**本科生毕业设计（论文）**

**文献综述**

姓 名： 李娟

学 号： 221200219

学 院： 数学与计算机科学学院

专 业： 软件工程

年 级： 2012

指导教师： （签名）

2016 年 05 月 01 日

基于MapReduce的网络级并行推理引擎

**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，现有的推理大量的语义网数据和它的快速增长带来了在执行效率和可伸缩推理上大量计算的挑战。基于MapReduce的并行化语义推理引擎，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。

在本文中，我们提出一个分布式技术进行实例化基于RDFS和OWL ter Horst的推理。我们将显示一个简单的实现是无效的和不规模。我们的技术通过一组结合的,显著提高性能的算法来解决分布式推理的挑战。我们实现了WebPIE(网络级推理引擎),我们将演示其性能64个节点的集群上。我们评估系统使用大量真实数据集(LLD,Bio2RDF LDSR)和LUBM合成基准,扩大到1000亿三元组。结果表明,我们的实现线性扩展,大大优于当前系统的最大数据大小和推理速度。

**关键词：语义网、MapReduce、分布式、推理、高效**

## 知识背景

## 1.1语义Web概述

万维网创始人Tim Berners-Lee于1998年首先提出了语义Web的概念，接着在2000年的XML 2000的会议上提出了语义Web的体系结构，2001年在《Scientific American》杂志上发表题为《The Semantic Web))的论文，该论文主要叙述了从传统无语义的万维网到包含语义的文档的进化过程。从此语义Web正式被人们所了解，更多的研究机构和学者开始进行语义Web相关的研究。语义Web是在万维网基础上，赋予语义信息到万维网的文档中，使其具有语义信息，从而便于被计算机所识别和理解，提高计算机对信息的自动解析和处理能力。与万维网是一个巨大的分布式超文本系统类似，语义Web致力于建立一个巨大的分布式知识系统。语义Web希望共享数据而不是文档，换句话说，语义Web提供一个通用的框架便于应用系统、企业之间的共享和重用数据。

类似于网络协议栈中的TCP/IP或OSI体系，语义Web也有相应的分层栈结构[]。语义Web系统架构经过不断的演化和改进，产生了如图1.1所示的语义Web系统架构图，下面基于最新的语义Web系统结构进行介绍，自下而上分别为字符编码和标记层(Unicode + URI )、XML数据结构层(XML )、资源描述层(RDF)、本体层(Ontology Vocabulary)、逻辑层(Logic )、证明层(Proof)和信任层(Trust)七层结构，各层之间相互联系，底层为上面的一层提供服务，网络资源从最底层出发，经过各层的功能处理后形成一个可信赖的富含语义信息的语义Web资源。



图1.1语义Web系统架构图

## 1.2RDF资源描述框架

RDF (Resource Description Framework，资源描述框架)是一种用来描述万维网上的元数据的一种语言，独立于任何语言，适用于任何领域，成为处理元数据的基础，促进网络资源的自动化处理。其数据模型是由一系列的三元组<主语，谓语，宾语>组成的图模型。在RDF数据模型中，包含如下三种对象类型:

(1)资源(Resource):所有能够使用RDF表示的对象都称为资源，包括网络上的信息、现实事物和虚拟概念等。资源使用IRI来唯一标识，不同的资源拥有不同的IRI。

(2)谓语(Property ):用于描述资源的特征或资源间的关系。每个谓语都有其具体意义，用于定义资源在属性上的属性值、描述谓语所属的资源形态以及其他谓语或资源之间的关系。谓语一般也用IRI唯一标识，且谓语也是一个资源。

(3)声明(Statement ):一条声明也称为一条三元组，三元组的形式为<主语，谓语，宾语>，其中主语一定是个被描述的资源，由IRI唯一标识。宾语表示主语在谓语上的取值，可以为一个资源、文本或为空。

根据定义RDF采用三元组(S,P,O)来描述Web资源的元数据，一组RDF三元组可以构成一个RDF有向图，如图1-1为W3C官方一个用于描述Eric Miller的RDF有向图的例子。通过图我们将直观地看到RDF是如何以三元组的形式描述数据。

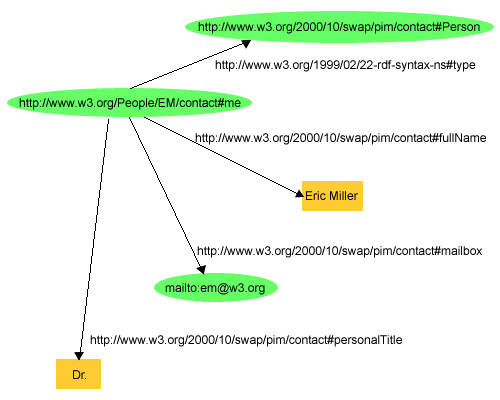


图1.2 一个描述Eric Miller的RDF图

其中椭圆表示一个资源，作为声明的主语或宾语，谓语用一条有向弧表示，有向弧由主语指向宾语。有向图的表示方法直观易懂，可是不容易被计算机所理解，声明还可以以“XML"、"N-TRIPLE"、"TURTLE”或“N3”格式的文件形式进行存储。

