computer Science Bibliography① (DataBase systems and Logic Programming). LUBM 测试 Benchmark 提供了 14 条不同的查询语句②，分别代表了不同的查询情况(如单个表的查询、包含 Union 操作的查询、多个表的 join 查询等)，我们在 WordNet 和 DBLP 上生成了 10 条不同的查询语句，同样覆盖了所有常见的查询类型. 实验平台环境中每个节点都配置了 24 GB 的内存，8 路 2.4 GHz 的 Intel Quad Xen 处理器，这些节点通过一个 1 Gbps 带宽的网络连接. 软件方面安装好了 RedHat Enterprise Linux Server 6.0 的操作系统以及 Hadoop 1.0.3 和 Spark 1.4.0.

本文利用拟合工具在实验中生成了 4 组不同规模的 LUBM 测试数据集：LUBM-1，LUBM-10，LUBM-100 和 LUBM-1000，分别包括 130000、1300000、13300000 和 133000000 条三元组，以测试在不同规模的语义数据下 RDF 数据查询系统的性能. DBLP 是德国特里尔大学开发的一个计算机类英文文献集成数据库系统，其以作者为核心并按照年代列出作者的科研成果. DBLP 中数据按照语义来组织，本文采用的 DBLP 数据集共含有 42 million 条三元组. WordNet 是普林斯顿大学研究人员设计开发的基于语义组织的大型英文词汇数据库，本文采用的 WordNet 3.1 共含有 8.9 million 条三元组.

7.2　优化性能分析

7.2.1　属性表构建

本节在实验环境所述的 12 个节点的集群上分别对 4 组数据(LUBM-1、LUBM-10、LUBM-100 到 LUBM-1000)来构建基于属性表的 RDF 数据存储，

① http://dblp.uni-trier.de/
② http://swat.cse.lehigh.edu/projects/lubm/queries-sparql.txt

以评估该方法的速度、扩展性、总体存储空间消耗等性能指标.

属性表构造时间的实验结果见图 14,其中坐标横轴代表四个数据集,坐标纵轴为构建属性表消耗的总时间.这里的构建属性表消耗的总时间包括前期的预处理(格式转换、编码等)、利用并行化频繁项目集挖掘生成属性表结构以及原始表转成属性表存储三部分构成.

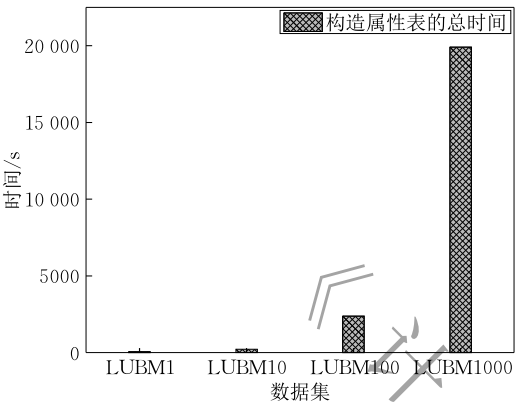

图 14 数据规模变化时构建属性表的总时间

图 14 中,可以看到在不同规模的数据下,构建属性表的总耗时增加的倍数与数据规模增加的倍数基本成正比,例如 LUBM-1000 相对于 LUBM-100 数据规模增加的倍数为 10,LUBM-1000 相对于 LUBM-100 构建属性表所耗总时间增加的倍数为 8.38.

存储空间消耗方面,我们对比评估了 RDF 语义数据的存储结构有四种主流形式:三元组存储结构、垂直划分存储结构、水平存储结构和本文采用的属性表存储结构.在表 6 中我们具体分析了这四种存储划分方案在不同规模数据集上的存储开销(整体数据都是持久化存储在底层基于磁盘的 HBase 上),可以看出采用本文的属性表存储结构的总体数据开销相当于原始数据的 3 倍左右.这是由于属性表的结构避免了很多稀疏化而导致的额外开销.因此,本文采用的属性表的索引结构的空间开销较低,可以很容易地将热点数据存储到分布式内存中,并将整体数据持久化存储在底层基于磁盘的 HBase 上.

表 6 四种存储划分方案的存储空间占用对比比例

数据集	原数据集	属性表	三元组	垂直划分	水平划分
LUBM-1	8	15.2	31	21	32
LUBM-10	102	374.85	696	416	612
LUBM-100	1,126.4	3,686.4	6993	4506	6634
LUBM-1000	11,264	35053.5	63612	45082	51878

另外,对于流式插入的数据,相同数据分布规律的数据集对属性表结构的影响不大,例如属性 1 和属性 2 在 LUBM-1 是频繁相关的,在其他的数据集中也是频繁相关的.因此,对于同一数据集只需构造一次属性表结构,当新的 RDF 语义数据流式插入到分层式 RDF 存储系统时,若新插入的数据具有新的属性值,会直接放到剩余表(left-over table)中,否则会插入到其对应的属性表当中,并且随着新 RDF 数据的插入,系统会定期更新属性表.

最后,为了评估基于 YAFIM 算法在计算节点增加时的性能扩展性,这里分别采用 LUBM-100 和 LUBM-1000 两个数据集以及真实数据集 WordNet 3.1 和 DBLP 在节点数为 2,4,6,8,10,12 上的生成关联规则的时间,每个节点上 Spark 进程使用的内存大小为 12GB(Executor Memory).图 15 中,坐标横轴为节点数目,坐标纵轴为使用 YAFIM 算法来生成关联规则的时间.

由图 15 可见,节点数为 2 时由于计算资源的有限,所以耗费时间较长,随着节点数目的增加,可用的计算资源不断增加,但同时节点间的网络开销带来的时间消耗也开始逐渐增加,时间变化曲线趋于平缓.总体上,关联规则的生成时间随着节点数目不断增多,在生成数据集和真实数据集上都呈现出近线性的扩展性.

