YARM:基于 MapReduce 的高效可扩展的语义推理引擎

顾 荣 王芳芳 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 210046)

摘 要 随着语义网的快速发展,RDF 语义数据大量涌现.大规模 RDF 语义数据推理的一个主要问题是计算量大、完成计算需要消耗很长的时间.显然,传统的单机语义推理引擎难以处理大规模的语义数据.另一方面,现有的基于 MapReduce 的大规模语义推理引擎,缺乏对算法在分布和并行计算环境下执行效率的优化,使得推理时间仍然较长.此外,现有的推理引擎大多存在可扩展性方面的不足,难以适应大规模语义数据的增长需求.针对现有的语义推理系统在执行效率和可扩展性方面的不足,文中提出了一种基于 MapReduce 的并行化语义推理算法和引擎 YARM.为了实现分布和并行计算环境下的高效推理,YARM 做出了以下 4 点优化:(1)采用合理的数据划分模型和并行化算法,降低计算节点间的通信开销;(2)优化推理规则的执行次序,提升了推理计算速度;(3)设计了简洁的去重策略,避免新增作业处理重复数据;(4)设计实现了一种新的基于 MapReduce 的并行化推理算法.实验结果表明,在真实数据集和大规模合成数据集上,YARM 的执行速度比当前最新的基于 MapReduce 的推理引擎快 10 倍左右,同时 YARM 还表现出更好的数据和系统可扩展性.

关键词 RDF;RDFS 推理;MapReduce;语义推理;分布式推理
中图法分类号 TP338;TP182 **DOI 号** 10.3724/SP.J.1016.2015.00074

YARM: Efficient and Scalable Semantic Reasoning Engine Based on MapReduce

GU Rong WANG Fang-Fang YUAN Chun-Feng HUANG Yi-Hua
(*State Key Laboratory for Novel Software Technology at Nanjing University, Nanjing* 210046)

Abstract The rapid development of the Semantic Web has produced massive amount of the RDF data. The major challenge for large scale RDF semantic reasoning is that it involves huge amount of computation. This makes the whole process very time-consuming. It is obvious that the traditional semantic reasoning engines are not efficient when dealing with the massive amount of RDF data. On the other hand, the state-of-art distributed semantic reasoning algorithms built with MapReduce lack of optimization for reasoning process in a distributed and parallelized environment. Thus, this still makes the reasoning process relatively time-consuming. In addition, most of existing reasoning engines lack of scalability. To solve these problems, we design and implement YARM, a new parallel semantic reasoning algorithm and engine that built with the MapReduce parallel model. YARM includes four major optimizations: first, it adopts a well-designed data partitioning schema and a corresponding reasoning algorithm to minimize the amount of data transferred among computing nodes; second, it optimizes the execution order of

收稿日期:2013-09-30;最终修改稿收到日期:2014-08-08.本课题得到国家自然科学基金专项基金(61223003)和美国 Intel Labs 大学研究项目资助.**顾 荣**,男,1988 年生,博士研究生,中国计算机学会(CCF)会员,主要研究方向为大数据并行处理技术和分布式数据挖掘等.E-mail:gurong@smail.nju.edu.cn.**王芳芳**,女,1989 年生,硕士,主要研究方向为大数据并行处理与云计算技术.**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与并行计算、多媒体文档处理、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处理与云计算技术、体系结构与并行计算.E-mail:yhuang@nju.edu.cn.

the reasoning rules to improve the computing speed; third, it uses an efficient way to remove duplicates yielded in reasoning process. This avoids the need of extra MapReduce jobs to do this work; forth, based on the optimizations above, we design and implement a new parallel reasoning algorithm on the Hadoop MapReduce framework. Experimental results on both real-world and synthetic datasets show that YARM is about 10 times faster than the latest MapReduce-based reasoning engine and also achieves better scalability.

Keywords　RDF; RDFS reasoning; MapReduce; semantic reasoning; distributed reasoning

1 引　言

近年来，行业应用数据规模的爆炸性增长推动了大数据技术的迅猛发展. 在常规大数据迅猛增长的同时，语义数据特别是 RDF 数据也以数亿甚至数十亿元组的规模大量涌现. 例如，截至 2012 年 3 月，LOD(Linked Open Data)项目①所收集的 RDF 数据集已经包含了超过 325 亿条 RDF 三元组. 大规模语义数据发展的这一趋势对 RDF 数据存储、查询和推理在可扩展性和效率方面提出了新的挑战[1].

RDF 推理所关注的主要目标是如何借助自动推理机从给定的知识演绎(推导)出一些结论，从而使隐含的知识外显出来[2]. 对隐含的语义数据的支持是 RDF 图与其他数据模型的本质区别，也是语义网发展的重要推动力[3]. 大数据技术背景下，随着 RDF 语义数据规模的迅速扩大，研究寻找高效的并行化语义推理机制和方法成为一个重要技术难题，成为大规模 RDF 数据管理系统迫切需要研究解决的重要问题.

利用并行计算技术解决大规模 RDF 数据相关问题已成为学术界和工业界的普遍共识[4]. 由 Google 提出的 MapReduce 并行计算模型以其高可扩展性和高易用性成为目前大数据处理最为成功的并行计算技术之一[5]. Hadoop 是 Google MapReduce 框架的一个开源实现，其提供了类似于 Google GFS 的分布式数据存储系统 HDFS 以及类似于 Google BigTable 的面向半结构化数据存储和管理系统 HBase.

目前，Hadoop 已成为大数据存储和并行处理实际上的工业标准，也成为目前大数据深度分析挖掘并行处理的主流技术和平台，在国内外诸多大型互联网企业和其他大数据分析行业应用中得到广泛应用. 因此，现实生活中很多涉及到大规模语义分析挖掘的大数据分析应用都可能需要在这个平台上实现，这就使得基于 MapReduce 完成大数据场景下快速的 RDFS 推理变得十分重要.

目前已有基于 MapReduce 的并行推理的研究工作[6-7]. 这些工作主要是将传统的推理技术直接迁移到 MapReduce 框架下，这种直接迁移的方式使得一次推理任务需多个链式 MapReduce 作业协同工作才能完成，其中包括增加额外处理重复数据的作业. 因此，现有的这些基于 MapReduce 的方法推理计算效率较低. 为了实现 MapReduce 下大规模 RDF 数据的高效推理，本文在对资源描述框架 RDF、RDF Schema(RDFS)等相关技术分析的基础上，实现了一种高效的基于 MapReduce 的 RDFS 推理算法和引擎 YARM(Yet Another Reasoning System with MapReduce). 通过采用合理的数据划分方案和优化的推理规则执行策略，YARM 将推理计算分解为多个相互间没有依赖关系的独立的推理任务，解决了传统推理算法的直接迁移所带来的大量数据移动问题. 此外，不同于传统的推理机，YARM 根据推理规则之间的依赖关系，优化了规则执行次序，避免了迭代计算，提高了推理过程的执行效率. 最后，YARM 还采取了有效的措施来解决分布式推理中的重复数据消除问题. 在大规模 LUBM 以及 WordNet 与 DBpedia 等模拟和真实实验数据集上的实验结果表明，相比于传统的基于 MapReduce 的推理引擎，YARM 在执行效率上提升了 10 倍左右，同时表现出良好的可扩展性.