## 1.3 Hadoop概述

Hadoop起源于一个名为Apache Nutch的网络搜索引擎，受到Google实验室发表的关于MapReduce和Google File System论文的启发开发了相应的HadoopMapReduce和Hadoop Distributed File System，成为目前学术界和工业界事实上的海量数据并行处理标准。

Hadoop是面向海量数据处理的分布式软件框架，具有可扩展性、经济性、可靠性和高效性等优点[1]。用户编写Map阶段和Reduce阶段相关的业务代码即可，不必关心框架的底层处理细节，提高了软件开发的效率。Hadoop包含很多元素，包括底层的HDFS文件系统、MapReduce并行计算框架以及HBase数据库等。Hadoop的架构如图1.3所示。

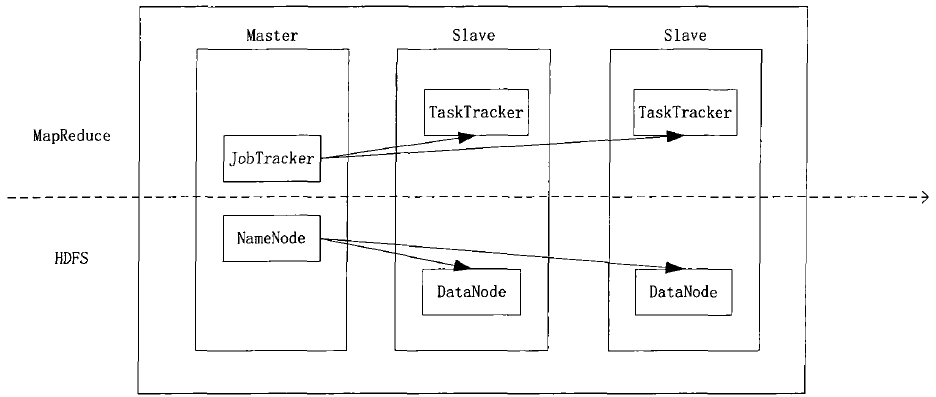


图1.3 Hadoop的架构如图

Hadoop相关的项目还有Zookeeper, Hbase和Pig等。Zookeeper是Hadoop的分布式协调服务，Hbase是一个构建在Hadoop上的分布式存储系统，而Pig是一种探索大规模数据集的脚本语言，提供了一套更强大的数据变换操作，包括在MapReduce中被忽视的连接操作。

(1) HDFS文件系统

Hadoop的HDFS是模仿Google GFS设计实现的，是一个构建在分布式节点文件系统上的一个逻辑上的文件系统，它将数据存储在物理上分布的每个节点上，但通过HDFS将整个数据形成一个逻辑上整体的文件。针对廉价硬件设计且能够存储海量数据，将数据保存到大量的分布式节点中。HDFS提供数据的高可靠性和容错能力，当单个或多个节点出现异常情况时，不会影响系统的整体运行，数据仍然可以使用。同时通过一定数量的数据备份存储保证数据的高可靠性和出错恢复能力。HDFS提供数据的快速访问能力，并提供良好的可扩展性，便于简单地加入更多服务器快速扩充系统容量，服务更多的客户端。与Google的GFS类似，HDFS是Hadoop的底层数据存储系统，同时保证数据尽可能的根据其本地局部性进行访问和计算。

(2) MapReduce并行计算框架

MapReduce借鉴Lisp函数式语言中的思想，使用Map和Reduce两个函数提供了高层的并行编程抽象模型。适用于相互间不具有计算依赖关系的大数据，将大任务分而治之，然后再将子任务的计算结果进行合并，计算出最终结果。MapReduce设计并提供了统一的计算框架，为程序员隐藏了绝大多数系统层面的处理细节，减轻程序员的开发负担，提高软件开发效率。MapReduce作业需要开发人员指定Map函数和Reduce函数，并指定相应的输入输出格式类型即可。Map函数根据一定的映射规则将输入的<key ，value>键值对转换成一个或一组<key ，value>输出，MapReduce框架将属于相同Key的数据划分到一个Reduce节点，Reduce节点的输入为<key, List<Value>>形式，对数据做一定的规约处理