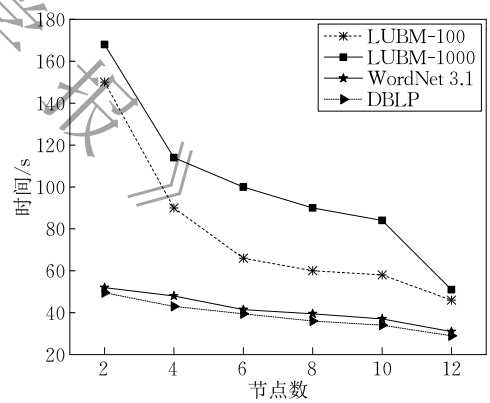

图 15 YAFIM 算法构建属性表的节点可扩展性

7.2.2 存储层优化

本节采用 6 组数据(LUBM-1、LUBM-10、LUBM-100 到 LUBM-1000、WordNet 3.1、DBLP)评估基于属性表存储结构的分层式 RDF 数据系统的查询性能,一共做了 3 次测试,最后取平均查询时间.此外,本节还使用循环三元组存储结构(SPO,OPS,PSO)进行对比实验.循环三元组存储结构是一种广为使用的以 SPO、POS、OSP 循环索引来应对给定不同 SPO 组合查询的表结构.

在 LUBM 所提供的 14 条测试查询语句、

WordNet 3.1 和 DBLP 生成的 10 条查询语句中，本节选取部分具有代表性的查询，实验结果如图 16～图 21 所示. 其中，坐标横轴代表了选取的标准测试查询语句，纵轴代表查询所消耗的时间，“循环三元组存储结构”表示循环三元组存储结构作为分层式 RDF 数据查询系统的存储层查询所需的时间，而“属性表存储结构”表示以属性表存储结构作为分层式 RDF 数据查询系统的存储层查询所需的时间.

由图 16～图 21 的实验结果可见，与循环三元组索引结构作为分层式 RDF 数据查询系统的存储层存储方式相比，利用属性表存储结构的存储方式在查询时间上更快. 并且，从实验中可以发现不同的查询语句，利用属性表存储结构提高 RDF 数据查询的效果也不一样，这和 RDF 数据在属性表中的分布有关. 如果查询语句中的几条子句要查询的属性都在一张属性表中，利用属性表存储结构作为存储层的 RDF 数据查询系统的查询速度就快得多；反之，如果子句中需要查询的属性都在不同的属性表中，这就需要表间连接，查询效果就会不理想.