与当前基于 MapReduce 的语义信息并行推理算法相比，论文的主要贡献可以归纳为如下 4 个方面：

(1) 研究设计了一种简洁的数据划分方案，将模式 RDF 和实例 RDF 数据分开处理，解决了传统推理算法的直接迁移而导致的多次数据通信问题.

(2) 研究并提供了一种优化的规则应用次序，提高了推理的执行效率.

① The Cooperative Association for Internet Data Analysis(CAIDA) [EB/OL]. http://www.linkeddata.org 2009,1,20

(3)结合 RDFS 规则特征和 MapReduce 框架特点设计了一种高效的重复数据检测和消除策略.

(4)设计实现了完整的 MapReduce 并行化推理算法.

本文第 2 节详细介绍 RDF、RDFS 以及推理问题的相关概念;第 3 节介绍 RDFS 推理并行化的相关工作;第 4 节对 RDFS 推理并行化处理的技术困难进行分析并给出解决方案;第 5 节讨论 YARM 的设计和实现;第 6 节采用多个数据集对 YARM 进行测试和分析;最后对全文进行总结.

2 语义推理相关概念

RDF(Resource Description Framework,资源描述框架)①是一种数据模型,是由万维网联盟(World Wide Web Consortium,W3C)组织的 RDF 工作组于 1999 年提出的描述 Web 信息的知识表示语言. RDF 是谓词逻辑的一种特殊形式,具有形式化的语义表示,有助于计算机理解它所表达的语义信息.

RDF 数据模型(RDF 图)的组成包括资源、属性和陈述. 资源以唯一的统一资源标识符(Uniform Resource Identifiers,URI)来表示,可以是网络上的任何信息、虚拟概念或现实事物等;属性则用来描述资源的特征以及资源之间的关系,每一属性都有其意义;陈述是一条三元组,表示为(主语、谓语、宾语)的形式. 其中主语是资源,由 URI 表示;谓语是属性,亦由 URI 表示;宾语是资源(URI)或文本.

RDF 的词汇描述语言 RDF Schema(RDFS)②利用 RDF 进一步定义了建模原语,RDFS 可以看作是针对 RDF 模型的一个基本类型系统,是一种简单的本体语言,即"一种对共享概念化的、明确的形式化说明"[8]. 使用 RDFS 可以定义属性的特征,定义被描述资源的类,并对类和关系的可能组合进行约束. RDFS 规则推理支持几乎所有的 RDFS 蕴含,并且具有良好的可计算性和表达能力,是最广泛使用的推理方法之一,得到了众多研究者的关注和研究[1,3,6-7,9].

RDF 图显式地表示了一个三元组的集合,同时结合 RDFS 语义定义的一组规则(如表 1 所示)还具有表示隐含数据的能力,这是 RDF 数据模型不同于传统数据模型的主要特征之一. 隐含数据的求解过程即为推理过程,亦即在给定一个 RDF 图 G 的情况下,根据 RDFS 推理规则加入一组新的三元组 T,从而得到更大的 RDF 图 G',这个过程称为推理过程或 RDFS 闭包的具体化[9]. G 所描述的数据是显式的 RDF 语义信息,称为显式数据;新加入的三元组集合 T 是隐含的 RDF 语义信息,称为隐含数据. 例如,设 RDF 图中有如下三元组{($Master$,$rdfs$:$subClassOf$,$Student$),($Student$,$rdfs$:$subClassOf$,$Person$)}. 根据规则 11 可知,RDF 图中还隐含着三元组($Master$,$rdfs$:$subClassOf$,$Person$).

事实上,表 1 中给出的 13 条规则是 RDFS 推理的一个标准规则集. 该规则集具有良好的计算能力和可判定性,包含了所有 RDFS 蕴涵,被广泛应用于各种领域[1]. RDFS 本身是一个原始本体语言,用于描述概念型的信息,而非特定的实例或对象,是具有广泛的抽象意义的,也就是说 RDFS 适用于任何领域,具有高度的通用性[9]. 基于 RDFS,用户可定义面向特定应用系统的概念对象和数据关系,创建相应的 RDF 数据.

推理可以在数据插入到 RDF 存储系统时执行,称为前向推理;也可以在查询发生时执行,称为后向推理. 前向推理(forward chaining)从一个初始的事实出发,不断应用规则得出结论. 前向推理在数据插入存储系统时被触发,推理得到的三元组和原三元组都保存在 RDF 存储系统中. 其优点是查询速度快;缺点是计算全部隐式数据需要的时间开销较大,而且推理结果的物化需要占用较多的磁盘空间. 当前有一些前向推理系统,如 Sesame 和 RStar. 后向推理(backward chaining)从查询目标出发,利用规则不断地进行目标消解来完成推理. 推理在查询时触发,从规则的结论出发依次归纳为规则的条件. 这种方法不需要大量的磁盘空间来保存无用的推理结果,也不需要额外的数据预处理时间,但是由于推理在查询时完成,因此会导致查询处理性能较低.

由于前向推理在查询响应时间方面的优势以及其在推理系统中的主流地位[10],同时也由于目前主流的大数据分析挖掘并行处理平台是 Hadoop MapReduce,本文主要研究基于 MapReduce 的前向推理并行化算法. 下文中的推理均指前向推理. 为了表达方便,本文使用实例数据和模式数据分别代表由 RDF 语义构造的三元组和使用 RDFS 语义构造的三元组.

① Resource Description Framework (RDF): Concepts and Abstract Syntax [EB/OL], http://www.w3.org/TR/rdf-concepts/2004

② RDF Vocabulary Description Language 1.0: RDF Schema [EB/OL], http://www.w3.org/TR/rdf-concepts/2004

表 1　RDFS 推理规则

规则	条件	结论(规则输出)
1	*s p o*	*_:n rdf:type rdfs:Literal*
2	*p rdfs:domain x & s p o*	*s rdf:type x*
3	*p rdfs:range x & s p o*	*o rdf:type x*
4	*s p o*	*s/o rdf:type rdfs:Resource*
5	*p rdfs:subPropertyOf q & q rdfs:subPropertyOf r*	*p rdfs:subPropertyOf r*
6	*p rdf:type rdf:Property*	*p rdfs:subPropertyOf p*
7	*s p o & p rdfs:subPropertyOf q*	*s q o*
8	*s rdf:type rdfs:Class*	*s rdfs:subClassOf rdfs:Resource*
9	*s rdf:type x & x rdfs:subClassOf y*	*s rdf:type y*
10	*s rdf:type rdfs:Class*	*s rdfs:subClassOf s*
11	*x rdfs:subClassOf y & y rdfs:subClassOf z*	*x rdfs:subClassOf z*
12	*p rdf:type rdfs:ContainerMembershipProperty*	*p rdfs:subPropertyOf rdfs:member*
13	*o rdf:type rdfs:Datatype*	*o rdfs:subClassOf rdfs:Literal*

3　相关工作

当前，在 RDFS 语义推理方面有着众多的研究[6,7,10-21]. 根据计算模式和实现方式的不同，可以分为以下几种：

(1) 基于单节点的推理方法. 典型的单机推理系统有 Jena[11]、Pellet[12]、Sesame[13]等. 这些系统为语义数据推理奠定了重要的技术基础，但是由于单节点运行环境的限制，在可扩展性和计算性能方面存在不足，它们难以处理大规模语义数据.

