基于 Hadoop 的大规模语义 Web 本体数据查询与推理关键技术研究

重庆大学博士学位论文

学生姓名：李 韧

指导教师：杨 丹 教 授

副 导 师：傅 鹂 教授, 胡海波 副教授

专 业：计算机科学与技术

学科门类：工 学

重庆大学计算机学院

二 O 一三年四月

Research on Key Technologies of Large-Scaled Semantic Web Onotologies Querying and Reasoning based on Hadoop

A Thesis Submitted to Chongqing University

in Partial Fulfillment of the Requirement for the

Doctor's Degree of Philosophy

By

Ren Li

Supervised by:Prof. Dan Yang

Advised by:Prof. Li Fu and Ass. Prof. Haibo Hu

Specialty:Computer Science and Technology

College of Computer Science,

Chongqing University, Chongqing, China.

April, 2013

摘　　要

语义 Web 是 Tim Berners-Lee 提出的下一代互联网远景，通过引入了哲学领域本体的概念，使得计算机能够理解 Web 上的资源，并能实现计算机之间的语义信息共享。在世界万维网联盟(World Wide Web Consortium, W3C)提出的语义 Web 体系结构中，基于 SPARQL 的资源描述框架(Resource Description Framework, RDF)数据查询、基于描述逻辑的 Web 本体描述语言(Web Ontology Language, OWL)一致性检测推理和基于语义 Web 规则语言(Semantic Web Rule Language, SWRL)的 OWL 本体规则推理构成了语义 Web 领域的研究核心。

然而，随着语义 Web 技术的不断快速发展，本体数据已呈现出大规模性、高速增长性、多样性等大数据特性。然而，传统的本体数据查询与推理工具由于设计运行于单机环境下，不可避免地存在计算性能和可扩展性不足等问题，影响了语义 Web 技术的进一步推广应用。

近年来，云计算因其具备高性能、易扩展的海量数据存储和计算能力已经成为产业界和学术界在信息技术领域的最新研究方向之一，其中开源 Hadoop 云计算工具已成为当前大数据处理的事实标准。目前，国内外研究人员已开始将 Hadoop 关键技术引入语义 Web 研究领域，以探寻分布式环境下的高效率本体数据查询与推理方法，并已逐步形成了以语义 Web 和云计算技术相结合的新研究方向，但其研究仍然处于起步阶段，存在许多关键问题尚待解决。

本文通过结合云计算和语义 Web 理论和关键技术，研究基于 Hadoop 的本体数据查询与推理并行化方法，为实现面向大规模语义 Web 本体的数据管理云服务奠定理论研究基础。主要研究内容和创新性成果包括以下五个方面：

(1) 以 W3C 提出的语义 Web 体系结构为基础，结合云计算 Hadoop 关键技术特性，提出了一种大规模语义 Web 本体数据查询与推理云计算框架。首先，对该框架进行了功能层级划分，自底向上分别由物理层、存储层、数据层、逻辑层、接口层、网络层和应用层组成。然后，基于本体查询与推理理论，设计了核心的逻辑层由数据预处理器、数据适配器、查询与推理分析器、查询与推理计划生成器、MapReduce SPARQL 查询引擎、MapReduce SWRL 规则推理引擎和 MapReduce Tableau 推理引擎构成。该框架的提出为实现高性能、易扩展的语义 Web 数据管理云服务提供体系结构和数据交互流程支持和借鉴，为进一步研究其中的关键技术理论奠定基础。

(2) 基于语义 Web 中 RDF 三元组数据特性和基于描述逻辑的 OWL 本体描述语言形式化语义，结合 HBase 基于列的数据存储模式特性，提出了由三个 HBase

数据表 T_OS_P、T_PO_S 和 T_SP_O 构成的本体数据分布式存储策略，分析了在进行基于 MapReduce 的本体查询和推理任务时的数据检索机制，并通过与现有的数据存储策略进行对比和分析，论证了本文提出方法能够在本体数据存储空间开销和检索性能方面实现较好的平衡。

(3) 基于 SPARQL 语法和形式化语义，结合 MapReduce 键值对的计算特性，提出了 SPARQL 复杂组图模式在 MapReduce 环境下的分布式查询方法。首先提出了 SPARQL 复杂组图模式查询的相关解析模型定义。然后提出了基于 MapReduce 的 SPARQL 复杂组图模式查询任务生成算法，实现了查询任务数的优化，并以此为基础，提出了在 map 和 reduce 函数中的 SPARQL 复杂组图模式分布式查询算法。最后，通过使用语义 Web 研究领域广泛采用的 SP2Bench 本体测试数据集和标准测试语句，对提出方法与现有的 Jena、Sesame 和 RDF-3X 查询引擎进行了对比实验和可扩展性实验。实验结果表明，提出方法在面向大规模 RDF 数据的 SPARQL 复杂组图模式进行查询时，其计算性能和可扩展性均优于传统的单机环境下运行的查询引擎。

(4) 基于 OWL Lite 本体所对应的描述逻辑 SHIF 语义及其 Tableau 推理算法，结合 MapReduce 键值对的数据计算特性，提出了基于 MapReduce 的 OWL 本体一致性分布式检测推理方法。首先定义了 OWL 本体一致性检测的相关解析模型。然后提出了基于 MapReduce 的 OWL Lite 本体数据划分方法和分布式 Tableau 推理算法。最后通过使用 LUBM 本体测试数据集，对提出方法与现有 Pellet、RacerPro 和 HermiT 推理引擎进行了对比实验和可扩展性实验，证明了提出方法在进行大规模 OWL 本体的一致性检测推理时，在计算性能和可扩展性方面均优于传统单机环境下运行的描述逻辑推理引擎。

(5) 基于 SWRL 规则语法和形式化语义，结合 MapReduce 键值对的数据计算特性，提出了基于 MapReduce 的 SWRL 规则分布式推理方法。首先提出了 SWRL 规则推理的相关解析模型定义。然后提出了基于 MapReduce 的 SWRL 规则推理计划生成算法，实现了推理任务数的优化。其次，为保证推理的可判定性，提出了 DL-safe 限制下 SWRL 规则在 map 和 reduce 函数中的分布式推理算法。最后通过使用 LUBM 本体数据集和自定义 SWRL 测试规则，对提出方法与 Jess 和 Pellet 推理引擎进行了对比实验和可扩展性实验，证明了在处理大规模 OWL 本体的 SWRL 规则推理时，提出方法较传统规则推理引擎具备更好的计算性能和可扩展性。

关键词：语义 Web，云计算，本体，Hadoop，MapReduce

ABSTRACT

Semantic Web, which is proposed by Tim Berners-Lee, is the vision of next generation of Web. Through combining the concepts of ontology from philosophy into computer science domain, computers can understand the information published in the Semantic Web, and it is possible to exchange semantic information among computers. In the World Wide Web Consortium (W3C) proposed semantic web stack, SPARQL-based Resource Description Framework (RDF) data querying, description logic-based Web Ontology Language (OWL) reasoning, and Semantic Web Rule Language (SWRL)-based OWL ontologies rule reasoning are the core contents of semantic web research.

However, large-scaled ontologies have existed with the explosion of the semantic web technologies, and the amounts of it is rapidly growing ever year. Therefore, these conventional semantic web data querying and reasoning tools do not scale well for large amounts of ontologies because they are designed for use on a single-machine context.

Recent years, cloud computing has become one of the latest research area in both academe and IT industry because of its high-performance and scalability for storing and computing on large-scaled data. Nowadays, Hadoop technologies have become the de-facto standard of Big Data processing. Several researchers have started to combine cloud computing and semantic web technologies to explore high-performance ontology querying and reasoning solutions in the distributed computing context. However, this novel research area is still in the initial stage, lots of key problems need to be solved.

To overcome the drawbacks, this thesis researches on the approaches of distributed querying and reasoning for large-scaled ontology data by utilizing cloud computing technologies. This thesis can establish the theoretical research basis for implementing large-scaled semantic web ontology data management cloud services in the future. The main research contents and innovative results are listed as follows.

(1) Based on the W3C proposed semantic web stack, MapReduce distributed computing model and HBase distributed database technology, this thesis proposes a architecuture of large-scaled semantic web ontology data management cloud service. First, the author designs the architecuture according to querying and reasoning functions, the layers from bottom to up consists of physical layer, storage layer, data layer, logical layer, interface layer, network layer and application layer. And then, the author designs

the logical layer, which is the core component of proposed architecutre, to be consisted of data preprocessor, data adapter, querying and reasoning analyser, querying and reasoning plan generator, MapReduce SPARQL querying engine, MapReduce SWRL rule reasoning engine and MapReduce Tebleau reasoning engine. The proposed framework can provide a completed architecture and data exchange workflow to implement high-performance and scalable ontology data management cloud service in the future, and it can establish the basis for the key technologies researching.

(2) Based on the features of RDF triple and the formalized semantics of description logic-based OWL ontologies, this thesis proposes a novel data storage solution for large-scaled semantic web ontologies according to the HBase distributed database schema. The ontologies are designed to store in three HBase tables named T_OS_P, T_PO_S and T_SP_O, respectively. The MapReduce-based querying and reasoning approach is analysed as well. Through comparing with the existing ontology storage schema, this thesis prove that the proposed schema can achieve the balance of the data storage space and performance.

(3) Based on the syntax and semantics of SPARQL and the features of MapReduce key-value pairs, this thesis proposes a novel MapReduce-based SPARQL Graph Patterns distributed querying approach for large-scaled RDF data. First, the author defines several data models to represent RDF and SPARQL queries. Second, to reduce the number of MapReduce jobs and optimize the performance, a query plan generation algorithm is proposed to determine jobs based on a greedy selection strategy. Furthermore, several query algorithms are also presented to answer SPARQL Graph Pattern queries in MapReduce paradigm. An experiment on a simulation cloud computing environment shows that our approach is more scalable and efficient than traditional approaches when storing and retrieving large volumes of RDF data.

(4) Based on the semantics of OWL Lite ontologies and the Tableau algorithm of description logic SHIF, this thesis proposes a novel MapReduce-based distributed Tableau reasoning approach to check the consistency of large OWL ontologies. First, by exploiting MapReduce paradigm, OWL individual assertions are first partitioned into multiple independent units with the form of key-value pair, and then the consistency of each unit with respect to the OWL terminologies is checked in parallel. Last, through using LUBM benchmark and comparing with Pellet, RacerPro and HermiT reasoners, an experiment on a simulation cloud computing environment shows that our approach is more scalable and efficient than traditional tools when reasoning over large-scaled

OWL ontologies.

(5) Based on the syntax and semantics of SWRL rules, this thesis proposes a novel MapReduce-based SWRL distributed reasoning approach. First, some novel data models for representing SWRL rules and intermediate key-value data are defined. Second, a MapReduce paradigm based distributed SWRL reasoning algorithm is proposed under DL-safe restriction. Last, through using LUBM benchmark and self-defined SWRL rules, an experiment on a simulation environment shows our approach is more efficient and scalable than conventional rule engines Jess and Pellet when reasoning over large-scale of OWL data.

Keywords: Semantic Web, Cloud Computing, Ontology, Hadoop, MapReduce

目　录

图表目录

1 绪 论

1.1 研究背景

近年来，以 Internet 技术为基础的万维网(World Wide Web)已成为人们日常工作、生活娱乐和获取信息不可或缺的重要途径之一。然而，传统的 Web 建立在超文本标记语言(Hyper-Text Markup Language, HTML)基础之上，信息以面向人类的沟通和理解而发布。尽管多年来计算机自然语言处理技术已得到一定的发展，但 Web 中大量信息仍然很难被计算机所理解，也无法实现对已有信息的自动化推理验证和隐含知识自动化发掘。例如，现有的 Google、Yahoo!和百度等搜索引擎均以基于文本关键字的方式对 Web 信息进行搜集和检索，尚无法理解 Web 信息的语义，不能实现基于语义的知识共享。为解决上述问题，万维网之父 Tim Berners-Lee 在现有 Web 标准和技术基础之上，通过结合传统哲学领域中的本体概念和人工智能领域中描述逻辑理论，在 2001 年首次提出了语义 Web (Semantic Web)作为实现下一代互联网的远景[1]。

语义 Web 采用本体的概念对某一领域的共有知识进行描述，并使用基于描述逻辑的自动化推理技术实现隐含知识的发现和检测[2]。近年来，世界万维网联盟(World Wide Web Consortium, W3C)相继发布了用于描述语义 Web 资源之间语义关系的资源描述框架(Resource Description Framework, RDF)[3]和 RDF 词汇描述语言(RDF-Schema, RDF-S)[4]。但由于 RDF 和 RDF-S 存在表达能力上的不足，例如不能自定义新的类，也缺乏表达能力来描述属性具有的传递性或对称性等，W3C 在 RDF 和 RDF-S 的研究基础之上，于 2004 年首次提出了基于描述逻辑[5]的 Web 本体描述语言(Web Ontology Language, OWL)[6]，将其作为语义 Web 中的标准本体描述语言。根据表达能力和推理复杂度的不同[7]，OWL 分为了 OWL Lite、OWL DL 和 OWL Full 三种子语言[8]。2012 年，W3C 提出了新一版本的 OWL 2[9]，其包含了对应于描述逻辑 SROIQ 的 OWL 2 DL，以及 OWL 2 RL 、OWL 2 EL 和 OWL 2 QL 子语言来应对不同应用领域需求的本体建模[10]与推理[11]。为实现语义 Web 本体数据查询检索，研究人员为 RDF 数据设计了 OWL-QL[12]、RQL[13]、RDQL[14]和 SPARQL[15]等语言。其中，SPARQL 被 W3C 推荐为 RDF 数据查询语言标准。另外，由于 OWL 无法对例如“如果，那么”等一般形式的规则进行描述，W3C 将 OWL 子语言 OWL DL、OWL Lite 与 RuleML 相结合，提出了具备更强逻辑表达能力的语义 Web 规则描述语言(Semantic Web Rule Language, SWRL)[16]，但其推理不可判定[17]。在语义 Web 标准体系结构中，RDF 和 RDF-S 构成了数据层，OWL 为本体层，SPARQL 和 SWRL 为语义 Web 本体的查询和规则推理标准[18]，并且上

述五种推荐标准构成了语义 Web 数据管理研究的核心问题，是能否成功地推广应用语义 Web 的关键。

目前，语义 Web 领域研究人员已研发出多个本体查询和推理工具。如基于 SPARQL 语法标准和形式化语义[19]设计研发的 SPARQL 查询工具：Jena[20]、Sesame[21]和 RDF-3X[22]等。在 OWL 本体的自动化推理验证方面，当前主要使用基于描述逻辑推理工具[23]进行本体的概念可满足性推理和一致性检测，主要包括了 Fact++[24]、Racer[25]、Pellet[26]和 Hermit[27]等，它们均基于描述逻辑 Tableau 算法[28]并进行了大量的优化，提供了对本体公理和断言完备的推理服务。近年来，虽然学术界也出现了一些基于定理证明器的一阶逻辑 OWL 推理机，如基于 Vampire[29]的 Hoolet、基于 Otter 的 Surnia 等，但这些工具对于复杂的本体在推理效率方面仍不如专用的描述逻辑推理机[30]。在面向 OWL 本体的 SWRL 规则推理方面，目前产业界和学术界还没有一个专用的 SWRL 规则推理引擎，研究人员主要使用 protégé 工具[31]的 SWRLTab 插件将 OWL 本体映射入 Jess 事实库，将 SWRL 规则映射入 Jess 规则库，并调用 Jess 规则引擎对已知 OWL 个体进行可判定的 SWRL 规则推理[32]。同时，Pellet 和 KAON2[33]等本体推理引擎也提供了在 DL-safe 限制[34]下的可判定 SWRL 推理支持。

随着语义 Web 技术的快速发展，目前 Web 中已存在大规模的本体数据，并且该数量正在逐年快速增长。根据 Link Open Data 的统计，2009 年语义 Web 中包含 RDF 三元组数目约为 44 亿条，2010 年该数目增长至 130 亿条，而 2011 年 9 月统计的 RDF 三元组总数已超过 310 亿条[35]。然而，传统的语义 Web 查询与推理工具均设计在单机环境下运行，当在对大规模本体数据进行计算处理时，会不可避免地出现内存空间和计算性能上的瓶颈，影响了语义 Web 技术的进一步发展和应用。此外，传统的关系型数据库因其需要预先设定数据表的存储模式，已很难适用于具有描述动态演变特性的语义 Web 本体存储和计算任务需求。因此，本体数据呈现出的**规模性、高速增长性、多样性**等大数据(Big Data)[36][37]特性，使得如何实现高性能、可扩展的大规模本体存储、查询与推理成为语义 Web 研究领域最亟需解决的问题之一[38]。

自 2007 年 Google 和 IBM 首次提出云计算[39]概念以来，云计算已经成为产业界和学术界在信息技术领域的最新研究方向之一[40], [41], [42]。目前，基于 Google File System (GFS)分布式文件系统模型[43]、MapReduce 分布式计算模型[44]和 BigTable 分布式数据库模型[45]实现的开源 Hadoop 平台[46]因具备的高性能、可扩展的海量数据存储和计算能力，及高容错、支持异构环境、较低使用成本等技术特性，已成为云计算研究领域中最为广泛使用的数据密集型计算和存储模型[47]，某种程度上可以认为 **Hadoop 已经成为大数据处理事实上的标准**[36]。

1.2 研究问题和目标

目前，语义 Web 本体存在的大数据特性已受到了学术界的广泛关注[48]。为实现高性能、可扩展的本体数据存储、查询与推理，研究人员已开始将具备大数据处理能力的云计算 Hadoop 技术引入语义 Web 研究领域，并逐步形成了以语义 Web 和云计算相结合的新研究方向[49]。例如，自 2008 年开始，研究人员组织了面向大规模本体数据查询与推理研究的 Semantic Web Challenge，以共享该方向最新的研究成果。同时，为验证本体数据查询和推理工具的计算性能，学术界已提出了如 LUBM[50]、SP2Bench[51]和 BSBM[52]等的标准测试数据集。

如何对传统的本体查询与推理算法作出并行化的改进，使其适应云计算的分布式计算框架并具备可扩展性，成为了该研究方向需要解决的关键科学问题。本文的研究工作也是以此为背景，而拟定的研究计划。

近年来，虽然已有研究人员结合 Hadoop 技术提出了一些语义 Web 本体数据查询与推理方法，并证明了其在计算性能和可扩展性方面优于传统单机环境下的工具，但该方向的研究仍然处于起步阶段，存在不少的关键问题需要进一步研究，具体体现在以下几个方面：

(1) 尚未提出一种大规模本体数据查询与推理云计算体系结构

目前，已有研究成果仅从方法论层面去探寻云计算环境下的某一特定本体查询或推理并行化方法，而尚未涉及一种本体数据管理云服务体系结构研究，也尚未提出体系结构中涉及的功能层级划分和核心计算模块之间的交互关系，进而使得本领域的研究过程中缺少一种体系结构标准，也不利于理论研究成果向云计算服务的实际产业化应用转化。

(2) 云计算环境下大规模本体数据的存储策略有待改进

现阶段，语义 Web 研究领域多采用文件系统或传统关系型数据库作为本体数据的存储载体。其中，基于文件系统的存储方式很难实现对大规模本体数据的快速随机访问和修改；基于关系型数据库的存储策略因其需要在设计阶段对数据库表结构进行设定，因而很难适应本体在对特定领域资源进行描述时结构动态、多变的特性。另外，在云计算与语义 Web 相结合的研究领域，虽然有研究人员提出了基于 HDFS 分布式文件系统和 HBase 分布式数据库的本体存储策略，但已有方法仍存在存储空间开销和查询性能之间的平衡不足等问题。

(3) 云计算环境下的大规模 RDF 数据查询方法研究存在不足

目前，本领域的已有研究成果仅局限于大规模 RDF 数据的 SPARQL 基本图模式查询，而未涉及 SPARQL 可选图模式 OPTIONAL 和多图模式 UNION 运算符在 MapReduce 模型下的计算方法研究，SPARQL 复杂组图模式在 MapReduce 模型下的并行化方法也尚未开展研究。

(4) 云计算环境下的大规模 OWL 本体一致性检测方法研究存在不足

在面向大规模 OWL 本体的推理问题研究方面，虽然研究人员已提出了基于 MapReduce 的 RDF-S 规则、OWL Horst 规则和 OWL 2 EL 本体分类的并行化推理方法，但学术界尚未对描述逻辑 Tableau 算法在 MapReduce 模型下并行化推理方法开展研究，进而无法实现大规模 OWL Lite、OWL DL 等基于描述逻辑语言的本体可满足性、一致性检测。

(5) 云计算环境下的 SWRL 规则分布式推理方法研究存在不足

目前，语义 Web 领域对 SWRL 规则推理工具的研究相对滞后，且目前尚不存在一种专用的 SWRL 标准测试规则集，而在面向大规模 OWL 本体数据的 SWRL 规则推理问题研究方面，也没有关于一种有效的解决方案。

本论文以改进上述研究工作中存在的不足为总体研究目标，通过结合已成功应用于大数据计算的 Hadoop 技术，研究大规模本体在云计算环境下的分布式数据查询与推理关键技术。拟解决的具体问题和研究目标分别是：

(1) 面向大规模本体数据查询与推理任务的云计算框架体系结构

云计算技术涉及了计算机硬件环境配置、网络服务和云计算基础平台软件设置、云计算安全等诸多问题，并且语义 Web 本体查询和推理工具需根据其理论基础而设计特有的功能体系划分。在本文研究中，针对云计算框架的体系结构问题，主要目标是在云计算 Hadoop 平台体系结构基础之上，探询符合整个语义 Web 本体查询与推理任务的框架功能模块划分机制及数据计算业务流程，并为后续关键技术的理论研究奠定体系结构标准。

(2) 云计算环境下的大规模本体数据分布式存储方法

如何合理地拆分原始的大规模本体 RDF 和 OWL 元数据以便实现云存储，并且能够无损的重新组合原有的本体，是实现云计算环境下的本体数据分布式存储的基础。在本文研究中，针对大规模本体数据的存储问题，主要目标是基于本体数据模型及其形式化语义，结合 HBase 数据表特性，探询如何在 HBase 中对本体数据进行高效存储的方法，实现所需数据空间与数据查询性能之间的平衡。在实现这一目标的基础之上，进而为实现高性能、可扩展的本体数据查询与推理提供支撑。

(3) 大规模 RDF 数据在云计算环境下的分布式组图模式查询方法

SPARQL 是 W3C 推荐的 RDF 数据标准查询语言，但由于其复杂组图模式查询需要由多个 MapReduce 任务组合完成，而过多的查询任务会造成计算性能和时间的不必要开销。因此，如何根据 SPARQL 组图模式查询条件尽可能减少查询任务数，并在此基础之上，探寻符合 MapReduce 键值对模型的复杂组图模式分布式查询方法，是本文拟解决的关键科学问题和研究目标之一。

(4) 大规模 OWL 本体在云计算环境下的分布式一致性检测方法

OWL 本体可规约为相应的描述逻辑语言，而 Tableau 算法是进行本体一致性检测的标准推理算法。如何实现本体数据的合理拆分和组合，并探寻 Tableau 算法在 MapReduce 下的并行化方法是本文拟解决的关键科学问题和研究目标之一。

(5) 大规模 OWL 本体在云计算环境下的 SWRL 规则分布式推理方法。

为了保证推理任务在有限时间内完成，需要对 SWRL 规则进行基于 DL-Safe 的限制。在本文研究中，针对 SWRL 规则在云计算环境下的分布式推理问题，主要目标是探寻 DL-Safe 限制下 SWRL 规则在 MapReduce 键值对计算模型下的查询任务生成机制和并行化推理方法，进而实现高性能、可扩展的 SWRL 规则分布式推理。

1.3 论文研究内容

基于上述研究问题和目标，本文的研究工作拟在以下五个方面开展：

(1) 研究基于 Hadoop 的本体数据查询与推理云计算框架体系结构

以现有描述逻辑推理引擎和规则引擎、SPARQL 查询引擎的体系结构研究成果为基础，充分结合云计算 Hadoop 技术特性，按照功能维度的不同对框架体系结构进行分层设计，对涉及核心计算功能的组件进行详细功能模块划分，研究各个组件之间的分布式计算业务流程。

(2) 研究基于 HBase 的大规模本体数据分布式存储策略

研究 RDF 和 OWL 本体数据在 HBase 表中行键、列名及其数据单元中的数据存储策略，分析在进行 SPARQL 组图模式查询、一致性检测推理和 SWRL 规则推理时的数据映射机制，探寻本体存储空间开销与数据查询性能之间的平衡点，并通过与现有本体存储策略进行分析对比，论证提出方法具备的优势与不足。

(3) 研究基于 MapReduce 的 SPARQL 复杂组图模式分布式查询方法

根据 SPARQL 语法和形式化语义，研究组图模式运算符 AND、UNION、OPTIONAL 和 Offset、Order、Distinct 等查询限制条件在 MapReduce 键值对模型下的计算方法及 SPARQL 组图模式查询任务生成算法，并在此基础之上，研究各个查询任务在 map 和 reduce 函数中键值对的选择机理以及查询的并行化方法，并通过与现有单机查询工具进行对比和可扩展性实验，对提出方法进行充分论证。

(4) 研究基于 MapReduce 的 OWL 本体一致性分布式检测方法

根据 OWL 形式化语义，研究 MapReduce 键值对计算模型下大规模 OWL 本体数据划分与重构机制，并在此基础之上，研究描述逻辑 SHIF 的 Tableau 算法推理在 MapReduce 中的并行化方法，并通过与现有单机描述逻辑推理工具进行对比和可扩展性实验，对提出方法进行充分论证。

(5) 研究基于 MapReduce 的 SWRL 规则分布式规则方法

根据 SWRL 规则形式化语义，研究其在 MapReduce 的键值对模型下的解析和推理任务生成方法，并以此为基础，研究 SWRL 各类规则原子在推理过程中的键值对选择机制，以及 MapReduce 任务各个 map 和 reduce 函数中的分布式规则推理算法。通过与现有单机规则推理工具进行对比和可扩展性实验，对提出方法进行充分论证。

1.4 论文研究意义

本论文将语义 Web 与云计算关键技术相结合，研究基于 Hadoop 的大规模语义 Web 本体数据查询与推理的关键科学问题，弥补本领域已有研究工作中存在的不足，论文研究具有如下的学术及实用意义：

(1) 本论文对云计算环境下大规模本体数据分布式存储策略的研究，在提出一种新的本体数据存储理论方法的同时，可实现本体存储空间开销与数据检索性能的优化，为实现高效率的大规模本体数据查询和推理奠定理论和应用基础。

(2) 本论文面向本领域关键科学问题，研究并提出 SPARQL 组图模式查询、OWL 本体一致性检测及 SWRL 规则推理在云计算 Hadoop 环境下新的并行化方法，具有重要的学术意义。同时，本论文的研究可改进传统单机环境下 SPARQL 查询工具、描述逻辑推理工具和规则推理工具在对大规模本体数据处理时存在的计算性能和可扩展性不足问题，具有重要的实用意义。

(3) 本论文关键技术理论研究过程中，提出的查询和推理任务生成算法，可实现 MapReduce 任务迭代执行次数的优化，能够为今后涉及 MapReduce 任务数判定的科学理论研究提供借鉴作用。此外，本论文的理论研究成果还可对知识表示和人工智能领域在进行大数据计算研究时提供借鉴。

(4) 本论文对大规模本体数据查询与推理云计算框架体系结构的研究，将理论研究和实际应用紧密结合，能够为今后高性能、易扩展的大规模语义 Web 本体数据查询与推理云服务的实际应用奠定框架体系结构支撑，为本领域后续的理论研究提供体系结构的参考标准。

1.5 论文组织结构

根据对论文研究内容的设计，全文共分 7 个章节，组织结构如下：

第 1 章引出论题，即指出本论文的研究背景，通过对问题和目标的分析，阐述了论文的研究意义和主要研究内容。

第 2 章通过国内外文献调查，首先对语义 Web 体系结构及其关键术语概念、描述逻辑、云计算关键技术进行综述，然后对云计算环境下的语义 Web 本体数据

查询与推理的研究现状进行了分析和总结。

第 3 章论述了基于 Hadoop 的本体数据查询与推理云计算框架体系结构，并提出了基于 HBase 的本体数据分布式存储策略。首先根据现有语义 Web 基本理论及 Hadoop 平台技术特性，对框架的层次结构及功能模块进行了划分，阐述了各个层级的功能和核心功能模之间的交换关系，并对提出体系结构进行了论证。然后根据 HBase 数据库表基于列的数据结构特性，提出新的 RDF 数据和 OWL 本体存储策略，最后对所提出策略进行深入对比和分析。

第 4 章论述了云计算环境下基于 MapReduce 的 SPARQL 复杂组图模式分布式查询方法。首先对 SPARQL 的语法和形式化语义进行了论述，并以此为基础，提出了 SPARQL 复杂组图模式的相关解析模型定义；然后分析了 SPARQL 查询运算符 OPTIONAL 和 UNION 在 MapReduce 的键值对计算模型下的计算方法；其次，在已有研究基础之上，提出了基于 MapReduce 的 SPARQL 复杂组图模式查询任务生成算法，进而提出了在 SPARQL 复杂组图模式在 MapReduce 模型下的分布式查询算法；最后通过使用 SP2Bench 本体测试数据集和标准测试语句，对本章所提出方法和已有 SPARQL 查询引擎进行了对比实验和可扩展性实验，并对实验结果进行了详细分析。

第 5 章论述了 MapReduce 分布式计算模型下基于描述逻辑的 OWL 本体一致性分布式检测方法。首先对 OWL Lite 的形式化语义和对应描述逻辑 SHIF 的 Tableau 算法进行了论述，并以此为基础，定义了在进行 OWL 本体一致性检测时的相关解析模型；然后，详细论述了 OWL Lite 本体在 MapReduce 键值对计算模型下的划分方法，进而提出了基于 MapReduce 的 Tableau 推理算法并行化方法；最后通过使用 LUBM 本体数据生成工具生成了大规模的 OWL Lite 本体数据，并以此作为测试数据集，对本章方法和已有描述逻辑推理引擎进行了对比和分析。

第 6 章论述了云计算 MapReduce 分布式计算模型下的 SWRL 规则推理方法。首先对 SWRL 的语法和形式化语义进行了论述，并以此为基础，提出了 SWRL 规则的相关解析模型定义；其次，提出了基于 MapReduce 的 SWRL 推理计划生成算法，进而提出了在 DL-Safe 限制下 SWRL 规则在 MapReduce 键值对计算模型下的并行化推理算法。最后通过使用 LUBM 本体数据生成工具生成了大规模的 OWL Lite 本体数据，并自定义了测试 SWRL 规则，对本章所提出方法和传统 SWRL 规则推理引擎进行了对比分析。

第 7 章对全文工作进行了总结，并讨论了当前工作的不足，以及下一步研究的计划和思路。

根据对论文研究内容规划和章节组织结构安排，主要研究工作的逻辑依赖关系如图 1.1 所示。

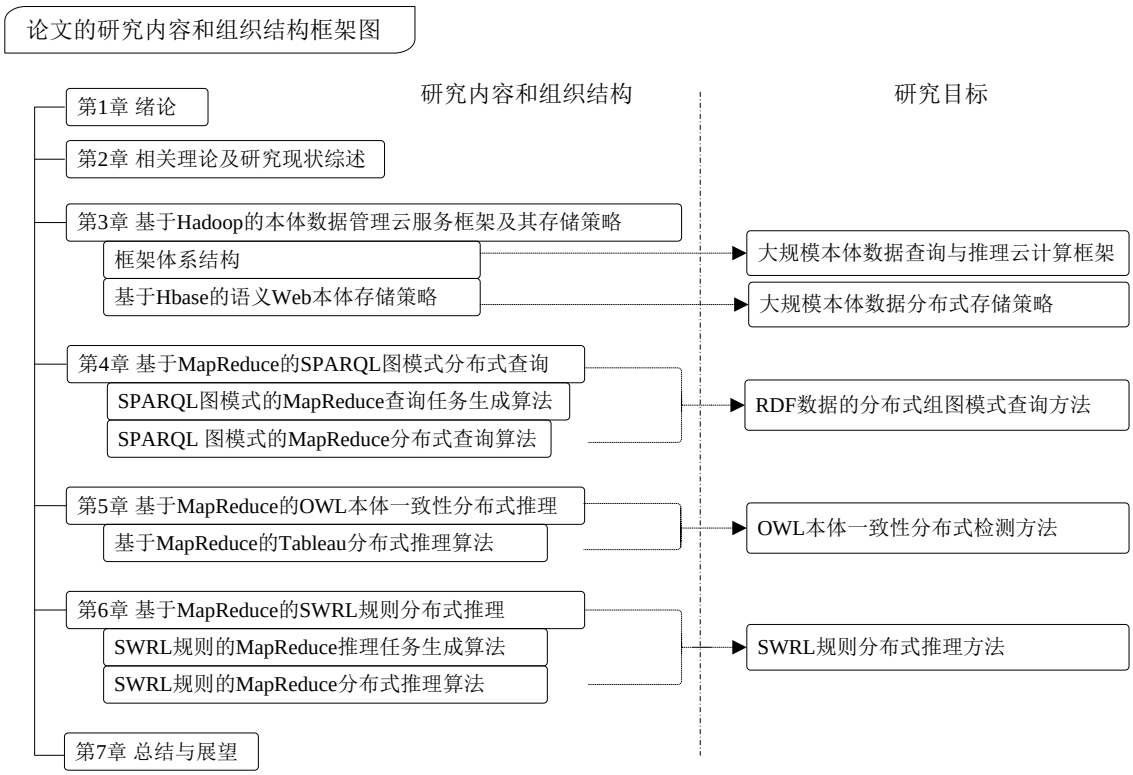


图 1.1 论文的研究内容和组织结构框架

Figure 1.1 Frameworks of Research Topics and Organization for the Thesis

2 相关理论及研究现状综述

本章首先综述论文中涉及的语义Web相关理论以及云计算数据密集型存储和计算模型，并对国内外云计算环境下的大规模本体数据查询与推理研究现状进行综述和分析，为后续章节中研究内容的论述奠定基础。