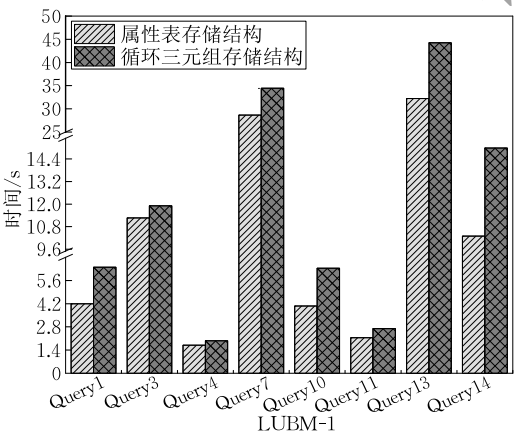

图 16　LUBM-1(0.13M 条三元组)查询性能对比

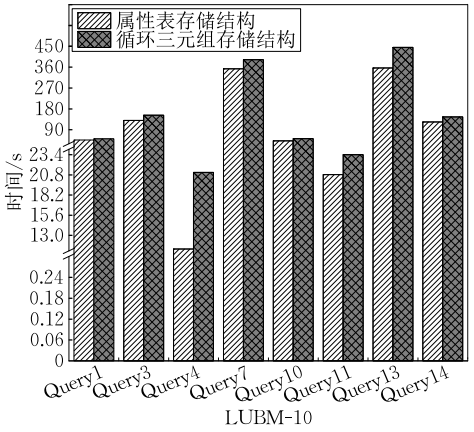

图 17　LUBM-10(1.3M 条三元组)查询性能对比

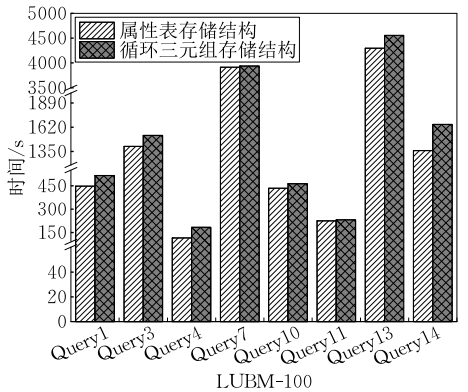

图 18　LUBM-100(13.3M 条三元组)查询性能对比

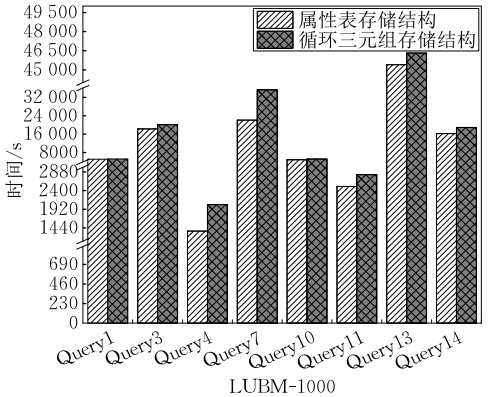

图 19　LUBM-1000(133M 条三元组)查询性能对比

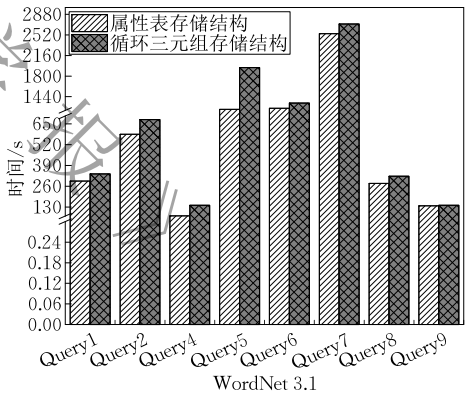

图 20　WordNet 3.1(8.9M 条三元组)查询性能对比

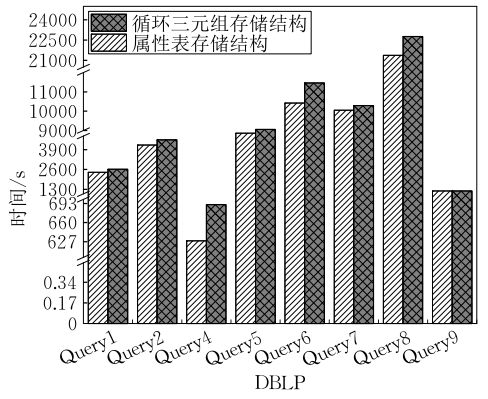

图 21　DBLP(42M 条三元组)查询性能对比

7.2.3 查询层优化

进一步地,为了验证本文所提出的查询层优化的效果,本节采用 6 组数据(LUBM-1、LUBM-10、LUBM-100 到 LUBM-1000、WordNet 3.1、DBLP)分别做了 3 趟测试,最后取平均运行时间,实验结果如图 22～图 27 所示.其中,横轴代表的是不同的标准测试查询语句,纵轴代表的查询所需要执行的时间."查询层哈希转换优化"代表的是在分层式 RDF 数据查询系统的查询层增加哈希转换优化后的系统,"查询层未优化"代表的是查询层未优化的分层式 RDF 数据查询系统.这里选取了部分具有代表性的查询语句,如图所示.