(2) 基于关系型数据库的推理方法[14-15]. 这类方法采用关系数据库作为 RDF 存储系统，结合现有的单机推理系统对语义数据进行推理，满足了语义信息的存储和查询要求. 与单机推理系统类似，这类系统在推理速度和可扩展性方面也存在不足.

(3) 基于分布式哈希技术的推理方法. 文献[16]采用了基于分布式哈希技术的并行推理方法. 该方法将数据存储在分布式哈希表中，推理过程需不断访问哈希表以避免数据重复. 该类方法存在的主要问题是负载不均衡，因此在处理大规模数据的场景下，容易产生严重的性能瓶颈.

(4) 基于数据划分的推理方法. 这类方法主要通过对语义数据或规则进行划分来实现语义数据推理的并行化. MacCartney 等人[17]提出了一种基于图划分的一阶逻辑推理算法；Soma 和 Prasanna[10]提出了一种通过数据划分来实现并行推理的技术，但是这些方法并没有提供大数据集下的实验结果. Weaver 与 Hendler[18]提出了一种数据划分模型，在该模型下推理任务可被分解为相互独立的多个并行子任务，各节点对本地的数据应用所有的推理规则，直到不再有新的数据产生，最后通过合并各节点局部结果得到全局解. 该方案的缺陷是没有检测和过滤重复数据，另外该方案在每个数据单元上使用传统的推理机，计算效率较低.

(5) 基于 P2P 网络的推理方法. 文献[1,19-20]提出了基于 P2P 自组织网络的并行化推理方法 Marvin. Marvin 以“divide-conquer-swap”的方式执行前向推理，蕴含规则以渐进方式执行直到不会再产生新数据为止，整个推理过程也需要多次数据通信[21].

(6) 基于 MapReduce 的推理方法. 这类方法在底层采用 Hadoop MapReduce 方法和平台，结合数据划分方法来实现大规模语义数据的推理. 文献[7]第一次提出基于 MapReduce 的推理算法，但是并未给出任何实验结果. Urbani 等人[6]提出的基于 MapReduce 物化 RDFS 闭包的分布式推理算法 reasoning-hadoop，是目前为止最快的大规模并行化推理算法. 该算法通过分析 RDFS 规则之间的依赖关系，分析出 RDFS 规则依赖图，在该图的指导下将原来需要多次迭代才能完成的推理工作简化为只需 4 次 MapReduce 作业. 但是该方案中每个步骤使用相同的数据划分方案，不同的只是计算部分，这导致了大量静态数据重复传输和中间结果不能有效利用的问题，此外 reasoning-hadoop 中每个作业一般都是较短的数据密集型作业，推理过程中的大部分时间都消耗在任务的创建、数据传输以及磁盘输入输出上，处理代价高，计算性能较低.

基于 MapReduce 的 RDFS 推理算法提出后，得到了学术界的广泛关注，众多研究者开始探讨基于 MapReduce 的特定领域或其他规则库的推理算法. Liu 等人[22]提出了基于MapReduce的大规模模糊 RDF 语义数据的分布推理引擎；Tachmazidis 等人[23]提出了基于MapReduce的非单调规则推理的并行化设计方案；而 Wu 等人[24]提出了基于 MapReduce 的分布式规则执行机制，但并未给出实验数据；Jang 和 Ha[25]针对规则推理中的传递性关系的计算提出了特定

的解决方案;Maeda 等人[26]提出了基于 MapReduce 的规则系统的设计和实现. 这些工作的研究进一步拓展了 MapReduce 在语义推理领域的应用,为大规模分布式语义推理做出了重要贡献.

本文所提出的并行化推理算法和对应实现的引擎 YARM 采用了与 Weaver 和 Hendler 类似的数据划分模型,并根据 RDFS 规则依赖关系优化了推理过程. YARM 采用不同数据划分模型,使推理在一次 MapReduce 作业之内即可完成,显著提升了计算效率. 在单个任务上,YARM 采用与 reasoning-hadoop 类似的规则优化策略进行推理计算,以避免多次迭代处理. 此外,YARM 还结合 MapReduce 的特点设计了高效的重复数据检测和消除机制.

4 RDFS 推理并行化问题和解决方案

4.1 并行化问题

大规模 RDF 语义数据推理的一个主要问题是计算量大、完成计算需要消耗很长的时间,因此,需要研究并行化的 RDFS 推理算法.

RDFS 推理过程实质上是一个对 RDF 输入数据反复应用 RDFS 推理规则进行推理的过程. RDFS 推理规则如表 1 所示. 其中,规则 1、4、6、8、10 只有一个条件语句,对推理并行化算法改造没有影响;而规则 12 和 13 在 RDF 数据中是极少出现的,在大多数推理工作中均不进行讨论[6,14-15,18,21]. 因此,影响推理并行化改造的主要规则归结在表 2 中.

表 2 影响推理并行化改造的 RDFS 规则子集

规则	条件	结论
2	$p\ rdfs:domain\ x$ $s\ p\ o$	$s\ rdf:type\ x$
3	$p\ rdfs:range\ x$ $s\ p\ o$	$o\ rdf:type\ x$
5	$p\ rdfs:subPropertyOf\ q$ $q\ rdfs:subPropertyOf\ r$	$p\ rdfs:subPropertyOf\ r$
7	$s\ p\ o$ $p\ rdfs:subPropertyOf\ q$	$s\ q\ o$
9	$s\ rdf:type\ x$ $x\ rdfs:subClassOf\ y$	$s\ rdf:type\ y$
11	$x\ rdfs:subClassOf\ y$ $y\ rdfs:subClassOf\ z$	$x\ rdfs:subClassOf\ z$

这些规则在推理中被表示为三元组的连接(join)操作,并且需要反复执行. 应用于处理大规模 RDF 数据时,将会面临多次大规模数据集的自连接操作,导致巨大的计算量,因此,传统的推理机不能直接应用于大规模数据的处理. 此外,在推理过程中会产生大量重复数据,这些重复数据既可能位于同一节点,也可能位于不同节点,如图 1 所示. 如何检测和消除这些重复数据也是推理算法并行化改造时需要研究并解决的一个重要问题.