2.1 语义 Web

为了解决传统Web中所发布信息不能被计算机所理解，进而不能实现自动化推理和隐含知识发现等问题，Tim Berners-Lee于2001年首次提出了语义Web的概念。语义Web是通过为Web上资源引入清晰的语义标注和结构化描述，即引入本体的概念以实现计算机之间基于语义的信息交换，是关于下一代互联网的设想。

本体源于哲学概念，是对客观存在系统的解释或说明，并关注于客观现实的抽象本质[53]。在计算机科学领域，本体被认为是对概念模型的明确规范说明[54]，是显示的概念化规范[55]，具有共享性，常用于描述共同认可的结构化知识。语义Web需要形式化规范地说明概念模型[56]，因此，本体适合语义Web上的知识表示与推理，具有重要的研究价值。

2.1.1 语义 Web 体系架构

随着语义Web理论和技术的不断完善和应用，W3C推出了语义Web的体系结构。如图2.1所示，该体系基于层次结构，由底至上逐步建立规范，并具备向下可兼容性和向上可理解性。

首先，W3C规定采用UNICODE[57]格式对语义Web所发布的内容进行编码，并使用通用资源标识符(Uniform Resource Identifier, URI)[58]对所有资源进行唯一性标记。在此基础之上，为使得发布资源适合在Web上传递，并具有广泛的兼容性和允许用户自定义发布内容，W3C规定所有语义Web的结构化网络文档基于可扩展标记语言(Extensible Markup Language, XML)[59]格式。为了定义网络资源之间的关系，实现机器可理解，应用之间互操作，以及提供软件Agent自动处理能力，W3C于2004年提出了RDF，并将其推荐为语义Web的基本元数据模型。在RDF层之上，W3C提出了RDF-S，定义了资源的类以及类和属性的层次划分等，初步具备了简单语义推理和规则推理特征。为提供更强大的描述能力和可判定的推理能力，W3C于2007年提出了Web本体描述语言OWL，并将其指定为语义Web的核心技术，提供可以用于各种应用领域建模的本体描述和推理能力。OWL采用面向对象的方式来描述领域知识[60]，即通过类和属性来描述对象，通过公理来描述类和属性的特征及关系。另外，SPARQL是W3C推荐的RDF数据查询标准语言，SWRL是用于对OWL表达能

力进行规则扩展的标准规则描述语言。RDF、RDF-S、OWL、SPARQL和SWRL一起构成了目前语义Web领域的研究重点。除此以外，在语义Web体系结构中还包含了证明(Proof)层、信任(Trust)层和数字签名等面向用户和接口终端应用，属于实际应用开发范畴[61]，本文在此对上述内容不做过多论述。

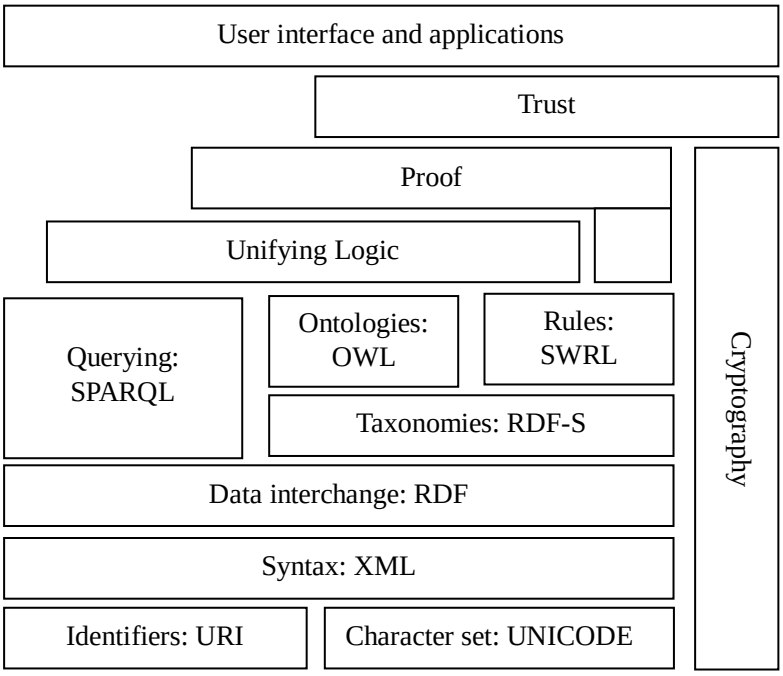


图 2.1 语义 Web 体系结构

Figure 2.1 Architecture of the Semantic Web Stack

2.1.2 RDF 与 RDF-S

RDF 是关于 Web 资源及其属性的陈述，代表了 Web 中任意两个资源间的某一特定关系，具体表现形式为 RDF 三元组(RDF Triple)数据模型(*s*, *p*, *o*)，其中 *s* 代表主语(Subject)，*p* 代表谓语(Predicate)，*o* 代表宾语(Object)，*s* 和 *p* 用 URI 指定唯一资源，*o* 可以为 URI 资源、XML Schema 数据类型或特定值的字面量[62]。可以将三元组(*s*, *p*, *o*)看作是主语 *s* 和宾语 *o* 之间关于二元谓词 *p* 的逻辑公式 *p*(*s*, *o*)。另外，三元组的集合称为 RDF 图，每个三元组中的主语和宾语表示两个结点，谓词为其之间的链[63]。当前，由于 RDF 在语义 Web 体系结构中位于 XML 层之上，因此，除了基于 XML 格式的 RDF/XML 以外，研究人员还提出了 N-Triple[64]、N3[65]等 RDF 语法格式，但所有格式均存在相同的描述语义，并可使用 RDF 文件解析器实现相互转换[66]。

由于 RDF 仅提供了基本数据模型，且不存在面向任何领域的语义，W3C 在 RDF 基础之上定义了 RDF-S，并提供了将网络对象组织成层次结构的建模原语，

其中包括了类(rdfs:Class)、属性(rdf:Property)、子类(rdfs:subClassOf)、子属性(rdfs:subPropertyOf)、定义域(rdfs:domain)和值域(rdfs:range)等[67]。因此，RDF-S 可看作是 RDF 的词汇表扩展，并根据 RDF-S 推理规则，具备了针对特定领域初步的语义和推理能力。

2.1.3 SPARQL 查询语言

根据 W3C 提出的语法规约，SPARQL 查询语句由查询数据集(Query DataSet)、查询模式(Query Pattern)、解序列修改器(Sequence Solution Modifier)和查询表单(Query Form)四部分组成[68]。其中，查询数据集用于指定所需查询的 RDF 数据源。查询模式用于对目标 RDF 图进行匹配，其基本查询单元为三元组模式(Triple Pattern)，一个或多个三元组模式可组合为图模式(Graph Pattern)[69]。当前，SPARQL 定义了五种图模式，分别为：由必须匹配的三元组模式集合构成的基本图模式(Basic Graph Pattern)、必须匹配所有的图模式的组图模式(Group Graph Pattern)、由 OPTIONAL 运算符连接的可选图模式(Optional Graph Pattern)、由 UNION 运算符连接的多图模式(Alternative Graph Pattern)、用于模式和命名图匹配的命名图模式(Pattern on Named Graphs)[70]。在本文研究内容中，将原有的组图模式、可选图模式和多图模式统称为**复杂组图模式**。

SPARQL 基本图模式用于对多个三元组模式查询条件同时进行匹配，复杂组图模式可在多个图模式之间实现 AND、OPTIONAL 和 UNION 查询操作。解序列修改器提供了 Order By、Distinct、Offset、Limit 等限制符对查询模式返回的无序查询结果进行排序和限制。最后，查询表单根据例如 SELECT、ASK、DESCRIBE 或 CONSTRUCT 的选择符返回最终查询结果[71]。在最新的 SPARQL 1.1 规范中，W3C 还引入了 INSERT、DELECT 和 UPDATE 等查询表单选择符，以实现对 RDF 数据的添加、删除和修改操作[72]。将在本文第四章中详细描述 SPARQL 语言的形式化语义。

目前，语义 Web 领域已提出了多个 SPARQL 查询工具。例如，Jena 是一个基于 Java 实现的开源 SPARQL 编程环境，它采用 ARQ 查询引擎[73]，提供了在内存中的高效 RDF 数据访问与查询机制。同时，为实现对多种存储环境的支持，Jena 还提供了基于关系型数据库模型的 Jena SDB 数据存储访问模型。Sesame 是一个支持存储和查询大规模 RDF 数据的通用框架，它支持 SPARQL 和 SeRQL 查询语言，并能提供 RDF 三元组、关系型数据库和远程 Web 服务[74]的数据存储。RDF-3X 是一个基于 RISC 的 RDF 查询引擎[75]，由于采用了基于直方图(Histograms)和概要统计(Summary Statistics)的查询优化策略，RDF-3X 被认为是目前性能最佳的 RDF 查询工具，但目前版本的 RDF-3X 还不能提供 OPTIONAL 运算符查询。

2.1.4 OWL 本体描述语言与描述逻辑

由于 RDF 和 RDF-S 仅提供简单的本体描述能力，尚不能自定义新的类，也缺乏表达能力来描述传递性和对称性等，W3C 在 OIL[76]和 DAML+OIL[93]的研究基础之上，由 2007 年制定了 Web 本体描述语言 OWL 1.0，并推荐其为语义 Web 的本体建模标准。2012 年 W3C 发布了新的 OWL 2 标准[9]。

OWL 1.0 分为了 OWL Lite、OWL DL 和 OWL Full 三种子语言，其描述能力逐步提高，推理复杂度也随着提升。其中 OWL Lite 支持分类体系和简单约束，并对应了描述逻辑 SHIF[78]；OWL DL 提供了更强的表达能力，具有计算完备性和可判定性，它对应描述逻辑 SHOIN[79]；OWL Full 提供了最强大的表达能力和使用 RDF 语法的最大自由，但其推理计算过程不可判定[80]。OWL 2 EL 可映射为描述逻辑 EL+，其所有标准推理任务均具备多项式时间算法，但为了保证推理性能，降低了表达能力，特别适用于大本体[81]；OWL 2 QL 支持采用传统关系型数据库技术的合取查询，适用于采用轻量级本体组织大规模的个体，或者采用 SQL 关系查询的应用场景[82]；OWL 2 RL 采用规则扩展的直接运行在 RDF 三元组上的数据库技术，具备多项式时间复杂度的推理算法，适用于轻量级本体组织大规模个体或者直接操作 RDF 三元组的场景[83]。借助于人工智能领域描述逻辑理论和工具，OWL Lite、OWL DL 和 OWL 2 DL 是当前语义 Web 领域最广泛使用的本体建模语言。在本文中，主要面向大规模 OWL Lite 本体的一致性检测和规则推理展开研究。

描述逻辑[84, 85]是一阶谓词逻辑的可判定子集，是在框架系统(Frame)、语义网络(Semantic Networks)[86]和面向对象[87]等方法基础之上形成的知识表示工具，具有形式化语义表示和逻辑推理的能力。描述逻辑的知识表示系统包含一个知识库及其提供的推理服务，其中知识库由描述应用领域词汇的术语集(Terminology Box, TBox)和个体断言集(Assertion Box, ABox)组成。图 2.2 描述了描述逻辑知识库系统的体系结构。

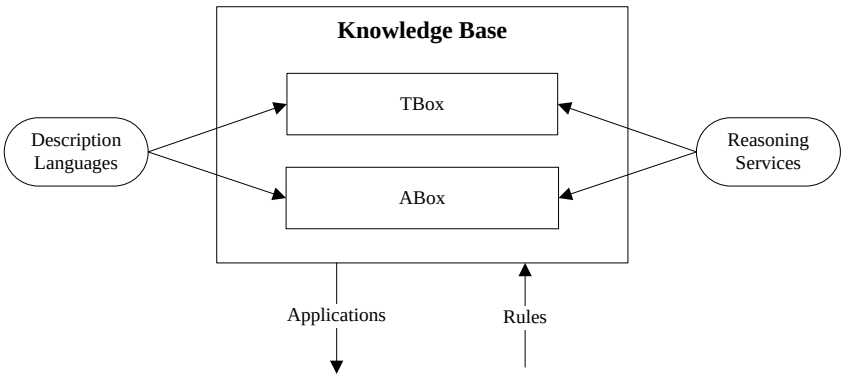


图 2.2 基于描述逻辑的知识表示系统体系结构

Figure 2.2 Architecture of a knowledge representation system based on Description Logics

在知识描述方面，描述逻辑以 ALC[88]为基础，表达能力逐步提升，其中的某个特定语言成员由所允许使用的构造子对特定应用领域进行递归定义。常用的构造子包括概念的交、并、反或补；角色的全局和扩展限定、逆、函数关系、复合关系等等。不同的描述逻辑语言成员可以使用不同的构造子来形成语法体系，使得语义表示能力不同，但表达能力的增强会对推理性能和可判定性带来负面影响。

设 C 和 D 为描述逻辑原子概念，R 和 S 为原子角色，TBox 是概念和角色包含关系 $C \sqsubseteq D$ 和 $R \sqsubseteq S$ 的有限集合，因此，复杂概念和角色可由原子概念和原子角色进行描述。设 x 和 y 为描述逻辑个体，ABox 是关于个体的概念断言和角色断言 $C(x)$和 $R(x, y)$的有限集合。描述逻辑中解释 $I = <\Delta^I, (.)^I>$，其中，解释域 Δ^I 是一个非空的个体集，$(.)^I$ 是解释函数。对于每个概念 C，都有 $C^I \subseteq \Delta^I$，对每个 R，都有二元关系 $R^I \subseteq \Delta^I \times \Delta^I$。在表 2.1 中以最基本的描述逻辑 ALC 为例，列举了其句法及语义关系。将在本文第五章中描述 OWL Lite 本体语言及其描述逻辑 SHIF 的形式化语义描述。

表 2.1 描述逻辑 ALC 语法及语义的对应关系

Table 2.1 Relationship between Syntax and Semantics of Description Logic ALC

ALC 语法	**ALC 语义**
$C \sqcap D$	$C^I \bigcap D^I$
$C \sqcup D$	$C^I \bigcup D^I$
$\forall R.C$	$\{x \mid \forall y. < x, y > \in R^I \rightarrow y \in C^I\}$
$\exists R.C$	$\{x \mid \exists y. < x, y > \in R^I, y \in C^I\}$
$\neg C$	$\Delta^I \setminus C^I$

作为 OWL 语言的语义表达和逻辑推理基础，表 2.2 列举了 OWL DL 语言中的主要的标签原语与描述逻辑算子间的语义映射关系。

在推理服务方面，早期描述逻辑系统主要采用结构包含算法(Structural Subsumption Algorithm)[89]，但其只能针对简单描述逻辑语言。当前，描述逻辑主要基于 Tableau 算法[90]，分别提供了针对于 TBox 和 ABox 的推理服务[91]。TBox 推理形式中主要包括了概念的可满足性、包含关系、等价关系和不相交关系验证，ABox 推理主要涉及一致性推理服务。并且根据 Tableau 算法思想，所有推理过程均可规约为对 ABox 的一致性验证。

表 2.2 OWL DL 中类构造符与描述逻辑 SHOIN 语法的对应关系

Table 2.2 Relationship between OWL DL Class constructors and DL SHOIN Syntax

OWL 标签原语	**DL 描述**
owl:subClassOf	$C \subseteq D$
owl:intersectionOf	$C \sqcap D$
owl:unionOf	$C \sqcup D$
owl:complementOf	$\neg C$
owl:disjointWith	$C \sqcap D = \varnothing$
owl:allValuesFrom	$\forall R.C$
owl:someValuesFrom	$\exists R.C$
owl:maxCaidinality	$\leq nR.C$
owl:minCardinality	$\geq nR.C$
owl:cardinality	$= nR.C$
owl:hasValues	R:x

2.1.5 SWRL 规则描述语言

SWRL 的基础包括 OWL 1.0 子语言 OWL DL 和 OWL Lite，以及一元/二元 Datalog 规则标记语言的子语言 RuleML[92]。SWRL 规则采用 Horn 子句形式的规则对 OWL 的公理集进行了扩展，可以将 Horn 子句形式的规则和 OWL 知识库相结合。

SWRL 规则采用蕴含的形式表示，包括一个前件或规则体，以及一个后件或规则头。其含义是：当前件中的条件满足时，后件中的条件一定满足。将在本文第六章中描述 SWRL 规则的句法和形式化语义。

SWRL 与 OWL 一样，均采用开放世界假设[93]，即对两个属于相同类型的 OWL 个体并不假设两个个体名不同，它们就是不同的个体。但在 SWRL 规则中，不同的两个变量?x 和?y 可以匹配同一个体。另外，SWRL 和 OWL 类似，仅支持单调推理，即不能用 SWRL 规则来修改本体中的信息。

DL-Safe 规则[94]是 SWRL 的一个限制子集，其仅限于对 OWL 本体中已知的个体进行推理，因此使得 DL-Safe 限制下的 SWRL 规则推理具有可判定性[95]。但由于 DL-Safe 限定了 SWRL 规则的表达能力[96]，可能得到不完备的推理结果，即不一定能得到所有正确的结果，但所有得到的推理结果都是正确的。

2.2 云计算

2.2.1 云计算概念及其特性

目前，云计算已成为当前计算机研究领域最热门的研究方向之一。然而，学术界和工业界对云计算概念还没有一个统一的定义[97]。美国国家标准与技术研究

院认为，云计算是一种按使用量付费的模式，这种模式提供可用的、便捷的、按需的网络访问，进入可配置的计算资源共享池(资源包括网络、服务器、存储、应用软件、服务)，这些资源能够被快速提供，只需投入很少的管理工作，或与服务供应商进行很少的交互[98]。IBM 认为：“云计算一词用来同时描述一个系统平台或者一种类型的应用程序。一个云计算的平台按需进行动态地部署(provision)、配置(configuration)、重新配置(reconfigure)以及取消服务(deprovision)等。在云计算平台中的服务器可以是物理的服务器或者虚拟的服务器.高级的计算云通常包含一些其他的计算资源，例如存储区域网络(SANs)，网络设备，防火墙以及其他安全设备等。云计算在描述应用方面，它描述了一种可以通过互联网 Internet 进行访问的可扩展的应用程序。‘云应用’使用大规模的数据中心以及功能强劲的服务器来运行网络应用程序与网络服务。任何一个用户可以通过合适的互联网接入设备以及一个标准的浏览器就能够访问一个云计算应用程序”[99]。我们认为，云计算是建立在传统分布式计算、效用计算、虚拟化技术、Web 服务和网格计算等技术的研究基础之上，根据用户需求，对其提供大规模数据存储和高性能计算服务的新型计算模式。

当前，根据云计算所服务应用场景的不同，将云计算分为了基础架构即服务(Infrastructure as a Service, IaaS)、平台即服务(Platform as a Service, PaaS)和软件即服务(Software as a Service, SaaS)三种服务模式[100], [101]，并且形成了以 Google、Amazon、Yahoo!等为代表的云计算服务应用。在本文的研究中，本文认为针对大规模语义 Web 本体数据的查询和推理属于 SaaS 范畴。

为保障向用户提供按需服务顺利实施，在云计算服务端具备了针对大数据的高性能、可扩展存储及计算能力，这一重要特性也引起了学术界的广泛关注。其中，由 Google 公司提出的 GFS 分布式文件系统[43]、MapReduce 分布式计算模型[44]、BigTable 分布式数据库模型[45]已成为云计算研究领域最广泛使用的数据密集型分布式存储和计算模型。Apache 公司将上述三种模型在 Hadoop 平台中进行了开源实现，有针对性地提出了 Hadoop File System (HDFS)分布式文件系统、MapReduce 分布式编程框架和 HBase 分布式数据库系统，并已在多个学术领域得到了成功应用[102],[103], [104],[105]。例如，研究人员已开始使用 MapReduce 对传统数据挖掘算法进行改进，并实现了 PB 级的大规模数据挖掘[106]。Chu 等人基于 MapReduce 模型，改写了现有机器学习算法，实现了高性能、易扩展的学习计算[107]。

2.2.2 GFS 与 HDFS 分布式文件系统

GFS 是 Google 公司针对其大规模 Web 数据存储应用需求而提出的分布式文件系统模型，是 MapReduce 和 BigTable 模型的基础。GFS 充分考虑了文件系统的性能、可伸缩性、可靠性和可用性，并具备了容错和自动恢复功能。图 2.3 描述了

GFS 的系统架构。

一个 GFS 集群包含一个主服务器(Master)和多个块服务器(Chunk Server)。大文件被分割成固定尺寸的块,块服务器把块作为 Linux 文件保存在本地硬盘上，并根据指定的块句柄和字节范围来读写块数据。为了保证可靠性，每个块被缺省保存 3 个备份。主服务器管理文件系统所有的元数据，包括名字空间、访问控制、文件到块的映射、块物理位置等相关信息。HDFS 是 GFS 的开源实现，具有相似的架构，本文在此不做过多描述。

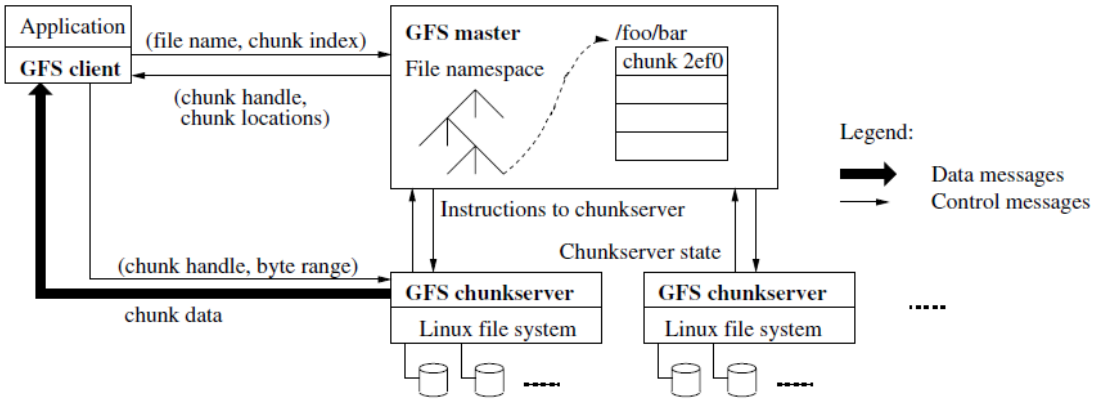


图 2.3 Google File System 系统架构

Figure 2.3 Architecture of the Google File System

2.2.3 MapReduce 分布式计算模型

以 GFS 文件系统为基础，MapReduce 是一个在普通计算机集群中运行的并行和分布式海量数据处理框架。

MapReduce 的核心思想是：(1) 将问题分而治之；(2) 将计算推到数据，可有效避免数据传输过程中产生的大量通信开销。

首先，MapReduce 将 GFS 文件系统中的数据块作为输入，并将输入数据转换成键值对，在由 map 和 reduce 函数组成的 MapReduce 任务(Job)中进行处理，而复杂的计算任务也可以在多个互相链接的 MapReduce 任务中完成计算。在每个 MapReduce 任务中，Master 节点将输入数据分割成若干独立的数据块，并将它们传递到 Map 节点。之后，在 Mapping 阶段，每个 Mapper 计算节点接受一定数量的数据块，然后根据用户自定义的 map 函数，产生一系列中间键值对。最后，Master 节点通知将中间数据的存储位置通知 Reducer 结点，每 Reducer 结点合并具有相同键值的中间数据，并根据用户定义的 reduce 函数生成一系列的键值对数据。

在 Hadoop 开源平台下的 MapReduce 编程环境中，还包含了 MapReduce 任务链、MapReduce 任务依赖等优化技术以实现对复杂计算任务的支持[108]。

图 2.4 描述了 MapReduce 模型的数据处理原理。

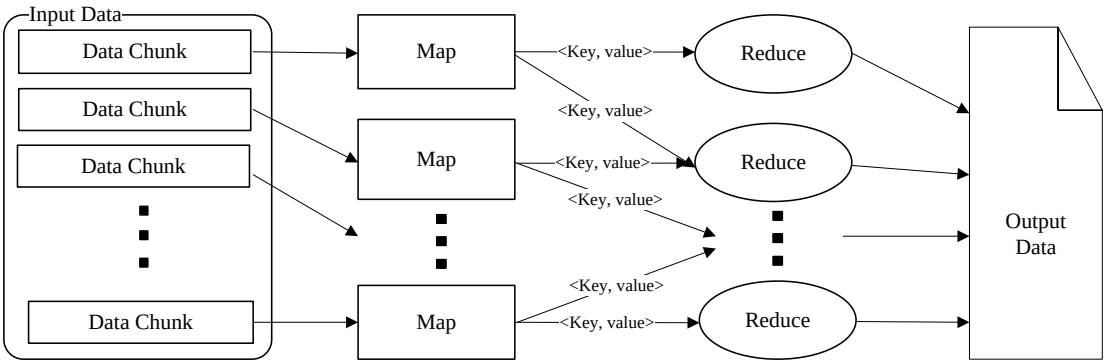


图 2.4 MapReduce 分布式计算模型

Figure 2.4 MapReduce Paradigm

2.2.4 BigTable 与 HBase 分布式数据库

BigTable 分布式数据库模型的基本思想是在计算机集群中维护一组多维数据表。每个数据表的数据行由唯一的行键(Row Key)和任意数量的列组成，多个列可以合并为一个列族(Column Family)。当确定某一行键和列名(Column Name)时，可检索到某一数据单元值(Cell Value)，并且此数据单元可根据时间戳(Timestamp)区分存储多个版本的数据。其形式化描述如下：

给定某一行键 *row_key*、列名 *column _name* 和时间戳 *Timestamp*，可确定唯一单元值 *Cell_Value*，如式(2.1)：

$$(row\_key, column\_name, Timestamp) \rightarrow Cell\_Value \tag{2.1}$$

另外，每个数据表中可定义多个列族，每个列族可由多个列构成，并且列名及列数目可由程序动态任意指定，并且每列可存放基于时间戳的多版本数据值。作为 BigTable 模型的开源实现，Hadoop 平台下的 HBase 分布式数据库还提供了类似 B+树[109]的行键索引，对数据进行检索时，还可以根据给定行键或键范围进行处理。

图 2.5 描述了 HBase 分布式数据库表的数据模型，其具备以下特性：

①HBase 表中存储的数据无需预定义数据类型。表中所有数据均以字节数组形式存储，在初始化定义 HBase 表时无需为行或列指定具体的数据类型。

②HBase 表无固定结构。每个 HBase 表可随时动态添加或删除行或列，且其数目不设置上限，因此 HBase 能适应数据结构多变的应用场景，并且能对海量数据进行存储。

③HBase 表结构稀疏。每个 HBase 表是一个稀疏的数据存储单元，当表中任一单元值为空时，不占用存储空间，因此，HBase 的这一特性可减少不必要的

存储空间消耗。

④HBase 的存储能力具备强大的可伸缩性。HBase 基于 HDFS 分布式文件系统构建，是建立在普通商用计算机集群之上的分布式数据库，其存储能力可根据实际应用需要，通过横向扩展存储计算机节点实现存储空间的线性增长。

RowKey	Column Family (*M*)				Timestamp
	Col 1	Col 2	...	Col *N*	
RowKey 1	*Value*	*Value*	...	*Value*	Timestamp *T_1*
...	*Value*	*Value*	...	*Value*	...
RowKey *i*	*Value*	*Value*	...	*Value*	Timestamp *T_i*

图 2.5 HBase 表数据模型

Figure 2.5 Data Model of HBase Table

表 2.3 给出了 HBase 数据库与传统关系数据库数据模型之间的差异。正是由于 HBase 可适应数据结构多变的应用场景，且基于分布式计算机集群构建，其在对大规模数据进行存储时的可扩展性更强，因此，相较于关系型数据库，HBase 更适应于大规模语义 Web 本体的存储需求。但是，当前版本的 HBase 数据库尚不支持存储过程及事务处理机制，并且当在对多个 HBase 数据表进行管理时，在一定程度上会带来数据集成和数据冗余的问题。因此，为避免上述问题，本文在进行框架体系结构的研究过程中，设计了数据适配器模块，作为访问 HBase 数据库的唯一接口。

表 2.3 HBase 数据库与关系数据库的不同点

Table 2.3 The difference between HBase table and relational database table

关系数据库	HBase 数据库
二维映射表	多维映射表
有固定数据存储模式	无固定的模式
表中每一列预定义数据类型、长度	数据统一以字节数组存储
面向行、稠密的、表中列数存在限制	面向列、稀疏的、支持大规模的列数
集中式存储	分布式存储
数据关联性、多行事务性	数据无关联、单行事务性

2.3 云计算环境下大规模本体数据查询与推理研究现状

2.3.1 基于 MapReduce 的大规模 RDF 数据分布式查询

目前云计算环境下的大规模语义 Web 本体数据查询领域的已有研究成果主要

面向 SPARQL 基本图模式在 Hadoop 平台下的分布式查询。

其中，Choi 等人提出了一个基于 HBase 和 MapReduce 的分布式 RDF 数据查询框架 SPIDER[110]，并论述了该系统的总体架构，简单描述了 RDF 图在 MapReduce 模型下的分布式查询机制。但是，Choi 等人并没有详细论述 RDF 三元组在 HBase 数据库中的存储策略，没有提出具体的 MapReduce 查询算法，也没有报告任何的实验结果。

Myung 等人提出了一种 SPARQL 基本图模式的 MapReduce 算法[111]。由于在 MapReduce 环境下运行多个查询任务会带来很多额外的开销，例如每次查询任务的初始化需对计算资源信息重新设定，以及 map 和 reduce 函数中均会对数据进行排序处理等，因此 Myung 等人采用贪心策略实现了 SPARQL 基本图模式查询语句的连接键(Join key)选择，并将多路链接(Multi-way Join)方法应用到 MapReduce 查询任务，减少了不必要的任务迭代。通过在模拟云计算环境下进行对比实验，Myung 等人证明了在处理大规模 RDF 数据进行 SPARQL 查询时，他们所提出的方法在计算性能和可扩展性方面均优于传统 SPARQL 查询工具。

Husain 等人提出了一个基于 MapReduce 的启发式(Heuristics)大规模 RDF 查询处理框架[112]。首先，Husain 等人设计了一个全新的基于 HDFS 文件系统的 RDF 数据存储策略，并设计了 SPARQL 基本图模式、RDF 三元组及框架内术语的表示模型。为生成查询计划，Husain 等人还提出了一个计算复杂度以总查询变量的 log 值为上界的查询计算生成算法 Relaxed-Bestplan。通过采用语义 Web 研究领域中广泛使用的测试数据集 LUBM 和 SP2Bench 进行模拟实验，Husain 等人证明了该提出框架在处理大规模 RDF 数据时，计算性能和易扩展性均优于传统单机环境下的 SPARQL 查询工具。本文的研究过程中，主要对 Relaxed-Bestplan 进行改进和扩展，以实现面向 SPARQL 复杂组图模式和 SWRL 规则的推理计划生成。

然而，文献[111]和[112]所提出的解决方案均是以 HDFS 文件系统作为 RDF 数据存储载体，因而很难对大规模的 RDF 数据进行随机的修改操作，尤其是当在涉及到最新的 SPARQL 1.1 中 Update 或 Delete 等运算符的计算处理时，该存储机制显然存在不足之处。因此，采用 HBase 等分布式数据库系统作为本体存储载体，可实现随机的数据访问和修改。

基于 Hexaxtore 的存储策略[113]，Sun 等人提出了在 HBase 数据库环境下的 RDF 数据存储方案和基于 MapReduce 的 SPARQL 基本图模式查询方法[114]。Sun 等人根据三元组主谓宾的不同组合形式设计了六个 HBase 表来对各种可能的 SPARQL 三元组模式进行匹配，并提出了基于贪心策略的连接键选择方法。但是，在 Sun 等人所提出解决方案中，RDF 数据需要被复制六次进行存储，进而增大了本体数据存储空间的开销，尤其是当需要对 RDF 数据进行修改或删除时，需要同时对所

有六个表进行访问，在 HBase 数据库没有提供事务处理功能的情况下，存在数据处理的同步问题。

Franke 等人提出了一个基于 HBase 的分布式语义 Web 数据管理框架[115]，设计了一个使用两张 HBase 数据表的 RDF 存储策略并提出了一个实现 SPARQL 基本图模式查询的 MapReduce 算法。通过与 MySQL 集群数据库进行对比实验，证明了云计算环境下的语义 Web 数据管理方法在可扩展性和数据查询效率等方面优于传统关系型数据库存储模式。但 Franke 等人的解决方案仍局限于对 SPARQL 基本图模式的计算处理。

因此，云计算环境下大规模语义 Web 本体数据分布式查询的研究在国内外仍处于起步阶段，已有的研究成果均只考虑了最基本的 SPARQL 基本图模式查询处理，但还无法实现对 SPARQL 复杂组图模式的查询。

2.3.2 基于 MapReduce 的大规模 OWL 本体分布式推理

在云计算环境下的大规模 OWL 本体数据分布式推理研究领域，Urbani 等人提出了基于 MapReduce 的分布式本体推理引擎 WebPIE，实现了云计算环境下的 RDF 图闭包计算及 OWL Horst 推理[116]。根据 RDF-S 推理规则，Urbani 等人分析了其在 MapReduce 下推理时存在的并行化先决条件，并提出了将术语本体载入各个计算节点内存和对本体数据进行预处理等优化方案。另外，Urbani 等人在实现了复杂 OWL Horst 规则在 MapReduce 模型下的分布式推理的同时，还提出了对 OWL Horst 本体的优化方法。通过使用 FactForge[117]、Linked Life Data[118]和 LUBM 等测试数据集与现有单机环境下的描述逻辑推理引擎进行对比实验和可扩展性试验，证明了基于 MapReduce 的大规模 OWL 本体推理在计算性能和可扩展性方面均优于传统推理工具。

Mutharaju 等人提出了针对 OWL 2 EL 本体分类的 MapReduce 推理算法[119]。首先，Mutharaju 等人将现有 OWL2 EL 本体推理算法 CEL 进行改进，使其适用于 MapReduce 模型的键值对计算环境，并详细论述了 MapReduce 推理算法的计算步骤。然而，Mutharaju 等人并没有对提出方法进行实验验证，也没有给出具体的实验结果。

到目前为止，国内外学术界尚未开展针对于大规模 OWL 本体的描述逻辑 Tableau 推理在 MapReduce 下的并行化研究，并且在大规模 OWL 本体的 SWRL 规则推理方面，国内外学术界仍然缺少一种高性能、易扩展的解决方案。

基于上述研究现状，本课题充分结合云计算研究领域中 MapReduce 和 HBase 等分布式计算和数据存储技术，对 RDF 和 OWL 本体在云计算环境下的分布式存储模式，SPARQL 复杂组图模式在 MapReduce 下的并行化查询机制，基于 Tableau 算法的 OWL 本体一致性推理和基于 SWRL 规则的 OWL 本体推理等关键技术开展

深入研究，以实现语义 Web 中大规模本体数据的高性能、可扩展的查询与推理这一研究目标。

2.4 本章小结

本章首先对论文中主要涉及的语义 Web 体系结构及 RDF、RDF-S、OWL、SPARQL 和 SWRL 术语基本概念进行了描述，然后阐述了云计算 HDFS 分布式文件系统模型、MapReduce 分布式计算模型和 HBase 分布式数据库系统模型的基本理论及其技术特性。最后，综述了当前国内外学术界在大规模语义 Web 本体数据查询与推理方向的研究现状，详细分析了该方向当前研究成果存在的不足。

结合本领域研究背景和研究现状，可得出以下的分析结果：(1) 基于 SPARQL 的 RDF 数据查询、基于描述逻辑的本体一致性检测推理和基于 SWRL 的本体规则推理是语义 Web 领域研究的核心，而现有的本体大数据为传统单机环境下的查询和推理工具带来了严峻的挑战，对现有理论和方法在分布式计算环境下开展并行化研究是该领域的必然趋势。(2) 云计算 Hadoop 关键技术具备优秀的大数据分布式存储和计算能力，是当前大数据处理的事实标准，并已在语义 Web 领域得到了一定程度上的成功应用。(3) 语义 Web 与云计算相结合的研究领域当前处于起步阶段，在大规模本体的分布式查询、分布式一致性检测及分布式规则推理研究方面均存在不足。

3　基于 Hadoop 的本体数据查询与推理云计算框架

3.1　引言

以 W3C 推荐的 RDF、RDF-S、OWL、SPARQL 和 SWRL 标准为基础，对大规模语义 Web 本体数据进行高效率的存储组织、查询和推理是实现语义 Web 远景的关键。近年来，研究人员已研发出多个单机环境下的语义 Web 本体查询与推理系统，如 Jena、Pellet、HermiT 等，通过分析已有文献得知[120]，上述工具的一般系统结构主要由本体解析器、查询解析器、推理查询引擎和结果输出模块构成，其系统结构如图 3.1 所示。其中，本体解析器负责读取并解析本体文件，查询解析器用于解析用户的查询命令，推理查询引擎负责具体查询与推理计算流程的执行，结果输出模块将查询或推理结果返回给用户。在本体数据的存储组织方面，除了基于 RDF 和 OWL 文件形式，当前语义 Web 研究领域主要采用传统关系型数据库进行本体数据的底层存储。

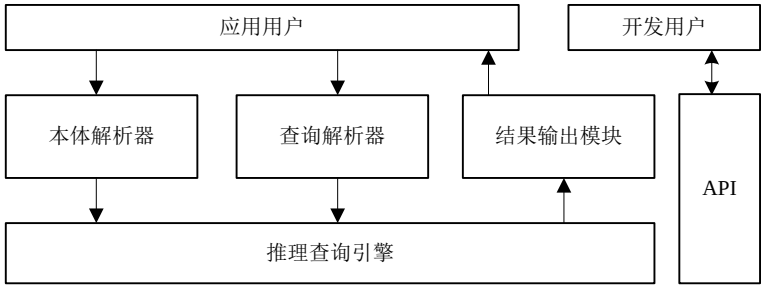


图 3.1 当前本体查询工具一般应用系统结构

Figure 3.1 General Architecture of Current Ontology Querying and Reasoning Tools.