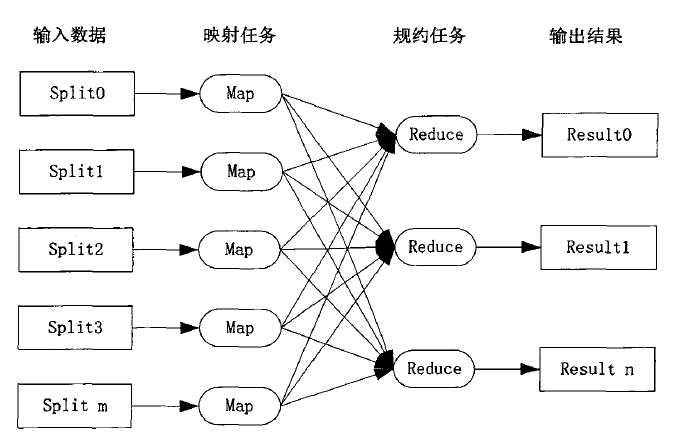
并输出最终结果。

图1.4 MapReduce计算模型图

(3) HBase数据库

HBase是一个分布式的、面向列的开源数据库，该技术来源于 Fay Chang 所撰写的Google论文“Bigtable：一个结构化数据的[分布式存储系统](http://baike.baidu.com/view/1911305.htm" \t "_blank)”。就像Bigtable利用了Google文件系统（File System）所提供的分布式数据存储一样，HBase在Hadoop之上提供了类似于Bigtable的能力。HBase是Apache的Hadoop项目的子项目。HBase不同于一般的关系数据库，它是一个适合于非结构化数据存储的数据库。HBase位于结构化[存储](http://baike.baidu.com/view/87682.htm" \t "_blank)层，Hadoop HDFS为HBase提供了高可靠性的底层存储支持，Hadoop MapReduce为HBase提供了高性能的计算能力，Zookeeper为HBase提供了稳定服务和failover机制，利用HBase技术可在廉价PC Server上搭建起大规模[结构](http://baike.baidu.com/view/160039.htm" \t "_blank)化[存储](http://baike.baidu.com/view/87682.htm" \t "_blank)集群。

与关系数据库相比，HBase具有如下特征:

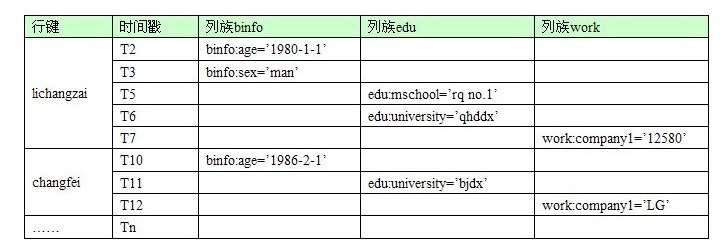
1.数据类型:关系数据库中提供多种数据类型供用户选择，而HBase只有简单的字节数组，用户在读写HBase的过程中需要把数据转换成字节数组。

2.存储模型:关系数据库是面向行存储的，而HBase是基于列式存储，每个列族作为一个文件单独存储。面向列存储的特性使HBase适用于非结构化数据的存储，避免了关系数据库存储稀疏数据造成的空间浪费情况。

3.数据操作:关系数据库的表之间有各种各样的关联操作，而HBase的表与表之间是分离的操作。关系数据库的表之间有各种各样的关联操作，而HBase的表没有表与表之间的关联操作，只有简单的插入、查询和删除操作。

4.数据更新:HBase数据库的每条信息都有一个指定的时间戳信息，当数据进行更新时，并没有像关系数据库一样更新原来的内容，而是新添加了一条数据，以最近时间戳的信息表示为最新的信息，这样也便于数据恢复，但耗费了更多的存储空间。

5.可伸缩性:HBase根据存储机器的数量改变在各个机器内存储文件的数目，大大提高了对硬件的容错性，而传统的关系数据库需要其他技术才能实现类似的功能。HBase中的表是以HTable形式存在的，如表2.1所示，一个HTable逻辑上由行和列组成，所有的列存储于某一个列族，而行和列的交叉点称为cell, cell是以时间戳而版本化的。每个表都有一个row key，且表按row key的字典序进行排序，row key和cell都是一段不可分割的字节数组。一个表可以有多个列族，而一个列族可以包含任意多个列，列族里的列以标签(qualifer)加以区分，列名由列族、“:”和标签连接而成。