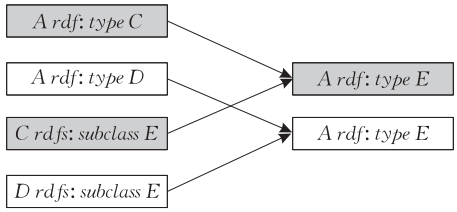

图 1 重复数据(灰色部分和白色部分既可能位于同一节点,也可能位于不同节点)

4.2 推理并行化解决方案

4.2.1 规则依赖关系

为了设计出高效可扩展的并行化推理引擎,需要对 RDFS 的规则进行深入分析. 首先按规则的输出对其分类,可分为如下几类:

(1) 输出三元组以 $rdfs:subPropertyOf$ 为谓词的规则:规则 5.

(2) 输出三元组以 $rdfs:subClassOf$ 为谓词的规则:规则 11.

(3) 输出三元组以 $rdf:type$ 为谓词的规则:规则 2、3、9.

(4) 输出三元组无固定模式的规则:规则 7.

另一方面,依据规则输入可将规则分为 4 类:

(1) 输入三元组中至少有一条谓词为 $rdfs:subPropertyOf$ 的规则:规则 5、7.

(2) 输入三元组中至少有一条谓词为 $rdfs:subClassOf$ 的规则:规则 9、11.

(3) 输入三元组中至少有一条谓词为 $rdf:type$ 的规则:规则 9.

(4) 输入三元组中至少有一条为任意实例三元组的规则:规则 2、3、7.

按以上推理规则的输入输出关系重新对这些规则进行组织,可得到如图 2 所示的规则依赖关系图.

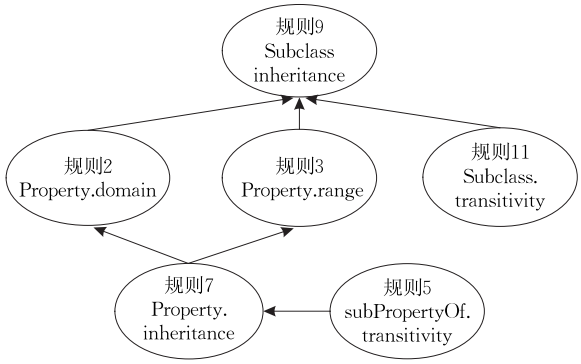

图 2 规则依赖图

RDFS 规则还具有如下的特征：(1) RDFS 中每一条规则的输入至多有两个三元组，且至少一个为 RDFS 模式三元组；(2) 据统计，RDF 数据集中模式三元组的数量只占极少的一部分，大部分数据为实例三元组. 这说明 RDFS 推理中不存在实例数据之间的连接操作，只存在模式数据之间或模式数据与实例数据之间的连接操作[6,14-15,18,21].

4.2.2　数据划分模型

解决数据通信问题的方法是设计一个好的数据划分策略[10]. 根据对 RDFS 规则的分析可知，RDF 数据集可以被分成两部分：一部分是小规模的模式数据；另一部分是大规模的实例数据. 模式数据和实例数据之间存在连接关系，而实例数据之间是没有数据依赖关系的，因此在划分数据时，将小规模模式数据作为全局数据，复制到每个节点的内存中；然后，我们对大规模的实例数据进行划分，并采用分布式存储的方式，以此解决大规模 RDF 实例数据的存储管理和并行化推理计算.

在这种数据划分模型下，一方面，规则匹配由原来一个大表的自连接操作转换为两个集合间的连接(一个小规模模式数据集与一个大规模实例数据集的连接)，例如，规则 9($s\ rdf:type\ x$) & ($x\ rdfs:subClassOf\ y$)→($s\ rdf:type\ y$)可通过对模式数据集和实例数据集在 x 上做连接实现；另一方面，各计算节点的任务之间无关联关系，整个输入数据上的推理可分解成多个可同时并行执行的推理任务. 该数据划分模型在文献[18]中被首次提出，并经过了严格的证明，YARM 借鉴上述数据划分模型，以减少 MapReduce 环境下推理时节点间的数据通信开销.

4.2.3　消除迭代

在推理过程中迭代涉及到两个方面：第一，单个规则需要迭代运算，如规则 5、7、9、11，这类规则的特点是规则输出又可以作为自身的输入，称为传递规则. 传递规则的推理计算实质上是求解输入数据的传递闭包. 第二，不同规则间存在依赖关系(图 2)，整个推理过程需要多次迭代直到不再产生新的隐含数据为止.

通过对规则间依赖关系的分析，我们得到了一个非循环的规则依赖图(图 2)，该图存在一种拓扑排序(比如规则“5,11,7,2,3,9”，这里的序列不是表达前后两项的一一依赖关系，而是图论中的拓扑排序关系. 具体地，指在这个序列中不存在任何一项依赖于该项之后出现的项的情况)，使得我们对每条规则只应用一次即可推导出所有的隐含数据，推理过程不再需要多次迭代. 另一方面，对于单个传递规则来说，通过在规则 7 之前执行规则 5，可以消除规则 7 的迭代计算，同样在规则 9 之前执行规则 11，可以消除规则 9 的迭代计算[15]. 而同时规则 5 和 11 的推理只涉及小规模模式数据，其推理计算可在内存中完成.

4.2.4　消重策略

在 RDFS 推理中重复数据存在两种类型：第一种是新推导出的结果和原数据存在重复(图 3)；第二种是新推导出的数据之间存在重复(图 1). 为了消除结果中的重复数据，需要将原始数据和新推导出的数据进行比较. 因为 Hadoop 能自动地将 Map 端输出的键值对按键值组织在一起，因此在将三元组设置为键值的情况下，相同三元组会被组织在一起，可以很容易地使用 Combine 和 Reduce 来实现重复数据的检测和消除.

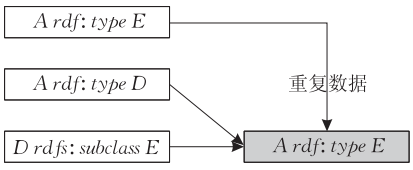

图 3　重复数据(白色框内为原始数据，黑色框内为推理出的重复数据结果)

由表 2 可知，RDFS 规则的输出存在 4 种模式：

(1) 模式 1(?x,$rdf:type$,?y)，即谓词为 $rdf:type$ 的三元组；

(2) 模式 2(?x,$rdfs:subPropertyOf$,?y)，即谓词为 $rdfs:subPropertyOf$ 的三元组；

(3) 模式 3(?x,$rdfs:subClassOf$,?y)，即谓词为 $rdfs:subClassOf$ 的三元组；

(4) 模式 4(?x,?$subPropertyRelated$,?y)，即谓词具有 $rdfs:subPropertyOf$ 属性的三元组.

其中，与模式 1 和模式 4 匹配的三元组只可能为实例三元组；与模式 2 和模式 3 匹配的三元组只可能为模式三元组.

可见，RDFS 规则的输出遵循固定的模式，因此，只需与原始 RDF 图中匹配这些模式的部分数据进行比较，即可检测出所有与原始 RDF 三元组存在重复的推理结果. 这种方式会减少大量不必要的数据传输，对提升推理性能起到重要作用.