如本文第二章所述，由于学术界关于云计算环境下的大规模本体数据存储、查询与推理研究仍处于起步阶段，业界还缺乏一种云计算服务框架体系结构，并且已有的本体存储策略仍需改进。

本章首先在现有语义 Web 本体数据查询与推理引擎体系结构的研究基础之上，结合云计算 HDFS 分布式文件系统、MapReduce 分布式计算环境和 HBase 分布式数据库技术，提出基于 Hadoop 的大规模本体分布式查询与推理云计算框架体系结构；然后，根据 RDF 三元组数据特性和 OWL 本体形式化语义，设计了框架中本体数据在 HBase 分布式数据库中的存储策略；最后，本章对所提出存储策略与现有方法进行了对比和分析。

3.2 框架体系结构

本章所提出的基于Hadoop的大规模本体分布式查询与推理云计算框架体系结构由客户端的应用层，中间的网络层，以及云服务端三大模块构成，如图3.2所示。其中应用层为用户提供访问接口和客户端UI界面，负责本体数据和查询、推理命令的上传及结果显示。网络层基于Internet网络技术实现客户端和云服务端的信息传递[121]。云服务端是框架的核心，负责大规模本体数据的分布式存储，及查询、推理任务的分布式计算。

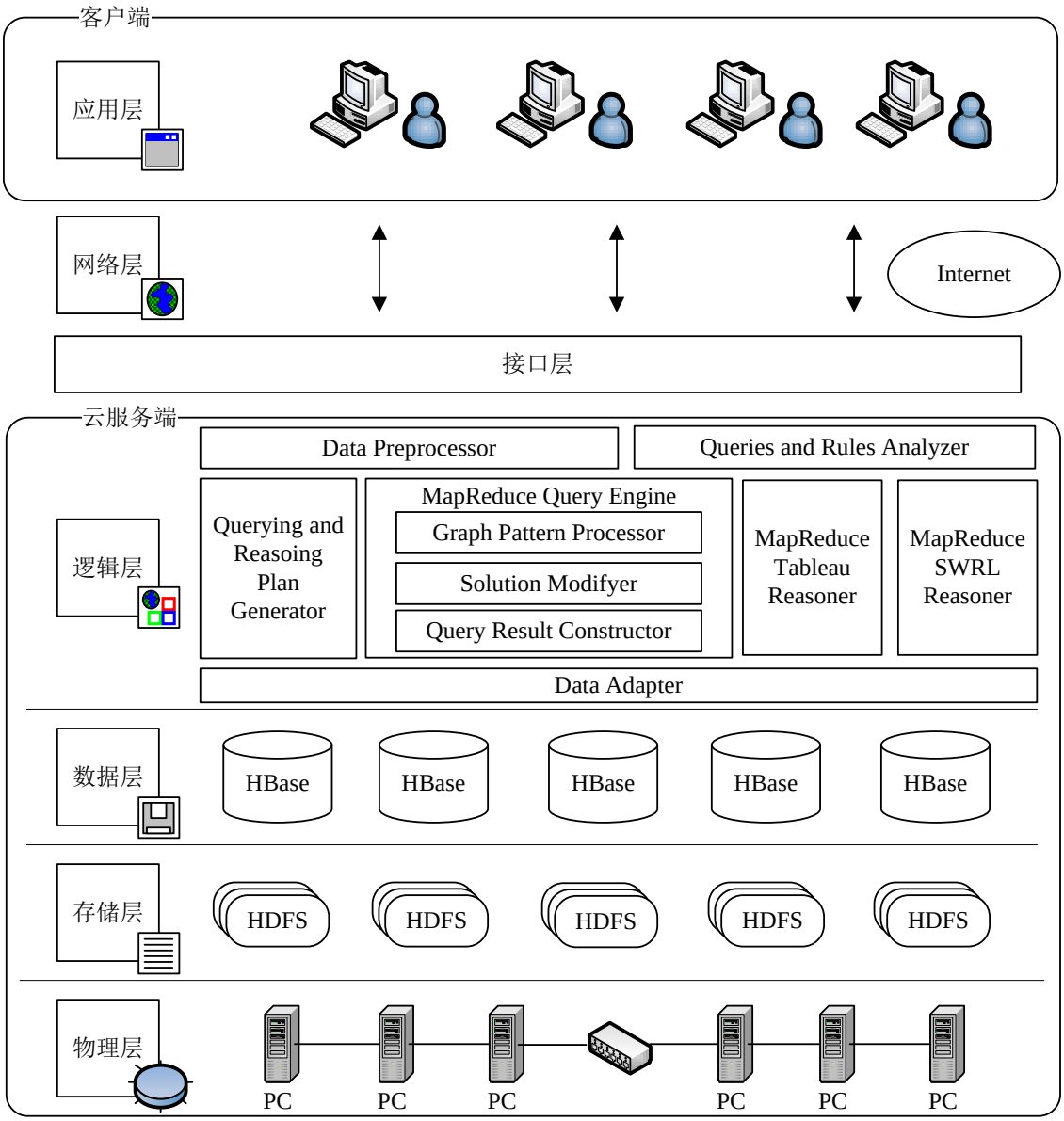


图 3.2 大规模本体查询与推理云计算框架体系结构

Figure 3.2 Architecture of Large-Scaled Ontology Querying and Reasoning Framework based on Cloud Computing Technologies

因本文重点研究该框架下的大规模本体查询与推理关键技术，本小节重点阐述提出框架中云服务端各个模块的架构情况，客户端应用的开发属软件研发工程领域，在此从略。

如图 3.2 所示，在所提出框架中，云服务端设计基于 Hadoop 云计算平台构建，自底向上分别由物理层、存储层、数据层、逻辑层和接口层组成。各功能层次的详细描述如下。

(1) 物理层，指由多台普通计算机和网络交换机构成的局域网，是部署和运行云计算 Hadoop 平台的硬件基础。其中，包含了一台 Master 计算节点，若干台 Slave 计算节点。由于 Hadoop 平台自身具备优秀的可伸缩性，Slave 节点数可根据实际计算任务中涉及的数据量大小任意扩展。在本文后续章节对各种关键技术的研究过程中，使用了一台 Master 节点、八台 Slave 节点的配置来构建 Hadoop 实验云计算环境。

(2) 存储层，指运行于物理层计算机局域网之上的 HDFS 分布式文件系统。该层提供了框架体系结构中 HBase 分布式数据库以及 MapReduce 分布式计算的基础结构，并可实现本体文件以数据块的方式进行分布式存储。本体文件的分发、容错、备份等功能均由 Hadoop 平台自带功能实现。另外，该层还为框架中逻辑层在进行 MapReduce 分布式计算时提供临时文件交换基础。

(3) 数据层，指运行于存储层 HDFS 之上的 HBase 分布式数据库，对用户上传的大规模 RDF 或 OWL 本体数据进行存储与管理，实现对本体数据的随机快速访问、添加、修改及删除。该层 HBase 分布式数据库中本体存储策略将在本章 3.3 小节中进行详细阐述。

(4) 逻辑层，基于 Hadoop 的 MapReduce 分布式计算环境构建，实现对大规模本体数据的查询和推理任务，是本章提出框架的核心组件。如图 3.2 所示，根据语义 Web 本体数据查询与推理任务需求，并以软件高内聚、低耦合的设计规范为基础，该层设计包含了七个子功能模块，其具体功能描述如下。

数据预处理器(Data Preprocessor)基于 Jena API 技术对用户上传的 RDF 或 OWL 数据进行预处理，将基于 N-Triple、N3 或 XML 格式的本体文件转换为框架中对应的本体数据解析模型。数据预处理器设计运行于云服务端 MapReduce 环境下的 Master 计算结点中。

数据适配器(Data Adapter)是逻辑层中实现对底层的数据层和存储层进行数据交换的唯一接口。一方面，数据适配器以数据预处理器转换后的本体数据解析模型为输入，实现本体数据在 HBase 数据库或 HDFS 文件系统中的并行化录入和分布式存储；另外，数据适配器根据框架中的查询和推理计算任务，实现对 HBase 数据库或 HDFS 文件系统中本体数据的并行化访问获取。数据适配器功能模块设

计运行于云服务端的 MapReduce 分布式计算环境中。

查询与规则分析器(Queries and Rules Analyzer)基于 Jena API 将用户输入的 OWL 本体推理任务、SPARQL 查询命令或 SWRL 规则转换为本框架中定义的解析模型。查询与规则分析器设计运行于云服务端 MapReduce 环境下的 Master 计算结点中。

查询与推理任务生成器(Querying and Reasoning Plan Generator)以查询与规则分析器输出的解析模型为输入，根据本文第四章提出的 SPARQL 查询任务生成算法以及本文第六章提出的 SWRL 规则推理任务生成算法，生成相应的查询和推理任务模型。查询与推理任务生成器设计运行于云服务端 MapReduce 环境下的 Master 计算结点中。

MapReduce 查询引擎(MapReduce Query Engine)根据 MapReduce 查询任务模型，在 MapReduce 环境下执行大规模 RDF 数据的 SPARQL 分布式查询。该模块由三个子模块组成构成，分别是：图模式处理器(Graph Pattern Processor)、解修改器(Solution Modifier)和查询结果构造器(Query Result Constructor)。根据本文第四章提出的基于 MapReduce 的 SPARQL 复杂组图模式分布式推理算法，图模式处理器以查询计划模型为基准，迭代地执行一组 MapReduce 查询任务，以实现 MapReduce 环境下的 SPARQL 复杂组图模式匹配；解修改器根据 SPARQL 查询条件中的顺序限制符对图模式处理器查询生成的匹配解(Solutions)进行从新排序；最终查询结果构造器根据输入 SPARQL 查询条件中的查询表单生成查询结果。MapReduce 查询引擎设计运行于云服务端的 MapReduce 环境中。

MapReduce Tableau 推理引擎(MapReduce Tableau Reasoner)是在 MapReduce 环境下的 Tableau 算法分布式本体推理机。该模块根据本文第五章提出的基于 MapReduce 的 Tableau 分布式推理算法，以数据层中的大规模 OWL 本体数据为输入，首先生成相应的推理模型，并对 OWL 本体的一致性进行验证。MapReduce Tableau 推理引擎设计运行于云服务端的 MapReduce 环境中。

MapReduce SWRL 推理引擎(MapReduce SWRL Reasoner)是 MapReduce 环境下的 SWRL 规则分布式推理机。该模块以框架数据层中大规模 OWL 本体数据和查询与规则分析器生成的 SWRL 规则模型为输入，并根据查询与推理任务生成器生成的推理计划以及 MapReduce 环境下的 SWRL 规则分布式推理算法，执行一组 MapReduce 任务实现 SWRL 规则的分布式规则推理，并返回推理结果。MapReduce SWRL 推理引擎设计运行于云服务端的 MapReduce 环境中。

(5) 接口层，是云服务端与外界实现信息传递和数据交换的接口，负责云服务端的负载均衡以及访问的身份验证、权限分配以及云计算安全监控与管理等。接口层的关键技术超出了本文的研究范围，在此不做过多论述。

上述框架体系结构在具备基本的本体数据存储、查询和推理功能特性的同时，还存在以下特性：(1) 基于现有 Internet 和云计算 Hadoop 开源平台等关键技术构建，保证框架的可行性；(2) 框架从云计算涉及的硬件配置、文件系统、数据库存储及分布式计算等不同维度进行分层架构，保证了体系结构的全面性；(3) 以语义 Web 查询与推理工具的成熟体系结构为基础，并充分结合 Hadoop 技术特性，保证了体系结构的科学性。

3.3 大规模语义 Web 本体存储策略

目前，语义 Web 研究领域主要以文件系统和数据库的方式对本体进行存储。其中，基于文件系统的存储方式通过将本体文件解析入计算机内存，在进行本体数据查询和推理时具有较高的效率，但该种方式每次都需要载入所有本体数据，不易进行数据的随机访问和添加修改，因此仅适用于对小规模本体数据的存储。

另一方面，关系型数据库技术已得到长足的发展和广泛的应用，在数据的随机访问、数据容错性，数据一致性、完整性、安全性、事务处理等方面均具有成熟的软件产品支持，其已成为当前语义 Web 领域进行本体存储的主要方式。

目前，研究人员已提出三元组表模式、垂直存储模式、水平存储模式、模式生成等基于关系数据库的本体存储策略。但由于关系型数据库通常部署在单机环境下运行，且数据表结构需要在数据库设计初期确定，因此，其很难适应大规模本体数据的可扩展性和模式动态变化需求。

随着语义 Web 和云计算技术融合研究的不断深入，已有研究人员提出了将本体数据存储于云计算 HBase 分布式数据库的存储策略，但其仅针对于面向 RDF 数据的 SPARQL 基本图模式查询需求，在数据存储空间开销和查询效率的平衡方面还有待改进。因此，本文面向本体的 SPARQL 复杂组图模式查询、SWRL 规则推理和基于描述逻辑的一致性验证，并结合 HBase 表模式特性，提出一种新的、针对大规模本体数据的存储策略。

3.3.1 基于 HBase 的本体存储策略

在语义 Web 体系结构中，RDF 三元组(s, p, o)是本体的基本数据表示模型，即虽然 RDF 与 OWL 在语义和表达能力方面上存在差异，但是 OWL 本体可采用 RDF 三元组格式进行描述。在本文所提出框架中，结合 SPARQL 复杂组图模式查询匹配和基于描述逻辑的 OWL 本体及其规则推理的特性，设计了针对于 RDF 数据和 OWL 本体的 HBase 数据表存储策略。

首先，在面向大规模 RDF 数据的分布式存储和查询方面，设计了三个 HBase 表来满足所有可能组合形式的 SPARQL 三元组模式查询匹配条件需求，分别是：表 T_SP_O、T_PO_S 和 T_OS_P，表结构分别如图 3.3、图 3.4 和图 3.5 所示，其

中 S 表示主语，P 表示谓语，O 表示宾语，m 和 n 分别为 HBase 表中列数和行数，且 $m, n \geq 0$。

表 T_SP_O 将主语值和谓语值的有序对<S, P>联合作为行键，其对应的 n 个宾语值 O_n 作为列名包含于一个列族中，每个表单元设计为空值。表 T_PO_S 和 T_OS_P 具有与表 T_SP_O 相似的结构，分别将谓语和宾语、宾语和主语的有序对作为行键，对应的主语值和谓语值作为列名，并存在于一个列族中。

Row Key	Column Family				Timestamp
	O_1	O_2	...	O_n	
<S_1 , P_1>	*null*	*null*	...	*null*	Time T_1
<S_2 , P_2>	*null*	*null*	...	*null*	Time T_2
⋮	⋮	⋮	⋮	⋮	⋮
<S_m , P_m>	*null*	*null*	...	*null*	Time T_m

图 3.3 HBase 表 T_SP_O 存储结构

Figure 3.3 Storage Structure of HBase Table T_SP_O

Row Key	Column Family				Timestamp
	S_1	S_2	...	S_n	
<P_1 , O_1>	*null*	*null*	...	*null*	Time T_1
<P_2 , O_2>	*null*	*null*	...	*null*	Time T_2
⋮	⋮	⋮	⋮	⋮	⋮
<P_m , O_m>	*null*	*null*	...	*null*	Time T_m

图 3.4 HBase 表 T_PO_S 存储结构

Figure 3.4 Storage Structure of HBase Table T_PO_S

Row Key	Column Family				Timestamp
	P_1	P_2	...	P_n	
<O_1 , S_1>	*null*	*null*	...	*null*	Time T_1
<O_2 , S_2>	*null*	*null*	...	*null*	Time T_2
⋮	⋮	⋮	⋮	⋮	⋮
<O_m , S_m>	*null*	*null*	...	*null*	Time T_m

图 3.5 HBase 表 T_OS_P 存储结构

Figure 3.5 Storage Structure of HBase Table T_OS_P

本章所提出的 RDF 存储策略虽然需要将数据复制三次，但云计算环境下的数据存储空间开销可以通过扩展计算节点数来平衡，并且由于上述所有的 HBase 表中数据单元均为空值，根据 HBase 数据库对数据单元中空值不占用存储空间的特性，所提出存储策略也可减少数据冗余和节省不必要的数据空间开销。另外，作为 SPARQL 查询基本单元的三元组模式有八种可能的变量组合形式，通过采用上述存储策略，每种查询方式都可快速、直接地进行匹配。

表 3.1 描述了不同种 SPARQL 三元组模式查询变量条件和上述 HBase 表之间的查询映射关系，其中?*s*、?*p* 和?*o* 分别表示出现在主语、谓语和宾语位置的变量，*s*、*p* 和 *o* 表示常量。

表 3.1 SPARQL 三元组模式查询条件与提出存储策略中 HBase 表映射关系

Table 3.1 The Querying Relationship between the Proposed Storage Schema and Triple Patterns

HBase 表	Triple Patterns
T_SP_O	(*s*, *p*, ?*o*), (*s*, ?*p*, ?*o*), (*s*, *p*, *o*), (?*s*, ?*p*, ?*o*)
T_PO_S	(?*s*, *p*, *o*), (?*s*, *p*, ?*o*)
T_OS_P	(*s*, ?*p*, *o*), (?*s*, ?*p*, *o*)

基于表 3.1 所示的映射关系，当对三元组模式 (?*s*, *p*, *o*)、(*s*, ?*p*, *o*)或(*s*, *p*, ?*o*)进行查询时，可分别将其中两个已知 RDF 术语设定为检索条件，并分别对表 T_PO_S、T_OS_P 和 T_OS_P 的行键进行匹配；当对三元组模式(*s*, *p*, *o*)或(?*s*, ?*p*, ?*o*)进行查询时，前者需要验证对应的 RDF 三元组数据是否存在表 T_SP_O 中，后者则仅需检索出表 T_SP_O 中的所有 RDF 三元组数据；当对三元组模式(?*s*, ?*p*, *o*)、(?*s*, *p*, ?*o*)或(*s*, ?*p*, ?*o*)进行查询时，可利用 HBase 数据库中 PrefixFilter 技术包含的 Scan 区域检索和 Get 数据检索机制，分别通过为表 T_OS_P、T_PO_S 或 T_SP_O 指定检索条件为匹配所有 *o*、*p* 和 *s* 的行键区间，实现其余两项变量的查询。

另外，在面向大规模 OWL 本体数据的存储和推理方面，结合 SWRL 规则推理和基于 Tableau 算法的描述逻辑推理计算特性，设计将大规模的 OWL 个体断言集存放于表 T_SP_O 和 T_PO_S 中。此外，由于 OWL 术语集通常数据量较小，设计将其直接存储于 HDFS 文件系统中。

根据 SWRL 规则语法，由于 SWRL 规则原子 *C*(*x*)和 *P*(*x*, *y*)与 OWL 类和属性之间存在一一对应关系，而基于 RDF 的 OWL 可用三元组形式描述，且具备相同的语义。另外，在进行 SWRL 规则推理时需要首先检索到满足各个规则原子的 OWL 本体个体，而规则原子因变量位置的不同，可能出现四种不同的表达形式。

表 3.2 描述了 SWRL 规则原子的三元组形式语义映射关系，以及基于上述存

储策略进行 SWRL 规则原子的本体数据检索时，各类规则原子对应的 HBase 数据表，其中?*x* 和?*y* 表示变量，*a* 和 *b* 表示 OWL 个体或常量。

对规则原子 *C*(?*x*)，由于其谓语和宾语分别已知为 rdf:type 和 OWL 概念 *C*，因此可以将 rdf:type 和 *C* 作为检索条件，与表 T_PO_S 的行键进行匹配，进而得出满足概念 *C* 的所有 OWL 个体。同理，对规则原子 *P*(?*x*, ?*y*)、*P*(*a*, ?*y*)和 *P*(?*x*, *b*)，分别将规则中已知项作为检索条件，并与提出存储策略中对应 HBase 表的行键进行匹配，可查询出满足条件的三元组数据。

表 3.2 SWRL 规则原子的三元组语义映射及其对应的 HBase 表

Table 3.2 Semantics Mapping from SWRL Atoms to Triples and HBase Tables

SWRL 规则原子	三元组表示	HBase 数据表
C(?*x*)	(?*x*, rdf: type, *C*)	T_PO_S
P(?*x*, ?*y*)	(?*x*, *P*, ?*y*)	T_PO_S
P(*a*, ?*y*)	(*a*, *P*, ?*y*)	T_SP_O
P(?*x*, *b*)	(?*x*, *P*, *b*)	T_PO_S

在进行 OWL 本体的一致性验证时，由于需要根据本体库中 OWL 术语对所有 OWL 个体断言进行验证，因此可依据本文提出存储策略的 HBase 表 T_SP_O 或 T_PO_S 中检索出所有断言，然后根据 HDFS 中本体术语集进行计算。

3.3.2 存储策略对比与分析

在基于关系型数据库的本体存储策略方面，三元组表模式是最简单的存储组织方案，所有本体数据直接存储在一张由三个列构成的表中，分别对应 RDF 三元组的主语、谓语和宾语。初期的很多语义 Web 系统均采用该方案,如 Sesame, Jena, CODE[122],YARS[123]等。但这种策略在处理 SPARQL 查询时将存在大量自连接的 SQL 查询，在存储大规模本体数据时随着自连接次数增加，查询时间也会随之显著增长，存在系统查询效率低的问题。

垂直存储方案是在三元组表存储模式基础之上的一种优化。它根据谓语对三元组表进行划分，将拥有相同谓语的三元组存储到同一张表中。由于该表中所有三元组的谓语都相同，因此将 RDF 三元组谓语作为表名，表中仅保留两列，分别对应主语和宾语，例如 CODERS[125]就采用了垂直存储方案。图 3.6 描述了垂直存储方案的表结构的示例。

垂直存储方案由于不用对 RDF 三元组谓词进行存储，可在一定程度上降低存储空间开销，并由于其将数据拆分到多个表中进行存储，降低了单个数据表对大规模本体数据进行存储时所带来的存储和数据查询压力，同时当一个资源在某个

谓词上存在多个属性值时，可以在该谓词所对应的数据表中进行多行存储，进而也避免了空值和多属性值问题。

然而，当存在新的三元组谓语时，则需要创建新的表结构来存储三元组数据，即为每个新谓词添加一个数据表，因此垂直存储方案会造成数据库中存在大量的数据表，增加数据库维护的难度。同时在进行不同数据表之间的数据连接查询操作时，会存在过多的表之间连接操作，造成了查询效率的下降。

cd:artist

S	O
cd:Empire Burlesque	Bob Dylan
cd:Hide your heart	Bonnie Tyler

cd:country

S	O
cd:Empire Burlesque	USA
cd:Hide your heart	UK

cd:company

S	O
cd:Empire Burlesque	Columbia
cd:Hide your heart	CBS Records

cd:price

S	O
cd:Empire Burlesque	10.90
cd:Hide your heart	9.90

图 3.6 RDF 数据的垂直存储方案

Figure 3.6 The Vertical Partition of RDF Data

水平存储方案将所有的谓词作为列名存储列在一张数据表中，即表中第一列存储 RDF 三元组的主语，与该主语相关的谓语设定为列名，对应的宾语值存储在数据单元中。图 3.7 描述了水平存储方案的表结构示例。

Subject	cd:artist	cd:country	cd:company	cd:price
cd:Empire Burlesque	Bob Dylan	USA	Columbia	10.90
cd:Hide your heart	Bonnie Tyler	UK	CBS Records	9.90
cd:Beatles	null	Japanese	null	null

图 3.7 RDF 数据的水平存储方案

Figure 3.7 The Horizontal Storage of RDF Data

水平存储方案能够一次性地对与某一主语关联的本体数据进行查询，进而避免了多表连接或自连接的问题，该方案能够提高查询的效率。目前在语义 Web 数据管理研究中尚未见到采用这种存储方案，而在 Web 数据存储的研究中，已有研究者采用这种方案存储 Web 数据[126]。由于语义 Web 数据是 Web 数据的一种扩展，因此从长远来看，这种方案也会成为一种选择。然而，当某个资源在某列属性上

不存在值时，其对应的数据单位值为空。RDF 数据本身的特点就是半结构化、数据稀疏，不同资源的属性存在较大差异，水平存储方案会存在大量的空值，并且当某个资源在某个属性上存在多值时会使得存储结构较为复杂。

水平存储方案在一定程度上避免了自连接和多表连接的问题，但由于将数据存储在一张表中又会造成表数据量过大，进而影响查询效率，同时该方案会存在大量空值。垂直存储方案避免了空值问题，但会存在大量的数据表，且在查询过程中存在大量多表连接，互相之间不能权衡。于是有研究人员提出了模式生成方案，即根据 RDF 数据的空值分布情况将三元组拆分到多个表中进行存储，即把常用的属性或大多数三元组资源共有的属性存储在一个表中，不常用属性则存储在另一张表中。模式生成方案需要对表做垂直切分和水平切分，形成多个表，使得切分后各表内的空值较少。通过这种切分机制，一方面可以降低空值的存储开销，另一方面，本体数据存储在多张表中，而每张表中数据量的减少又降低了查询连接操作的计算开销。文献[127]描述了模式生成方案在 Jena 系统中的设计和实现，它将三元组信息转换成一些属性表。但是，模式的生成任务需要由数据库管理员人工执行，因此该方案会存在过多的人工干预，不适用于针对海量本体数据的管理任务。

目前，由于学术界关于云计算环境下的大规模本体数据管理研究仍处于起步阶段，仅有 Sun 等人[114]和 Franke 等人[115]提出了基于 HBase 的本体数据分布式存储方案。

其中，Sun 等人基于 Hexaxtore 的存储策略，并根据三元组主谓宾的不同组合形式设计了六个 HBase 表：S_PO、P_SO、O_SP、PS_O、SO_P 和 PO_S，用于匹配各种可能的 SPARQL 三元组模式。其中，表 S_PO 以 RDF 三元组的主语值作为行键，以谓语和宾语值的数据对作为列名。表 P_SO 和 O_SP 有与 S_PO 相似的表结构，分别将谓语和宾语作为行键，其对应的<主语, 宾语>数据对和<主语, 谓语>对作为列名。表 PS_O、SO_P 和 PO_S 与本文提出存储策略相似。但是，在他们所提出解决方案中，RDF 数据需要被复制六次进行存储，进而增大了存储空间的开销。

Franke 等人设计了使用两张 HBase 数据表 T_{sp} 和 T_{op} 的方法对 RDF 数据进行存储。图 3.8 和 3.9 分别描述了其表数据组织结构。其中，表 T_{sp} 将三元组主语值作为行键，谓词值作为列名，对应的宾语值存储在数据单元中。相似的，表 T_{op} 将三元组宾语值作为行键，谓语值作为列名，对应的主语值在数据单元中进行存储。基于上述存储策略，RDF 数据将会被复制两次进行存储。另外，在进行 SPARQL 三元组模式匹配时，当主语或宾语处为变量时，可分别对表 T_{sp} 和 T_{op} 的行键进行匹配。但当三元组模式为中谓词为变量时，则需对其中任一表进行全盘扫描，并

返回表中所有本体数据，必然会对查询过程带来额外开销并影响其性能。

Row Key	Column Family				Timestamp
	P_1	P_2	...	P_n	
S_1	$O_{(1, 1)}$	$O_{(1, 2)}$	...	$O_{(1, n)}$	Time T_1
S_2	$O_{(2, 1)}$	$O_{(2, 2)}$	...	$O_{(2, 2)}$	Time T_2
⋮	⋮	⋮	⋮	⋮	⋮
S_m	$O_{(m, 1)}$	$O_{(m, 2)}$	...	$O_{(m, n)}$	Time T_m

图 3.8 Franke 等人提出的表 T_{sp} 存储结构

Figure 3.8 Storage Structure of Table T_{sp} Proposed by Franke et al.

Row Key	Column Family				Timestamp
	P_1	P_2	...	P_n	
O_1	$S_{(1, 1)}$	$S_{(1, 2)}$	...	$S_{(1, n)}$	Time T_1
O_2	$S_{(2, 1)}$	$S_{(2, 2)}$	...	$S_{(2, 2)}$	Time T_2
⋮	⋮	⋮	⋮	⋮	⋮
O_m	$S_{(m, 1)}$	$S_{(m, 2)}$	...	$S_{m, n)}$	Time T_m

图 3.9 Franke 等人提出的表 T_{op} 存储结构

Figure 3.9 Storage Structure of Table T_{op} Proposed by Franke et al.