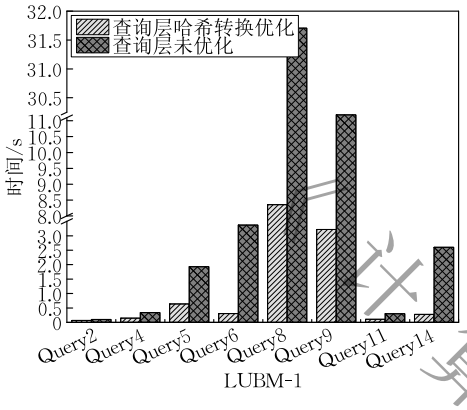

图 22 LUBM-1(0.13M 条三元组)的查询时间对比

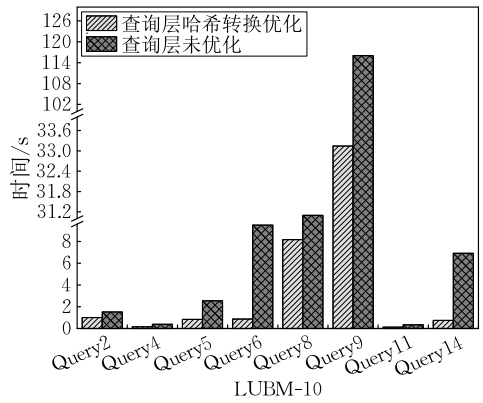

图 23 LUBM-10(1.3M 条三元组)查询时间对比

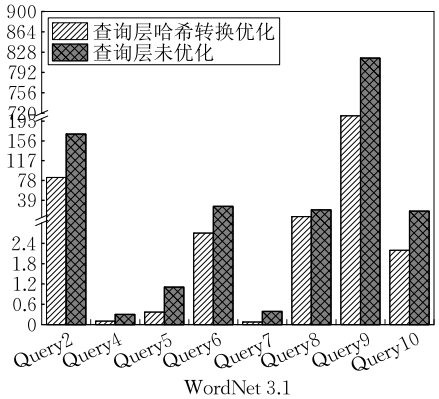

图 24 LUBM-100(13.3M 条三元组)查询时间对比

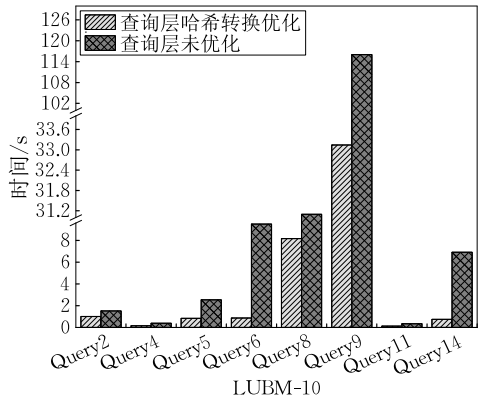

图 25 LUBM-1000(133M 条三元组)查询时间对比

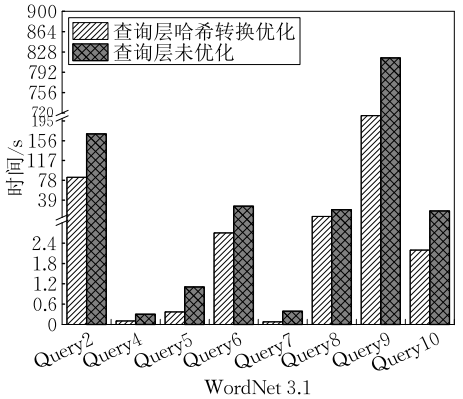

图 26 WordNet 3.1(8.9M 条三元组)查询时间对比

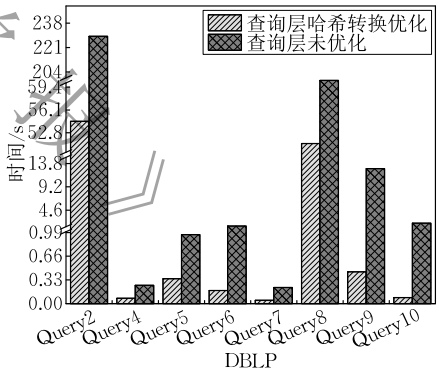

图 27 DBLP(42M 条三元组)查询时间对比

由实验结果可见,与查询层未增加查询转换优化的分层式 RDF 数据查询系统相比,通过增加哈希转换层优化后的 RDF 数据查询系统比原系统查询时间平均快 1～7 倍.从图中可以看到,在 LUBM-1 规模的 RDF 数据集下,优化效果最快的 Query6 的查询速度提高了 9 倍,返回结果数 7790 条;优化效果最慢的 Query2 的查询速度提高了 0.5 倍,返回查询结果数 0 条.在 LUBM-10 规模的 RDF 数据集下,优化效果最快的 Query6 的查询速度提高了 9 倍,

返回结果数 99 566 条；优化效果最慢的 Query2 的查询速度提高了 0.5 倍，返回查询结果数 0 条. 在 LUBM-100 规模的 RDF 数据集下，优化效果最快的 Query6 的查询速度提高了 9 倍，返回查询结果 1048532 条；优化效果最慢的 Query2 的查询速度提高了 1 倍，返回查询结果数 0 条. 在 LUBM-1000 规模的 RDF 数据集下，优化效果最快的 Query13 的查询速度提高了 19 倍，返回结果数 6078 条；优化效果最慢的 Query4 的查询速度提高了 1.3 倍，返回结果数 34 条.

WordNet 3.1 数据集中，优化效果最快的 Query6 的查询速度提高了 10 倍，返回结果数 7532 条；优化效果最慢的 Query2 的查询速度提高了 2 倍，返回结果数 8 条. DBLP 数据集中，优化效果最快的 Query9 的查询速度提高了 28 倍，返回结果数 1230 条；优化效果最慢的 Query4 的查询速度提高了 3 倍，返回结果数 30 条.

因此，在 RDF 数据查询任务中，查询的数据集规模越大、返回的查询结果数越多，本文提出的查询层的哈希转换优化的效果越明显.

7.2.4　系统扩展性

为了评估随着 RDF 数据规模的增长，对比以属性表存储结构的 RDF 数据查询系统和以循环三元组存储结构的 RDF 数据查询系统在查询时间上的变化情况，同时也对比查询层优化后的分层式 RDF 数据查询系统和未优化的原系统在查询时间的变化情况. 这里选用数据规模不同的四组 LUBM 数据集进行测试，每个规模级别下分别作 3 趟测试，最后取平均运行时间

(1) 索引结构扩展性

实验结果见图 28～图 31，图中纵坐标为查询时间，横坐标为 LUBM 数据集规模. “属性表存储结构”表示以属性表存储结构作为存储层的 RDF 数据查询系统，“循环三元组存储结构”表示以循环三元组存储结构作为存储层的 RDF 数据查询系统. 在这里，选取的具有典型代表的 Query3，Query4，Query7 和 Query14 进行分析. 从图中可以发现，随着 RDF 数据测试数据集规模的增长，用属性表存储结构的 RDF 数据查询系统随着 RDF 测试数据集的增大，查询速度均优于循环三元组存储结构.