5 YARM 的设计与实现

5.1 MapReduce 并行化推理算法设计与实现

根据第 4 节的推理算法并行化解决方案，在 YARM 中 RDFS 前向推理可归结为 3 个阶段. 在阶段 I 中，将小规模模式数据复制到每个节点的内存中，同时将实例数据划分为多个互不重叠的分区. 每个节点拥有全部的模式数据和部分实例数据，在推理执行过程中，将不需要从其他节点得到数据或者其他控制信息，亦即各节点之间是相互独立的，可以并行地执行推理计算. 在阶段 II 中，各节点并行地对本地数据执行推理计算过程，该阶段主要负责依据规则依赖图对输入数据应用推理规则进行推理处理. 阶段Ⅲ负责对各节点的推理结果进行合并，并消除结果中的重复数据. 可以看出，阶段 I 相当于数据划分阶段，阶段 II 相当于推理计算阶段，而阶段Ⅲ相当于合并阶段，因此这 3 个阶段可以方便地用 MapReduce 并行计算模型加以实现. YARM 执行框架如图 4 所示.

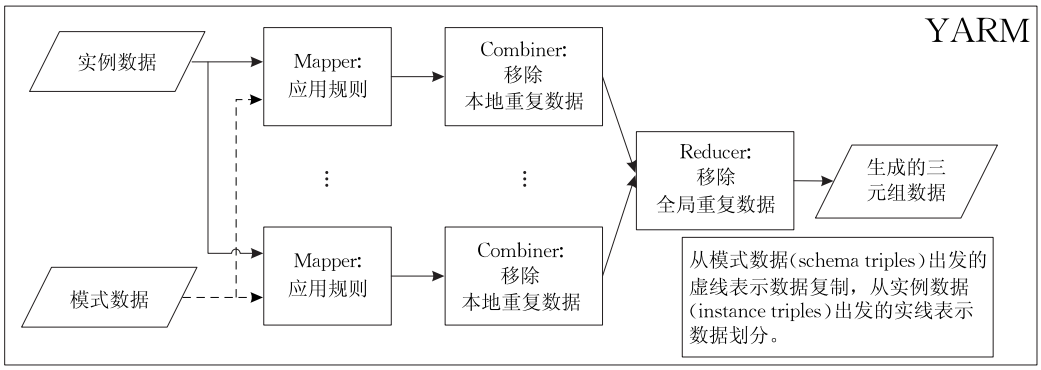

图 4 YARM：基于 MapReduce 的并行化推理算法与处理过程

5.2 数据划分

在 YARM 中，我们假定整个 RDF 三元组数据集按行存储在 HDFS 中，实例数据和模式数据分别作为一个文件. 首先，将利用 Hadoop MapReduce 中的 Distributed Cache 机制，在 *map*() 的 *setup*() 初始化函数中将模式数据复制到每个节点的本地内存中；然后将实例数据作为 *map*() 函数的输入，这样可以利用 HDFS 自身的存储机制把数据集切分成固定大小的 N 个子数据块(默认为 64 MB)，这些子数据块分布在集群内不同的节点上.

为了节省数据导入时间和存储空间，我们对 RDF 三元组进行了编码压缩，模式三元组和实例三元组在编码阶段分别输出到不同的文件，同时，考虑到程序的实现，存储文件采用 Hadoop SequenceFile 格式，每条序列以键值(*key-value*)对的形式存储，直接以三元组作为键(*key*)，值(*value*)设置为空. 由于所有三元组的大小被编码为相同的长度，因此各节点得到的三元组数量基本相同，保证了各节点间的负载均衡.

其次，为了便于 Map 阶段的处理，依据所匹配的规则，模式数据被组织成四类相应的内存数据结构. 其中与规则 2 匹配的三元组(即谓词为 *rdfs*:*domain* 的三元组)存放在 *domainMap* 数据集中，例如，三元组(*masterDegreeFrom*，*rdfs*:*domain*，*Master*)的谓词为 *rdfs*:*domain*，则〈*masterDegreeFrom*，*Master*〉被添加到 *domainMap* 中；与规则 3 匹配的三元组(谓词为 *rdfs*:*range* 的三元组)存放在 *rangeMap* 数据集中；与规则 5 和 7 匹配的三元组(谓词为 *rdfs*:*subClassOf* 的三元组)存放在 *subClassMap* 数据集中，例如，三元组(*Master*，*rdfs*:*subClassOf*，*Student*)的谓词为 *subClassOf*，则〈*Master*，*Student*〉被添加到 *subClassMap* 中；而与规则 9 和 11 匹配的三元组(谓词为 *rdfs*:*subPropertyOf* 的三元组)存放在 *subPropMap* 数据集中.

Mapper 端 *setup*() 装载和组织数据的实现如算法 1 所示. 其中，*setup*() 函数从 HDFS 中读取所有的模式三元组，并根据三元组所匹配的规则逐条进行处理，将其组织为不同的内存数据结构，方便之后各推理规则的执行；最后，因为规则 5 和规则 11 的推理只涉及模式数据，并且处于规则依赖图的底层，应该首先被执行，因此这两条规则的推理在模式数据装载到内存后便立即以批处理方式加以执行，在执行过程中重复的模式数据也会得到处理. 而其余涉及到实例数据的推理规则在 *map*() 函数中逐行

加以处理.

算法 1.　模式数据预处理.

输入：模式三元组集合 *Schema*

输出：模式三元组分类集合

```
setup()
FOR Schema.next()!=null
  triple=Schema.next();
  IF triple.predicate==rdfs:domain
    domainMap.add (triple.subject,triple.object);
  ELSE IF triple.predicate==rdfs:range
    rangeMap.add (triple.subject,triple.object);
  ELSE IF triple.predicate==rdfs:subPropertyOf
    subpropMap.add(triple.subject,triple.object);
  ELSE IF triple.predicate==rdfs:subClassOf
    subclassMap.add(triple.subject,triple.object)
  END IF
END FOR
subpropMap.transitiveClosure(); //执行规则 5 的推理
subclassMap.transitiveClosure(); //执行规则 11 的推理
```

5.3　Map 阶段的规则推理计算

Map 计算阶段(阶段 II),用来执行规则 2、3、7、9 的推理. Map 端的输入是从实例数据文件中读取出来的〈*spo-null*〉键值对. 如算法 2 所示：在 *map*() 函数中,对读入的实例三元组确定所匹配的规则,并根据规则依赖图触发新的推理直到到达规则依赖图的顶端,即可得到该实例三元组的全部推理结果. 同时 *map*()还对原数据进行过滤,以便在 Reduce 端进行重复数据的检测和消除.