综上所述，不难得出结论，本章所提出的本体数据存储策略，由于充分考虑了数据存储与 SPARQL 查询、OWL 一致性验证和规则推理性能之间的平衡问题，进而更适应于大规模本体数据的管理。

3.4 本章小结

本章以现有语义 Web 本体数据存储、查询和推理理论为基础，结合云计算 HDFS 分布式文件系统、MapReduce 分布式计算环境和 HBase 分布式数据库技术特性，提出了大规模本体分布式查询与推理云计算框架体系结构，阐述了框架中各个层次和模块的功能及其相互之间的逻辑关系。然后，以实现大规模本体数据的 SPARQL 复杂组图模式查询、SWRL 规则推理和基于描述逻辑的一致性验证为目的，提出了基于 HBase 的本体数据存储策略，最后对已有的方法和研究成果进行了对比分析。

结果表明，本章所提出框架为大规模本体数据管理中存储、查询与推理核心任务提供了较为全面的支持，所提出基于 HBase 的存储策略能够在本体数据存储

空间开销和查询推理性能之间找到平衡点，能够为实现大规模本体数据管理云服务提供体系结构和理论方法支撑。

4 基于 MapReduce 的 SPARQL 复杂组图模式分布式查询

4.1 引言

如本文第二章所述，基于 SPARQL 的 RDF 本体数据查询是语义 Web 领域研究的核心内容之一。随着语义 Web 的快速发展和应用，传统单机环境下运行的 SPARQL 查询引擎已无法适应大规模 RDF 数据的查询任务需求，学术界已开始将语义 Web 与云计算 Hadoop 技术相结合，研究如何实现高性能、可扩展的大规模 RDF 数据查询，并已提出了基于 MapReduce 的 SPARQL 基本图模式分布式查询方法，但在针对 SPARQL 复杂组图模式查询方面仍未开展研究。

本章的工作主要是基于 W3C 提出的 RDF 和 SPARQL 形式化语义，结合本文第三章中论述的框架体系结构和基于 HBase 的 RDF 数据存储策略，提出一种新的基于 MapReduce 的 SPARQL 复杂组图模式分布式查询方法。

首先，本章 4.2 小节对 RDF 和 SPARQL 的形式化语义进行了描述，并给出了面向 SPARQL 复杂组图模式查询时的相关解析模型定义；其次，为尽可能减少 MapReduce 查询任务数，实现查询性能的优化，本章 4.3 小节提出了基于 MapReduce 的 SPARQL 复杂组图模式查询任务生成算法，并以此为基础，在 4.4 小节设计了 MapReduce 模型下 map 和 reduce 函数中的 SPARQL 复杂组图模式分布式查询算法；最后通过使用语义 Web 研究领域广泛采用的 SP2Bench 本体公共测试数据集，在 4.5 小节中对本章所提出方法和现有单机环境下的 SPARQL 查询引擎进行了实验对比和分析。

4.2 相关术语及解析模型

4.2.1 SPARQL 语法及其形式化语义

作为语义 Web 体系结构中本体数据描述基础模型的 RDF 三元组具有以下的形式化语义。

假设存在三个有序不相交的有限集合 I，B 和 L，其中 I 是 URI 的集合，B 是空结点的集合，L 是字面量的集合。称三元组$(s, p, o)\in(I\cup B)\times I\times(I\cup B\cup L)$为 RDF 三元组，其中 s 表示主语，p 表示谓语，o 表示宾语。称集合$(I\cup B\cup L)$中的任一元素为一个 RDF 术语(RDF term)。

根据 W3C 的 SPARQL 语法结构定义，位于 SPARQL 查询语句 *WHERE* 片段的查询模式基于图模式对目标 RDF 数据进行匹配，是 SPARQL 查询语句的核心。在此基于文献[19]中提出的 AND、UNION、OPTIONAL 和 FILTER 二元运算符，给出 SPARQL 复杂组图模式表达式的形式化描述。

假设存在一个 SPARQL 变量的有限集合 V，且 V 与集合 I，B 和 L 不相交，SPARQL 图模式表达式可迭代地定义为：

① 若存在三元组$(s,\ p,\ o)\in(I\cup L\cup V)\times(I\cup V)\times(I\cup L\cup V)$，称该三元组为一个 SPARQL 图模式或三元组模式。

② 如果存在 P_1 和 P_2 是图模式，那么称表达式$(P_1$ AND $P_2)$、$(P_1$ OPTIONAL $P_2)$和$(P_1$ UNION $P_2)$为复杂组图模式。

③ 如果 P 是图模式，且 R 是一个 SPARQL 内置条件，则表达式$(P$ FILTER $R)$是一个图模式。其中，R 可由集合$(I\cup L\cup V)$的元素、常数、逻辑连接符$(\neg, \wedge, \vee)$、计算符$(<, \leq, >, \geq, =)$和一元谓词(bound、isBlank、isIRI)构成。

在提出 SPARQL 复杂组图模式的形式化语义之前，先阐述以下概念的定义。

① 设存在一个映射函数 $u:V\rightarrow T$，即 u 将集合 V 中元素映射到集合 T；u 的定义域 $dom(u)\subseteq V$。

② 对任一 SPARQL 三元组模式 t，存在 SPARQL 变量 $v\in t$，$u(t)$为根据映射函数 u 将 v 映射为对应 RDF 术语的三元组。

③ 有两个映射函数 u_1 和 u_2，当所有 $x\in dom(u_1)\cap dom(u_2)$满足 $u_1(x)=u_2(x)$时，称 u_1 和 u_2 兼容(Compatible)；当 u_1 和 u_2 不相交时，u_1 和 u_2 也兼容；对某一映射函数 $u_\varnothing$，当 $dom(u_\varnothing)$为空时，$u_\varnothing$与其它所有映射函数兼容。

④ 令 Ω_1 和 Ω_2 为映射函数的集合，Ω_1 和 Ω_2 的连接(Join)、并集(Union)、差集(Different)和左外连接(Left Outer-Join)操作分别定义为符号：$\bowtie$、$\cup$、$\setminus$和$:\bowtie$，且具有语义：

① $\Omega_1 \bowtie \Omega_2 = \{\ u_1\cup u_2 \mid u_1\in\Omega_1, u_2\in\Omega_2$ 是兼容映射函数 $\}$；

② $\Omega_1 \cup \Omega_2 = \{u \mid u_1\in\Omega_1$ 或者 $u_2\in\Omega_2\ \}$；

③ $\Omega_1 \setminus \Omega_2 = \{\ u\in\Omega_1 \mid$ 对$\forall u'\in\Omega_2$，u 和 u'是不兼容函数 $\}$；

④ $\Omega_1 :\bowtie \Omega_2 = (\Omega_1 \bowtie \Omega_2) \cup (\Omega_1 \setminus \Omega_2)$。

基于上述概念定义，SPARQL 复杂组图模式及其 AND、UNION 和 OPTIONAL 运算符的形式化语义可描述为：

① 设 D 是 T 中的一个 RDF 数据集，t 为一个 SPARQL 三元组模式，P_1 和 P_2 为 SPARQL 图模式，记$[\cdot]_D$为图模式$(\cdot)$在 D 上映射结果的匹配解；

② $[t]_D = \{u \mid dom(u) = \mathrm{var}(t)$，且 $u(t)\in D$，其中 $\mathrm{var}(t)$是 t 中变量的集合$\}$；

③ $[(P_1$ AND $P_2)]_D = [P_1]_D \bowtie [P_2]_D$；

④ $[(P_1$ OPTIONAL $P_2)]_D = [P_1]_D :\bowtie [P_2]_D$；

⑤ $[(P_1$ UNION $P_2)]_D = [P_1]_D \cup [P_2]_D$。

对表达式$(P_1$ OPTIONAL $P_2)$，令 u_1 为$[P_1]_D$ 中的一个映射，如果存在一个映射 $u_2\in[P_2]_D$ 且 u_1 和 u_2 兼容，则$(u_1\cup u_2)\in[P_1$ OPTIONAL $P_2]_D$。但如果没有这样一个

映射 u_2 存在，那么 $u_1 \in [P_1 \text{ OPTIONAL } P_2]_D$。因此，运算符 OPTIONAL 允许将可能存在的检索到信息添加到一个映射 u 中。

另外，给定一个映射函数 u 和一个内置条件 R，当满足以下条件时称 u 满足 R，记为 $u \vDash R$，如果：

① R 为 bound(?x)，且?$x \in dom(u)$；

② R 为?$x = c$，?$x \in dom(u)$且 $u(?x) = c$；

③ R 为?x = ?y，?$x \in dom(u)$，?$y \in dom(u)$，且 $u(?x) = u(?y)$；

④ R 为$(\neg R_1)$，R_1 为内置条件，且不满足 $u \vDash R_1$；

⑤ R 为$(R_1 \vee R_2)$，R_1 和 R_1 为内置条件，且 $u \vDash R_1$ 或 $u \vDash R_2$；

⑥ R 为$(R_1 \wedge R_2)$，R_1 和 R_1 为内置条件， $u \vDash R_1$ 且 $u \vDash R_2$。

因此，FILTER 表达式的语义可定义为：给定一个 RDF 数据集 D，一个图模式 P，及一个 FILTER 表达式(P FILTER R)，那么，

$$[(P \text{ FILTER } R)]_D = \{ u \in [P]_D \mid u \vDash R \}。$$

对任意图模式 P_1、P_2、P_3 和内置条件 R，运算符 AND、UNION 和 OPTIONAL 满足以下计算性质，本文在此对该定理不作详细证明，其证明过程见文献[19]。

① 运算符 AND 和 UNION 可交换并可结合，OPTIONAL 不可交换；

② $(P_1 \text{ AND } (P_2 \text{ UNION } P_3)) \equiv ((P_1 \text{ AND } P_2) \text{ UNION } (P_1 \text{ AND } P_3))$；

③ $(P_1 \text{ OPTIONAL } (P_2 \text{ UNION } P_3)) \equiv ((P_1 \text{ OPTIONAL } P_2) \text{ UNION } (P_1 \text{ OPTIONAL } P_3))$；

④ $((P_1 \text{ UNION } P_2) \text{ OPTIONAL } P_3) \equiv ((P_1 \text{ OPTIONAL } P_3) \text{ UNION } (P_2 \text{ OPTIONAL } P_3))$；

⑤ $((P_1 \text{ UNION } P_2) \text{ FILTER } R) \equiv ((P_1 \text{ FILTER } R) \text{ UNION } (P_2 \text{ FILTER } R))$。

为更好地说明上述 SPARQL 复杂组图模式的计算语义，图 4.1 中描述了示例 RDF 数据集 S。

(A, name, *a*),	(A, email, a@cqu.edu.cn)
(B, name, *b*),	(B, email, b@cqu.edu.cn)
(C, name, *c*),	(C, webpage, http://c.com)
(D, name, *d*),	(D, phone, 666666)
(D, webpage, http://d.com),	(D, email, d@cqu.edu.cn)

图 4.1 示例 RDF 数据集 S

Figure 4.1 A Sample RDF Dataset S

对以下四个 SPARQL 图模式 P_1、P_2、P_3 和 P_4，设其均为 SELECT 查询方式，并返回所有变量对应 RDF 术语映射结果的匹配解，其针对 S 的检索结果$[P_1]_S$、$[P_2]_S$、$[P_3]_S$ 和$[P_4]_S$，如图 4.2 所示。

(1) P_1= (?*X*, name, ?*Y*) AND (?*X*, webpage, ?*Z*)；

(2) P_2= (?*X*, name, ?*Y*) OPTIONAL (?*X*, phone, ?*Z*)；

(3) P_3= (?*X*, name, ?*Y*) UNION (?*X*, phone, ?*Z*)；

(4) P_4= ((?*X*, name, ?*Y*) OPTIONAL (?*X*, phone, ?*Z*)) FILTER ¬ bound(?*Z*)。

$[P_1]_S$ =

	?*X*	?*Y*	?*Z*
u_1:	C	*c*	http://c.com
u_2:	D	*d*	http://d.com

(a)

$[P_1]_S$ =

	?*X*	?*Y*	?*Z*
u_1:	A	*a*	
u_2:	B	*b*	
u_3:	C	*c*	
u_4:	D	*d*	666666

(b)

$[P_1]_S$ =

	?*X*	?*Y*	?*Z*
u_1:	A	*a*	
u_2:	B	*b*	
u_3:	C	*c*	
u_4:	D	*d*	
u_5:	D		666666

(c)

$[P_1]_S$ =

	?*X*	?*Y*	?*Z*
u_1:	A	*a*	
u_2:	B	*b*	
u_3:	C	*c*	

(d)

图 4.2 示例 SPARQL 图模式的查询结果

(a) P_1 查询结果；(b) P_2 查询结果 (c) P_3 查询结果；(d) P_4 查询结果

Figure 4.2 Query Results of Sample SPARQL Graph Patterns

(a) Query results of P_1; (b) Query results of P_2; (c) Query results of P_3; (d) Query results of P_4

4.2.2 相关解析模型

由于 MapReduce 分布式计算模型要求所有数据在键值对模型下进行处理，因此在进行基于 SPARQL 的 RDF 数据查询时需要对相关 RDF 和 SRARQL 查询语句解析为适应于 MapReduce 的数据模型。

在详细论述基于 MapReduce 的 SPARQL 复杂组图模式分布式查询方法之前，本小节描述框架中关于相关的解析模型定义。另外，为更好地进行描述，选取 SP2Bench 公共测试数据集中提出的标准测试查询语句 Query 8 作为示例，并根据

先后顺序对其中的三元组模式标识为 TP_1，TP_2,...，TP_8，将内置条件分别标识为 R_1 和 R_2，如图 4.3 所示。其中，TP_3 和 TP_4 参与了两次查询计算。

```
SELECT DISTINCT ?name
WHERE {
   ?erdoes rdf:type foaf:Person. (TP1)
   ?erdoes foaf:name "Paul Erdoes"^^ xsd:string. (TP2)
   {
      ?document dc:creator ?erdoes. (TP3)
      ?document dc:creator ?author. (TP4)
      ?document2 dc:creator ?author. (TP5)
      ?document2 dc:creator ?author2. (TP6)
      ?author2 foaf:name ?name (TP7)
      FILTER (?author!=?erdoes &&?document2!=?document
        && ?author2!=?erdoes && ?author2!=?author) (R1)
   } UNION{
      ?document dc:creator ?erdoes. (TP3)
      ?document dc:creator ?author. (TP4)
      ?author foaf:name ?name (TP8)
      FILTER (?author!=?erdoes) (R2)
   }
}
```

图 4.3 SP2Bench 标准测试查询语句 Query 8

Figure 4.3 Query 8 in SP2Bench Benchmark

根据 SPARQL 语法和语义，文献[68]分析了 AND、UNION 和 OPTIONAL 运算符查询的特性，并提出可将 SPARQL 查询语句解析为一个操作树结构的思想。因此，本文结合 MapReduce 模型特性，提出以下三种数据模型用于对所提出框架中 SPARQL 语句的三个组成部分分别进行描述。

定义 4.1：图模式树(Graph Pattern Tree，GPT)，是一个用于描述 SPARQL 语句 *WHERE* 片段中所对应复杂组图模式的树结构。GPT 树模型中每个树节点可为三元组模式节点(Triple Pattern Node, TPN)、元组节点(Tuple Node, TN)或运算符节点(Operator Node, OPN)，其中，

- TPN 是一个三元组(s, p, o)，用于表示 SPARQL 查询语句中的任一三元组模式，且 TPN 的每个组元 s、p 和 o 可以为变量或常量值。
- TN 是一个 n+1 元组(t_1, t_2, ..., t_n, *JobN*)，n≥1，用于描述图模式 P 针对 RDF 数据集 D 映射结果的匹配解$[P]_D$中的所有变量，其中 t_i 是一个 SPARQL 变量名，1≤i≤n；另外，每个 TN 的属性值 *JobN*，用于存储该 TN 节点所对应的 MapReduce 任务数序号。
- OPN 对应于 SPARQL 运算符 AND、UNION 或 OPTIONAL，并且每个 OPN

可附 *m* 个 FILTER 内置条件，*m*≥0；当某一 OPN *O* 的所有子节点均为 TPN 节点时，称 *O* 为一个叶 OPN 节点(Leaf OPN，LF-OPN)。

- 定义 Depth(*LF-OPN*)为任一 L-OPN 节点在其所存在的 GPT 树 *T* 中的深度值，Tree-Depth(*T*)为的 *T* 中所有 L-OPN 的最大深度值。即对任一 GPT 树 *T*，若 *T* 中存在 *x* 个 L-OPN 节点：$LF\text{-}OPN_1$, $LF\text{-}OPN_2$, ..., $LF\text{-}OPN_x$，*x*≥1，那么 Tree-Depth(*T*) = Max{ Depth($LF\text{-}OPN_1$),..., Depth($LF\text{-}OPN_x$) }。

定义 4.2：解序列修改器(Solution Sequence Modifier，SSM)，SSM=<*Modifier*, *Parameters*>，对应于 SPARQL 查询语句中的解序列修改指令。其中，参数 *Modifier* 可以赋值为操作符 Distinct、Order By、Projection、Reduced、Offset 或 Limit；参数 *Parameters* 为 *Modifier* 对应的操作对象。

定义 4.3：查询构成器(Query Form，QF)，是一个对 SPARQL 复杂组图模式的匹配解生成最终查询结果集的操作符，其可被赋值为 SPARQL 语句中查询表单的关键字 SELECT、CONSTRUCT、ASK 或 DESCRIBE。

基于上述定义，示例查询语句 Query 8 中每条三元组模式可映射为一个 TPN 节点，并可定义 Query 8 的 SSM 模型和 QF 模型分别为 SSM=(DISTINCT, ?*name*)，QF={SELECT}。因此，可将 Query 8 解析为：[GPT, (DISTINCT, ?*name*), SELECT]。

图 4.4 描述了 Query 8 对应的 GPT 模型，其中 TPN 节点用圆形表示，OPN 节点用矩形表示。由于 OPT 节点 AND_1 和 AND_3 的所有子节点均为 TPN 节点，因此，AND_1 和 AND_3 为 LF-OPN 节点，且分别依附 FILTER 内置条件 R_1 和 R_2。OPT 节点 AND_2 子节点中由于包含了 AND_1 节点，因此 AND_2 不是 LF-OPN 节点。同理，根 OPT 节点 UNION 也不为 LF-OPN 节点。

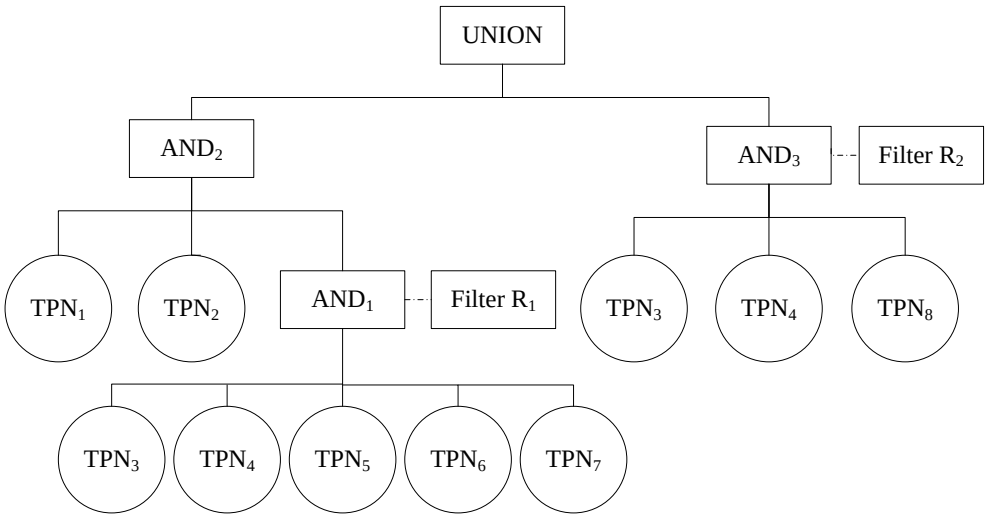


图 4.4 示例查询语句 Query 8 的 GPT 模型

Figure 4.4 GPT Model of Sample Query 8

基于本文第三章提出的框架体系结构和基于 HBase 的本体数据存储策略，以及上述解析模型定义，最直接的 RDF 数据查询方式即对每一个 TPN 节点分别在 HBase 中进行 RDF 术语匹配。例如，针对 Query 8，可依次对 TPN_1,TPN_2,...,TPN_8 分别在 HBase 数据库中查询与变量相对应的 RDF 术语匹配解，但由于该方法会进行多次 HBase 数据库访问和 MapReduce 任务迭代，且每次 MapReduce 任务均涉及任务初始化、中间数据的存储、排序和分发等步骤，会不可避免地带来额外的 I/O 数据传递开销和计算性能损耗，因此，本章基于以下数据模型提出了一种更为高效的 RDF 数据查询方法。

定义 4.4：查询模式模型(Query Pattern，QP)。对任一 GPT 模型，如果存在 n 个 TPN 节点：$TPN_1=(s_1, p_1, o_1)$, $TPN_2=(s_2, p_2, o_2)$,..., $TPN_n=(s_n, p_n, o_n)$，$n\geq 2$，且满足其中所有主语、谓语和宾语均为变量，或分别具有相同固定值 V_s、V_p 和 V_o 时，用三元组 QP=(s, p, o)来描述 TPN_1, ..., TPN_n 的共享检索信息。当主语 $s_1, s_2,..., s_n$、谓语 $p_1, p_2,..., p_n$ 或宾语 $o_1, o_2,..., o_n$ 为变量时，对应的 QP 模型 s、p 和 o 赋值为 *Var*；否则，s、p 和 o 分别赋值为 V_s、V_p 和 V_o。

在 Query 8 中，TPN_3、TPN_4、TPN_5 和 TPN_6 的主语和宾语均为变量，谓语具有相同的固定值 *dc:creator*，因此，对 TPN_3、TPN_4、TPN_5 和 TPN_6 可定义一个 QP 模型：QP_1=(*Var*, *dc*:*creator*, *Var*)。同理，可定义 TPN_7 和 TPN_8 的 QP 模型为 QP_2=(*Var*, *foaf*:*name*, *Var*)。

通过使用 QP 模型，通过将固定值设定为查询匹配条件，可在一次 HBase 访问中对多个 TPN 节点所对应的 RDF 术语进行匹配。例如，当设定 QP_1 中固定值 *dc*:*creator* 为查询条件时，可在一个 MapReduce 任务中匹配到满足 TPN_3、TPN_4、TPN_5 和 TPN_6 的 RDF 术语匹配解，而不是分别使用四次 MapReduce 任务迭代。然而，由于其谓语和宾语均有不同值，所以不能定义针对 TPN_1 和 TPN_2 的 QP 模型，仍需分别进行数据查询。

定义 4.5：共享变量(Shared Variable，SV)，指存在于同一个 OPN 节点下 TPN 节点或 TN 节点中的变量。

定义 4.6：MapReduce 查询计划模型(MapReduce Query Plan，MRQP)，是一个五元组(*JobID*, *Opt*, *SV*, *Tag*, *Flt*)，其中 *JobID* 标识 MapReduce 任务序号，*Opt* 为 OPN 节点对应运算符 AND、OPTIONAL 或 UNION，*Tag* 为目标 TPN 或 TN 节点的集合，*SV* 是 *Tag* 中的共享变量，*Flt* 为附属于 *Opt* 的 FILTER 内置条件集合。

基于上述定义可知，Query 8 中变量?*document* 是 TPN_3 和 TPN_4 节点的共享变量，变量?*author* 是 TPN_4 和 TPN_5 节点共享变量。对存在于 TPN_7 和 TPN_8 节点中的变量?*name* 而言，由于其分别隶属于不同的 OPN 节点，因此?*name* 不是共享变量。

另外，假设 TPN_4 和 TPN_5 在第一次 MapReduce 任务中进行计算，可定义一个

MRQP 模型为 $MRQP_{(4,5)}$=(1, AND, ?*author*, {TPN_4,TPN_5}, R_1)。当框架中查询引擎接收到 $MRQP_{(4,5)}$的查询计划后，Mapper 计算结点以与 TPN_4 和 TPN_5 匹配的 RDF 三元组作为输入，并以变量?*author* 及其对应 RDF 术语匹配解为键生成一组中间键值对数据。然后，具有相同键值的中间数据被分发到一个 Reducer 计算结点，再在 reduce 函数中进行 AND 和 FILTER 计算。图 4.5 描述了本文所提出框架在执行 SPARQL 查询时的计算流程。

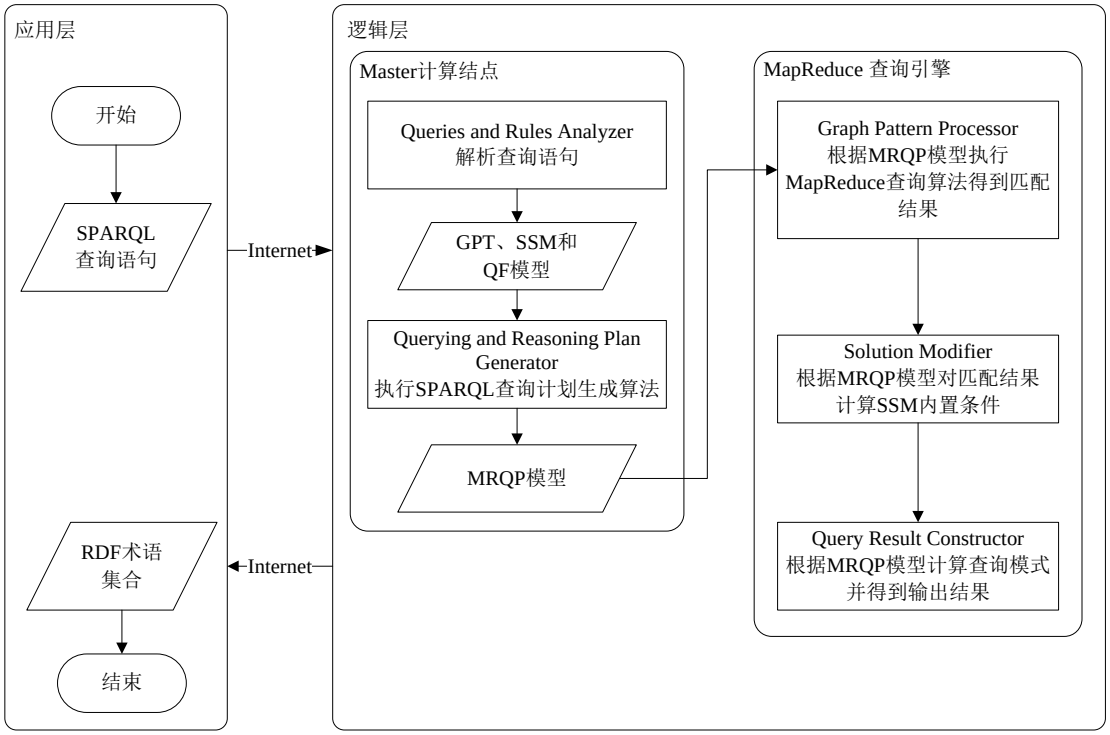


图 4.5 框架中 SPARQL 分布式查询计算流程

Figure 4.5 Workflow of SPARQL Querying in the Proposed Framework.

4.3 SPARQL 复杂组图模式的 MapReduce 查询任务生成算法

以本章 4.2 小节中定义的解析数据模型为基础，本文所提出框架中的查询和推理计划生成器根据用户输入的 SPARQL 查询语句计算生成相应的 MapReduce 查询任务计划。本小节首先提出 MapReduce 计算环境下 SPARQL 复杂组图模式运算符 AND、OPTIONAL 和 UNION 的计算方法。然后，由于 MapReduce 任务以键值对格式对数据进行处理，存在于同一个 TPN 或 TN 结点中的两个或多个共享变量不能在一个 MapReduce 任务中进行计算，对 SPARQL 复杂图模式查询任务不可避免地需要进行多次 MapReduce 任务迭代。为尽可能降低 MapReduce 任务的执行次数，以实现查询性能的优化，通过扩展文献[112]中提出的 Relaxed-Bestplan 算法，提出 MapReduce 环境下面向 SPARQL 复杂组图模式查询的查询计划生成算法。

根据 SPARQL 语法和语义，运算符 UNION 对两个或多个图模式所对应的 RDF 术语匹配解进行合并。在 MapReduce 计算模型下，一个 UNION 运算符可以在一个 MapReduce 任务中处理。例如，对由两个图模式构成的 UNION 表达式：{(?*a*, ?*b*, ?*c*)} UNION {(?*a*, ?*b*, ?*d*)}，MapReduce 任务中的 map 函数首先获取分别与(?*a*, ?*b*, ?*c*)和(?*a*, ?*b*, ?*d*)匹配的 RDF 术语，并生成一组对应于四元组(?*a*, ?*b*, ?*c*, ?*d*)的解。在这种情况下，不需要执行 reduce 计算。

相似地，OPTIONAL 运算符将一个可选解附加在另一个图模式上，可在一个 MapReduce 任务中进行计算。例如，假设存在两个图模式：P_1={(?*a*, b, ?*c*)}和 P_2={(?*a*, d, ?*d*)}，当在对可选图模式表达式 P_1 OPTIONAL P_2 进行计算时，map 函数首先查询得到与分别与 P_1 和 P_2 相匹配的 RDF 术语解，然后将共享变量?*a* 及其匹配的术语设置为键，将变量?*c* 和?*d* 及其相匹配的 RDF 术语联合设置为值，并生成一组键值对中间数据。Reducer 计算结点接收到具有相同键的中间数据后，在 reduce 函数中生成一组三元组(?*a*, ?*c*, ?*d*)所对应的映射解集。

在文献[112]中，Husain 等人证明了当在进行 SPARQL 基本图模式查询时，会存在呈指数数量级的连接运算。同时，运算过程中的概要统计(Summary Statistics)过程会带来很大的计算时间和空间开销。除此以外，通过进行实验对比，Husain 等人发现在对相同数据进行计算时，运行多个 MapReduce 任务会带来更多的 I/O 数据传递，并导致计算性能的下降和计算时间的增加。因此，Husain 等人基于贪心选择策略设计了用于生成最少 MapReduce 任务数的 Relaxed-Bestplan 算法。

在面向多个图模式的 AND 运算时，其语义和计算方法与 SPARQL 基本图模式查询相同。因此，对于某一 AND 运算符，设其对应了 N 个 TPN 或 TN 模型，并包含了 K 个共享变量，Relaxed-Bestplan 算法生成的查询计划包含的 MapReduce 任务为 J，则

$$J=\begin{cases}0 & N=0,\\ 1 & N=1 \text{ or } K=1,\\ \min([1.71\log_2 N], K), & N, K>1.\end{cases} \tag{4.1}$$

另外，由于一个 FILTER 内置条件可以在 MapReduce 任务的 map 或 reduce 函数进行计算，因此，可以得出以下结论。

对一个 OPN 节点 O，设对 O 进行计算时需要的 MapReduce 任务数为 SJ，则

$$SJ=\begin{cases}1, & \text{当}O\text{为OPTIONAL或UNION节点时},\\ J, & \text{当}O\text{为AND节点时}。\end{cases} \tag{4.2}$$

基于顺序的计算方式是最直接的 OPN 节点运算方法。以 Query 8 为例，根据式 4.1，计算节点 TPN_3, TPN_4, TPN_5, TPN_6 和 TPN_7 所对应的 OPN 节点 AND_1 所需的 MapReduc 任务数为 3；计算节点 TPN_3, TPN_4 和 TPN_8 所对应的 OPN 节点 AND_1

所需的 MapReduc 任务数为 2；计算节点 TPN_1、TPN_2 和 AND_1 需要的 MapReduce 任务数为 1；位于根节点位置的 UNION 节点需要 1 个 MapReduce 任务。因此，如果按照以上顺序执行的思想，在对 Query 8 进行计算时总共需要的 MapReduce 任务数为 7。显然该方法会涉及过多的 MapReduce 任务数。

由于 OPN 节点 AND_1、AND_2 和 AND_3 分别独立地对其对应的 TPN 节点进行计算，所以基于贪心策略的计算方法能够在一个 MapReduce 任务中同时对 AND_1、AND_2 和 AND_3 进行计算。例如，在前三个 MapReduce 任务中，在对 AND_1 进行计算的同时，也可以对 AND_2 进行计算。然后，针对 OPN 节点 AND_3 和 UNION 的计算任务可被顺序执行。因此，整个计算过程需要的 MapReduce 任务数为 5。基于上述贪心选择策略，在算法 4.1 中提出面向 SPARQL 复杂组图模式的 MapReduce 查询任务生成算法。

算法 4.1 以一个 GPT 模型树 *T* 作为输入，计算生成一个 MRQP 模型的集合 *QPlan* 作为查询计划输出。首先，初始化 *QPlan* 为空，并将树 *T* 的深度值赋值给参数 *L*，然后对每一层的 OPN 节点按自底向上的顺序依次进行迭代计算推理计划。设当前计算层次数为 *a*，对 $a=L, (L-1),..., 1$，将第 *a* 层的所有 OPN 节点存储到集合 *N* 中，$N=\{OPN_1, ..., OPN_n\}$，$n\geq 1$。再对 *N* 中的任一节点 OPN_i，$1\leq i\leq n$，设集合 $CN=\{Node_1, ..., Node_m\}$用于存储 OPN_i 的所有子节点，$m\geq 1$，如果所有节点均为 TPN 类型，即 OPN_i 为 L-OPN 节点，那么将当前 MapReduce 任务序号 *t* 设置为 1；否则，由函数 Max_Job_Nums 获取 *CN* 集合中所有子节点中的最大任务序号，并将 *t* 设置为 Max_Job_Nums(CN)+1。然后，对集合 *CN* 的所有节点中所包含的共享变量根据其 E-count 值[112]按升序排列，并存储在集合 *S* 中。接下来，查询计划可根据当前的 OPN 对应的运算符进行生成。