表1.1 HBase逻辑表实例

## 2 课题的研究现状及意义

万维网促进了全球的信息共享，引起了人类史上的一场信息革命，全世界的人们通过这一媒介首次实现了全球广泛的相互交流。但是随着信息量的急剧增长，人们能够获取的信息越来越多，面对海量数据，信息检索的效率却越来越低，无法满足人们的快速、有效获取知识的需要。随着语义Web的发展，资源描述框架(RDF)得到了广泛的应用。但语义Web数据量的急剧增长，使RDF[1]数据的存储和检索面临严峻的挑战。幸运的是，Hadoop[2]平台的MapReduce[3]并行框架和分布式推理可以在大大提高时间效率并且有很好的系统可扩展性。

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。2009年初,语义网是估计包含44亿三元组。一年之后,语义网的大小扩大三倍到130亿三元组，和当前的趋势表明,这种增长速度并没有改变。2012年3月，LOD项目所收集的RDF数据集已经包含了超过325亿条RDF三元组。而可伸缩的推理是语义网的一个至关重要的问题。而分布式推理具有良好的可伸缩性。利用并行计算技术解决大规模RDF数据相关问题已经成为学术界和工业界的普遍共识。

当前有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。Hadoop是Google MapReduce框架的一个开源实现，其提供了类似于GoogleGPS的分布式数据存储系统HDFS以及类似于Google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。

目前基于MapReduce的并行推理的研究工作是将传统的推理技术直接迁移到MapRedece框架下，这种方法的效率低下。为了实现MapReduce下大规模RDF数据的高效推理。本文在RDFS和OWL Horst等相关技术分析的基础上，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，能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，解决传统推理算法的直接迁移带来的大量数据移动问题。同时，优化规则的执行次序，避免了迭代计算，提高了推理过程的执行效率。

## 3 基于MapReduce的推理算法

## 3.1 RDFS推理

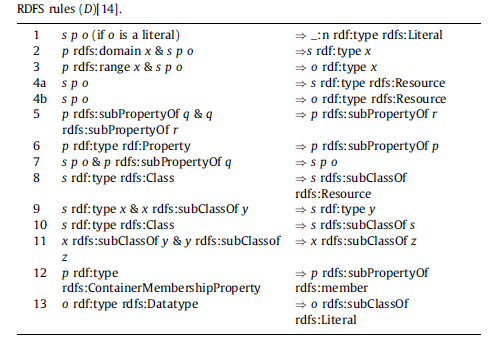


表3.1RDFS规则集

## 3.1.1算法原理1：在内存中加载模式三元组

由于模式三元组远远少于实例三元组，观察表3.1可知，先行词中至多只有一个实例三元组。这意味着所有的RDFS之间连接两个模式三元组或一个模式的三元组和一个实例的三元组。因此我们可以加载模式三元组在内存中，执行连接时，以流媒体的方式输入三元组。

在内存中加载模式三元组的优点：(i)三元组连接时不需要重新分配数据,意味着大大减少了数据传输;(2)同时，可以实现负载平衡,因为可以从任何节点输入来处理任何三元组;(3)在几乎所有的情况下，可以在避免迭代

## 3.1.2算法原理2：数据预处理

为了节省数据导入时间和存储空间，对三元组进行编码压缩，并采用Hadoop SequenceFile格式进行存储。数据预处理后，大大提高算法的整体性能。

## 3.1.3算法原理3：整理RDFS的应用规则

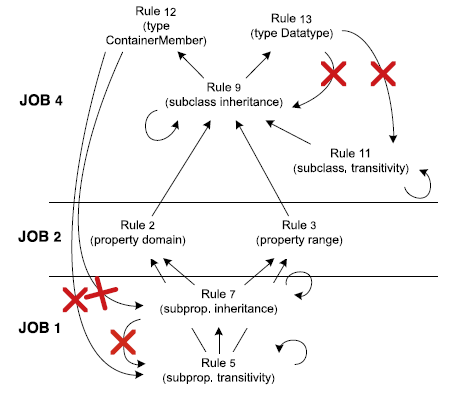


图3.2RDFS规则集的关系

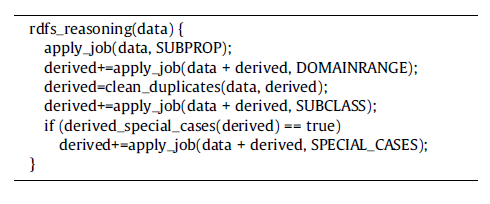
根据RDFS规则集的数据依赖关系，我们可以整理规则，限制迭代的次数来达到算法的完全关闭。

基于规则集的输出：

* 规则5和12谓语为rdfs:subPropertyOf;
* 规则11和13谓语为rdfs:subClassOf;
* 规则2、3和9谓语为rdf:type;
* 规则7可能产生任意的三元组。