匹配规则的确定是通过检查三元组的谓词(模式)实现的. 例如,如图 5 所示,RDF 图中有如下三元组{(*Jolin*, *masterDegreeFrom*, *University*0), (*masterDegreeFrom*, *rdfs*:*domain*, *Master*), (*Master*, *rdfs*:*subClassOf*, *Student*)},其中模式三元组(*masterDegreeFrom*, *rdfs*:*domain*, *Master*)和实例

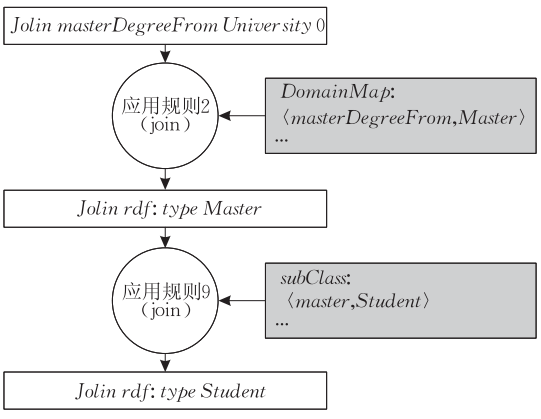

图 5　推理示例

三元组(*Jolin*, *masterDegreeFrom*, *University*0)与规则 2 匹配,因此首先要对其应用规则 2;规则 2 的输出(*Jolin*, *rdf*:*type*, *Master*)和模式三元组(*Master*, *rdfs*:*subClassOf*, *Student*)与规则 9 匹配,因此规则 2 会继续触发规则 9 的执行;规则 9 在规则依赖图中是最上层的规则,推理计算至此完成.

为了检测和消除重复数据,YARM 需要将输入的原始三元组和新推导出的三元组进行比较,所以 Map 端输出新三元组的同时,还需要将原始三元组传递到 Reduce 端. 为了尽可能减少传输不必要的静态数据,YARM 在 Map 阶段对原数据进行过滤. 通过对 RDFS 规则的输出进行分析,看到只有谓词为 *rdf*:*type* 以及谓词具有 *rdfs*:*subPropertyOf* 属性的实例三元组才存在与新推导出的三元组产生重复的可能性,因此 Map 只需过滤出这部分数据并将其传递给 Reduce 即可. Map 会输出两种类型的数据(原始的和新推导出的三元组),为了对两者进行区分以便后续的消重处理,YARM 使用(s, p, o)三元组作为键,使用标志作为值来识别三元组的类型.

算法 2.　Map 阶段的规则推理算法.

输入：*key* 为实例三元组；*value* 为空(null)

输出：*key* 为输出三元组；*value* 表示 *key* 是否为原实例三元组集合的三元组

```
map(key,value)
T={key}; //T 存放各步的推理结果,初始化为原始三元组
IF key.predicate=="rdf:type" or subprop.contains
                    (key.predicate)
    emit(key, original_flag); //原始数据的过滤
END IF
FOR (rule:RDFS rules)
    FOR (t:T)
      IF t.match(rule)
        apply rule to t;
        add derived triple to T;
      END IF
    END FOR
END FOR
T.delete(key);
FOR (t:T)
    emit(t,derived_flag);
END FOR
```

5.4　合并阶段的去重处理

合并阶段的任务是收集所有新推导出的三元组,同时对 RDFS 推理中存在的重复数据进行处理,该阶段使用 Reduce 和 Combine 来实现.

Combine 先对本地的重复数据进行处理,以减少数据传输量;Reduce 确保推理结果中没有重复推理结果.如算法 3 和 4 所示:因为同一键值的元组会被 Hadoop 自动组织在一起,所以 combine()和 reduce()只需要判断该元组对应的数据集中是否存在原始三元组,若存在则直接丢弃,否则对每个三元组输出唯一一份.

算法 3. Combiner 类.

输入:算法 2 中 *map* 函数的输出结果

输出:*key* 为推理结果三元组;*value* 表示输出的 *key* 是原始三元组还是新推理出的三元组

```
combine(key,valueList)
IF valueList.contains(original_flag)
    emit(key,original_flag);
ELSE
    emit(key,derived_flag);
END IF
```

算法 4. Reducer 类.

```
reduce(key,valueList)
```

输入:算法 2 和算法 3 的输出结果

输出:*key* 为推理结果三元组;*value* 为空

```
IF !valueList.contains(original_flag)
    emit(key,null);
END IF
```

6 实 验

6.1 实验环境与设置

我们从执行效率、数据扩展性和机器可扩展性三个方面分别评估 YARM 的性能.实验过程中,我们采用 reasoning-hadoop[6] 与 YARM 进行对比,据我们所知,reasoning-hadoop 是目前最快的并具有高可扩展性的大规模 RDF 数据推理引擎,通过与 reasoning-hadoop 做比较,可间接地取得与其他方法比较的结果.

我们采用的实验集群由 1 个主控制节点(JobTracker)和 16 个计算节点(TaskTracker)组成,集群的节点配置参见表 3.

表 3 计算节点配置信息

项目	配置说明
CPU	4 Core Intel Xeon 2.4GHz×2
Memory	24GB
Disk	2TB SAS×2
Network Bandwidth	1Gbps
OS	Red Hat Enterprise Linux Server 6.0
JVM Version	Java 1.6.0
Hadoop Version	Hadoop 1.0.3

实验所用数据集选用了一个合成的数据集 LUBM(Lehigh University Benchmark)[27] 大学语义数据,两个真实的数据集 WordNet[28] 电子词典语义数据以及 DBpedia 数据集[29].所有数据经编码压缩后以 Hadoop SequenceFile 文件格式存储在 HDFS 中,默认的分块大小为 64MB,均匀地分布在集群内的各计算节点上.

LUBM 数据集是当前公认的面向大规模本体应用环境的语义 Web 知识库系统测试基准.实验中利用拟合工具生成了 5 组规模的测试数据:LUBM-100、LUBM-250、LUBM-500、LUBM-750 和 LUBM-1000,分别含有 100、250、500、750 和 1000 个大学的语义数据,三元组规模分别为 13million、33million、66million、100million 和 133million.

WordNet 是一个真实的语义数据集,是美国普林斯顿大学认知科学实验室于 1985 年起开发的大型英文词汇数据库,在 WordNet 中词语按照语义来组织,共含有 1942887 条三元组.

DBpedia 是一个被广泛使用的 RDF 数据集,数据是从维基百科网页中抽取出的结构化信息,包含了众多领域的实体信息,被广泛用作 RDF 数据查询的一个标准数据集.实验中选用了 4 组不同大小的 DBpedia 数据集:DBpedia-1(35 million triples),DBpedia-2(85 million triples),DBpedia-3(140 million triples)和 DBpedia-4(210 million triples).

6.2 执行性能测试

我们对 10 组数据(5 组 LUBM 数据,1 组 WordNet 以及 4 组 DBpedia 数据)分别在 reasoning-hadoop 和 YARM 下做了 5 趟测试,最后取平均运行时间,实验结果如表 4 所示.