如算法 4.1 中第 9 行所示，如果当前 OPN 节点对应的运算符为 OPTIONAL，那么生成一个 MRQP 模型(*t*, OPTIONAL, *S*, *CN*, FILTER)，用于指定当前任务所需执行的计算操作，其中 *t* 是 MapReduce 任务序号，*CN* 是目标 *TPN* 或 *TN* 节点的集合，*S* 是存在于 *CN* 中的共享变量，*FILTER* 为附属于当前 OPTIONAL 节点的内置条件。相似地，如果当前 OPN 节点为 UNION 类型，那么如算法 4.2 中第 13 行所示，生成一个 MRQP 模型为(*t*, UNION, null, CN, FILTER)，并且将其添加入最终的查询任务计划集合 *QPlan*。如果当前 OPN 节点对应的运算符为 AND，则通过改进 Relaxed-Bestplan 算法实现 MRQP 模型的迭代生成。在算法 4.1 中第 16 行开始的 While 循环中，如果集合 *CN* 非空，对集合 *S* 中包含的共享变量 sv_k，$1\leq k\leq s$，如果 sv_k 能够被完全或部分消去，那么将所有对于 sv_k 的 TPN 或 TN 节点存储于集合 *TP* 中，并生成一个 MRQP 模型为(*t*, AND, sv_k, TP(*CN*, sv_k), FILTER)。

算法 4.1：SPARQL 复杂组图模式的 MapReduce 查询任务生成

输入：GPT 模型 T

输出：MRQP 模型集合 *QPlan*

(1) 初始化 *QPlan* ←∅, *L*←Tree-Depth(*T*)；
(2) For *a* = *L* to 1
(3) *N*={OPN$_1$, ..., OPN$_n$}←第 *a* 层的所有 OPN 节点；
(4) For (*i*=1 to *n*)
(5) *CN* ← OPN$_i$ 节点的所有子节点；
(6) If OPN$_i$ 为 L-OPN 节点, 执行 *t* ← 1；
(7) Else *t* ← Max_job_Nums(CN)+1；
(8) *S*={sv$_1$,...,sv$_s$} ← *CN* 中所有共享变量按 E-count 值进行升序排列；
(9) If OPN$_i$ 为 OPTIONAL 节点
(10) *MRQP* ← (*t*, OPTIONAL, *S*, *CN*, *FILTER*)；
(11) *QPlan* ← *QPlan* ∪{*MRQP*}；
(12) Else If OPN$_i$ 为 UNION 节点
(13) *MRQP* ← (*t*, UNION, null, *CN*, *FILTER*)；
(14) *QPlan* ← *QPlan* ∪{ *MRQP* }；
(15) Else If OPN$_i$ 为 AND 节点
(16) While(*CN* ≠ ∅)
(17) For *k*=1 to *s* 且如果 Can-Eliminate(*CN*, sv$_k$)
(18) TP(*CN*, sv$_k$) ←All TPNs or TNs in *CN* containing variable sv$_k$；
(19) MRQP←(*t*, AND, sv$_k$, TP(*CN*,sv$_k$), FILTER)；
(20) *QPlan*←*QPlan* ∪{MRQP}；
(21) Temp←Temp ∪ Join_Result (TP(*CN*, sv$_k$))
(22) CN←CN–TP(*CN*, sv$_k$)；
(23) End For
(24) *t*←*t*+1；
(25) *CN*←*CN*∪Temp；
(26) End While
(27) End If
(28) Replace OPN$_i$ with new TN(*CN*, *t*)；
(29) End For
(30) End For
(31) Return *QPlan*

在进行下一个共享变量的计算之前，所有已经参与过计算的 TPN 或 TN 节点被从集合中除去，并将中间计算结果存储在参数 *Temp* 中。当所有共享变量均被计算之后，参数 *t* 和 *CN* 被更新，并进入下一次 While 计算循环。在当前 OPN 节点的计算完成后，如算法 4.1 中第 28 行所示，根据 *CN* 中所有变量名，生成一个新的 TN 节点，用其来替换当前的 OPN 节点，并将属性 *JobN* 设置为 *t*。最后，当所有层次上的 OPN 节点的计算过程均完成后，算法 4.1 返回一个 MRQP 查询计划集合 *QPlan*。

根据算法 4.1 生成的 MRQP 模型 *QPlan*，MapReduce 查询过程从每个 L-OPN 开始，在根 OPN 节点处终止。因此，将每个 L-OPN 节点作为起点，存在一条通往根节点的路径。假设存在两条路径 $Path_1$ 和 $Path_2$ 在一个 OPN 节点 *O* 处交互，并且设 $Path_1$ 和 $Path_2$ 所需要的 MapReduce 任务数分别为 PJ_1 和 PJ_2，设节点 *O* 自身需要的 MapReduce 任务数为 *SJ*。由于对 *O* 的计算依赖于之前节点，即需要在 $Path_1$ 和 $Path_2$ 的计算完成后进行计算，所以对 *O* 进行计算所需的 MapReduce 任务总数为 *SJ* 与 PJ_1、PJ_2 的最大值之和。

因此，给定一个包含 *n* 个 L-OPN 节点的 GPT 模型，存在 *n* 个路径 $P_1, P_2, ..., P_n$，每条路径以一个 L-OPN 节点作为起点，以 GPT 树中根 OPN 节点作为终点。算法 4.1 对该 GPT 生成的 MapReduce 任务数为 *SumJ*，那么，

$$SumJ = \max(PJ_1, PJ_2, ..., PJ_n) \tag{4.3}$$

其中，PJ_i 是路径 P_i 所需的总任务数，$1 \le i \le n$。

另外，假设 GPT 树的深度为 *L*，并且设某一路径 *P* 的 L-OPN 节点位于 GPT 树的第 *D* 层，$D \le L$，那么该路径 *P* 所需的任务数 *PJ* 为：

$$PJ = \sum_{j=D}^{1} SJ_j \tag{4.4}$$

其中 *SJ* 为根据式(4.2)定义生成的路径中 OPN 所需 MapReduce 任务数。

以 Query 8 为例，存在两个 L-OPN 节点：AND_1 和 AND_2。因此，存在两条路径 P_1 = <AND_1, AND_3, UNION> 和 P_2 = <AND_2, UNION>可以同时进行计算。根据式(4.2)，AND1 需要的任务数为 3，AND2 需要的任务数为 2，AND3 和 UNION 分别需要 1 个 MapReduce 任务来完成计算。因此 PJ_1=5，PJ_2=3。算法 4.1 生成的任务数为 *SumJ* =max(5, 3)，即需要 5 次 MapReduce 任务来完成对 Query 8 的查询。

在提出的算法 4.1 中，位于第 2 行的外层 for 循环最多执行 *L* 次，第 4 行的 for 循环最多执行 *N* 次，其中 *L* 是输入 GPT 树的深度，*N* 是树所有层次中 OPN 节点数的最大值。位于第 16 行的 While 循环最大执行次数为 *J*，位于 17 行的内层 for 循环最大执行次数为 *K* 次，*J* 为当 OPN 为 AND 运算符时的最大 MapReduce 任务数，*K* 为所有 OPN 节点中的最大共享变量数。因此，所提出的查询计划生成算法

的复杂度为 $O(LNK(J + \log K))$。

4.4 SPARQL 复杂组图模式的 MapReduce 分布式查询算法

在查询计划生成之后，MapReduce 环境下的分布式查询引擎根据本小节提出的基于 MapReduce 的 SPARQL 复杂组图模式分布式查询算法，通过执行一组 MapReduce 任务来实现面向大规模 RDF 数据的 SPARQL 复杂组图模式查询。

首先，框架中数据适配器模块在 HBase 数据库中查询出所有与 TPN 节点匹配的 RDF 术语解，并根据所对应的 TPN 节点分别将其存储在 HDFS 文件中。RDF 数据的匹配过程如算法 4.2 所示。

算法 4.2：基于 MapReduce 的 RDF 术语匹配
输入：GPT 模型 *T*
输出：一组 RDF 术语匹配解 *solution*

(1) 对 *T* 中所有 TPN 节点，构造 QP 模型的集合 $Q=\{QP_1, ..., QP_n\}$, $n\geq 0$；
(2) 构造一个用于存储 *T* 中剩余 TPN 节点的集合 $T=\{TPN_1,..., TPN_s\}$, $s\geq 0$；
(3) 初始化一个支持多表查询(Multi-table)的 MapReduce 任务 *Job*；
(4) 基于 *Q* 和 *T*，为 *Job* 赋值查询参数；
(5) 在 MapReduce 环境中执行 *Job*，以从 HBase 中查询出 RDF 术语解 *solution*；
(6) 将 *solution* 存储在对应 TPN 节点的 HDFS 文件 $QP_1, ..., QP_n, TPN_1,..., TPN_s$ 中。

如本文第 4.2.2 小节所述，能够为 Query 8 中的 TPN_3、TPN_4、TPN_5 和 TPN_6 及 TPN_7 和 TPN_8 分别定义两个 QP 模型：QP_1 = (*Var, dc*:*creator, Var*)和 QP_2 = (*Var, foaf*:*name, Var*)，同时 TPN_1 和 TPN_2 需要单独进行处理。因此，数据适配器模块将 $Q=\{QP_1, QP_2\}$和 $T=\{TPN_1, TPN_2\}$中的固定值设置为查询参数，并且执行一个 MapReduce 任务从 HBase 中查询出匹配的 RDF 术语匹配解 *solution*。最后，所有对应的匹配解被分别存储在四个 HDFS 文件中：QP_1, QP_2, TPN_1 和 TPN_2。

接下来，MapReduce 查询引擎在 Master 计算结点中执行如算法 4.3 所示的中央控制算法(Central Control Algorithm)。该算法根据算法 4.1 计算生成的查询计划实现 MapReduce 查询任务的初始化、执行和调度，以及整个查询过程进度的监控。

算法 4.3：中央控制算法(Central Control Algorithm)
输入：MRQP 模型的集合 *QPlan*
输出：查询任务的最终计算结果

(1) 初始化 $J\leftarrow 1$, *InputFiles* $\leftarrow\varnothing$, *SumJ* $\leftarrow$ *QPlan* 集合中 MapReduce 任务数；
(2) While ($J \leq SumJ$)

(3) For each 集合 *QPlan* 中的 MRQP 模型 *M*, 满足 *M.JobID* = *J*, 执行
(4) *InputFiles* ← *InputFiles* ∪ {*M.Tag*}；
(5) *CurrentOPN*← *CurrentOPN* ∪ {*M*}；
(6) EndForeach
(7) *Job.Location* ← *InputFiles*；
(8) DictributedCache ← *CurrentOPN*；
(9) If *J* = *SumJ*, 执行 Distributed Cache ←{SSM, QF}；
(10) 执行 *Job*，并赋值 *InputFiles* ←∅, *CurrentOPN*←∅, *J*←*J*+1；
(11) End While

算法 4.3 以 MRQP 模型集合 *QPlan* 作为输入，迭代地执行一组 MapReduce 任务，直至集合 *QPlan* 为空。首先，初始化两个参数 *J* 和 *InputFiles*，分别标识当前的任务序号和目标 RDF 数据文件。然后，在 While 循环中，算法 4.3 根据每个 MRQP 模型的顺序号初始化并执行一个 MapReduce 任务。

如算法 4.3 的第 3 行所示，对于 *QPlan* 中的任一 MRQP 模型 *M*，如果其属性 *JobID* 与当前任务序号 *J* 相等，那么将 *M* 的属性 *Tag* 中存储的目标 RDF 文件设置为当前任务的 map 函数需处理的数据源 *InputFiles*，并将 *M* 存储到集合 *CurrentOPN* 中，用以标识当前查询任务的具体运算过程。当所有当前步骤中的 MRQP 模型均被处理后，如算法 4.3 第 7、8 行所示，将当前 MapReduce 任务的数据源设定为 *InputFiles*，并将集合 *CurrentOPN* 分发到 Hadoop 环境的 Distributed Cache 对象中。然后，如果当前任务序号 *J* 等于 *QPlan* 集合中的总任务数，即当前 MapReduce 任务为最后一个任务，将 SSM 和 QF 模型存储到 Distributed Cache 对象中。最后，算法 4.3 执行当前 MapReduce 任务，并更新参数 *InputFiles* 和 *J*，直至完成所有的查询任务，并得到最终的查询结果。

当 Master 计算结点分配和执行查询计算 MapReduce 任务后，算法 4.3 指定的 RDF 数据文件被分发到各个 Mapper 计算结点，然后各个 Mapper 计算结点独立地执行 map 函数查询算法，并生成一组中间键值对数据，算法 4.4 描述了每个 map 函数的计算流程。

算法 4.4：Map 函数查询算法(Query Algorithm in Map Function)
输入：HDFS 文件中一组 RDF 术语匹配解 *solution*
输出：一组中间键值对(key, value)

(1) *OPTs*←*CurrentOPN* in Distributed Cache；
(2) If *solution* matched to TPN or TN of one MRQP model *M.Tag* in *OPTs*

(3) If *M.Opt* equals UNION
(4) Construct a new solution *S* that consists of all the variable names in MRQP；
(5) *S*←the corresponding RDF terms in *solution*；
(6) Output *S* to one HDFS file；
(7) Else do
(8) key←(*M.SV*, RDF term of *M.SV* in *solution*)；
(9) value←*solution*；
(10) Emit(key, value)

如算法 4.4 所示，对 MapReduce 查询任务中的每个 map 函数而言，首先读取 Distributed Cache 对象中的 MRQP 模型集合 *CurrentOPN*，并将其存入参数 *OPTs* 中。当 Mapper 计算结点接收到的 RDF 术语匹配解能与 *OPTs* 中某一 MRQP 模型 *M* 的 *Tag* 属性包含的 TPN 或 TN 结点相匹配时，判断 *M* 的 *Opt* 属性。如果 *M.Opt* 为 UNION 运算符，用 *M* 中的所有变量名和对应的 RDF 术语构造一个新的术语解 *S*，并将 *S* 存储到一个 HDFS 文件中。如果 *M. Opt* 为 AND 或 OPTIONAL 运算符，如算法 4.4 的第 8 和第 9 行所示，以 *M* 中变共享变量 *M.SV* 和当前获取的术语解中对应的 RDF 术语为联合键，以术语解作为值，生成中间键值对数据(key, value)，并将其分发到 Reducer 结点进行计算。

在任一 MapReduce 查询任务的 Reduce 阶段，所有具有相同键值的中间键值对数据被分发到一个 Reducer 计算结点，并在 reduce 函数中执行如算法 4.5 所示的计算流程。

首先，每个 Reducer 计算结点从 Distributed Cache 对象中获取当前查询任务一组的 MRQP 模型，并将其存储在参数 *OPTs* 中。根据 MRQP 模型的数量 n，$n \geq 1$，初始化 n 个集合 $U_1, U_2, ..., U_n$，用于将接收到的每一个中间键值对的值部分存储在对应的集合 U_i 中，$1 \leq i \leq n$。然后，算法 4.4 基于文献[19]中提出的运算符 AND 和 OPTIONAL 的计算语义，根据 MRQP 模型中的 *Opt* 属性和 FILTER 内置条件对 $U_1, U_2, ..., U_n$ 进行迭代地计算，并将计算结果存储在集合 U_R 中。最后，如果从 Distributed Cache 对象中获取的 SSM 和 QP 模型不为空，则生成最后的查询结果并输出到 HDFS 文件系统。

算法 4.5：Reduce 函数查询算法(Query Algorithm in Reduce Function)
输入：一组具有相同键值的中间键值对数据
输出：一组 RDF 匹配解

(1) *OPTs*←*CurrentOPN* in Distributed Cache；
(2) *SSM* ← Sequence Solution Modifier models in Distributed Cache；

(3) $QF \leftarrow$ Query Form models in Distributed Cache；

(4) For each MRQP model M in $OPTs$

(5) Initial collections U_1,..., U_n, , n is the TPN or TN numbers of MRQP model；

(6) For each key–value pair P that corresponds to M, do $U_i \leftarrow P$.value, $1 \le i \le n$；

(7) Execute $M.Opt$ for U_1,..., U_n, and store the computational results in U_R；

(8) Execute $M.Filter$ for U_R if required；

(9) Execute SSM for U_R if $SSM \neq \varnothing$；

(10) Output solutions U_R to HDFS according to QF if $QF \neq \varnothing$.

4.5 实验结果及分析

4.5.1 实验环境搭建

本章采用SP2Bench公共本体数据集对提出方法的计算性能和可扩展行进行了实验验证。SP2Bench 是语义 Web 研究领域在对 SPARQL 查询引擎的计算性能进行验证时，研究人员所广泛采用的基础数据集。与仅包含简单 SPARQL 基本图模式测试查询语句的 LUBM 和 BSBM 数据集不同，SP2Bench 提供了面向复杂 SPARQL 组图模式查询的 16 个标准测试语句。

其中，选取了 SP2Bench 中的 6 条语句作为测试标准，它们分别是：Query 2、Query 3a、Query 4、Query 6、Query 8 和 Query 9。未采用其他的测试语句的原因是：Query 1、Query 5a、Query 5c、Query 10 和 Query 11 为简单 SPARQL 基本图模式查询语句，而未涉及 UNION 或 OPTIONAL 运算符。Query 3b 和 Query 3c 具有与 Query 3a 相似的查询条件和语句结构，而仅在 FILTER 内置条件限制上存在细微差别。另外，由于本文所实现的方法中还不能支持关键字 bound 和 ASK，因此未选用 SP2Bench 中测试语句 Query 7、Query 12a、Query 12b 和 Query 12c。

为了模拟云计算环境，本文将 9 台 PC 机通过 Ethernet 连接构成为一个实验云计算环境，以此作为框架的 Hadoop 平台硬件基础。每个 PC 计算结点具有如下的硬件配置：Pentium IV 3.00 GHz CPU，1.5 GB 内存和 80 GB 硬盘空间。为进行对比实验，将 Jena、Sesame 和 RDF-3X 查询引擎部署在一个硬件配置为 Intel i5 2.50 GHz dual core processor, 8 GB 内存空间和 4 TB 硬盘空间的计算机之上。

在软件设置方面，采用 Hadoop-0.20.2 和 HBase-0.20.6 版本作为本章所提出方法的基础，并选取了 Jena-2.6.4 In-memory 和 SDB，Sesame 2.6.3 main memory 和 RDF-3X 0.3.7 版本。另外，MySQL 5.0 版本的关系型数据库管理系统被选为了 Jena SDB 的数据库基础。在本章后续实验结果报告中，将本章方法以 CloudSPARQL 命名。

4.5.2 实验效果及分析

在实验过程中，由于学术界尚不存在基于 MapReduce 的 SPARQL 复杂组图模式分布式查询方法，本章进行了与单机环境下 SPARQL 查询引擎 Jena In-memory 和 SDB、Sesame 和 RDF-3X 的对比实验以及可扩展性实验。

在对比实验中，使用 SP2Bench 的 RDF 数据生成器分别构造了包含 400 万、800 万、1200 万、1500 万、2000 万和 3000 万条的 RDF 三元组作为测试数据集。在可扩展性实验方面，分别生成了 1000 万、2000 万、3000 万和 4000 万条的 RDF 三元组作为测试数据集。

首先，以 SP2Bench 标准查询语句 Query 2、Query 3a、Query 4、Query 6、Query 8 和 Query 9 作为输入 SPARQL 查询语句，对 CloudSPARQL 与 Jena In-Memory 和 Jena SDB 进行了对比实验，计算生成了相同的查询结果。上述 6 条查询语句中既存在复杂的语法结构，也包含了 AND、UNION 和 OPTIONAL 运算符，以及 FILTER 内置条件限制。表 4.1 描述了 CloudSPARQL 与 Jena In-memory 的对比结果，表 4.2 描述了 CloudSPARQL 与 Jena SDB 的对比结果。其中，查询用时的计量单位为秒。

表 4.1 CloudSPARQL 与 Jena In-memory 的对比实验结果

Table 4.1 Comparison between Jena In-memory and our approach CloudSPARQL

	8 Million Triples		12 Million Triples		15 Million Triples		30 Million Triples	
	In-Mem	CloudS	In-Mem	CloudS	In-Mem	CloudS	In-Mem	CloudS
Query 2	174.75s	252.96s	286.23s	394.53s	Failed	458.08s	Failed	820.71s
Query 3a	95.85s	51.50s	140.26s	83.58s	204.17s	116.78s	Failed	260.78s
Query 4	Failed	370.43s	Failed	478.40s	Failed	579.98s	Failed	887.69s
Query 6	Failed	375.47s	Failed	517.89s	Failed	582.79s	Failed	1154.99s
Query 8	78.75s	547.77s	136.83s	949,52s	194.44s	1245.74s	Failed	2365.54s
Query 9	82.36s	285.37	139.96s	402.95s	199.02s	502.32s	Failed	916.42s

表 4.2 CloudSPARQL 与 Jena SDB 的对比实验结果

Table 4.2 Comparison between Jena SDB and our approach CloudSPARQL

	4 Million Triples		8 Million Triples		12 Million Triples		15 Million Triples	
	SDB	CloudS	SDB	CloudS	SDB	CloudS	SDB	CloudS
Query 2	192.57s	179.05s	4090.97s	252.96s	Failed	394.53s	Failed	458.08s
Query 3a	172.93s	44.54s	1287.33s	51.50s	Failed	83.58s	Failed	116.78s
Query 4	Failed	243.88s	Failed	370.43s	Failed	478.40s	Failed	579.98s
Query 6	Failed	257.26s	Failed	375.47s	Failed	517.89s	Failed	582.79s
Query 8	Failed	376.54s	Failed	547.77s	Failed	949,52s	Failed	1245.74s
Query 9	223.78s	171.80s	1881.46s	285.37s	Failed	402.95s	Failed	502.32s

Query 2 中前九条三元组模式进行 AND 运算，其结果再与最后一条三元组模式进行 OPTIONAL 运算。在第一条三元组模式中存在两个固定的 RDF 术语值，并且所有的三元组模式共享一个变量?*inproc*。因此，Query 2 具备高选择度特性，即参与计算的数据及其查询结果集的数据量较小。当输入 RDF 数据量大小为 800 万和 1200 万条时，Jena In-memory 在计算性能方面优于本章方法，但是当输入 1500 万和 3000 万条 RDF 三元组数据时，Jena In-memory 出现内存溢出错误。对输入的所有 4 个数据集，本章方法在计算性能方面均优于 Jena SDB，特别是当对 1200 万和 1500 万条输入 RDF 三元组数据进行处理时，Jena SDB 工具出现内存溢出错误。

Query 3a 的语法结构相对简单，仅涉及针对于两个三元组模式的 AND 运算符，以及一个 FILTER 内置条件，并查询返回 RDF 数据集中所有具有属性 *swrc*:*pages* 的文章信息(articles)。然而，由于 Query 3a 中第二个三元组模式的主语、谓语和宾语均为变量，因此 Query 3a 具有低匹配度特性，查询生成一个数据量较大的结果集。对实验中输入的四个 RDF 数据集，本章方法相较于 Jena In-memory 和 Jena SDB 具有更好的计算性能。当输入数据量增加时，Jena In-memory 和 Jena SDB 均不能完成对 Query 3a 的查询。

Query 4 查询生成 SP2Bench 数据集中所有在同一期刊中发表文章的作者和其文章的数据对，因此会产生一个数据量较大的结果集。与 Query 3a 的测试结果类似，针对 Query 4 和四个输入数据集，本章方法均优于 Jena In-memory 和 SDB。

Query 6 的语法结构较 Query 4 更为复杂且具有低匹配度，查询产生一个大结果集。Jena In-memory 和 SDB 均不能完成输入的四个数据集，但是本章方法能够在可接受的时间内计算出结果。

Query 8 包含了对多个图模式的 AND 和 UNION 运算，以及 FILTER 内置条件，其具有复杂的语法结构。当在对 800 万、1200 万和 1500 万条的输入 RDF 数据进行查询时，Jena In-memory 优于本章方法。但是，当输入 RDF 数据量增加到 3000 万条时，Jena In-memory 抛出内存溢出异常不能完成计算。Jena SDB 对所有的四个输入数据集均不能完成 Query 8 查询任务。

Query 9 查询生成 SP2Bench 数据集中收入和支出资产，并且因为 Distinct 关键字的限制，生成较小的结果集。当输入 800 万、1200 万和 1500 万条 RDF 三元组时，Jena In-memory 处理 Query 9 的计算性能优于本章方法，但其不能完成输入数据量为 3000 万条时的 Query 9 处理任务。对于四个输入数据，本章方法计算 Query 9 时的性能均优于 Jena SDB。

根据以上实验结果可得出如下结论：Jena In-memory 在处理小数据量的输入 RDF 数据，且查询语句因存在固定值而具有高匹配性时，在计算性能方面优于本章方法。然而，由于存在内存限制，Jena In-memory 不能完成输入数据量增加时的

计算需求。同时，对于查询语句中因存在非固定值而具有低匹配度特性时，本章方法较 Jena In-memory 具有更好的计算性能。另外，由于 Jena SDB 无法对大数据量的 RDF 数据执行 SPARQL 查询，因此本章方法比 Jena SDB 有更好的计算性能和可扩展性。

由于当前版本的 RDF-3X 未能提供对 OPTIONAL 运算符的支持，因此，选取了 Query 3a, 4, 8 和 9 作为测试语句，并用 SP2Bench 生成了 400 万、800 万、1200 万和 1500 万条的 RDF 三元组数据来进行与本章方法的对比实验。另外，由于 RDF-3X 在执行查询任务之前，需要先将所有 RDF 数据载入计算机内存中，所以在实验计算时间的统计过程中包含了初始的数据载入时间和查询时间。

图 4.6 描述了本章方法与 RDF-3X 的对比实验结果。其中，X 轴表示输入 RDF 三元组数据量大小，计量单位为万条，Y 轴表示查询总响应时间，计量单位为秒，用符号“×”标识内存溢出错误。

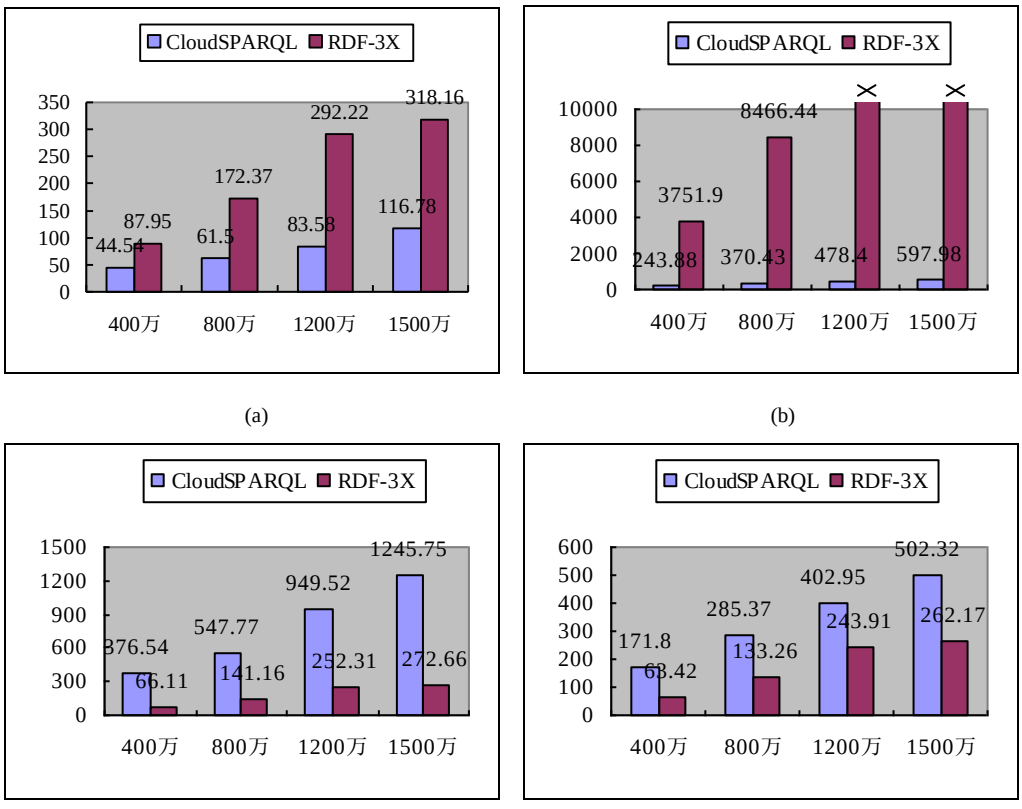


图 4.6 CloudSPARQL 与 RDF-3X 的对比实验结果

(a) SP2Bech Query 3a 计算用时；(b) SP2Bech Query 4 计算用时

(c) SP2Bech Query 8 计算用时；(d) SP2Bech Query 9 计算用时

Figure 4.6 Comparison between RDF-3X and our approach CloudSPARQL

(a) Response time for SP2Bech Query 3a; (b) Response time for SP2Bech Query 4;

(c) Response time for SP2Bech Query 8; (d) Response time for SP2Bech Query 9;

如图 4.6(a)所示，在对 Query 3a 进行查询时，本章方法对四种输入数据量大小均优于 RDF-3X 的计算性能。由于 Query 4 查询生成一个大数据量的结果集，RDF-3X 耗费了大量的时间来生成最终结果，与之相反地，本章方法只需要更少的时间就能得到结果。当输入数据集包含 400 万条 RDF 三元组时，RDF-3X 耗费了 3751.9 秒来得到查询结果，而本章方法仅需要 243.88 秒。当输入数据量增加到 1200 万和 1500 万条时，实验过程分别运行 RDF-3X 超过了 5 和 8 小时，但由于内存溢出没能输出最终结果，但本章方法仅使用了 478.4 和 597.98 秒。

在对 Query 8 和 Query 9 进行查询时，由于查询语句中具有高匹配度特性，因此查询结果集较小，RDF-3X 相较于本章方法，耗用了更少的时间得到查询结果。

图 4.6 的实验结果表明，当对具有高匹配度的 SPARQL 组图模式语句进行查询时，RDF-3X 相较于本章方法具有更好的查询性能。但当在查询低匹配度的 SPARQL 语句或输出结果集数据量较大时，本章方法具有更好的查询性能。

第三个实验是将本章方法分别与 Sesame main-memory 和 Sesame database 工具进行了对比测试。分别生成了基于 SP2Bench 的 800 万、1200 万、1500 万和 3000 万条 RDF 三元组作为测试数据集，并选用了 Query 2、Query 3a、Query 4、Query 6、Query 8 和 Query 9 作为测试 SPARQL 语句。然后，由于 Sesame Database 工具存在复杂的 RDF 数据存储策略，当试图将 800 万条 RDF 三元组存储入数据库时，整个过程持续了超过 12 小时，因此中断了该实验过程，并由此可得出 Sesame Database 工具不适应于对大规模 RDF 数据进行存储和查询的结论。表 4.3 描述了本章方法与 Sesame main-memory 工具之间的实验对比数据。

表 4.3 CloudSPARQL 与 Sesame main-memory 的对比实验结果

Table 4.3 Comparison between Sesame main-memory and our approach CloudSPARQL

	8 Million Triples		12 Million Triples		15 Million Triples		30 Million Triples	
	Sesame	CloudS	Sesame	CloudS	Sesame	CloudS	Sesame	CloudS
Query 2	143.99	252.96	240.79	394.53	294.97	458.08	Failed	82071
Query 3a	145.28	51.50	236.80	83.58	313.43	116.78	Failed	260.78
Query 4	Failed	370.43	Failed	478.40	Failed	579.98	Failed	887.69
Query 6	Failed	375.47	Failed	517.89	Failed	582.79	Failed	1154.99
Query 8	144.95	547.77	235.03	949,52	285.78	1245.74	Failed	2365.54
Query 9	143.81	285.37	229.26	402.9	283.91	502.32	Failed	916.42

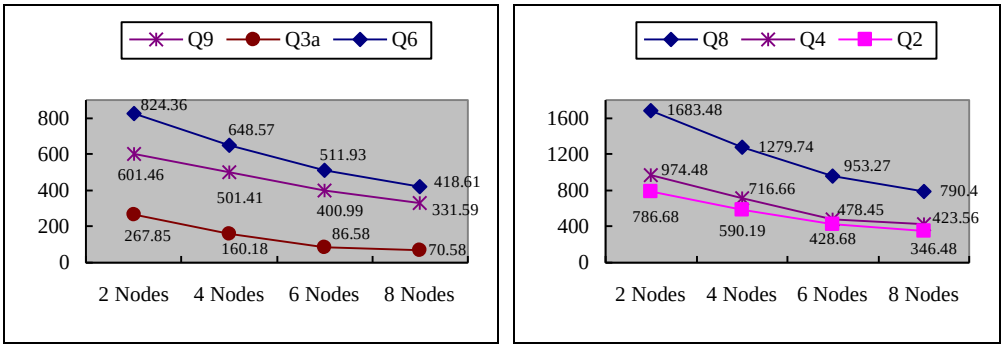
由于 Query 2、Query 8 和 Query 9 的高可匹配性和小数据量的结果集，Sesame main-memory 在实验中比本章方法更快地完成了查询任务。对 Query 3a，Sesame

在前三个数据集中分别使用了 145.28、236.80 和 313.43 秒的响应时间来完成查询。此外，由于结果集的数据量增大，Sesame main-memory 工具未能完成基于实验中四个数据集的 Query 4 和 Query 6 查询任务。与之相反，本章方法在合理时间内完成了查询计算。当 RDF 三元组数量增加到 3000 万条时，与使用 Jena In-memory 模型类似，Sesame main-memory 由于存在内存限制无法实现对所有四个测试查询语句的计算任务。

基于如表 4.3 所示的实验结果，当输入 RDF 三元组数据量较小时，Sesame main-memory 在对具有高可匹配性和结果数据集的数据量较小的 SPARQL 查询具有良好的计算性能，但当数据量增加时，其可扩展性不如本章方法。对低可匹配度和大结果集的 SPARQL 图模式查询，本章方法比 Sesame main-memory 有更好的计算性能。

除了上述的对比实验以外，还对本章方法的可扩展性进行了测试。首先，继续选取对比实验中的 SP2Bench 测试语句，并基于 1000 万条 RDF 三元组数据，并设定参与计算的计算结点数由 2 台递增为 8 台，来验证计算结点的增长对本章方法在计算性能方面的影响，图 4.7 描述了该项实验的结果数据统计。

如图 4.7 所示，对相同的查询基准 RDF 数据和测试 SPARQL 语句，本章方法的计算用时随着计算结点数的增加而递减。例如，对于 Query 6，当计算结点数为 2 时，本章方法查询用时为 824.36 秒；当计算结点数为 4 时，计算用时为 648.57 秒；当计算结点数为 6 时，计算用时为 511.93 秒；当计算结点数为 8 时，计算用时为 418.61 秒。对实验中的其他测试语句，具有同样的实验效果。因此，通过增加 Hadoop 环境中的计算结点数，本章方法能够获得更佳的计算性能和可扩展性。



(a) (b)

图 4.7 CloudSPARQL 递增计算结点的可扩展性实验结果

(a) Query 3a, Query 6, Query 9 计算用时；(b) Query 2, Query 4, Query 8 计算用时

Figure 4.7 CloudSPARQL experimental results for increasing the number of computing nodes

(a) Response time for Queries 9, 3a, and 6; (b) Response time for Queries 8, 4, and 2

最后，选取 Query 2、Query 4、Query 6 和 Query 9 作为测试语句，并使用 1000 万、2000 万、3000 万和 4000 万条的 RDF 三元组数据，在 8 台计算结点中验证本章方法对输入 RDF 数据量增加时的计算性能和可扩展性。图 4.8 描述了该项实验的结果数据，其中 X 轴为输入 RDF 三元组数据量大小，单位为条，Y 轴为查询用时，单位为秒。

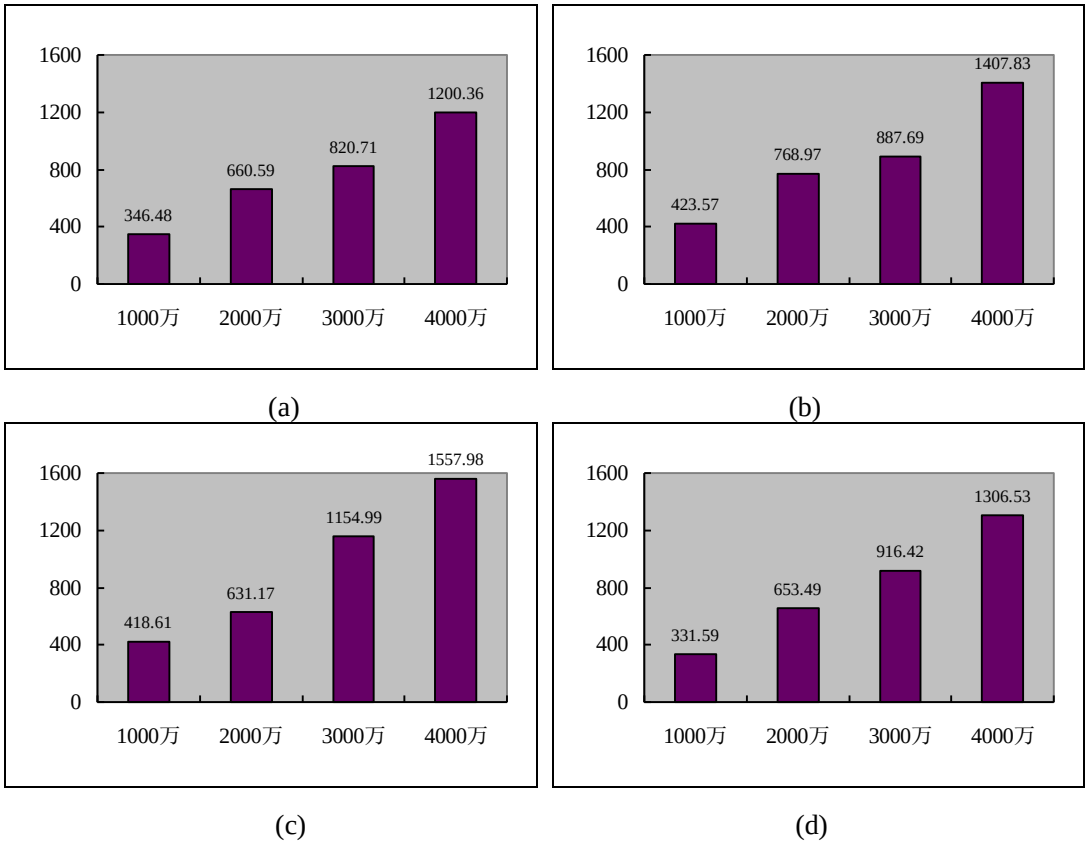


图 4.8 CloudSPARQL 递增 RDF 三元组数的可扩展性实验结果

(a) Query 2 计算用时；(b) Query 4 计算用时; (c) Query 6 计算用时；(d) Query 9 计算用时

Figure 4.8 CloudSPARQL experimental results for increasing the number of RDF triples

(a) runtime for Query 2; (b) runtime for Query 4; (c) runtime for Query 6; (d) runtime for Query 9.