由以上总结的规则可知，RDFS规则的输出遵循固定的模式。因此我们根据规则的输出来定义规则的执行顺序如图3.2所示。

由以上三个算法原理，我们推出了RDFS算法：



SUBPROP适用规则5和7；

DOMAINRANGE适用规则2和3；

SUBCLASS使用于规则9；

**SUBPROP的分布式算法：**

map(key, value):

if (subproperties.contains(value.predicate) && // for rule 7

max(value.timestamp,

subproperties.get(value.predicate).timestamp) >

last\_execution\_timestamp)

key = ’’1’’ + value.subject + ’’-’’ + value.object

emit(key, value.predicate)

if (subproperties.contains(value.object) && // for rule 5

value.predicate == ’’rdfs:subPropertyOf’’ &&

max(value.timestamp,

subproperties.get(value.predicate).timestamp) >

last\_execution\_timestamp)

key = ’’2’’ + value.subject

emit(key, value.object)

reduce(key, values):

values = values.unique // filter duplicate values

switch (key[0])

case 1: // we are doing rule 7: subproperty inheritance

for (predicate in values)

// iterate over the predicates emitted in the map and

collect superproperties

superproperties.add(subproperties.recursive\_

get(value))

for (superproperty in superproperties)

// iterate over superproperties and emit instance triples

emit(null, triple(key.subject, superproperty, key.object)

case 2: // we are doing rule 5: subproperty transitivity

for (predicate in values)

// iterate over the predicates emitted in the map, and

collect superproperties

superproperties.add(subproperties.recursive\_

get(value))

for (superproperty in superproperties)

// emit transitive subproperties

emit(null, triple(key.subject, ’’rdfs:subPropertyOf’’,

superproperty))

**DOMAINRANGE的分布式算法：**

map(key, value):

if (domains.contains(value.predicate)) then // for rule 2

key = value.subject

emit(key, value.predicate + ’’d’’)

if (ranges.contains(value.predicate)) then // for rule 3

key = value.object

emit(key, value.predicate +’’r’’)

reduce(key, values):

values = values.unique // filter duplicate values

for (predicate in values)

switch (predicate.flag)

case ’’r’’: // rule 3: find the range for this predicate

types.add(ranges.get(predicate))

case ’’d’’: // rule 2: find the domain for this predicate

types.add(domains.get(predicate))

for (type in types)

emit(null, triple(key, ’’rdf:type’’, type))

**SUBCLASS的分布式算法：**

map(key, value):

if (value.predicate = ’’rdf:type’’)

key = ’’0’’ + value.predicate

emit(key, value.object)

if (value.predicate = ’’rdfs:subClassOf’’)

key = ’’1’’ + value.predicate

emit(key, value.object)

reduce(key, values):

values = values.unique // filter duplicate values

for (class in values)

superclasses.add(subclasses.get\_recursively(class))

switch (key[0])

case 0: // we are doing rdf:type

for (class in superclasses)

if !values.contains(class)

emit(null, triple(key.subject, ’’rdf:type’’, class))

case 1: // we’re doing subClassOf

for (class in superclasses)

if !values.contains(class)

emit(null, triple(key.subject, ’’rdfs:subClassOf’’,

class))

## 3.1.3算法原理4：优化RDFS的应用规则

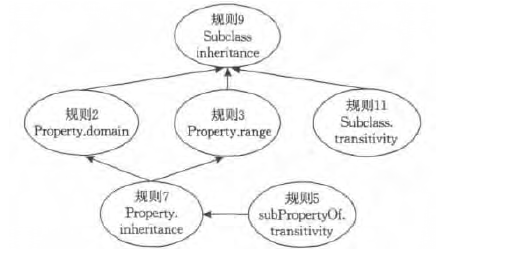
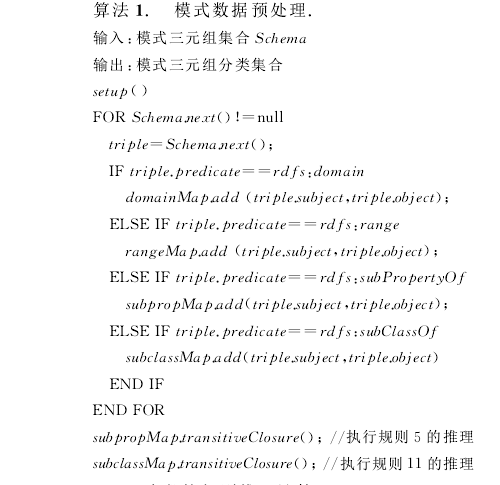
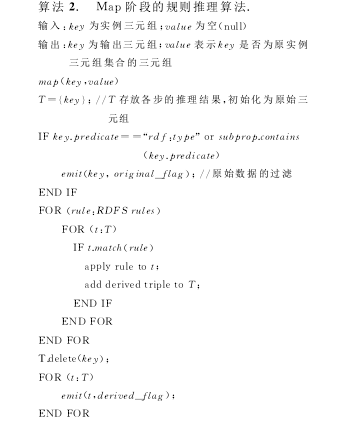
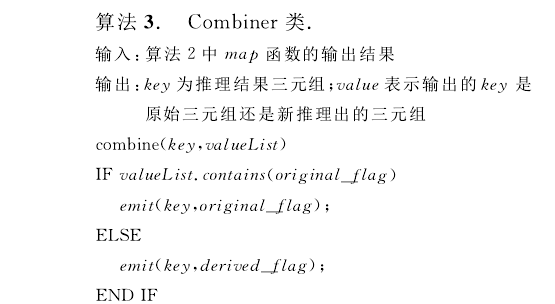
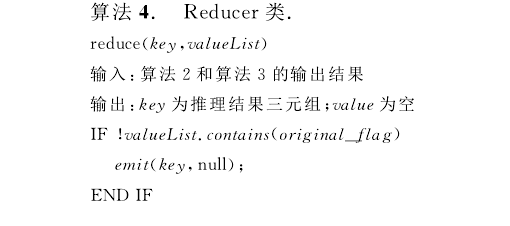