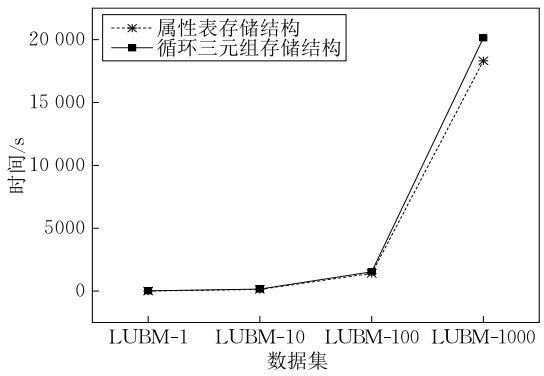

图 28　数据规模变化时 Query3 可扩展性对比

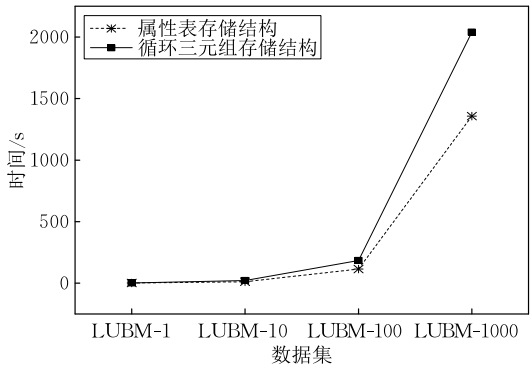

图 29　数据规模变化时 Query4 可扩展性对比

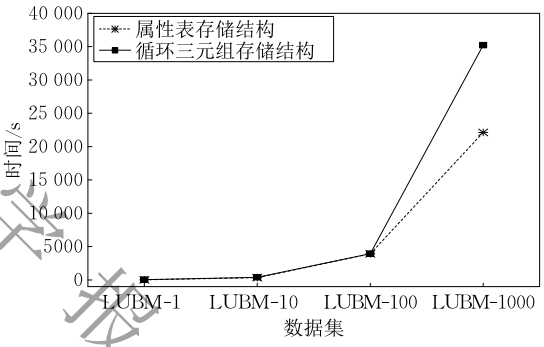

图 30　数据规模变化时 Query7 可扩展性对比

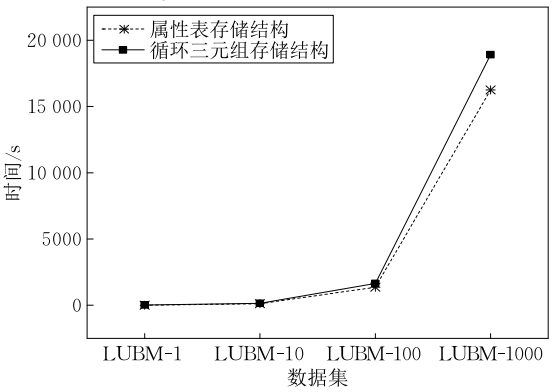

图 31　数据规模变化时 Query14 可扩展性对比

(2) 哈希优化扩展性

实验结果见图 32～图 35. 图中纵坐标为查询执行时间，横坐标为 RDF 数据的规模. “查询层哈希转换优化”代表的是在分层式 RDF 数据查询系统的查询层增加哈希转换优化后的系统，“查询层未优化”

代表的是查询层未优化的分层式 RDF 数据查询原系统.由图中可见,随着 RDF 数据规模的增长,查询层增加哈希转换优化后的系统比原系统查询时间快 1 倍～7 倍.

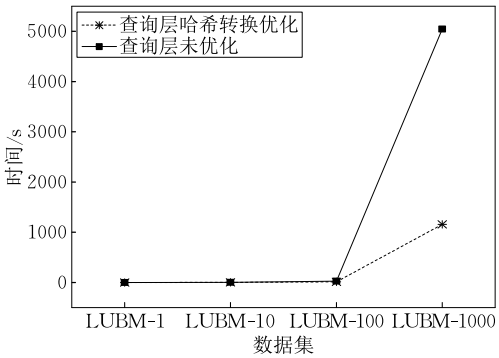

图 32 数据规模变化时 Query2 查询时间对比

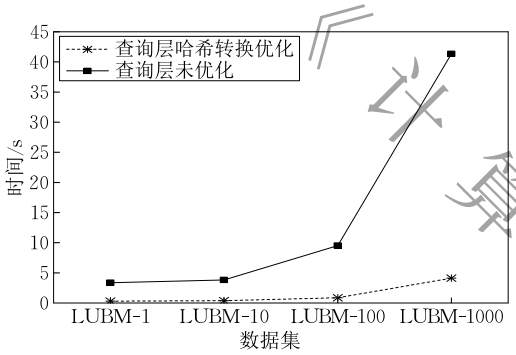

图 33 数据规模变化时 Query6 查询时间对比

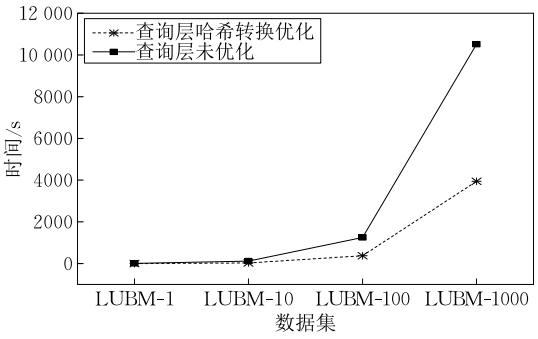

图 34 数据规模变化时 Query9 查询时间对比

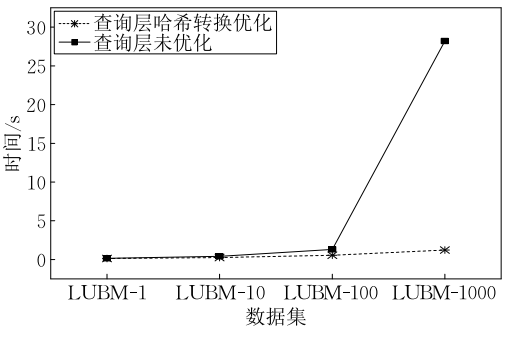

图 35 数据规模变化时 Query13 查询时间对比

在图中,随着 RDF 数据测试数据集规模的增长,对 Query2 查询语句,查询层优化后的系统的查询时间比原系统的查询时间快 1 倍左右;对 Query6 查询语句,查询时间快 9 倍左右;对 Query9 查询语句,查询时间快 1 倍左右;对 Query13 查询语句,在 LUBM-1 到 LUBM-100 规模的 RDF 数据集上,查询时间快 2 倍左右,在 LUBM-1000 时,性能提升 19 倍.

在查询语句 Query2,Query6,Query9 和 Query13 中,查询层哈希转换优化后的分层式 RDF 数据查询系统的查询时间随着 RDF 数据规模的增长均低于未优化的分层式 RDF 数据查询系统的查询执行时间,并且查询返回的 RDF 数据结果越多,查询层进行哈希转换优化的效果越明显.

7.3 系统对比分析

为了评估采用了上述所有优化后的 Goldfish 系统的查询性能,本节采用已有的查询性能较好的语义数据查询系统 Jena-HBase,SHARD 和 Rainbow 与之进行对比. Jena-HBase,SHARD 和 Rainbow 系统都是基于类似平台(Hadoop/HBase)的分布式大规模 RDF 数据的查询系统.我们在数据集 LUBM-10 和 WordNet 3.1 中选取了部分查询语句,对每条查询在 Jena-HBase,SHARD,Rainbow 和 Goldfish 系统分别做了 3 趟测试,最后取平均运行时间,实验结果如图 36 和图 37 所示.