表 4 推理执行时间

数据集	Triple 条数/M	reasoning-hadoop/s	YARM/s	速度提升倍数
LUBM-100	13.0	630.066	46.547	13.5
LUBM-250	33.0	742.164	54.486	13.6
LUBM-500	66.0	834.114	58.512	14.2
LUBM-750	100.0	911.106	64.760	14.0
LUBM-1000	133.0	964.143	72.732	13.2
WordNet	1.9	373.548	70.718	5.3
DBpedia-1	35.0	146.651	22.548	6.5
DBpedia-2	85.0	383.295	33.639	11.4
DBpedia-3	140.0	662.131	40.642	16.3
DBpedia-4	210.0	734.424	43.646	16.8

由实验结果可见,与 reasoning-hadoop 相比,YARM 在执行时间上要快 10 倍左右.此外,虽然 WordNet 的数据集规模小于 LUBM-100,但是 YARM 在 WordNet 上的推理时间大于在 LUBM-

100 上的推理时间，这是因为 WordNet 的推理复杂性要远远大于 LUBM，实验结果说明影响 YARM 执行效率的是推理复杂性，影响 reasonging-hadoop 的因素则侧重于数据集规模. 因此在大规模 RDF 推理任务中，YARM 比 reasoning-hadoop 具有更好的性能.

6.3　数据可扩展性测试

为了观测数据规模增长时 YARM 实际运行时间的变化情况，并且尽量避免因采用复杂性不同的数据集对结果造成的影响，我们选用复杂性相同但大小不同的 5 组 LUBM 和 4 组 DBpedia 数据集进行测试. LUBM 的规模从 100 个大学增加到 1000 个，每个规模级别下分别作 5 趟测试，最后取平均运行时间.

DBpedia 选用了不同大小的 4 组数据：分别为 35 M 条记录、85 M 条记录、140 M 条记录和 210 M 条记录，每组数据下也分别做 5 趟测试，最后取平均运行时间. 实验结果见图 6 和图 7，图中纵坐标为执行时间，横坐标为数据规模. 可见，随着数据规模的增长，YARM 推理时间呈现出近似于线性增长的趋势，与 reasoning-hadoop 相比，YARM 具备了更理想的数据可扩展性.

6.4　系统可扩展性测试

为了观察集群规模增大时，YARM 的性能变化情况，我们还进行了系统可扩展性实验.

实验中采用了 LUBM-1000 数据集，集群计算节点规模从 1 个节点增长到 16 个节点. 实验结果数据处理后如图 8 所示. 图 8 显示了计算节点数目变化时 YARM 的运行性能变化曲线. 其中横坐标为集群内计算节点数目，纵坐标为执行时间. 由图中曲线可见，YARM 运行时间在小于 8 个结点时随集群内节点数目的增加呈接近于线性的下降，但超过 8 个节点后则下降趋于平缓并接近于一个下限，这个下限取决于一个数据块由单个处理器处理所需的计算时间加上整个算法并行化的额外开销.

另外，我们计算了算法的相对加速比和计算效率. 相对加速比即同一算法在单个处理器上运行的时间除以在多个处理器上运行的时间，计算效率是指加速比和处理器数目的比值. 我们发现随着集群中节点数目的增加，YARM 加速比快速上升. 与 reasoning-hadoop 相比，加速比上升得更快，到达极限的速度更慢. 本实验中加速比相对的执行时间基础是 YARM 在与集群内计算节点配置相同的单节点环境下，以 local 模式运行的 Hadoop 上的运行时间.

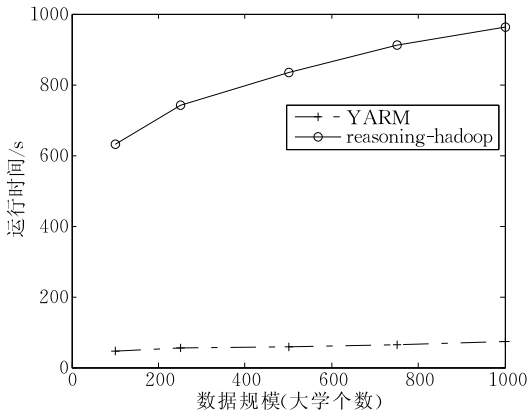

图 6　LUBM 数据规模变化时运行时间对比图

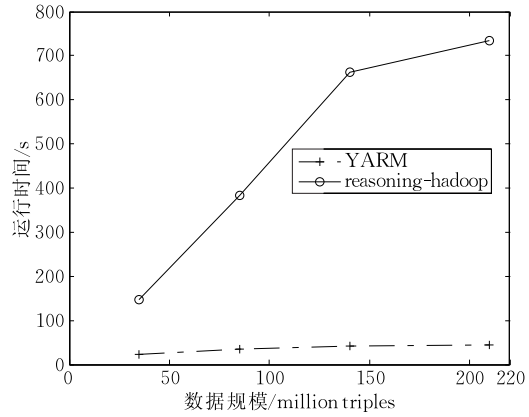

图 7　DBPedia 数据规模变化时运行时间对比图

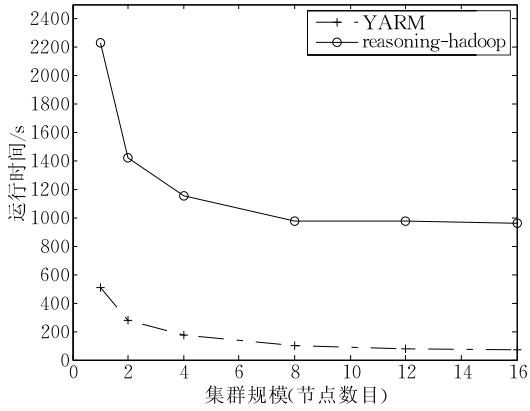

图 8　计算节点数目变化时运行时间对比图

计算效率定义为加速比和处理器数目的比值，计算效率反应了集群系统中计算资源的利用率的高低. 在计算量确定的前提下，随着并行计算系统规模的增长，计算效率会有所下降. 经计算发现，与 reasonong-hadoop 相比，YARM 的计算效率曲线下降的更平缓.

总的说来，从图 8 以及相应的计算结果可以发现，随着处理器数目的增长，YARM 运行时间的下降和加速比的上升都十分显著，并行执行的效果较

好.同时,随着处理器规模的增大,YARM 计算效率下降的趋势也是比较缓慢的,即算法有比较好的可扩展性.

7 总 结

为了实现大规模 RDF 数据的推理计算,本文设计和实现了一种基于 MapReduce 的高效并行化推理执行引擎 YARM.它将整个 RDF 图的推理分解成多个相互独立的更小规模的推理任务,增加了任务执行的并行度,使得 RDFS 推理在一次 MapReduce 作业之内即可完成.同时,对单个节点上的本地数据使用优化的推理执行机制.实验结果表明,YARM 的计算速度比当前最快的基于 MapReduce 的并行推理引擎(reasoning-hadoop)高出 10 倍左右,同时 YARM 也具备了良好的可扩展性.

在未来的工作中,我们将继续对 YARM 进行改进.我们将进一步研究高效的分布式 OWL 推理引擎.另外,我们还拟研究如何利用基于内存的分布式计算平台来设计和实现 RDFS 及 OWL 推理算法和系统.