如图 4.8 所示，当输入 RDF 三元组数据量增加时，完成上述四个查询语句的用时逐步递增，但其增长的速率小于数据量增加的速率。例如，针对于 Query 2，当对 1000 万条 RDF 三元组进行查询时本章方法用时为 346.88 秒，而对 4000 万条 RDF 三元组进行查询时的用时为 1200.36 秒，即当数据量增长为 4 倍时，查询的用时仅增加为 3.46 倍。对 Query 4、Query 6 和 Query 9，本项实验具有相似的结果。

综上所述，本章方法在针对于大规模 RDF 数据的 SPARQL 复杂组图模式查询时，较传统的查询引擎具有更好的查询效率和可扩展性。

4.6 本章小结

本章面向语义 Web 中存在的大规模 RDF 数据查询问题，提出了一种新的基于 MapReduce 的 SPARQL 复杂组图模式分布式查询方法。

首先，基于 RDF 和 SPARQL 形式化语义，提出了复杂组图模式在 MapReduce 模型下的相关解析模型定义、查询任务生成算法及并行化查询方法；然后，通过使用语义 Web 领域广泛采用的 SP2Bench 测试数据集与传统的单机环境下运行的查询引擎进行对比实验和可扩展性实验，证明了本章方法在对大规模 RDF 数据进行处理时，具备更优的计算性能和可扩展性。

5 基于 MapReduce 的 OWL 本体一致性分布式检测

5.1 引言

如本文第二章所述，基于描述逻辑的 OWL 是 W3C 推荐的语义 Web 本体描述语言标准。根据描述能力和推理判定能力的不同，OWL 1.0 分为了基于描述逻辑 SHIF 的 OWL Lite，基于描述逻辑 SHOIN 的 OWL DL[78]，及具备最强描述能力但推理不可判定的 OWL Full。在推理服务方面，描述逻辑 Tableau 算法可实现针对于 TBox 的概念可满足性、包含关系和等价性等的推理，同时上述推理服务均可规约为针对 ABox 的一致性检测推理。

然而，随着语义 Web 的快速发展和应用，传统单机环境下运行的描述逻辑推理引擎已无法适应于面向大规模 OWL 本体的推理任务需求。目前，语义 Web 研究领域已开始将云计算技术与本体推理理论相结合，研究如何实现高性能、可扩展的本体数据推理，并已提出了基于 MapReduce 的 RDF-S 规则推理方法 WebPIE 和 OWL EL 本体推理方法，但在针对基于描述逻辑的 OWL 本体推理方面仍未开展研究。

本章以实现云计算 MapReduce 环境下的大规模 OWL Lite 本体一致性分布式检测推理为主要研究内容，根据 OWL Lite 可映射为的描述逻辑 SHIF 形式化语义及其 Tableau 推理算法，并结合本文第三章中提出框架体系结构和基于 HBase 数据库的本体存储策略，提出一种新的基于 MapReduce 的描述逻辑 SHIF 分布式 Tableau 一致性检测推理算法。

首先，本章 5.2 小节对 OWL Lite 的形式化语义和对应描述逻辑 SHIF 的 Tableau 算法进行了描述，并给出了相关的解析模型定义；然后，在本章 5.3 小节中提出了基于 MapReduce 的描述逻辑 SHIF 分布式 Tableau 推理算法；最后通过使用基于描述逻辑 SHIF 的 LUBM 本体公共测试数据集，在 5.4 小节中对本章所提出方法和现有单机环境下的描述逻辑推理引擎进行了对比和分析。

5.2 相关术语及解析模型

5.2.1 OWL Lite 和描述逻辑 SHIF 的形式化语义

OWL Lite 是在 OWL DL 的表达能力基础之上限制了枚举类(owl:oneOf)、类不相交陈述(owl:disjointWith)、任意基数约束(owl:maxCardinality、owl:minCardinality 和 owl:cardinality 中的 *n* 只能为 0 或 1)等的本体描述语言，虽然其表达能力不如 OWL DL，但其增强了 OWL DL 中的公理约束，可以保证快速高效的推理，因此，OWL Lite 在语义 Web 本体建模领域已得到了广泛的应用。

表 5.1 描述了 OWL Lite 语言中的构子及各个构子与描述逻辑 SHIF 概念及其计算符号之间的映射关系，其中 C 和 D 表示描述逻辑概念名，R 和 S 为角色名。

表 5.1 OWL Lite 构子与描述逻辑 SHIF 语法的对应关系

Table 5.1 Relationship between OWL Lite constructors and DL SHIF Syntax

OWL 构子	**DL 描述**	**示例**
owl:Class	C, D	Man, Human
owl:ObjectProperty	R, S	*hasMother*, *hasParent*
owl:DatatypeProperty	P	*string*
rdfs:subClassOf	$C \sqsubseteq D$	Man $\sqsubseteq$ Human
rdf:subPropertyOf	$R \sqsubseteq S$	*hasMother* $\sqsubseteq$ *hasParent*
owl:inverseOf	R^-	owl:inverseOf(*teach*) = *was_Taught*
owl:TransitiveProperty	*Trans*(R)	owl:TransitiveProperty(*subRegionof*)
owl:intersectionOf	$C \sqcap D$	Man $\sqcap$ Human = Male
owl:allValuesFrom	$\forall R.C$	$\forall$*hasNationality*.CHINA
owl:someValuesFrom	$\exists R.C$	$\exists$*hasDegree*.PhD
owl:maxCaidinality	$\leq nR.C$ (n= 0 or 1)	≤n*hasChild*.Boy, (n= 0 or 1)
owl:minCardinality	$\geq nR.C$ (n= 0 or 1)	≥ n*hasChild*.Boy, (n= 0 or 1)
owl:cardinality	$= nR.C$ (n= 0 or 1)	=n*hasChild*.Boy, (n= 0 or 1)

描述逻辑 SHIF 在描述逻辑 ALC 的基础之上允许部分角色具有传递性，加入角色的逆算子，纳入角色分层扩展并允许函数性约束。描述逻辑 SHIF 概念的形式化描述如下。

设 N_C 是概念名的集合，N_R 是全部角色名的集合，N_R 包括一般性角色名 N_{R+} 和传递性角色名 N_{RP}，同时 $N_{R+} \cap N_{RP} = \varnothing$；$R^-$是逆角色，SHIF 角色集合是集合 $N_R \cup \{ R^- \mid R \in N_R \}$。定义 SHIF 概念集合为满足下列条件的最小集合。

① 若概念名 $C \in N_C$，则 C 是 SHIF 概念；

② 若 C, D 是 SHIF 概念，R 是 SHIF 角色，则$(C \sqcap D)$，$(C \sqcup D)$，$(\neg C)$，$(\forall R.C)$和$(\exists R.C)$都是 SHIF 概念；

③ 形如 $R \sqsubseteq S$ 称为角色包含公理，其中 R, S 属于 SHIF 角色集合。设 R 是由角色包含公理组成的一个集合，则称 R^+为一个角色分层，$R^+ = (R \cup \{Inv(R) \sqsubseteq Inv(S) \mid R \sqsubseteq S \in R\}, \dot{\subseteq})$，其中$\dot{\subseteq}$是$\sqsubseteq$在 $R \cup \{Inv(R) \sqsubseteq Inv(S) \mid R \sqsubseteq S \in R\}$上的自反传递闭包。

④ 若 R 是一个简单角色，则$(\leq 1R)$，$(\geq 2R)$是 SHIF 概念；

⑤ 若 R 是一个简单角色，如果$\neg Trans(R)$而且对任意 S，$S \dot{\subseteq} R$，S 也是一个简单角色。

描述逻辑 SHIF 解释 I 的形式化描述如下。

SHIF 解释 $I=<\Delta^I,(.)^I>$，其中，论域 Δ^I 是一个非空的个体集，$(.)^I$ 是解释函数。函数 $(.)^I$ 将每个概念映射为 Δ^I 的一个子集，将每个角色映射为 $\Delta^I \times \Delta^I$ 的一个子集。同时满足下列语义：

① $(C \sqcap D)^I = C^I \cap D^I$；

② $(C \sqcup D)^I = C^I \cup D^I$；

③ $(\neg C)^I = \Delta^I \setminus C^I$；

④ $(\forall R.C)^I = \{x \in \Delta^I \mid$ 对任意 $y \in \Delta^I$ ，如果$<x, y> \in R^I$，那么 $y \in C^I\}$；

⑤ $(\exists R.C)^I = \{x \in \Delta^I \mid$ 存在一 $y \in \Delta^I$，满足$<x, y> \in R^I$，而且 $y \in C^I\}$；

⑥ 对任意 $P \in N_R$，$<x, y> \in P^I$ 当且仅当$<y, x> \in (P^-)^I$ 对任意 $R \in N_{R^+}$，如果存在 $<x, y> \in R^I$ 而且$<y, x> \in R^I$，那么$<x, z> \in R^I$；

⑦ 对任意角色 R 和 S，$R \dot{\subseteq} S$，如果$<x, y> \in R^I$，那么$<x, y> \in S^I$；称解释 I 是 R^+ 的一个模型，当且仅当对所有 $R \sqsubseteq S \in R^+$ 都成立 $R^I \subseteq S^I$；

⑧ $(\leq 1R)^I = \{ x \in \Delta^I \mid$ 对于所有的 y 和 z，如果$<x, y> \in R^I$ 且$<x, z> \in R^I$，那么 $y = z \}$；

⑨ $(\geq 2R)^I = \{ x \in \Delta^I \mid$ 存在某两个 y 和 z，满足$<x, y> \in R^I$ 且$<x, z> \in R^I$，并有 $y \neq z\}$。

对某一解释 $I=<\Delta^I,(.)^I>$，如果使得 $C^I \neq \varnothing$，称 I 满足概念 C，I 为 C 的一个模型(Model)。对某一 TBox T，当 I 满足 T 内所有的包含关系，即对所有 $C \sqsubseteq D$，都有 $C^I \subseteq D^I$ 成立时，称 I 为 T 的一个模型。

对由一组断言 $C(a)$或 $R(a, b)$组成的 ABox A，其中 C 和 R 分别是 T 中概念和角色，并且 a 和 b 为个体，当 T 的所有模型 I 都满足 $x^I \in C^I$ 时，称 $C(a)$相对 T 被满足；如果 T 的所有模型 I 都满足$(a^I, b^I) \in R^I$，称 $R(a, b)$相对 T 被满足。当一个解释 I 满足 A 中所有断言时，称 I 为 A 的一个模型。

如果一个 ABox A 如果存在一个模型，称 A 是一致的(Consistent)；反之，如果不存在这样一个模型，则称 A 是非一致的(Inconsistent)。

在描述逻辑研究领域，Tableau 算法是进行 TBox 和 ABox 推理时的标准算法，并且针对 TBox 的概念可满足性、包含关系等问题的推理服务均可规约为对 ABox 的一致性问题的推理，因此，Tableua 算法的基本思想是判断 ABox 相关于 TBox 是否存在一个模型，即对输入 ABox 寻找一个解释，其证明过程详见文献[5]。

5.2.2 描述逻辑 SHIF 的 Tableau 算法

在对 ABox 的一致性问题进行推理验证时，描述逻辑 SHIF 的 Tableau 算法以 ABox 中所有的断言和其相关联的 TBox 为输入，首先根据 De Morgen 规则，在线性时间内将所有概念转化为否定范式(Negation Normal Form)，即否定符(¬)仅出现

在概念名前，并以此做为 Tableau 树的根节点 $L(x)$，其中 L 为概念的否定范式，x 为任一个体名。然后，Tableau 算法根据如表 5.2 所示的 Tableau 规则对 $L(x)$进行展开，其中关于阻塞、邻居和先驱的概念定义详见文献[5]。最后，直到没有规则可以再继续应用时，Tableau 算法构造一个完整的、无冲突的 Tableau 树，其中树节点为对应的解释，并称该 ABox 是相关与输入 TBox 一致的。反之，若在计算过程中检测到对某一个体 x，有冲突(Clash)存在时 Tableau 算法停止计算，并称该 ABox 是不一致的。

表 5.2 描述逻辑 SHIF 的 Tableau 算法规则

Table 5.2 Tableau rules for Description Logic SHIF

规则名	规则	
⊓-规则	如果	(1) $C_1 \sqcap C_2 \in L(x)$，且 x 没有被直接阻塞，
		(2)$\{C_1, C_2\} \not\subseteq L(x)$，
	那么	$L(x) \rightarrow L(x) \cup \{C_1, C_2\}$。
⊔-规则	如果	(1) $C_1 \sqcup C_2 \in L(x)$，且 x 没有被直接阻塞，
		(2) $\{C_1, C_2\} \cap L(x) = \varnothing$，
	那么	$L(x) \rightarrow \{C\}$，对于某个 $C \in \{C_1, C_2\}$。
∃-规则	如果	(1) $\exists\, S.C \in L(x)$，而且 x 没有被阻塞，
		(2) x 没有一个 S 邻居 y，使得 $C \in L(y)$，
	那么	新增一个节点 y，赋值 $L(\langle x, y\rangle) = \{S\}$且 $L(y) = \{C\}$。
∀−规则	如果	(1) $\forall S.C \in L(x)$，而且 x 没有被阻塞，
		(2) x 有一个 S 邻居 y，使得 $C \in L(y)$
	那么	新增一个节点 y，赋值 $L(\langle x, y\rangle) = S$ 且 $L(y) = \{C\}$。
$\forall'_+$-规则	如果	(1) $\forall S.C \in L(x)$，而且 x 没有被阻塞，
		(2)存在一个传递性角色 R，且 $R \dot{\subseteq} S$，
		(3) x 有一个 R 邻居 y，但$\forall R.C \notin L(y)$
	那么	$L(y) \rightarrow L(y) \cup \{\forall R.C\}$。
≥-规则	如果	(1) $\geq 2\, S \in L(x)$，而且 x 没有被阻塞，
		(2) x 没有 S 邻居 y 使得 $A \in L(y)$，
	那么	新增两个节点 y_1 和 y_2，赋值 $L(y_1)=\{A\}$，$L(y_2)=\{\neg A\}$，
		且 $L(\langle x, y_1\rangle)=\{S\}$，$L(\langle x, y_2\rangle)=\{S\}$。
≤-规则	如果	(1) $\leq 1 S \in L(x)$，而且 x 没有被直接阻塞，
		(2) x 有两个 S 邻居 y 和 z，且 y 不是 z 的先驱
	那么	(1) $L(z) \rightarrow L(z) \cup L(y)$，
		(2)如果 z 是 y 的先驱，那么 $L(\langle z, x\rangle) \rightarrow L(\langle z, x\rangle) \cup Inv(L(\langle x, y\rangle))$，
		否则 $L(\langle x, z\rangle) \rightarrow L(\langle x, z\rangle) \cup L(\langle x, y\rangle)$。

文献[5]中论述了描述逻辑 SHIF 的 Tableau 算法的可终止性、完备性和可靠性证明，本文在此不再做证明。

5.2.3 相关解析模型

由于在 MapReduce 环境下输入数据被划分为若干独立的数据块并基于键值对的格式进行处理，同时在每个 MapReduce 任务的计算过程中，各个计算结点之间不能进行消息传递，因此，在 MapReduce 环境中进行大规模 ABox 一致性检测推理的前提是将输入 ABox 数据划分为若干独立的部分。在给出 MapReduce 分布式计算模型下 Tableau 分布式推理算法的具体描述之前，本小节基于上述描述逻辑 SHIF 的语义及其 Tableau 算法扩展规则，先提出本文框架内的相关解析模型定义。

根据 OWL 本体基于描述逻辑的语义及 Tableau 推理算法，在对一个 OWL 本体库 ABox 的一致性问题进行推理验证时，Tableau 算法对某一个体 x 验证是否存在一个或多个冲突，即如果存在概念 C，有$\{\neg C, C\}\in L(x)$；或者存在角色 R 和 S，$R\dot{\subseteq}S$ 有$\{(\leq 1R), (\geq 2S)\}\subseteq L(x)$，称 Tableau 树 $L(x)$存在冲突。

然而，当两个不同个体 a 和 b 不具备语义相关性时，即便存在 $C(a)$和$\neg C(b)$，也不会导致冲突的产生，因此，可将 ABox 根据语义无关的个体划分为若干个子集。即在本体库中，若个体 x 和 y 的所有断言均不存在语义相关性，那么可将个体 x 和 y 所对应的断言划分为两个独立的部分，并以此为输入，参与 MapReduce 分布式计算。

定义 5.1 个体依赖(Individual Dependence)：对给定的 TBox T 和 ABox A，有 $C(x)\in A$，$D(y)\in A$，$R(x, y)\in A$，其中 C 和 D 为 T 中概念，R 为 T 中角色，x 和 y 为个体名，称个体 y 关于角色 R 依赖于个体 x。

基于上述定义，可将一个输入 ABox A 划分为 $A=\{A_1, A_2,..., A_n\}$，$n\geq 1$，其中 A_i 是与某一特定个体 a 相依赖的所有断言的集合，$1\leq i\leq n$。同时，对个体 a 而言，在 A 中必定存在一个概念断言 $C(a)$，以及 k 个角色断言 $R_1(a, b_1), R_2(a, b_2),..., R_k(a, b_k)$ 依赖于 a，$k\geq 1$。

定义 5.2 依赖断言集(Dependence Assertions of Individual x，DAI(x))：对给定的 TBox T 和 ABox A，有 $C(x)\in A$，C 为 T 中概念名，$\mathrm{DAI}(x) = \{ C, R_1(y_1), R_2(y_2),..., R_k(y_k), D_1(y_1), D_2(y_2),..., D_k(y_k) \}$，$k\geq 1$，其中 R_i 为 T 中角色，D_i 为 y_i 的概念声明，y_i 关于角色 R_i 依赖于 x，$1\leq i\leq k$。

因此，ABox 中的每个 DAI 模型可作为一个独立的推理数据单元 A_i，在每个 MapReduce 计算结点中进行分布式的计算。

定义 5.3 中间断言键值对(Intermediate Assertion Key-value Pair，IAKP)：对给定的某一 TBox T 和 ABox A，用键值对模型 $\mathrm{IAKP} = < x, \mathrm{DAI}(x)>$ 表示针对于 A 一致性推理 MapReduce 任务中的中间数据键值对，其中键 x 为 A 中的某一个体名，

值 DAI(*x*)为个体 *x* 在 *A* 中的依赖断言集。

定义 5.4 一致性键值对(Consistency Key-value Pair，CKP)：对给定的某一 TBox *T* 和 ABox *A*，键值对 CKP = < *Individual*, *Consistency*>，其中键 *Individual* 为 *A* 中的某一个体名，值 *Consistency* 为判定一致性的布尔值 *True* 或者 *False*。

CKP 模型用于表示 Tableau 分布式推理算法执行后，对每个独立的 ABox 单元 A_i 的一致性检测结果。

为更好地说明上述模型的含义及 MapReduce 环境下的分布式 Tableau 推理算法流程，基于可映射为描述逻辑 SHIF 的 LUBM 本体作为示例，其术语集在 univ-bench.owl 本体文件中描述，表 5.3 描述了部分概念概念断言和角色断言。

表 5.3 基于 LUBM 数据集的示例断言集

Table 5.3 Sample ABox from LUBM Benchmark

概念断言	角色断言
GraduateStudent(*GS0*)	advisor(*GS0*, *AP3*)
ResearchAssistant(*GS0*)	advisor(*GS1*, *AP0*)
University(*U3*)	takesCourse(*GS0*, *GC16*)
AssistantProfessor(*AP3*)	takesCourse(*GS1*, *GC33*)
AssistantProfessor(*AP0*)	teachingAssistantOf(*GS1*, *C25*)
GraduateStudent(*GS1*)	teachOf(*AP3*, *GC44*)
ResearchAssistant(*GS1*)	undergraduateDegreeFrom (*GS1*, *U145*)
University(*U145*)	undergraduateDegreeFrom (*GS0*, *U3*)
Course(*C25*)	
GraduateCourse(*GC16*)	
GraduateCourse(*GC33*)	
GraduateCourse(*GC44*)	

在表 5.3 描述的断言集中，存在两个概念断言 GraduateStudent(*GS0*)和 AssistantProfessor(*AP3*)，个体 *GS0* 和 *AP3* 之间存在角色断言 advisor(*GS0*, *AP3*)。根据定义 5.1，个体 *AP3* 关于角色 advisor 依赖于个体 *GS0*。同理，个体 *GC16* 关于角色 undergraduateDegreeFrom 依赖于 *GS0*，个体 *U3* 关于角色 takesCourse 依赖于个体 *GS0*。因此，可设定依赖于个体 *GS0* 的依赖断言集模型为：DAI(*GS0*) = {GraduateStudent, advisor(*AP3*), takesCourse(*GC16*), undergraduateDegreeFrom(*U3*), AssistantProfessor (*AP3*), GraduateCourse(*GC16*), University(*U3*)}。

以 DAI(*GS0*)为基础，可将表 5.3 断言集中关于个体 *GS0* 的所有相关联的断言

提取出来，并根据定义 5.3 生成一个中间断言键值对 IAKP=< *GS0*, DAI(*GS0*)>，并以此作为一个独立的数据块在 MapReduce 环境下进行分布式的 Tableau 计算，其算法将在本章 5.3 小节中详细阐述。推理检测结果被封装在一致性键值对 CKP 中，假设对 *GS0* 有 CKP=< *GS0*, *True*>，即与 *GS0* 相关的断言是一致的。

5.3 基于 MapReduce 的描述逻辑 SHIF 分布式 Tableau 推理算法

由于每个 MapReduce 任务以键值对的形式对输入数据进行划分和计算，基于描述逻辑 SHIF 的语义及 Tableau 算法规则，将 MapReduce 环境下的分布式 Tableau 推理过程设计为本体数据载入及划分和 Tableau 分布式推理两个计算阶段。图 5.1 描述了本文第 3 章所提出框架各个功能模块在执行 OWL 本体的一致性检测推理时的整个计算任务流程。

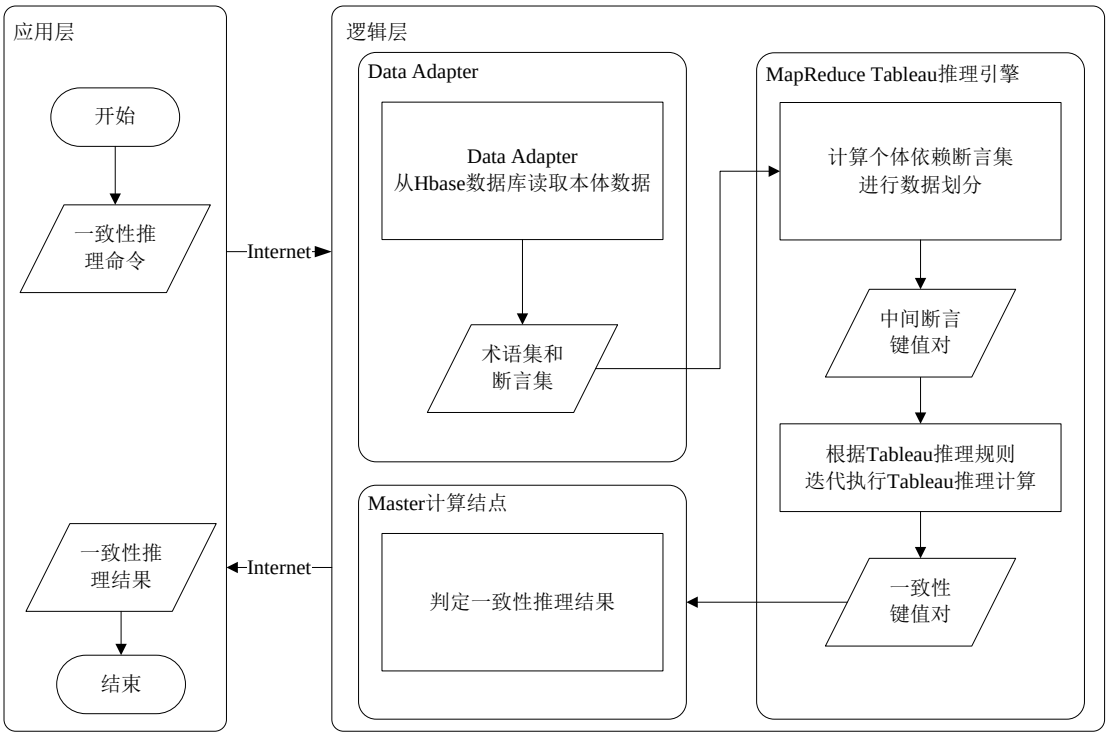


图 5.1 框架中 OWL 本体一致性推理计算流程

Figure 5.1 Workflow of OWL Consistency Checking in the Proposed Framework

算法 5.1 描述了基于 MapReduce 的描述逻辑 SHIF 分布式 Tableau 推理算法。为更好地进行描述，以表 5.3 中断言集和 LUBM 术语集作为示例对算法 5.1 进行详细阐述。算法 5.1 的步骤(1)至步骤(5)为数据载入及划分阶段，算法 5.1 的步骤(6)至步骤(9)为分布式 Tableau 推理阶段。

首先，MapReduce Tableau 推理引擎初始化一个 MapReduce 任务 Job_1，并在其

map 函数中调用框架中数据适配器以从 HBase 表 T_SP_O 中读取基于三元组格式的 OWL 本体断言集 *A* 和术语集 *T*，并将 *A* 存储到 HDFS 文件系统中，将 *T* 存储到 Distributed Cache 对象。在示例数据中，*T* 为 LUBM 术语集，*A* 为表 5.3 的断言集。

算法 5.1：基于 MapReduce 的描述逻辑 SHIF 分布式 Tableau 一致性检测

输入：本体术语集 *T* 和个体断言集 *A*

输出：*A* 相关于 *T* 的一致性检测结果

(1) 初始化 MapReduce 任务 Job_1，在 Job_1 的第一个 map 函数中，调用数据适配器模块读取 OWL 本体断言集 *A* 和术语集 *T*，执行 Distirbuted Cache ←*T*；

(2) 在 Job_1 的 *n* 个 Mapper 计算结点中，$n \geq 1$，执行第二个 map 函数：对 $C(x) \in A$，执行 DAI(*x*) ← DAI(*x*)∪{*C*}；对 $R(x, y) \in A$ 执行 DAI(*x*) ←DAI (*x*)∪{ *R*(*y*) }。生成 $IAKP_n$=<*x*, DAI(*x*)>，并将 $IAKP_n$ 存入 HDFS 临时文件 $Temp_1$；

(3) 初始化一个 MapReduce 任务 Job_2，并再次读取 OWL 本体断言集 *A*；

(4) 在 Job_2 的 map 函数中，对 $R(x, y) \in A$ 生成键值对<*y*, *R*(*x*)>，对 $D(y) \in A$ 生成键值对<*y*, *D*>；

(5) 在 Job_2 的 *m* 个 Reducer 计算结点中，$m \geq 1$，执行 reduce 函数：对键具有相同值 *y* 的<*y*, *D*>和<*y*, *R*(*x*)>，执行 DAI(*x*)←DAI(*x*)∪{*D*(*y*)}，并生成键值对 $IAKP_m$=<*x*, DAI(*x*)>，并将 $IAKP_m$ 存入 HDFS 临时文件 $Temp_2$；

(6) 初始化一个 MapReduce 任务 Job_3，在 map 函数中读取 $Temp_1$ 和 $Temp_2$，并从 Distributed Cache 中获取 *T*，继续执行步骤(7)；

(7) 对 *C*∈IAKP，根据 *C* 在 *T* 中概念定义生成否定范式 *C*'，构造键值对<*x*, *C*'>；对 *D*(*y*)∈IAKP，根据 *D* 在 *T* 概念定义生成否定范式 *D*'，构造键值对<*x*, *D*'(*y*)>；对 *R*(*y*) 概念定义，根据 *R* 在 *T* 中角色定义展开为原子角色 *R*'，构造键值对<*x*, *R*'(*y*)>；

(8) 在 Job_3 的 reduce 函数中，对具有相同键的< *x*, *C*'>, <*x*, *R*'(*y*)> 和 <*x*, *D*'(*y*)>，初始化 Tableau 树根节点 *L*(*x*)={*C*', *R*'(*x*, *y*), *D*'(*y*)}，根据表 5.2 的 Tableau 规则迭代地进行展开计算，如果存在冲突，生成 CKP=<*x*, *False*>，否则 CKP=<*x*, *True*>；

(9) Master 节点读取所有 CKP 模型，如存在一个 *False*，则推理结果为不一致，否则推理结果为一致。

然后，如算法 5.1 的步骤(2)所示，MapReduce Tableau 分布式推理引擎以 *A* 作为输入，在 MapReduce 环境中的 *n* 个 Mapper 计算结点中分别执行 Job_1 的第二个 map 函数，以计算个体之间的依赖关系。例如，当某一 Mapper 计算结点接收到输入概念断言 GraduateStudent(*GS0*)时，生成一个依赖断言集 DAI(*GS0*)，并将概念