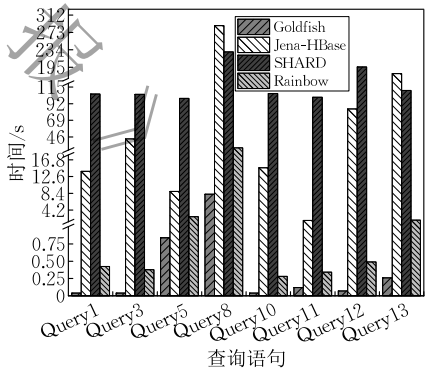

图 36 四种系统对 LUBM-10(1.3M 条三元组)的查询时间对比

(1) LUBM-10 下的四种系统对比

从图 36 可以看出,在 LUBM-10 规模的 RDF 数据集下,Goldfish 系统的查询时间均快于 Jena-HBase,SHRAD 和 Rainbow 这三种系统的查询时间.其中 Goldfish 系统对比 Jena-HBase 系统,优化效果最快的 Query3 的查询速度提高了 1000 倍,返回结果数 6 条;优化效果最慢的 Query5 的查询速度提高了 11 倍,返回查询结果数 678 条.

Goldfish 系统对比 SHARD 系统，优化效果最快的 Query12 的查询速度提高了 2700 倍，返回结果数 15 条；优化效果最慢的 Query8 的查询速度提高了 28 倍，返回查询结果数 7790 条. Goldfish 系统对比 Rainbow 系统，优化效果最快的 Query1 的查询速度提高了 11 倍，返回结果数 129466 条；优化效果最慢的 Query11 的查询速度提高了 3 倍，返回查询结果数 5 条.

(2) WordNet 3.1 下的四种系统对比

从图 37 可以看出，在真实语义数据集 WordNet 3.1 下，Goldfish 系统的查询时间均快于 Jena-HBase、SHRAD 和 Rainbow 这三种系统的查询时间. 其中 Goldfish 系统对比 Jena-HBase 系统，优化效果最快的 Query1 的查询速度提高了 348 倍，返回结果数 4 条；优化效果最慢的 Query10 的查询速度提高了 1.3 倍，返回结果数 12 条.

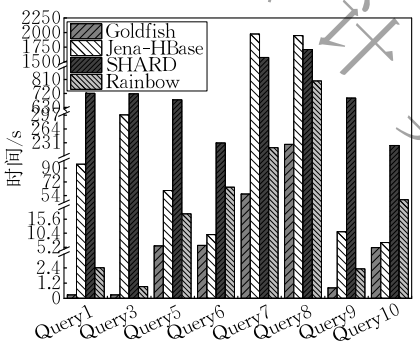

图 37　四种系统在 WordNet 3.1(8.9M 条三元组)下的查询时间对比

Goldfish 系统对比 SHARD 系统，优化效果最快的 Query1 的查询速度提高了 2642 倍，返回结果数 10 条；优化效果最慢的 Query8 的查询速度提高了 7.5 倍，返回结果数 7648 条. Goldfish 系统对比 Rainbow 系统，优化效果最快的 Query6 的查询速度提高 10.9 倍，返回结果数 7832 条；优化效果最慢的 Query9 的查询速度提高 2.8 倍，返回结果数 30 条.

因此，在大规模 RDF 数据查询任务中，Goldfish 系统比 SHARD、Jena-HBase 和 Rainbow 系统具有更好的性能.

8　相关工作

在过去，研究人员开发了诸如 Jena[11]、Sesame[12]、RDF-3X[13]等 RDF 数据存储和查询系统. 这类系统通常使用传统的关系型数据库作为底层存储，并通过一定的组织方式将语义数据存储在其中. 然后将 SPARQL 语义查询转换成传统的 SQL 语句进行数据查询[14]. 这样做得好处是单机范围内技术发展较为完善且系统稳定度高.

Erling 等人[15]基于 OpenLink Virtuoso① 实现了 RDF 三元组结构的存储. Yuan 等人[16]通过采用增量压缩、变长整数编码、基于数据块划分的存储等措施设计实现了 TripleBit. Franke 等人[17]在 HBase 中实现了基于水平存储 RDF 索引结构，并对比基于 MySQL 集群环境的 RDF 索引结构的性能. 基于 HBase 的方案在大多数的查询情况下均优于基于 MySQL 集群的方案. 但是在基于 HBase 的方案中存在两种格式的水平表，这样不仅造成冗余而且水平划分方案无法避免空值问题. 然而，现在随着 RDF 数据的爆炸性增长，基于传统单机数据库的 RDF 系统在处理语义的数据量和扩展性上存在一定的缺陷，使其无法满足大规模语义数据的应用场景. 传统的关系型数据库模型和 RDF 语义模型在设计和优化目标上并不完全一致，这往往会导致基于关系数据库的 RDF 数据系统在存储和查询语义数据时会有较大开销.

Khadilkar 等人[18]利用 Jena 提供的接口，开发出了 Jena-HBase 系统，使其能够适应大规模 RDF 数据处理的需求. 该系统使用 HBase 管理语义数据，解决了大规模 RDF 数据的存储问题. 但是，Jena-HBase 未根据 HBase 非关系型数据库的特性对查询引擎层未作出优化；并且，由于所有的 RDF 数据被存储在磁盘中，因而性能表现一般. 为了保证语义数据存储服务的可靠性以及查询服务的高效性，Gu 等人[19]研究提出了分层式 RDF 数据存储查询系统 Rainbow. 分层式 RDF 数据存储查询系统 Rainbow 分为两层：持久存储层和分布式内存存储层. 持久存储层通过 HBase 存储全局 RDF 数据，保证数据的高可靠性. 分布式内存存储层通过 Redis 存储经常被查询的热数据，将热数据存储在分布式内存中，以提供更高效的数据查询服务.

Rohloff 等人[20]基于 Hadoop 设计了 SHARD 系统，其将 RDF 数据存储在 HDFS 中以支持大规模语义数据的存储. 并且，为了提高大规模语义数据查询处理的效率，所有的查询处理全部转化为 MapReduce 任务来执行，能够很好地保证系统的容错性和可扩展性，但是 I/O 开销过大.