图3.3新规则依赖图









## 3.2 OWL推理

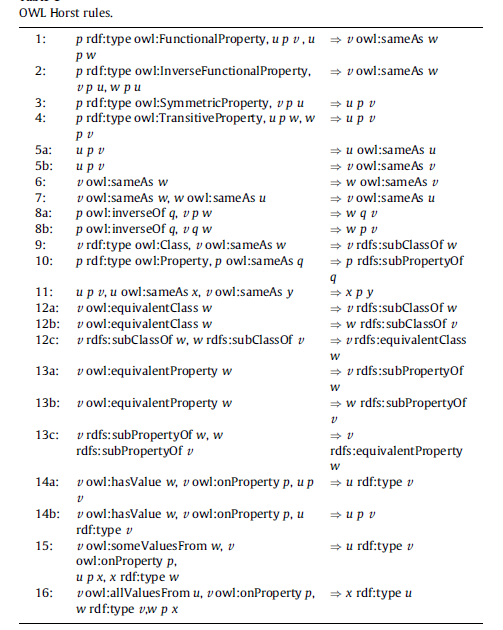
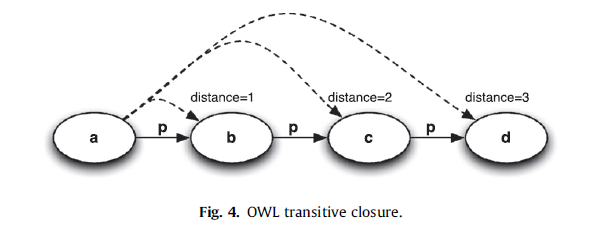


图3.4 OWL规则集

由图3.4可知，OWL规则的先行词中包含多个实例三元组，需要执行多次的连接操作。由此RDFS规则集的所有算法优化均都失效。必须重新构造新的算法原理。

## 3.2.1算法原理1：限制重复当执行实例三元组之间的连接



规则4可知连接中有多个重复的资源。限制重复，用规则4以一种简单的方式将导致大量的重复,因为每次应用的规则是,相同的关系将被推断。传递属性链的长度n,一个简单的实现将生成O(n^3)副本时最大输出只包含O(n^2)独特的一双。

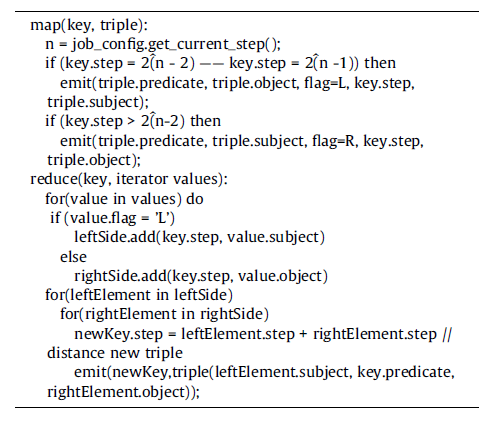
我们可以解决这个问题如果我们限制如何允许结合三元组。举例来说,考虑以下三个三元组:a p b,b p c和c p d,a,b和c是通用的资源和p是一个传递属性。图4图形化地给出相应的RDF图。应用规则4由计算传递闭包的链,即显式连接a与c和d和b和d连接。