GraduateStudent 作为元素并入 DAI(*GS0*)集合；对角色断言 *advisor*(*GS0*, *AP3*)，将角色和依赖个体组 *advisor*(*AP3*) 并入集合 DAI(*GS0*)。在对所有输入断言进行计算之后，针对特定个体 *GS0*，生成一个 IAKP 键值对模型：< *GS0*, DAI(*GS0*)>，最后将所有 IAKP 模型写入 HDFS 临时文件 $Temp_1$ 中。

算法 5.1 的步骤(3)至步骤(5)在第二个 MapReduce 任务 Job_2 中再次以 *A* 为输入数据源，对断言 *R*(*x*, *y*)和 *D*(*y*)计算生成依赖断言集，以实现个体 *x* 和 *y* 的之间的语义关联。例如，当 Job_2 中某一 Mapper 计算结点读取到断言 takesCourse(*GS0*, *GC16*)时，以个体 *GC16* 为键，以角色和个体的组合 takesCourse(*GS0*) 为值，生成中间键值对<*GC16*, takesCourse(*GS0*) >；当 Mapper 计算结点接收到断言 GraduateCourse (*GC16*)时，以个体 *GC16* 为键，以概念 GraduateCourse 为值，生成中间键值对<*GC16*, GraduateCourse>。然后，在 Job_2 的 reduce 函数中，如算法 5.1 步骤(5)所示，具有相同键值的<*GC16*, takesCourse(*GS0*)>和<*GC16*, GraduateCourse>被分发到一个 Reducer 计算结点，再通过构造一个集合 DAI(*GS0*)，将 GraduateCourse(*GC16*)并入 DAI(*GS0*)，并生成模型 IAKP=<*GS0*, DAI(*GS0*)>，最后将所有 IAKP 存入 HDFS 临时文件 $Temp_2$。

在 MapReduce 任务 Job_3 中，每个 Mapper 计算结点以 HDFS 文件 $Temp_1$ 和 $Temp_2$ 的数据块作为输入，并从 Distributed Cache 对象中获取本体术语集 *T*。然后，对输入键值对 IAKP 模型中的概念 *C*，根据其在 *T* 中的概念定义展开为原子概念描述，并转换为否定范式 *C*'。例如，设某一 Mapper 结点读取的 IAKP 模型中键为 *GS0*，值部分中包含了概念 GraduateStudent，则根据 LUBM 术语描述，将 GraduateStudent 替换为原子概念描述，即 GraduateStudent→∃takesCourse.GraduateCourse⊓Peron，并生成一个键值对<*GS0*, ∃takesCourse.GraduateCourse⊓Peron>。类似地，对 IAKP 中值部分存在的概念断言 *D*(y)，根据 *D* 在 *T* 中的概念定义展开为原子概念描述，并转换为否定范式 *D*'，然后生成键值对<x. *D*' (y)>。对 IAKP 中值部分存在的概念断言 *R*(*y*)，根据 *R* 在 *T* 中角色定义展开为原子角色 *R*'，构造键值对<*x*, *R*'(*y*)>。接下来，具有相同键的< *x*, *C*'>, <*x*, *R*'(*y*)>和<*x*, *D*'(*y*)>被分发到一个 Reducer 计算结点，并且每个 Reducer 根据接收到的断言生成一个 Tableau 树根节点 *L*(*x*)。

例如，在示例数据中与个体 *GS0* 相关的概念及角色断言 GraduateStudent, ResearchAssistant, University(*U3*), advisor(*AP3*), undergraduateDegreeFrom(*U3*) 和 AssistantProfessor(*AP3*)被分发到某一 Reducer 计算结点，因此，其 Tableau 根节点标注为 *L*(*GS0*)={GraduateStudent, ResearchAssistant, University(*U3*), advisor(*AP3*), undergraduateDegreeFrom(*U3*), AssistantProfessor(*AP3*)}。与个体 *GS1* 相关的概念及角色断言 GraduateStudent, ResearchAssistant, University(*U145*), advisor(*AP0*), AssistantProfessor(*AP0*), undergraduateDegreeFrom(*U145*), teachingAssistantOf(*C25*)

和 Course(*C25*)被分发到另一 Reducer 计算结点，其 Tableau 根节点标注为 *L*(*GS1*)={GraduateStudent, ResearchAssistant, University(*U145*), advisor(*AP0*), AssistantProfessor(*AP0*), undergraduateDegreeFrom(*U145*), Course(*C25*), teaching-AssistantOf(*C25*)}。同理，与个体 *AP3* 相关的断言 AssistantProfessor(*AP3*), teachOf(*GC44*)和 GraduateCourse(*GC44*)将被分发到第三个 Reducer 计算结点，其 Tableau 根节点标注为 *L*(*AP3*)={AssistantProfessor(*AP3*), teachOf(*GC44*), Graduate-Course(*GC44*)}。之后，reduce 函数根据表 5.2 的 Tableau 规则迭代地进行展开计算，如果存在冲突，生成 CKP=<*x*, *False*>，否则 CKP=<*x*, *True*>，并将推理结果写入 HDFS 文件。

最后，Master 节点读取所有 CKP 模型，并对所有推理结果进行统计。如果某一 Reducer 计算结点的推理结果 CKP 中包含了 False，那么 *A* 为不一致，否则 *A* 是一致性。

5.4 实验结果及分析

5.4.1 实验环境搭建

为验证本章方法的计算性能，而当前学术界尚不存在其它基于 MapReduce 的 OWL 本体一致性检测方法，首先进行了本章方法与 RacerPro、Pellet 和 HermiT 本体推理引擎的对比实验，然后对本章方法的可扩展性进行实验论证。其中，RacerPro、Pellet 是语义 Web 领域中在进行 OWL 本体推理时最广泛采用的描述逻辑推理机， HermiT 是当前语义 Web 领域中在进行 OWL 本体分类时推理效率最高的描述逻辑推理机。

实验数据选用了 LUBM 本体数据集，其基于 OWL Lite 描述且可映射为描述逻辑 SHIF，并已被语义 Web 研究领域广泛应用于语义 Web 推理工具性能测试。

在云计算硬件环境的搭建方面，采用了与本文第四章相同的实验环境。即将 9 台 PC 机通过 Ethernet 连接构成为一个模拟云计算环境，以此作为框架的 Hadoop 平台硬件基础。每个 PC 计算结点具有如下的硬件配置：Pentium IV 3.00 GHz CPU，1.5 GB 内存和 80 GB 硬盘空间。为进行对比试验，将 RacerPro、Pellet 和 HermiT 推理引擎部署在一个硬件配置为 Intel i5 2.50 GHz dual core processor，8 GB 内存空间和 4 TB 硬盘空间的计算机之上，并为 Jave 虚拟机分配了 6GB 的内存空间。

在软件设置方面，采用 Hadoop-0.20.2 和 HBase-0.20.6 版本作为本章所方法的软件基础，并选取了 Pellet 2.3.0、HermiT 1.3.5 和 RacerPro 1.9.0 教育版。

5.4.2 实验效果及分析

首先，在与 Pellet 和 HermiT 的对比实验中使用 LUBM 本体数据生成器生成了 LUBM(10)、LUBM(20)、LUBM(30)、LUBM(40)和 LUBM(50)的本体断言集。在

可扩展性实验中，生成了 LUBM(100)、LUBM(150)、LUBM(200)和 LUBM(250)的本体断言集。在本小节的后续部分，将本章方法标识为 CloudOWL，用符号“×”标识实验过程中出现的内存溢出错误。

图 5.2 描述了本章方法与 Pellet 的对比实验结果，其中 X 轴为输入 LUBM 本体数据量，Y 轴为推理用时，单位为秒。对五个实验输入数据，CloudOWL 和 Pellet 均返回推理结果为 *True*，即输入断言集是一致的。

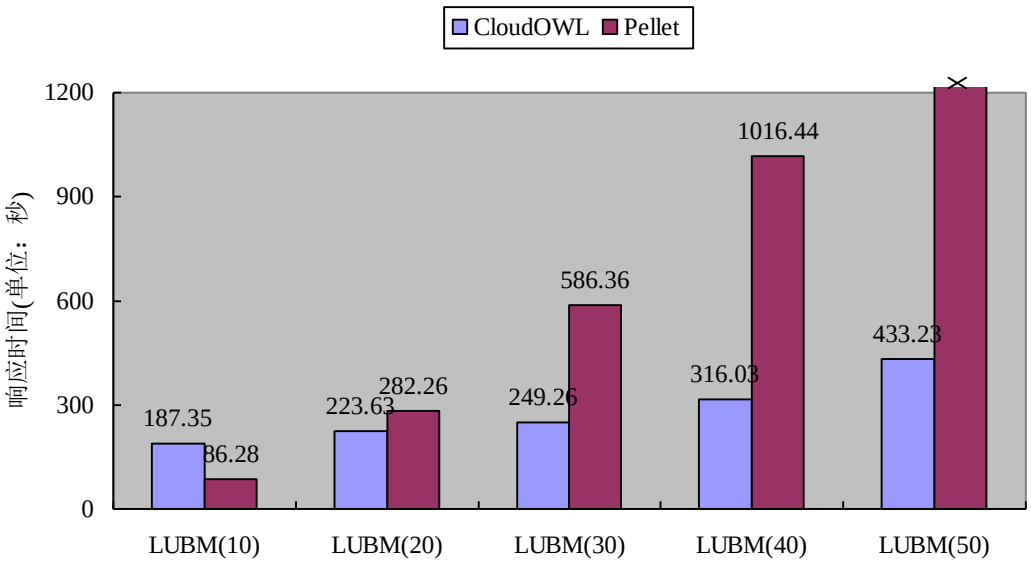


图 5.2 CloudOWL 与 Pellet 对比实验结果

Figure 5.2 Comparison results between Pellet and our approach CloudOWL.

LUBM(10)生成了大小约为 102MB 的 OWL 本体数据集，其中概念断言数为 263427 条，生成的角色断言数为 1052895 条。在对 LUBM(10)的一致性推理过程中，Pellet 用时为 85.92 秒，本章方法推理用时为 187.35 秒。

LUBM(20)生成了大小约为 217MB 的 OWL 本体数据集，其中概念断言数为 556572 条，生成的角色断言数为 2224750 条。如图 5.2 所示，当输入 OWL 本体数据量增加到 LUBM(20)时，本章方法推理用时 223.63 秒，而 Pellet 的推理用时则增至 282.26 秒。

LUBM(30)数据集中包含的概念断言数为 823170 条，角色断言数为 3284642 条，其数据集大小增加至 323MB。在实验过程中，Pellet 完成一致性检查用时为 586.36 秒，而本章方法仅用时 249.26 秒。

LUBM(40)分别生成了 1102205 条概念断言和 4391939 条角色断言，其数据集大小约为 431MB。在对 LUBM(40)的一致性检查实验中，Pellet 耗时 1016.44 秒，

而本章方法仅用时 316.03 秒。

LUBM(50)包含了 1381216 条概念断言和 5507426 条角色断言，其数据集大小约为 542MB。由于存在内存限制，在进行 LUBM(50)的一致性检测时，Pellet 在运行了较长时间后抛出内存溢出错误，即无法完成 LUBM(50)的一致性检测推理任务，而本章方法仅用时 433.23 秒。

以上实验结果表明，当输入 OWL 本体数据量较小时，由于 Pellet 在单机环境下运行，输入本体数据被直接载入内存，而本章方法由于需要对数据进行划分和网络传输，Pellet 在一致性检测推理性能方面优于本章方法，但在对大规模 OWL 本体数据进行一致性检查推理时，本章方法具有更好的计算性能。

图 5.3 描述了本章方法与 HermiT 推理机的对比实验结果。当输入 OWL 本体数据量较小时，如在对 LUBM(10)、LUBM(20)和 LUBM(30)的一致性进行推理验证时，HermiT 推理机因采用了 Hyper-Tableau 算法而具有较好的计算性能，实验过程分别用时 30.6 秒、42.67 秒和 59.55 秒，优于本章方法的 187.35 秒、223.63 秒和 249.26 秒。但随着输入数据量的增大，在对 LUBM(40)和 LUBM(50)进行计算时，HermiT 的 Hyper-Tableau 算法耗用了过多内存空间，而在单机环境下因为存在内存限制抛出而内存溢出错误，无法完成推理任务。相反，本章方法的计算用时在可接受范围之内，分别为 316.03 秒和 433.23 秒。因此，图 5.3 的实验结果表明，本章方法在对大规模 OWL 本体一致性问题推理时，比 HermiT 具有更好的计算性能。

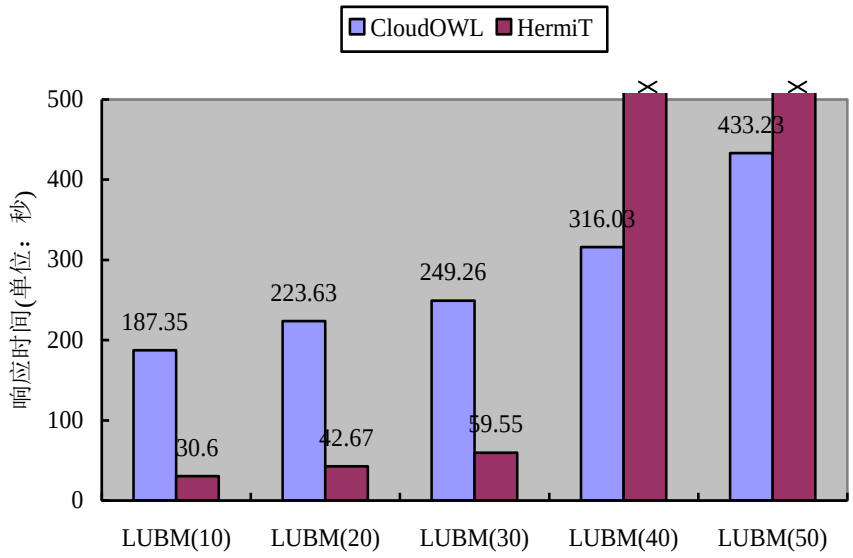


图 5.3 CloudOWL 与 HermiT 对比实验结果

Figure 5.3 Comparison results between HermiT and our approach CloudOWL

图 5.4 描述了本章方法与 RacerPro 推理机的对比实验结果。在对输入的五个 OWL 本体数据进行一致性检查时，RacerPro 的推理性能均不如本章方法。例如，

对 LUBM(10)数据集，RacerPro 用时 632.56 秒，而本章方法仅用时 187.36 秒；对 LUBM(20)数据集，RacerPro 用时 2426.78 秒，而本章方法仅用时 223.63 秒；当输入数据量增加至 LUBM(30)、LUBM(40)和 LUBM(50)时，RacerPro 由于内存空间限制无法完成推理任务，而本章方法均在可接受时间内返回推理结果。因此，图 5.4 的实验结果表明，RacerPro 推理机在对大规模的 ABox 一致性检测推理方面，其计算性能和可扩展性方面均存在不足。

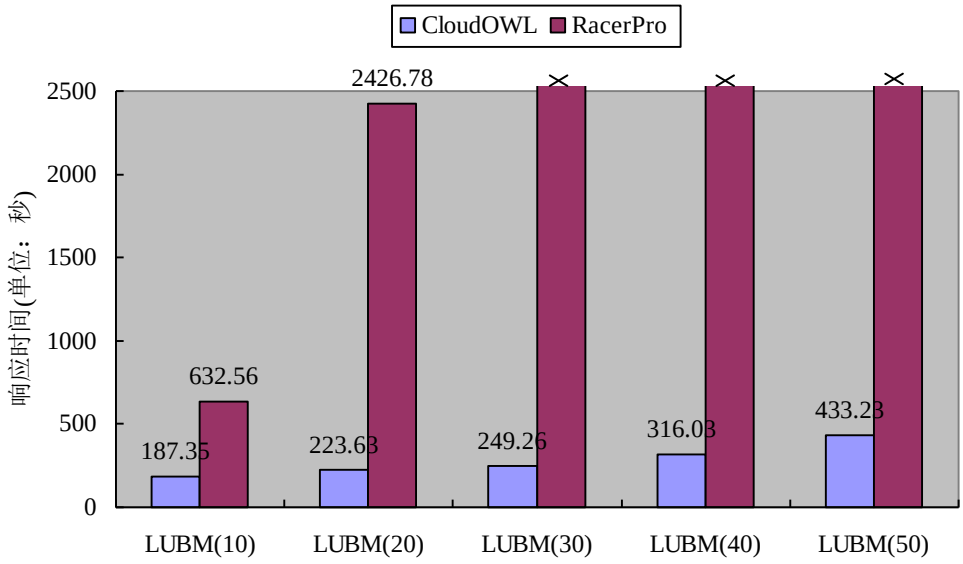


图 5.4 CloudOWL 与 RacerPro 对比实验结果

Figure 5.4 Comparison results between RacerPro and our approach CloudOWL

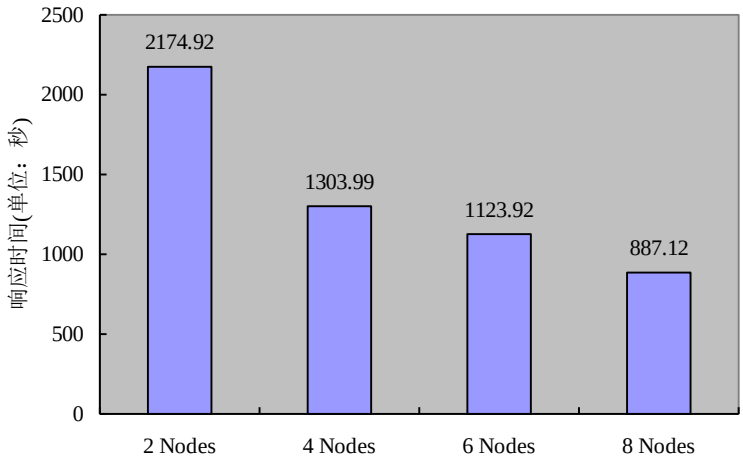


图 5.5 CloudOWL 递增计算结点数的可扩展性实验结果

Figure 5.5 CloudOWL experimental results for increasing the number of computing nodes

图5.5描述了计算结点数递增时的可扩展性实验结果。首先，生成了LUBM(100)数据集，其数据量超过了 1GB，并分别在 2、4、6、8 台计算结点的 MapReduce 实验环境中进行测试。

如图 5.5 所示，当仅有 2 台计算结点参与推理时，本章方法用时多达 2174.92 秒；当有 4 台计算结点参与计算时，本章方法用时为 1303.99 秒；当有 6 台计算结点参与计算时，本章方法用时为 1123.92 秒；当有 8 台计算结点参与计算时，本章方法用时为 887.12 秒。因此，通过增加 MapReduce 计算结点数，本章方法的计算性能可得到进一步提升。

最后，生成了更大规模的数据集 LUBM(100)、LUBM(150)、LUBM(200)和LUBM(250)来进一步验证本章方法的可扩展性。如图 5.6 所示，当输入本体数据量增加时，本章方法的推理用时同时递增，但递增的速率低于数据量增长的倍数。例如，当输入数据量由 LUBM(100)增加为 1.5 倍的 LUBM(150)时，本章方法推理用时从 887.12 秒增加至 1307.36 秒，用时增加约 1.47 倍；当输入数据量增加至LUBM(100)的2倍，即LUBM(200)时，本章方法推理用时由887.12秒增加至1723.26秒，仅增加 1.94 倍。因此，本章方法在处理大规模 OWL 本体的一致性推理任务时，具有较好的可扩展性。

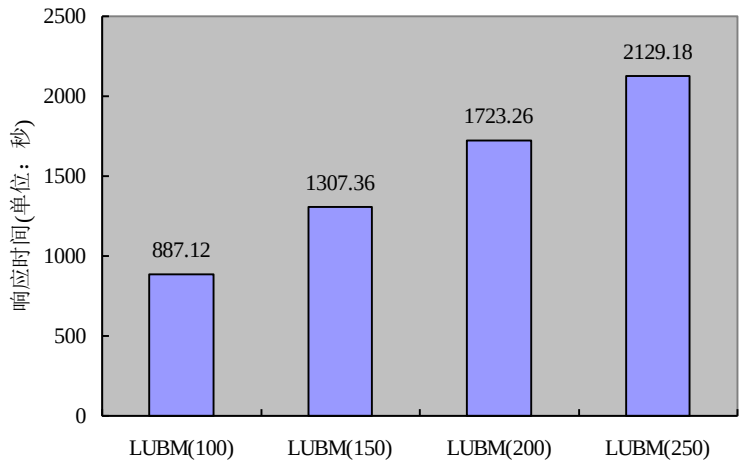


图 5.6 CloudOWL 递增输入本体数据量的可扩展性实验结果

Figure 5.6 CloudOWL experimental results for increasing the number of OWL ontologies

综合以上所有实验结果可得出如下结论。Pellet 和 HermiT 推理引擎在对输入数据量较小的 OWL 本体数据进行一致性推理时比本章方法具有稍好的计算性能，而 RacerPro 的一致性推理性能不如本章方法。在对大规模 OWL 本体数据的一致性进行推理时，传统单机环境下的 Pellet、HermiT 和 RacerPro 均存在内存限制，

无法完成推理任务，而本章方法均能在合理时间内完成计算，因此本章方法在可扩展性方面优于上述传统推理引擎。

5.5 本章小结

本章面向语义 Web 中存在的大规模 OWL 本体推理问题，提出了一种新的 MapReduce 环境下的大规模 OWL 本体一致性分布式推理方法。

首先，结合 OWL Lite 本体所对应的 SHIF 描述逻辑语义及其 Tableau 算法，论述了相关的模型定义以及基于 MapReduce 的 OWL 本体数据划分方法和分布式 Tableau 推理算法。然后，通过使用语义 Web 领域广泛采用的 LUBM 测试数据集与传统的单机推理引擎进行对比实验，证明了本章方法在对大规模 OWL 本体数据进行计算推理时，具备更优的计算性能和可扩展性。

6 基于 MapReduce 的 SWRL 规则分布式推理

6.1 引言

由于基于描述逻辑的OWL Lite和OWL DL本体描述语言需要保证其推理的可判定性，对本体的描述能力进行了限制，而无法对一般形式的规则进行描述。W3C将 OWL 子语言 OWL DL、OWL Lite 与 RuleML 相结合，提出了具备更强逻辑表达能力的语义 Web 规则描述语言 SWRL，并作为语义 Web 体系结构及其本体数据管理的核心内容之一。目前，语义 Web 研究领域还没有一个专用的 SWRL 规则推理引擎，研究人员主要使用的规则引擎 Jess 和 Pellet 来完成在 DL-Safe 限制下的 SWRL 规则推理任务。

但随着 OWL 本体数据量的不断增长，上述单机环境下运行的推理引擎由于需要将本体数据和规则载入内存，在对大规模本体数据进行 SWRL 推理时，存在计算性能和可扩展性不足等问题。近年来，为解决传统语义 Web 数据管理和推理工具在处理海量本体数据时存在的不足，学术界逐步形成了以语义 Web 和云计算研究领域相结合的新研究方向，但到目前为止，在应对大规模语义 Web 本体数据的 SWRL 规则推理方面，业界仍然缺少一种高性能、易扩展的解决方案。

本章的工作主要是基于 W3C 提出的 OWL 和 SWRL 形式化定义，并结合本文第三章中提出框架体系结构和基于 HBase 的 OWL 存储策略，研究并提出了云计算环境下的 SWRL 规则分布式推理框架 CloudSWRL。

首先，本章 6.2 小节对 SWRL 的形式化语义进行了描述，并给出了本文所提出框架中相关的术语和解析模型定义；其次，为尽可能减少 MapReduce 推理任务数，进而实现推理性能的优化，在本章 6.3 小节中提出了基于 MapReduce 的推理计划生成算法，并以此为基础，在 6.4 小节提出了在 DL-Safe 限制下的基于 MapReduce 的 SWRL 规则分布式推理算法。最后通过使用语义 Web 研究领域广泛采用的 LUBM 本体公共测试数据集，在 6.5 小节中对本章所提出方法和传统 SWRL 规则推理引擎进行了对比实验和可扩展性实验。

6.2 相关术语及解析模型

6.2.1 SWRL 规则语法及其形式化语义

根据 W3C 对 SWRL 语言的定义，每条 SWRL 规则由规则前件(Antecdent)和规则后件(Consequent)两部分组成，其含义为：当 SWRL 规则前件中指定的条件为真时，规则后件中指定的条件也一定为真。

每条 SWRL 规则的前件和后件均可由零或多个规则原子(*Atom*)的合取组成，

其形式化描述为：

$$\left(\bigwedge_{i=1}^{n} Atom_i\right)\rightarrow\left(\bigwedge_{j=1}^{m} Atom_j\right) \tag{6.1}$$

其中，$n\geq 0$，$m\geq 0$。每个规则原子可由 $C(x)$、$P(x, y)$、$Q(x, z)$、sameAs(x, y)、differentFrom(x, y)或 SWRL 内置函数构成，C 代表 OWL DL 描述，P 是一个 OWL DL 个体值属性(Individual-valued Property)，Q 是一个 OWL DL 数据值属性(Data-valued Property)，x 和 y 可为 SWRL 变量或 OWL 个体，z 可以为 SWRL 变量或者 OWL 数据常量值。在 OWL Lite 语境下，$C(x)$类型的规则原子中 C 被限定为 OWL 类名。

在 OWL 本体库 Ω 中，当 x 是类描述 C 的一个实例时，规则原子 $C(x)$成立；当 x 与 y(或 z)相关关于属性 P(或 Q)时，规则原子 $P(x, y)$(或 $Q(x, z)$)成立；当 x 和 y 被解释为同一对象时，规则原子 sameAs(x, y)成立；当 x 和 y 被解释为不同对象时，规则原子 differentFrom(x, y)成立。

SWRL 的模型论语义(Model-theoretic semantics)是对 OWL 语义的扩展。其基本思想是将绑定(Binding)定义为 OWL 解释的扩展，并将变量映射为论语的元素。一个 SWRL 规则被一个解释满足，当且仅当满足前件的解释同时满足后件。

根据 OWL 语义和抽象语法，一个 OWL 解释是一个六元组，$I = <R, EC, ER, L, S, LV>$，其中 R 是资源的集合，$LV\subseteq R$ 是字面量值的集合，EC 是一个映射，分别将类和数据类型映射为 R 和 LV 的子集，ER 是一个从属性到二元关系 R 的映射，L 是一个从确定字面量到 LV 元素的映射，S 是一个由个体名到 EC(owl: Thing)的映射。

给定一个 OWL 解释 I，一个 SWRL 绑定 $B(I)$是 I 的扩展，其中 S 将个体变量(i-variables)映射为 EC(owl: Thing)的元素，L 将数据变量(d-variables)映射为 LV 的元素。表 6.1 描述了一个规则原子被一个绑定 $B(I)$满足是需满足的条件，其中 C 是一个 OWL DL 描述，P 是一个 OWL DL 个体值属性，Q 是一个 OWL 数据值属性，x 和 y 是变量或 OWL 个体，z 是 OWL 数据值或变量。

一个绑定 $B(I)$满足一个前件 A 当且仅当 A 为空或 $B(I)$满足 A 中所有规则原子。一个绑定 $B(I)$满足一个后件 C 当且仅当 C 非空并且 $B(I)$满足 C 中每个规则原子。一个 SWRL 规则被一个解释 I 满足当且仅当每个绑定 $B(I)$满足前件，并且 $B(I)$满足后件。

表 6.1 SWRL 规则原子绑定的解释条件

Table 6.1 Interpretation Conditions of SWRL Atom

规则原子	解释条件
$C(x)$	$S(x) \in EC(C)$
$P(x, y)$	$<S(x), S(y)> \in ER(P)$
$Q(x, y)$	$<S(x), L(z)> \in ER(Q)$
sameAs(x, y)	$S(x) = S(y)$
differentFrom(x, y)	$S(x) \neq S(y)$

例如，有 SWRL 规则：

$$\text{parent}(?x, ?y) \wedge \text{brother}(?y, ?z) \rightarrow \text{uncle}(?x, ?z) \quad (6.2)$$

其中，parent, brother 和 uncle 为 OWL 类描述。那么给定一个解释 $I = <R, EC, ER, L, S, LV>$，一个绑定 $B(I)$扩展 S 为将变量?x, ?y, ?z 映射为 EC(owl:Thing)的元素 a, b 和 c。规则前件被 $B(I)$满足当且仅当$(a, b) \in ER$(parent)及$(b, c) \in ER$(brother)。规则后件被 $B(I)$满足当且仅当$(a, c) \in ER$(uncle)。因此，规则 6.2 满足解释 I，当且仅当每个绑定 $B(I)$，即$(a, b) \in ER$(parent)及$(b, c) \in ER$(brother)时，有$(a, c) \in ER$(uncle)。其形式化描述为：

$$\forall a, b, c \in EC(\text{owl:Thing}) \quad (6.3)$$

$$((a, b) \in ER(\text{parent}) \wedge (b, c) \in ER(\text{brother}) \rightarrow (a, c) \in ER(\text{uncle})) \quad (6.4)$$

6.2.2 相关解析模型

为实现在 MapReduce 环境下的 SWRL 规则分布式推理，需要将 SWRL 规则解析为适应于 MapReduce 键值对格式的数据模型。基于 6.2.1 小节中描述的 SWRL 句法和形式语义，提出本章中的相关模型定义。另外，为实现可判定的 SWRL 规则推理，本章主要针对在 DL-safe 限制下对本体库中已知 OWL 个体和公理进行推理。目前，本章方法支持 $C(x)$和 $P(x, y)$类型的 SWRL 规则原子，并规定 SWRL 规则前件至少包含一个规则原子，规则后件中有且仅有一个规则原子。

定义 6.1：SWRL 规则原子三元组(Atom Triple, AT)：$AT=(A(s), A(p), A(o))$对应了一个 SWRL 规则原子，其中 $A(p)$为 OWL 本体属性 P 或 Q，$A(s)$和 $A(o)$可为变量、OWL 个体、常量或 OWL 类名 C。

定义 6.2 SWRL 规则解析模型(*R*)：$R=(B, H)$对应了任一 SWRL 规则，其中 B 为前件规则原子三元组解析模型集合 $B=\{AT_1,\ldots, AT_n\}$，$n \geq 1$；H 为后件规则原子三元组解析模型集合 $H=\{AT_c\}$。

定义 6.3 SWRL 绑定模型：对 SWRL 规则原子三元组 AT，当 AT 中仅存在一个变量 v 时，AT 在本体库 Ω 中的绑定模型 $B(AT)=(v, a)$；当 AT 存在变量 v_1 和 v_2

时，$B(AT)=((v_1, a), (v_2, b))$。其中 a, b 为在本体库 Ω 中满足 A 中变量的 OWL 个体或常量，即 $C(a)$或 $P(a, b)$在 Ω 中成立。

定义 6.4 共同变量(*cv*)与非共同变量(*nv*)：若某一 SWRL 规则解析模型 R 中的规则原子三元组存在变量 v_1 和 v_2，当 v_1 同时存在于两个或多个规则原子三元组模型 $AT_1, AT_2,\ldots, AT_n$，$n\geq2$，称 v_1 为 $AT_1, AT_2,\ldots, A_n$ 的共同变量；当 v_2 仅存在于一个规则原子解析模型 AT 中，称 v_2 为非共同变量。

以规则 6.2 作为例说明上述模型定义。根据定义 6.1，前件规则原子 parent(?x, ?y)可解析为 AT_1=(?x, parent, ?y)，brother(?y, ?z) 可解析为 AT_2=(?y, brother, ?z)，后件规则原子 uncle(?x, ?z)可解析为 AT_c=(?x, brother, ?z)。因此，根据定义 6.2，规则 6.2 对应的 SWRL 规则解析模型为 R=({ AT_1, AT_2},{AT_c})。设在本体库中，个体 a 和 b 使得 AT_1 成立，即有 parent(a, b)存在于本体库中，那么 $B(AT_1)$=((?x, a), (?y, b))。同理，设个体 b 和 c 使得 AT_2 成立，即有 brother(b, c)存在于本体库中，那么 $B(AT_2)$=((?y, b), (?z, c))。另外，根据定义 6.4，由于变量?y 同时存在于 AT_1 和 AT_2 中，因此?y 为 AT_1 和 AT_2 的共同变量，而变量?x 和?z 为非共同变量。

对 SWRL 规则原子三元组 AT_1 和 AT_2，在本体库 Ω 中，若 AT_1 存在 SWRL 绑定模型的集合 $U_1=\{B_1(AT_1), B_2(AT_1),\ldots, B_j(AT_1)\}$，$j\geq1$，$AT_2$ 存在解释模型的集合 $U_2=\{B_1(AT_2),\ldots, B_k(AT_2)\}$，$k\geq1$，当 AT_1 和 AT_2 存在共同变量 v 时，使得 AT_1 和 AT_2 同时成立的绑定模型 $B(AT_1, AT_2)\in(U_1 \bowtie_v U_2)$，即对集合 U_1 和 U_2 进行以共同变量 v 为基准的连接操作；当 AT_1 和 AT_2 之间仅存在非共同变量时，$B(AT_1, AT_2)\in(U_1\times U_2)$，即对集合 U_1 和 U_2 取笛卡尔集。

因此，对规则解析模型 $R=(B, H)$，$B=\{AT_1, AT_2,\ldots, AT_n\}$，$n\geq1$，$H=\{AT'\}$，前件规则原子解析模型集合 B 的绑定模型 $B(AT_1, AT_2,\ldots, AT_n)$ 为：

$$(\{B(AT_1)\} \bowtie_v \{B(AT_1)\}\ldots\bowtie_v \{B(AT_j)\}\times\{B(AT_{(j+1)})\}\times\ldots\times\{B(AT_n)\}), \quad (6.5)$$

其中，$AT_1, AT_1,\ldots, A_j$ 之间存在共同变量，$A_{(j+1)}, A_{(j+2)},\ldots, A_n$ 中仅存在非共同变量。推理结果为 H 中规则原子解析模型 A'变量对应的绑定模型：

$$B(AT')\in\{ B(AT_1, AT_2,\ldots, AT_n) \}。 \quad (6.5)$$

由于 MapReduce 模型以键值对的形式对数据进行分布式计算，当规则中存在多个不同的共同变量时，SWRL 推理需要由多个 MapReduce 任务组合完成。

定义 6.5 SWRL 推理计划模型：$L=\{job_1, job_2,\ldots, job_n\}$，$n\geq1$，其中每个 job = (DS, {$subjob_1$, $subjob_2$,…, $subjob_m$})，$m\geq1$，DS 为标识符。当推理任务以 HBase 为数据源时，DS 标识为 D；当以 HDFS 为数据源时，DS 标识为 F。模型 $subjob$=(Tag, {AT_1, $AT_2,\ldots, AT_j$})，$j\geq1$，其中 AT 为规则原子三元组模型，Tag 为标识符，当 $AT_1, AT_2,\ldots, AT_j$ 中存在共同变量 v 时，Tag 标识为 v；当 $AT_1, AT_2,\ldots, AT_j$ 中仅存在非共同变量时，Tag 标识为 Neg。

定义 6.6 SWRL 推理中间键值对数据模型：$IKV=<(v, I[v]), (\{AT\}, B(\{AT\}))>$，其中 v 为变量名，$I[v]$为 $B(\{AT\})$中变量 v 的映射值，$B(\{AT\})$为满足规则原子三元组集合$\{AT\}$的绑定模型。

图 6.1 描述了本文第三章提出的体系结构中，各个功能模块在进行基于 MapReduce 的 SWRL 规则分布式推理时的计算流程。

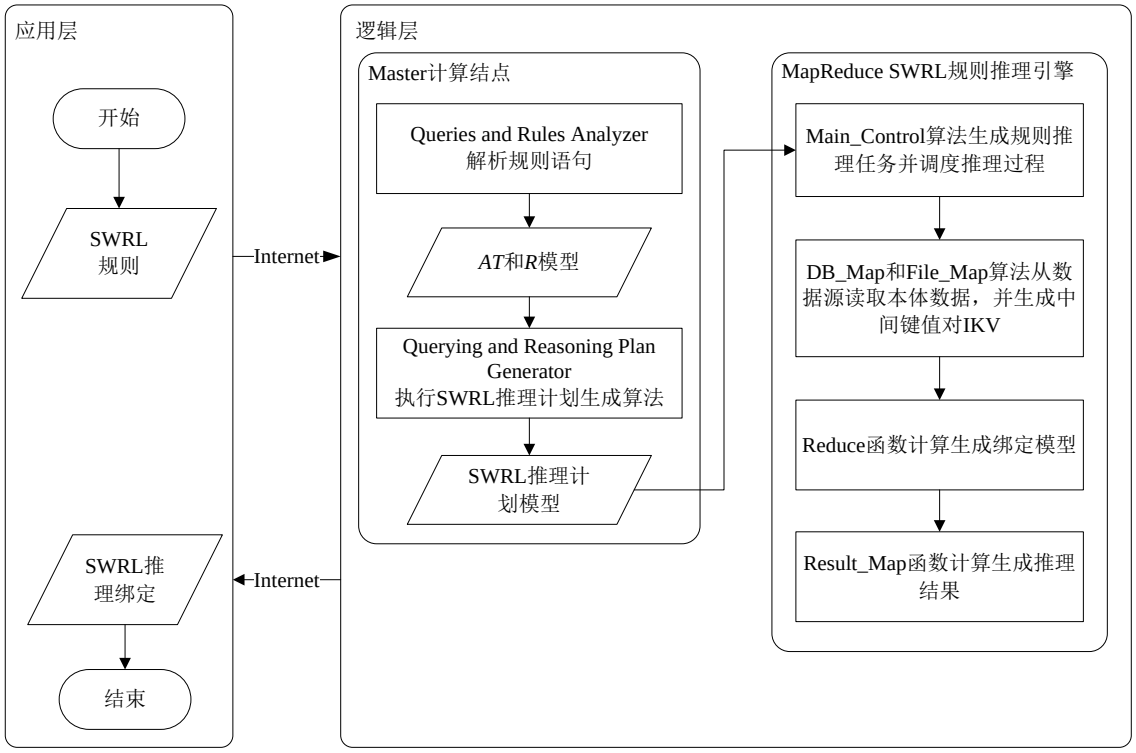


图 6.1 框架中 SWRL 规则分布式推理计算流程

Figure 6.1 Workflow of SWRL Reasoning in Proposed Framework.