① https://en.wikipedia.org/wiki/Virtuoso_Universal_Server

另外,其他一些研究者利用图划分优化技术来进行 RDF 图模式的匹配.例如,Huang 等人[21]他们的图划分存储在 RDF 图上达到了高查询性能.这种方法在 RDF 图是静态的,机器数是预先固定的情况下表现很好,但是当新批次的数据和机器加入时,为了更好的负载平衡整个 RDF 图就需要重新划分和分配.Shao 等人[22]基于 MPI[23]构建 RDF 图存储系统 Trinity,Trinity 的存储是完全基于分布式内存中的.Gurajada 等人[24]基于异步 MPI 设计了 TriAD,其通过分布式的 RDF 图划分索引结构对 join 查询进行预先剪枝,之后利用多线程进行异步的 join 操作,使其达到了非常高的查询性能.Wu 等人[25]基于 Hadoop 设计 SemStore,利用 K-means 构建一种特殊的子图进行图划分,并提出分区敏感查询分解算法和两阶段的动态优化技术来减少分布式 join 查询开销.Chen 等人[26]基于 Spark 设计了 SparkRDF,其将 RDF 依据关系和类别划分为多个子图,利用 Spark 内存计算的特性减小 I/O 开销,来达到较高的查询效率.

Hammoud 等人[27]设计了 DREAM,将数据共享到每个节点,把查询转换为多个子图并映射到一个独立的节点,通过节点间的通信来获取最终的查询结果,以此来避免大量的网络数据传输而带来的开销.Galárraga 等人[28]则通过分析历史查询日志,动态地对数据进行分配来提高查询效率.

9 总结与展望

随着语义网技术的不断发展与进步,语义网被广泛应用在医疗、地理信息等多个领域.大数据时代的来临,给语义网的研究带来了新的机遇和挑战.特别是大规模 RDF 语义数据的存储与查询是目前研究的热点.针对大规模 RDF 数据的存储与查询问题,本文的主要贡献为:以 OpenRDF Sesame 框架为基础,设计实现了以 HBase 作为持久存储层、Redis 存储热点数据的分层式 RDF 数据存储查询系统 Goldfish,并针对三元组表存储结构所表现出的查询效率和存储空间利用率低、可扩展性不佳的问题,在存储层以属性表作为 RDF 数据存储结构,替代了常用的循环三元组表存储结构;属性表可以通过布尔矩阵分解算法(ASSO)生成,但是该算法在大规模数据集下出现了构造缓慢等问题.根据布尔矩阵分解算法的特性,本文通过频繁项集挖掘算法对其进行求解,并采用了基于 Spark 的框架并行化频繁项集挖掘算法,以加速在大规模数据集下属性表的构造.在查询层增加哈希转换层,避免了频繁查询哈希表带来的查询性能的降低.实验表明,Goldfish 整体性能比 Rainbow 系统平均提升 6 倍左右,比 Jena-HBase 平均提升 500 倍左右,比 SHARD 系统平均提升 1200 倍左右.

下一步工作,我们将会针对以下两点进一步研究和改进:

(1)对跨多张属性表进行查询时的查询效率降低问题.

(2)基于数据切分的并行化查询.基于大规模 RDF 数据的数据切割并行化查询一直是 RDF 数据查询的难点.RDF 数据是一种图数据,具有图结构的特殊性;而现有的分布式框架,如 MapReduce,是基于数据切分的并行化框架.如何利用现有的分布式计算框架,将 RDF 数据查询与之相结合,实现并行化查询,涉及 RDF 数据的图切割问题,需要进一步研究.

参 考 文 献

[1] Gantz J, Reinsel D. Extracting value from chaos. Framingham, USA: IDC's Digital Universe Study: IDC-1142, 2011

[2] Berners-Lee T, Hendler J, Lassila O. The semantic Web. Scientific American, 2001, 284(5): 28-37

[3] Zaharia M, et al. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing//Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation. USENIX Association, USA, 2012: 2-2

[4] Zaharia M, et al. Spark: Cluster computing with working sets//Proceedings of the 2nd USENIX Conference on Hot Topics in Cloud Computing. USENIX Association, USA, 2010: 10-10

[5] Ni Chuan-Lei, Hu Wei, Gu Rong, et al. Based on Boolean matrix decomposition of RDF data storage layout and query optimization method//Proceedings of the 31st National Database Conference of China. Taiyuan, China, 2014(in Chinese)
(倪传蕾,胡伟等.基于布尔矩阵分解的 RDF 数据存储布局及查询优化方法//第 31 届中国数据库学术会议.太原,中国,2014)

[6] Miettinen P, Mielikainen T, Gionis A, et al. The discrete basis problem. IEEE Transactions on Knowledge and Data Engineering, 2008, 20(10): 1348-1362

[7] Chandola V, Kumar V. Summarization—compressing data into an informative representation. Knowledge and Information Systems, 2007, 12(3): 355-378

[8] Qiu Hong-Jian, Gu Rong, Yuan Chun-Feng, Huang Yi-Hua. YAFIM: A parallel frequent itemset mining algorithm with spark//Proceedings of the 28th International Parallel & Distributed Processing Symposium Workshops(ParLearning). Phoenix, USA,2014: 1664-1671

[9] 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): 158-182

[10] Miller G A. WordNet: A lexical database for English. Communications of the ACM, 1995, 38(11): 39-41

[11] McBride B. Jena: A semantic web toolkit. IEEE Internet Computing, 2002, 6(6): 55-59