我们定义两个后续资源传递链之间的距离随着大量的hops从第一个走到第二个。a和b的术语在我们的示例中有距离一个而条款a和c有距离两个,因为它们是连接通过b。当我们执行n MapReduce工作我们不希望获得三元组的一个图表距离小于或等于2^（n-2）,因为这些已经推导出在前面的执行。我们也想推导出新的三元组只有一次,而不是不同的组合。这条件以确保这是:

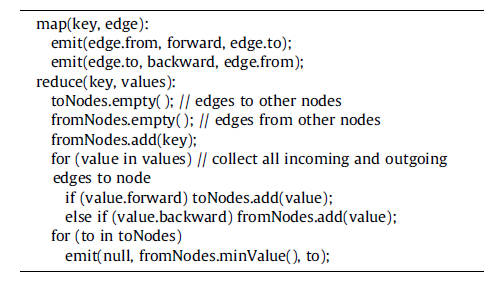
左边的连接(三元组的关键是对象)我们只允许三元组距离2^（n-2）和距离2^（n-2）

右边的连接(三元组的关键是主题)我们只允许与三元组距离大于2^（n-2）。

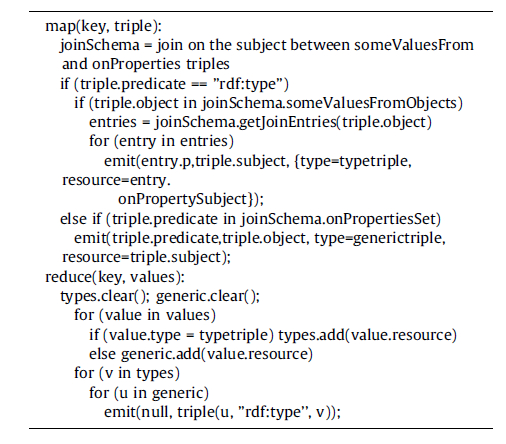
**算法原理实现如下：**



## 3.2.2算法原理2：构建sameAs表



3.2.3算法原理3：冗余连接,以避免执行负载平衡问题



## 4 结论

在本文中,我们展示了一个大规模可伸缩技术,并行OWL Horst特向前推理和证明推理1000亿三元组。

为了提高性能,引入了大量的关键优化处理一组特定的规则。这些优化:

——在内存中加载模式的三元组,在可能的情况下,执行加入动态而不是减少的阶段。

——在减少阶段执行连接和使用map函数组三元组,以避免重复。

——在特定的顺序执行RDFS规则MapReduce的岗位数量降到最低。

——限制重复执行实例三元组之间的连接时使用的上下文信息。

——限制OWL的指数推导:不同三元组建不同的表。

——执行冗余连接,以避免负载平衡问题。

此外,我们已经实现机制应用规则添加了不同的策略和支持对增量数据进行推理。

在这项工作中,我们实现了一个使用MapReduce编程范例推理方法。原则上,我们已经开发出的优化并不是MapReduce-specific可以应用在任何基础设施类似的工作。最初,MapReduce运行在两个离散阶段:地图阶段,数据分区和减少阶段,数据处理在一个给定的分区。在我们的一些优化,我们偏离这个模型通过执行连接在映射阶段(有效跳过划分阶段和操作任意部分的输入和输入的一部分,复制所有节点)。

技术提出了优化OWL-Horst规则。未来的工作在于推理在用户提供的规则集,系统会选择正确的实现对于每个规则和最有效的执行顺序,根据输入。此外,正如所有可伸缩三店,我们的方法不能有效地处理分布式数据。未来的工作应该扩展我们的技术来处理数据流从远程位置。

事实上,我们认为,建立计算可用集中一个非常大的数据集的闭包不再是一个重要的瓶颈,而且研究成果应该切换到其他的推理模式。查询驱动逆向链接推理在分布式数据集可能会更有前途比详尽的正向推理集中式存储。

## 5 参考文献

[1] Bio2RDF, <http://bio2rdf.org>, 2010.

[2] FactForge, <http://www.factforge.com>, 2010.

[3] Hadoop, <http://hadoop.apache.org>, 2010.

[4] Large triple stores wiki pagelarge triple stores wiki page, <http://esw.w3.org/topic/LargeTripleStores>, 2010.

[5] LarKC deliverable 5.2.2, <http://hadoop.apache.org>, 2010.

[6] Linked Data Semantic Repository (LDSR), <http://www.ontotext.com/ldsr/>,2010.

[7] Linked Life Data (LLD), <http://linkedlifedata.com>, 2010.

[8] WebPIE website, <http://www.cs.vu.nl/webpie>, 2010.

[9] D. Battré, A.H鰅ng, F. Heine, O. Kao. On triple dissemination, forward-chaining,and load balancing in DHT based RDF stores, in: Proceedings of the VLDB Workshop on Databases, Information Systems and Peer-to-Peer Computing (DBISP2P). 2006.