致 谢 感谢审稿人提出的宝贵意见及王善永同学对论文终稿进行的格式校验!

参 考 文 献

[1] Tsatsanifos G, Sacharidis D, Sellis T. On enhancing scalability for distributed RDF/S stores//Proceedings of the 14th International Conference on Extending Database Technology. Uppsala, Sweden, 2011: 141-152

[2] Grigoris Antoniou, Frank van Harmelen. A Semantic Web Primer. United States of America: Library of Congress Cataloging-in-Publication Data, 2004

[3] Salvadores M, Correndo G, Harris S, et al. The design and implementation of minimal RDFS backward reasoning in 4store//Proceedings of the 8th Extended Semantic Web Conference. Heraklion, Greece, 2011: 139-153

[4] Li Pei-Qiang. The quest for parallel reasoning on the semantic web//Proceedings of the 5th International Conference on Active Media Technology. Beijing, China, 2009: 430-441

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

[6] Urbani J, Kotoulas S, Oren E, Harmelen F. Scalable distributed reasoning using MapReduce//Proceedings of the 8th International Semantic Web Conference. Washington, USA, 2009: 634-649

[7] Mika P, Tummarello G. Web Semantics in the clouds. IEEE Intelligent Systems, 2008, 23(5): 82-87

[8] Studer R, Benjamins V R, Fensel D. Knowledge engineering: Principles and methods. Data & Knowledge Engineering, 1998, 25(1-2): 161-197

[9] Wu Gang. Research on Key Technologies of RDF Graph Data Management [Ph. D. dissertation]. Tsinghua University, Beijing, 2008(in Chinese)
(吴刚. RDF 图数据管理的关键技术研究[博士学位论文]. 清华大学, 北京, 2008)

[10] Soma R, Prasanna V K. Parallel inferencing for owl knowledge bases//Proceedings of the 37th International Conference on Parallel Processing. Portland, USA, 2008: 75-82

[11] Carroll J J, Dickinson I, Dollin C, et al. Jena: Implementing the semantic web recommendations//Proceedings of the 13th international World Wide Web Conference on Alternate Track Papers & Posters. New York, USA, 2004: 74-83

[12] Sirin E, Parsia B, Grau B C, et al. Pellet: A practical OWL-DL reasoner. Journal of Web Semantics, 2007, 5(2): 51-53

[13] 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 on The Semantic Web. Sardinia, Italy, 2002: 54-68

[14] Zhou J, Ma L, Liu Q L, et al. Minerva: A scalable OWL ontology storage and inference system//Proceedings of the 1st Asian Semantic Web Conference. Beijing, China, 2006: 429-443

[15] Hogan A, Harth A, Polleres A. Scalable authoritative OWL reasoning for the web. International Journal on Semantic Web and Information Systems, 2009, 5(2): 49-90

[16] Kaoudi Z, Miliaraki I, Koubarakis M. RDFS: Reasoning and query answering on top of DHTs//Proceedings of the 7th International Semantic Web Conference. Karlsruhe, Germany, 2008: 499-516

[17] MacCartney B, McIlraith S, Amir E, Uribe T E. Practical partition-based theorem proving for large knowledge bases//Proceedings of the 18th International Joint Conference on Artificial Intelligence. Acapulco, Mexico, 2003: 89-96

[18] Weaver J, Hendler J A. Parallel materialization of the finite RDFS closure for hundreds of millions of triples//Proceedings of the 8th International Semantic Web Conference. Washington, USA, 2009: 682-697

[19] Oren E, Kotoulas S, Anadiotis G, et al. Marvin: Distributed reasoning over large-scale semantic Web data. Web Semantics: Science, Services and Agents on the World Wide Web, 2009, 7(4): 305-316

[20] Muhleisen H, Dentler K. Large-scale storage and reasoning for semantic data using swarms. IEEE Computational Intelligence Magazine, 2012, 7(2): 32-44

[21] Norman H, Jeff Z P. RDFS reasoning on massively parallel hardware//Proceedings of the 11th International on the Semantic Web. Boston, USA, 2012: 133-148

[22] Liu C, Qi G, Wang H, et al. Large scale fuzzy pd* reasoning using MapReduce//Proceedings of the Semantic Web—ISWC 2011. Bonn, Germany, 2011: 405-420

[23] Tachmazidis I, Antoniou G, Flouris G, et al. Scalable nonmonotonic reasoning over RDF data using MapReduce// Proceedings of the Joint Workshop on Scalable and High-Performance Semantic Web Systems. Boston, USA, 2012: 75-90

[24] Wu H, Liu J, Ye D, et al. A distributed rule execution mechanism based on MapReduce in sematic web reasoning// Proceedings of the 5th Asia-Pacific Symposium on Internetware. Changsha, China, 2013: Article No. 6

[25] Jang B, Ha Y G. Transitivity reasoning for RDF ontology with iterative MapReduce//Proceedings of the 7th International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS). Taichung, China, 2013: 232-237

[26] Maeda R, Ohta N, Kuwabara K. MapReduce-based implementation of a rule system//Amelia Badica, Bogdan Trawinski, Ngoc Thanh Nguyen eds. Recent Developments in Computational Collective Intelligence. Switzerland: Springer International Publishing, 2014: 197-206

[27] Guo Y, Pan Z, Heflin J. LUBM: A benchmark for OWL knowledge base systems. Journal of Web Semantics, 2005, 3(2-3): 158-182

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

[29] Auer S, Bizer C, Kobilarov G, et al. Dbpedia: A nucleus for a web of open data//Proceedings of the Semantic Web. Busan, Korea, 2007: 722-735

GU Rong, born in 1988, Ph. D. candidate. His research interests include big data parallel computation models, large scale machine learning and data mining.

WANG Fang-Fang, born in 1989, M. S. Her research interests include big data processing and cloud computing.

YUAN Chun-Feng, born in 1963, professor. Her research interests include computer architecture, parallel computing, multi-media processing and Web information retrieval and mining.

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 recent years, with the rapid development of Semantic Web, many RDF data collections are produced. The demand on efficiently processing large-scale Semantic Web data has brought many challenges to RDF data management systems, especially for reasoning tasks due to large amount of computation. Traditional reasoning engines only work for limited amount of semantic data and thus are hard to work well for web-scale semantic data.

MapReduce brings new dawn to the reasoning process of web-scale RDF data. Nevertheless, due to the essential challenges in the computing model, the matured reasoning algorithm used in traditional RDF systems can neither be directly applied to massive data, nor be migrated to MapReduce in an easy way. This paper focuses on the design and implementation of an efficient and scalable reasoning engine, called YARM, which adopts MapReduce as the parallel model to achieve scalability and high performance.

YARM can map the reasoning tasks into MapReduce framework smoothly and make best use of the cluster computing resource. YARM outperforms the state-of-art method on both real-world and synthetic datasets without losing scalability.

This work is supported by the Special Research of National Natural Science Foundation of China under Grant No. 61223003 and USA Intel Labs URO Funding.