[12] Broekstra J, Kampman A, Van Harmelen F. Sesame: A generic architecture for storing and querying RDF and RDF schema// Proceedings of the 1st International Semantic Web Conference. Sardinia, Italy, 2002: 54-68

[13] Neumann T, Weikum G. The RDF-3X engine for scalable management of RDF data. The International Journal on Very Large Data Base, 2010, 19(1): 91-113

[14] Sakr S, Al-Naymat G. Relational processing of RDF queries: A survey. ACM SIGMOD Record, 2010, 38(4): 23-28

[15] Erling O, Mikhailov I. RDF Support in the Virtuoso DBMS// Proceedings of the Networked Knowledge-Networked Media. New York, USA, 2007: 59-68

[16] Yuan Ping-Peng, Liu Pu, et al. TripleBit: A fast and compact system for large scale RDF data. The Very Large Data Base Endowment, 2013, 4(7): 517-528

[17] Franke C, Morin S, Chebotko A, et al. Distributed semantic web data management in HBase and MySQL cluster// Proceedings of the 2011 IEEE 4th International Conference on Cloud Computing. Washington, USA, 2011: 105-112

[18] Khadilkar V, Kantarcioglu M, Thuraisingham B. Jena-HBase: A distributed, scalable and effcient RDF triple store//Proceedings of the 2012 International Semantic Web Conference (Posters & Demos). Aachen, Germany, 2012: 85-88

[19] Gu Rong, Hu Wei, Huang Yi-Hua. Rainbow: A distributed and hierarchical RDF triple store with dynamic scalability// Proceedings of the 2014 IEEE International Conference on Big Data. Washington, USA, 2014: 561-566

[20] Rohloff K, Schantz R E. High-performance, massively scalable distributed systems using the MapReduce software framework: The SHARD triple-store//Proceedings of the Programming Support Innovations for Emerging Distributed Applications. New York, USA 2010: 4.

[21] Huang J, Abadi D J. Scalable sparql querying of large rdf graphs. The Very Large Data Base Endowment, 2011, 4(11): 1123-1134

[22] Shao Bin, Wang Hai-Xun, Li Ya-Tao. Trinity: A distributed graph engine on a memory cloud//Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data. New York, USA, 2011: 505-516

[23] The MPI Forum. MPI: A message passing interface// Proceedings of the 1993 ACM/IEEE Conference on Supercomputing. New York, USA, 1993: 878-883

[24] Gurajada S, Seufert S, Miliaraki I, et al. TriAD: A distributed shared-nothing RDF engine based on asynchronous message passing//Proceedings of the ACM SIGMOD Conference on Management of Data. New York, USA, 2014: 289-300

[25] Wu Bu-Wen, Zhou Yong-Luan, Yuan Ping-Peng, et al. SemStore: A semantic-preserving distributed RDF triple store//Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management. New York, USA, 2014: 509-518

[26] Chen Xi, Chen Hua-Jun, Zhang Ning-Yu, et al. SparkRDF: Elastic discreted RDF graph processing engine with distributed memory//Proceedings of the 2014 International Conference on Posters & Demonstrations Track. Riva del Garda, Italy, 2014: 261-264

[27] Hammoud M, et al. DREAM: Distributed RDF engine with adaptive query planner and minimal communication. The Very Large Data Base Endowment, 2015, 8(6): 654-665

[28] Galárraga L, Hose K, Schenkel R. Partout: A distributed engine for efficient RDF processing//Proceedings of the 23rd International World Wide Web. Seoul, Korea, 2014: 267-268

GU Rong, born in 1988, Ph. D. candidate. His research interests include big data parallel processing and cloud computing.

QIU Hong-Jian, born in 1990, M. S. candidate. His research interests include big data parallel processing and cloud computing.

YANG Wen-Jia, born in 1991, M. S. candidate. His research interests include big data parallel processing and cloud computing.

HU Wei, born in 1982, associate professor. His main research interests include ontology engineering and data fusion.

YUAN Chun-Feng, born in 1963, professor. Her research interests include computer system architecture, big data processing and web information extraction and integration.

HUANG Yi-Hua, born in 1962, professor. His research interests include big data parallel processing and cloud computing, computer architecture and parallel computing.

Background

In the Big Data era, with the rapid development of the Internet applications and the semantic web technology, the semantic data is exploding. Many applications can provide better services based on the large amount of semantic data. Thus, it is significant to store and query over the large semantic data. However, the rapid increase of the semantic data brings new challenges on efficient storing and querying semantic data in the big data environment. The traditional ways for semantic data management is to store and query the data in relational database management systems. With the data increases, these traditional ways can hardly handle big data. Recently, researchers try to adopt the cutting-edge distributed system such as MapReduce and HBase to handling large scale semantic data. However, they put forward few indexing strategies, and systems like SHARD inherit the inefficiency of Hadoop MapReduce.

To address these problems, this paper proposed a distributed hierarchical storage architecture to store and query large-scale semantic data based on the OpenRDF Sesame framework, called Goldfish. The RDF storage and indexing mechanism is the triple store optimized by adopting the attribute table to replace the raw RDF tables. To handle generating the attribute table structure of the large scale semantic data, a parallel frequent itemset mining algorithm with Spark framework is adopt to process the large scale data. In addition, an optimized hash conversion layer is proposed to reduce time cost in frequent hash table search during the ad-hoc query stage.

Goldfish takes advantage of the cluster resource and performs good scalability. The experimental results show that Goldfish is around 8 times faster than Rainbow, 500 times faster than Jena-HBase and 1200 times faster than the MapReduce based RDF querying system SHARD.

This work is supported by Special Funds of the National Natural Science Foundation of China Grant (No. 61223003), the National Natural Science Foundation of China Grant (No. 61370019) and the Key Provience Technologies Research and Development Program of Jiangsu (No. BE2014131).