[10] M. Cai, M. Frank, RDFPeers: a scalable distributed RDF repository based on a structured peer-to-peer network, in: Proceedings of the International WorldWide Web Conference. 2004.

[11] J. Dean, S. Ghemawat, Mapreduce: simplified data processing on large clusters, in: Proceedings of the USENIX Symposium on Operating Systems Design & Implementation (OSDI), 2004, pp. 137–147.

[12] Q. Fang, Y. Zhao, G. Yang, W. Zheng, Scalable distributed ontology reasoning using DHT-based partitioning, in: Proceedings of the Asian Semantic Web Conference (ASWC), 2008.

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

[14] P. Hayes (Ed.), RDF Semantics, W3C Recommendation, 2004.

[15] A. Hogan, A. Harth, A. Polleres, Scalable authoritative OWL reasoning for the

web, International Journal on Semantic Web and Information Systems 5 (2) (2009).

[16] A. Hogan, J. Pan, A. Polleres, S. Decker, Saor: Template rule optimisations for distributed reasoning over 1 billion linked data triples, The Semantic Web -ISWC 2010,2010,pp.337–353.

[17] A. Hogan, A. Polleres, A. Harth, Saor: authoritative reasoning for the web, in: Proceedings of the Asian Semantic Web Conference (ASWC), 2008.

[18] H.J. ter Horst, Completeness, decidability and complexity of entailment for RDF schema and a semantic extension involving the OWL vocabulary, Journal of Web Semantics 3 (2–3) (2005) 79–115.

[19] M.F. Husain, P. Doshi, L. Khan, B. Thuraisingham, Storage and retrieval of large rdf graph using hadoop and mapreduce, in: M.G. Jaatun, G. Zhao, C. Rong (Eds.), Cloud Computing, vol. 5931, Springer, Berlin, Heidelberg, 2009, pp. 680– 686 (Chapter 72).

[20] Z. Kaoudi, I. Miliaraki, M. Koubarakis, RDFS reasoning and query answering on top of DHTs, in: Proceedings of the International Semantic Web Conference (ISWC), 2008.

[21] V. Kolovski, Z. Wu, G. Eadon, Optimizing enterprise-scale OWL 2 RL reasoning in a relational database system, in: The Semantic Web – ISWC 2010, 2010, pp. 436–452.

[22] S. Kotoulas, E. Oren, F. van Harmelen, Mind the data skew: distributed inferencing by speeddating in elastic regions, in: Proceedings of the WWW, 2010.

[23] R. Mutharaju, F. Maier, P. Hitzler, A mapreduce algorithm for el+, in: Proceedings of the 23rd International Workshop on Description Logics (DL2010), Waterloo, Canada, 2010.

[24] A. Newman, Y. Li, J. Hunter, Scalable semantics the silver lining of cloud computing, in: Proceedings of the 4th IEEE International Conference on eScience, 2008.

[25] E. Oren, S. Kotoulas, G. Anadiotis, R. Siebes, et al., Marvin: distributed reasoning over large-scale semantic web data, Journal of Web Semantics 7 (4) (2009) 305–316.

[26] A. Schlicht, H. Stuckenschmidt, Peer-to-peer reasoning for interlinked ontologies, International Journal of Semantic Computing (2010) (Special Issue on Web Scale Reasoning).

[27] R. Soma, V. Prasanna, Parallel inferencing for OWL knowledge bases, in: International Conference on Parallel Processing, 2008, pp. 75–82.

[28] P. Stutz, A. Bernstein, W. Cohen, Signal/collect: Graph algorithms for the (semantic) web, in: Proceedings of the ISWC, Shanghai, China, 2010.

[29] J. Urbani, S. Kotoulas, J. Maassen, N. Drost, et al., Webpie: a web-scale parallel

inference engine, 2010 (1st prize at the 3rd IEEE SCALE challenge at CCGrid).

[30] J. Urbani, S. Kotoulas, J. Maassen, F. van Harmelen, et al., Owl reasoning with mapreduce: calculating the closure of 100 billion triples, in: Proceedings of the ESWC 2010.

[31] J. Urbani, S. Kotoulas, E. Oren, F. van Harmelen, Scalable distributed reasoning using mapreduce, in: Proceedings of the ISWC, 2009.

[32] J. Urbani, J. Maassen, H. Bal, Massive semantic web data compression with mapreduce, in: HPDC ’10: Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing, ACM, New York, NY, USA, 2010, pp. 795–802.

[33] J. Weaver, J. Hendler, Parallel materialization of the finite rdfs closure for hundreds of millions of triples, in: 8th International Semantic Web Conference (ISWC2009), 2009.