6.3 SWRL 规则的 MapReduce 推理任务生成算法

当框架中应用层提交一个待推理的 SWRL 规则时，框架中逻辑层在执行具体的 SWRL 分布式推理之前，首先执行 SWRL 规则推理任务生成算法，以确定推理任务的执行流程，并以此减少推理过程中的 MapReduce 任务执行数，实现性能的优化。

为实现云计算环境下基于 SPARQL 的本体数据分布式查询，文献[16]中描述对查询语句中变量 v 取 E-count(v)值的定义，并基于贪心选择策略提出了查询计划的生成方法 Relaxed-Bestplan。本小节对该方法加以改进，提出一种针对 SWRL 规则的推理计划生成算法，其具体描述如算法 6.1 所示。

算法 6.1：基于 MapReduce 的 SWRL 规则推理计划生成

输入：SWRL 规则解析模型 R

输出：SWRL 推理计划模型 L

(1) 输入 $R=(B, H)$，$B=\{AT_1, AT_2,\ldots, AT_n\}$，$n\geq 1$，$H=\{AT_c\}$。

(2) 将 B 中所有变量 $v_1, v_2,..., v_f$，$1\leq f\leq 2n$，划分为共同变量集合$\{cv_1, cv_2,\ldots, cv_i\}$和非共同变量集合$\{nv_1, nv_1,\ldots, nv_j\}$，$i\geq 0$，$0\leq j\leq (2n-i)$。

(3) 设集合 CV、NV 和 U，对 k=1, 2,…, i，计算 $N_k\leftarrow$E-count(cv_k)，再执行 $CV\leftarrow CV\cup\{<cv_k, N_k>\}$。对 l=1,2,…, j，执行 $NV\leftarrow NV\cup\{nv_l\}$。执行 $U\leftarrow B$。

(4) 计算 $cl\leftarrow$length(CV)，cl 为集合 CV 中元素个数。当 cl=0 时，执行步骤 4；当 $cl\geq 1$ 时，对 CV 按 *E-count* 值进行升序排列，但若存在$<cv_s, N_s>\in CV$ 和$<cv_t, N_t>\in CV$，使得 $cv_s\neq cv_t$，$N_s=N_t$，则计算 cv_s 和 cv_t 在 B 中出现次数 T_s 和 T_t，若 $T_s>T_t$，将$<cv_s, N_s>$在 CV 中排序优先于$<cv_t, N_t>$，若 $T_s=T_t$，再计算 cv_s 和 cv_t 在 $A_p(s)$出现次数 S_s 和 S_t，p=1, 2,…, n， 如果 $S_s>S_t$，则将$<cv_s, N_s>$在 CV 中排序优先于$<cv_t, N_t>$。

(5) 设集合 *Temp*，参数 $e\leftarrow 1$。设 job_e 标识符 *DS* 为 D。对 x=1,…, cl，取 $cv_x\in CV$，$AT_p\in U,\ldots, AT_{(p+q)}\in U$，$p\geq 1$，$q\geq 1$，且满足 $cv_x\in AT_p,\ldots, cv_x\in AT_{(p+q)}$。再执行 $subjob_x\leftarrow(cv_x, \{AT_p,\ldots, AT_{(p+q)}\})$，$x\leftarrow x+1$，$U\leftarrow U-\{AT_p,\ldots, AT_{(p+q)}\}$，对 $z=p,\ldots, (p+q)$，若 AT_z 中存在除 cv_x 以外的共同变量 v_z 且 $v_z\notin Temp$，执行 $Temp\leftarrow Temp\cup\{v_z\}$。对生成的 x 个 *subjob* 模型，执行 $job_e\leftarrow(DS, \{subjob_1, subjob_2,\ldots, subjob_x\})$，$L\leftarrow L\cup\{job_e\}$，$e\leftarrow e+1$。

(6) 计算 $c\leftarrow$length($Temp$)，c 为集合 *Temp* 中元素个数。当 $c\geq 1$ 时，对 r=1,2,…, c，取 $v_r\in Temp$，$AT_d\in B,\ldots, AT_{(d+g)}\in B$，$d\geq 1$，$g\geq 1$，满足 $v_r\in AT_d,\ldots, v_r\in AT_{(d+g)}$。执行 $subjob_r\leftarrow(v_r, \{AT_d,\ldots, AT_{(d+g)}\})$，对 $u=d,\ldots, (d+g)$，若 AT_u 中存在除 v_r 以外的共同变量 v_u 且 $v_u\notin Temp$，执行 $Temp\leftarrow Temp\cup\{v_u\}$，$c\leftarrow c+1$。设 job_e 标识符 *DS* 为 F，对生成的 r 个 *subjob* 模型，执行 $job_e\leftarrow(DS, \{subjob_1,\ldots, subjob_r\})$，$L\leftarrow L\cup\{job_e\}$，计算 $e\leftarrow e+1$。

(7) 设集合 W，计算 $nl\leftarrow$length(NV)，nl 为集合 NV 中元素个数，并设 job_e 标识符 *DS* 为 D。对 y=1,2,…, nl，取 $nv_y\in NV$，$AT_y\in B$，满足 $nv_y\in AT_y$，执行 $W\leftarrow W\cup\{AT_y\}$，$U\leftarrow U-\{AT_y\}$。然后，执行 $subjob\leftarrow(Neg, W)$，$job_e\leftarrow(DS, \{subjob\})$，$L\leftarrow L\cup\{job_e\}$，计算 $e\leftarrow e+1$。

(8) 设 job_e 标识符 DS 为 F，对 $AT_c\in H$，执行 $subjob\leftarrow(Neg, H)$，$job_e\leftarrow(DS, \{subjob\})$，$L\leftarrow L\cup\{job_e\}$，输出 L。

算法 6.1 以框架中查询与规则分析器计算生成的 SWRL 规则解析模型 *R* 作为输入，在框架中的查询和推理计划生成器中计算生成一组 SWRL 推理计划模型 *L* 作为输出。为更好地阐述算法 6.1 的计算过程，以本章实验部分中基于 LUBM 本体数据集而定义的一条测试 SWRL 规则作为示例，其规则描述为：

$$Professor(?x) \wedge Department(?y) \wedge worksFor(?x, ?y) \wedge TeachingAssistant(?z) \wedge advisor(?z, ?x) \rightarrow worksFor(?z, ?y) \tag{6.6}$$

首先，算法 6.1 输入 SWRL 规则的解析模型 $R=(B, H)$，其中 $B = \{ AT_1, AT_2, AT_3, AT_4, AT_5\}$，$AT_1 = (?x, rdf{:}type, Professor)$, $AT_2 = (?y, rdf{:}type, Department)$, $AT_3 = (?x, worksFor, ?y)$, $AT_4 = (?z, rdf{:}type, TeachingAssistant)$, $AT_5 = (?z, advisor, ?x)$，$H = \{AT_c\}=\{(?z, worksFor, ?y)\}$。将变量?*x*, ?*y* 和?*z* 划分为共同变量集合{?*x*, ?*y*, ?z}，非共同变量集合为空。

其次，在算法 6.1 的步骤(3)中，计算所有共同变量的 E-count 值，并以共同变量名及其对应的 E-count 值构造键值对，并存入集合 *CV* 中。在式(6.6)的示例中，*CV*={<?*x*, 2>, <?*y*, 1>, <?*z*, 1>}，*NV*=∅。另外，将 *R* 模型中的 *B* 存储到集合 *U* 中，以供后续步骤计算。

然后，在算法 6.1 的步骤(4)中根据 E-count 值的大小对 *CV* 集合中各个共同变量进行排序，以此确定共同变量的计算顺序。在式(6.6)的示例中，*CV* 集合元素个数 *cl*=3，但由于 E-count(?*y*)和 E-count(?z)均等于 1，并且?*y* 和?*z* 在 *B* 中出现次数 $T_{?y}$=2 和 $T_{?z}$=2，因此计算?*y* 和?*z* 在各个 *AT* 模型中主语位置 *A*(*s*)中出现次数 $S_{?y}$=1 和 $S_{?z}$=2，即?*z* 排序优先于?*y*。重新排序后的 *CV*={<?*z*, 1>, <?*y*, 1>, <?*x*, 2>}，所以优先对变量?z 进行计算。

接下来，在算法 6.1 的步骤(5)中，根据集合 *CV* 中的共同变量计算生成 SWRL 推理计划模型。步骤(5)首先设置集合 *Temp* 用于存储因未完全消除而还需参与计算的共同变量，用参数 *e* 标识当前的 MapReduce 任务序号，初始化 *e* 为 1，并将 Job_1 的 *DS* 属性设置为 *D*，即第一个 MapReduce 任务以 HBase 中 OWL 本体数据作为数据源。然后，根据集合 *CV* 中共同变量的排列顺序，迭代地提取共同变量进行计算。在规则 6.3 的示例中，首先计算变量?*z*，从集合 *U* 中获取?*z* 所存在的 *AT* 模型：AT_4 和 AT_5，生成模型 $subjob_1 \leftarrow(?z, \{ AT_4, AT_5\})$，并将 AT_4 和 AT_5 从集合 *U* 中排除。但由于 AT_4 和 AT_5 中存在了除?*z* 意外的共同变量?*x* 且此时?$x \notin Temp$，而 $subjob_1$ 中未涉及?*x* 的推理，因此将?*x* 插入集合 *Temp*，即执行 $Temp \leftarrow Temp \cup \{?x\}$。在第二次迭代中对?*y* 进行计算，从当前的集合 *U* 中获取?*y* 所存在的 *AT* 模型：AT_2 和 AT_3，生成模型 $subjob_2 \leftarrow(?y, \{ AT_2, AT_3\})$，再将 AT_2 和 AT_3 从集合 *U* 中排除，由于 AT_2 和 AT_3 中除?*y* 以外的共同变量?*x* 已存在于 *Temp* 中，故停止本次迭代。同理，在第三次迭代中对同变量对?*x* 进行计算，生成模型 $subjob_3 \leftarrow(?x, \{AT_1\})$，再将 AT_1 从集

合 U 中排除。最终，步骤(5)将 Job_1 所包含的 $subjob_1$, $subjob_2$ 和 $subjob_3$ 存入 L，并设置 e 为 2。

算法 6.1 的步骤(6)对存储于集合 *Temp* 中的还未完全消除的共同变量计算生成推理计划模型。在式(6.6)的示例中，由于 *Temp* 中仅包含?x，因此只需进行一次迭代：取 B 中?x 所对应的 AT_1, AT_3, AT_5，并生成 Job_2 的 $subjob$=(?x, {AT_1, AT_3, AT_5})，用于表示在第二个 MapReduce 任务中将对{AT_1, AT_3, AT_5}基于?x 进行$\bowtie_{?x}$计算。然后，设置 Job_2 的 *DS* 属性为 F，即 Job_2 根据 HDFS 文件系统中的 Job_1 计算结果作为数据输入。最后，将 Job_2 添加入推理计划集合 L 中。

算法 6.1 的步骤(7)根据集合 *NV* 中的非共同变量计算生成 SWRL 推理计划模型。对 *NV* 集合中的所有非共同变量，依次获取其在 B 中对应的 *AT* 模型，并存入集合 W 中，然后生成 $subjob$←(Neg, W)，并根据该 $subjob$ 和设置为 D 的 *DS* 属性设置 *Job* 模型，最后添加入推理计划集合 L 中。在式(6.6)的示例中，由于 *NV* 集合为空，因此直接进入下一计算步骤。

算法 6.1 的步骤(8)对模型 R 中的 H 部分进行计算，即计算生成最后的推理结果。首先设置 *Job* 标识符 *DS* 为 F，再将 H 作为待处理的 *AT* 模型集合存入 *Job* 模型，并返回最终生成的 SWRL 推理计划模型 L。

因此，对于示例规则 6.3，基于 MapReduce 的 SWRL 规则推理任务生成算法完成推理任务总共仅需要进行 3 次 MapReduce 任务。

6.4 SWRL 规则的 MapReduce 分布式推理算法

以推理计划模型 L 和 HBase 中 OWL 本体数据为输入，框架中分布式规则推理模块实现基于 MapReduce 的 SWRL 规则分布式推理算法。该算法由五个子函数构成：Main_Control、DB_Map、File_Map、Reduce 和 Result_Map。其中，Main_Control 函数在 Hadoop 平台的 master 结点中负责推理任务的生成与调度，运行于各个 Mapper 计算结点的 DB_Map 和 File_Map 函数分别以 HBase 数据库和 HDFS 文件系统作为数据源，并生成相应的键值对数据。Reduce 函数在各个 Reducer 计算结点中对具有相同键的推理中间数据模型 *IKV* 进行计算。Result_Map 函数计算生成最终的 SWRL 规则推理结果。

算法 6.2 描述了 SWRL 规则推理任务生成与调度函数 Main_Control。该算法以 SWRL 推理计划模型 L 为输入，首先步骤(1)对 L 中每个模型 *AT* 的 $A(p)$值进行判断，若有某一 *AT* 中 $A(p)$值为 *rdf*: *type*，且 $A(o)=C$，那么从 HBase 的 OWL 术语集中查询出 C 的所有子概念，并以 C 为根构建其子类的层次结构树 T，并对 T 按深度优先策略排列为{C, C_1, C_2,…, C_h}，即若有某一个体 a 是 C_h 的实例，那么 a 也是 C 的一个实例。例如示例规则 6.2 中 AT_1 = (?x, *rdf*:*type*, *Professor*)，其 $A(o)$为

Professor，那么根据 LUBM 术语本体的定义，*T* 为由 *Professor* 及其子类按深度优先排列的树节点集合：*T* = {*Professor, AssistantProfessor, AssociateProfessor, Chair, Dean, FullProfessor, VisitingProfessor*}。然后，将 AT_1 替换为由 *Professor* 及其所有子类所对应的 *AT* 模型集合。类似地，若有某一 *AT* 中 *A*(*p*)为 OWL 属性 *P*，则获取 *P* 的所有子属性，并构造树 *T'*，并将 AT_1 替换为由 *P* 及其所有子属性所对应的 *AT* 模型集合。

算法 6.2：Master 结点 SWRL 推理任务生成与调度函数 Main_Control
输入：SWRL 推理计划模型 *L*
输出：SWRL 推理结果

(1) 对 *L* 中所有规则原子解析模型 *AT*，若 *A*(*p*)为 *rdf*: *type* 时，对 *A*(*o*)=*C*，从 HBase 数据库 T_SP_O 表中查询出以 *C* 为根的子类层次结构树 *T*，并对 *T* 按深度优先策略排列为{*C*, C_1, C_2,…, C_h}，*h*≥0，设 AT_1=(*A*(*s*), *A*(*p*), C_1),…, AT_h=(*A*(*s*), *A*(*s*), C_h)，执行 *AT*←{*AT*, AT_1,…, AT_h}。若 *A*(*p*)为 OWL 属性 *P* 时，从 HBase 数据库 T_SP_O 表中查询出以 *P* 为根的子属性层次结构树 *T'*，并对 *T'* 按深度优先策略排列为{*P*, P_1,…, P_l}，*l* ≥0，设 AT_1=(*A*(*s*), P_1, *A*(*o*)),…, AT_l=(*A*(*s*), P_l, *A*(*o*))，执行 *AT*←{*AT*, AT_1,…, AT_l }。

(2) 计算 *ln*←length(*L*)，*ln* 为 *L* 中模型 *job* 个数，设置 HDFS 文件夹 *JoinOut*、*TempOut*、和 *DcarIn*，初始化分布式缓存对象 *DC*。

(3) 对 *i*=1,2,..., (*ln*-1)，job_i∈*L*，执行 *DC*←job_i，当 job_i 标识符 *DS* 为 *F* 时，设置 File_Map 数据源为 *TempOut*，执行 File_Map 和 Reduce 函数；当 job_i 标识符 *DS* 为 *D* 时，计算 *sn* ←length(job_i)，*sn* 为 job_i 中 *subjob* 个数，对 *j*=1, 2,…, *sn*，计算 *an*←length($subjob_j$)，*an* 为 $subjob_j$ 中规则原子解析模型个数，对 *x*=1, 2,…, *an*，AT_x∈$subjob_j$，生成 AT_x 对应的数据表 $Table_x$ 及查询命令 $Comd_x$，执行 *Q*←*Q*∪{($Table_x$, $Comd_x$)}。设 DB_Map 数据源为 *Q*，当 $subjob_j$ 的 *Tag* 值为任一共同变量时，执行 DB_Map 和 Reduce 函数，当 $subjob_j$ 的 *Tag* 值为 *Neg* 时，仅执行 DB_Map 函数。

(4))对 *i*=*ln*，设 Result_Map 函数数据源为 HDFS 文件夹 *JoinOut* 和 *DcarIn*，并执行 Result_Map 函数。

算法 6.2 的步骤(2)首先获取 *L* 中的任务数 *ln*，然后初始化相应的 HDFS 临时文件和 Hadoop 分布式缓存对象 *DC*。

算法 6.2 的步骤(3)根据 *L* 中 *Job* 模型生成 MapReduce 推理任务，设置任务中调用的子算法并指定其输入数据源。例如，当某一 *job* 的标识符 *DS* 为 *F* 时，表明

该次 MapReduce 任务以之前计算任务在 HDFS 文件 *TempOut* 中的计算结果为输入，执行 File_Map 和 Reduce 函数。当某一 *job* 的标识符 *DS* 为 *D* 时，表明该次 MapReduce 任务需从 HBase 中检索出对应的断言集，因此对每个 *AT* 生成对应的数据检索命令并存储于集合 *Q* 中，再分别在该任务的 Map 和 Reduce 阶段执行 DB_Map 和 Reduce 函数。当 *ln*-1 个 MapReduce 推理任务计算完之后，算法 6.2 的步骤(4)设置 Result_Map 函数的数据源，并在最后一个 job 中计算得到最后的 SWRL 规则推理结果。

算法 6.3 描述了 DB_Map 函数中的数据查询及中间数据生成方法。

算法 6.3：DB_Map 函数的数据检索及中间数据生成

输入：OWL 断言数据检索命令集 *Q*

输出：一组 SWRL 推理中间键值对数据模型 *IKV*

---

(1) 根据集合 *Q*，检索得到规则原子解析模型在本体库中的解释数据集 $DL=\{data_1,\ldots, data_n\}$，其中 *data* 为 OWL 个体或常量 *a* 或二元组(*a*, *b*)。

(2) 当 $n=0$ 时，终止计算；当 $n\geq 1$ 时，读取 *DC* 中 *job* 模型，对 $i=1, 2,\ldots, n$，$data_i\in DL$，根据 $data_i$ 对应的 *Table* 和 *Comd*，取 $data_i$ 在 *job* 中对应模型 AT_i 和 $subjob_i$，当 $data_i = a$ 时，执行 $B_i\ (AT_i) \leftarrow (v, a)$，当 $data_i = (a, b)$ 时，执行 $B_i\ (AT_i) \leftarrow((v_1, a), (v_2, b))$，当 $subjob_i$ 标识符 *Tag* 为共同变量名 *v* 时，执行 $IKV_i \leftarrow <(v, I_i[v]), (A_i, B_i\ (AT_i))>$，当 $subjob_i$ 标识符 *Tag* 为 *Neg* 时，执行 $IKV_i \leftarrow <(v, I_i\ (A_i)),(A_i, B_i\ (AT_i))$。

(3) 对 $i=1, 2,\ldots, n$，当 $subjob_i$ 标识符 *Tag* 为共同变量名 *v* 时，将 IKV_i 传递给 Reduce 函数；当 $subjob_i$ 标识符 *Tag* 为 *Neg* 时，将 IKV_i 输出到 *DcarIn* 文件夹中 AT_i.out 文件。

算法 6.3 运行于各个 Mapper 计算结点上，并以 Main_Control 函数中的 OWL 断言数据检索命令集 *Q* 为输入，首先从 HBase 数据库中检索出与各个 *AT* 模型匹配的 OWL 数据集 $DL=\{data_1,\ldots, data_n\}$，若 $n=0$，说明各个 *AT* 模型不存在对应数据，那么终止计算。反之，当 $n\geq 1$ 时，根据每个 *data* 对应的数据表和检索命令确定其在 *job* 中对应模型 *AT* 和 *subjob*，然后构建 *data* 对应的 SWRL 绑定模型 *B*(*AT*)，并根据 *subjob* 的标识符 *Tag* 构建 SWRL 推理中间键值对数据模型 *IKV*。接下来，在步骤(3)中，再根据 *subjob* 标识符 *Tag* 设定 *IKV* 的输出地址。当 *Tag* 为共同变量名 *v* 时说明需对绑定值进行连接计算，因此将 *IKV* 传递给 Reduce 函数；当 *Tag* 为 *Neg* 时说明只需在最后的 Result_Map 函数中计算笛卡尔集，因此将 *IKV* 输出到 *DcarIn* 文件夹中，并以 AT_i.out 命名该文件。

算法 6.4 描述了 File_Map 函数的中间数据生成与分发算法。该算法以 HDFS 文件 *TempOut* 中的 *IKV* 模型集合 *KL* 及分布式缓存对象 *DC* 中 *job* 模型作为输入，若有 *IKV* 键所存储的变量 *v* 存在于某一 *subjob* 模型中，即 *v* 需在此次 MapReduce 任务中进行计算，则将该 *IKV* 模型传递给 Reduce 函数。

算法 6.4：File_Map 函数的 IKV 中间数据分发

输入：*TempOut* 中 *IKV* 模型的集合 $KL=\{IKV_1,\ldots, IKV_n\}$

输出：一组需计算的中间断言键值对 *IKV*

(1) 输入 *TempOut* 中 *IKV* 模型的集合 $KL=\{IKV_1,\ldots, IKV_n\}$，若 $n\geq 1$，读取 *DC* 中 *job* 模型，计算 $sl\leftarrow$length(*job*)，*sl* 为 *job* 模型中 *subjob* 个数。

(2) 对 $i=1,\ldots, n$，若 IKV_i 键中存在变量 $v\in subjob_k$，$1\leq k\leq sl$，将 IKV_i 传递给 Reduce 函数。

算法 6.5 描述了运行于各个 Reducer 计算结点中的 Reduce 函数的绑定模型连接计算算法。

算法 6.5：Reduce 函数的绑定模型连接计算方法

输入：具有相同键的推理中间数据模型 $IKV_1,\ldots, IKV_u$

输出：中间断言键值对集合

(1) 输入具有相同键的推理中间数据模型 $IKV_1,\ldots, IKV_u$，若 $u\geq 1$，读取 *DC* 中模型 *job*，根据 $IKV_1,\ldots, IKV_u$ 键中变量名 *v*，取 *job* 中对应 *subjob* 模型，计算 $cnt\leftarrow$length(*subjob*)，*cnt* 为 *subjob* 中规则原子解析模型个数，设集合 $E_1, E_2,\ldots, E_{cnt}$，分别用于存放 $AT_1, AT_2,\ldots, AT_{cnt}$ 的绑定模型。

(2) 对 $i=1,\ldots, u$，取 IKV_i 中绑定模型 $B(\{A_i\})$的变量 *v* 或(v_1, v_2)，当 *v* 或$(v_1, v_2)\in AT_j$ 时，执行 $E_j\leftarrow E_j\cup\{B(\{AT_i\})\}$，$1\leq j\leq cnt$。

(3) 计算 $E_1\bowtie E_2\bowtie\ldots\bowtie E_{cnt}$，得同时满足 *subjob* 中各规则原子解析模型的绑定模型 $B_1(E_1,\ldots, E_{cnt}),\ldots, B_r(E_1,\ldots, E_{cnt})$。当 $r=0$ 时，终止运行 Reduce 函数；当 $r\geq 1$ 时，对 $k=1,\ldots, r$，若 $B_k(E_1,\ldots, E_{count})$中存在仍未进行连接操作的变量 v_k，构造推理中间数据模型 $IKV_k=<(v_k, I(v_k)), (\{AT_1,\ldots, AT_{cnt}\}, B_k(E_1,\ldots, E_{cnt}))>$，当 *job* 标识符 *DS* 为 *D* 时，输出 IKV_k 到 *TempOut*；当 *job* 标识符 *DS* 为 *F* 且 $B_k(E_1,\ldots, E_{cnt})$中不存在仍未进行连接操作的变量时，将 $B_k(E_1,\ldots, E_{cnt})$输出到 *JoinOut*。

算法 6.5 以具有相同键的推理中间数据模型 *IKV* 和分布式缓存对象 *DC* 中 job 模型为输入，首先获取 job 中与各个 *IKV* 键中变量名 *v* 相对应的 *subjob* 模型，再

计算 *subjob* 模型的个数 *cnt*，并初始化 *cnt* 个集合 $E_1, E_2, \ldots, E_{cnt}$，分别用于存放 $AT_1, AT_2, \ldots, AT_{cnt}$ 的绑定模型，如步骤(2)所示。算法 6.5 的步骤(3)对具有共同变量的绑定模型集合基于该共态变量进行连接计算。当计算后存在仍未进行连接操作的共同变量时，再次构建推理中间数据模型 *IKV*，并将其输出到 TempOut 文件中以待下一次 MapReduce 任务的 File_Map 函数进行读取。反之，若计算后所有共同变量以完成连接计算时，将绑定模型输出到 HDFS 文件 JoinOut 中，以待最后一个 MapReduce 任务调用 Result_Map 函数计算生成最终的 SWRL 推理结果。

算法 6.6 描述了 Result_Map 函数的 SWRL 推理结果生成算法。

算法 6.6：Result_Map 函数的 SWRL 推理结果生成
输入：*JoinOut* 和 *DcarIn* 文件夹中绑定模型
输出：SWRL 推理结果绑定模型 *B*(*A'*)

(1) 设集合 *Result*，从 *JoinOut* 中读取绑定模型集合 $JI=\{B_1,\ldots, B_n\}$。从 *DcarIn* 各个文件中读取绑定模型集合$\{DI_1, \ldots, DI_m\}$，若 $n\geq 1$ 且 $m\geq 1$，计算 $Result \leftarrow JI \times DI_1 \times \ldots \times DI_m$；若 $m=0$ 且 $n\geq 1$，执行 $Result \leftarrow JI$；若 $n=0$ 且 $m\geq 1$，计算 $Result \leftarrow DI_1 \times \ldots \times DI_m$。

(2) 读取 *DC* 中模型 *job*，取 *job* 中后件规则原子解析模型 *A'*。根据 *A'*中变量，从 *Result* 绑定模型中取得对应的变量值，得到推理结果 *B*(*A'*)，并输出到 HDFS 文件 *ResultOut*。

算法 6.6 以 HDFS 文件 *JoinOut* 和 *DcarIn* 文件夹中绑定模型的集合作为输入，其中 *DcarIn* 中存储的是 SWRL 规则中非共同变量所对应的绑定模型。若共同变量绑定模型集合 *JI* 和非共同变量绑定模型集合中元素个数均大于等于 1 时，对其计算笛卡尔积；若不存在非共同变量绑定模型，则 *JI* 既为绑定结果；若不存在共同变量绑定模型，而存在非共同变量绑定模型，则结果集 *Result* 为非共同变量绑定模型之间的笛卡尔集。最后，算法 6.6 的步骤(2)根据 SWRL 规则后件的变量名，从结果集 *Result* 中取得对应的绑定值，并将最终结果输出到 HDFS 文件 *ResultOut* 中。

6.5 实验结果与分析

6.5.1 实验环境搭建

由于目前语义 Web 研究领域没有专用于 SWRL 规则引擎性能测试的 SWRL 规则集和 OWL 本体数据集，采用被学术界广泛应用于本体推理和查询性能测试的 LUBM 本体数据集，并根据 LUBM 的术语本体描述，自定义了测试 SWRL 规则，

如表 6.2 所示。由于当前学术界尚未提出分布式的 SWRL 规则推理方法，为验证本章方法的推理性能，进行了与 Jess 和 Pellet 推理引擎的对比实验，以及通过递增计算节点来验证本章方法可扩展性的实验。在本小节后续内容中，将本章方法表示为 CloudSWRL。

表 6.2 CloudSWRL 实验中基于 LUBM 数据集的 SWRL 测试规则

Table 6.2 LUBM Benchmark-based testing SWRL Rules for CloudSWRL

规则 ID	解释条件
Rule 1	*Professor*(?*x*)∧*Department*(?*y*)∧*worksFor*(?*x*,?*y*)∧*TeachingAssistant*(?z)∧*advisor*(?*z*, ?*x*) → *worksFor*(?*z*, ?*y*)
Rule 2	*ResearchAssistant*(?*x*)∧*GraduateCourse*(?*y*)∧*takesCourse*(?*x*, ?*y*)→*GraduateStudent*(?*x*)

在云计算硬件环境的搭建方面，采用了与本文第四章相同的实验环境，即将 9 台 PC 机通过 Ethernet 连接构成为一个实验云计算环境，以此作为框架的 Hadoop 平台硬件基础。每个 PC 计算结点具有如下的硬件配置：Pentium IV 3.00 GHz CPU，1.5 GB 内存和 80 GB 硬盘空间。

在对比实验中，使用 LUBM 工具分别了生成了如表 6.3 所示的本体数据集。将 Jess 71p2 和 Pellet 2.3.0 推理引擎部署于配置为双核 64 位 E7200 CPU， DDR2 6GB 内存，4TB 硬盘存储空间，安装 64 位 Windows 操作系统的计算机之上，并为 Java 虚拟机分配 5GB 内存空间。

表 6.3 CloudSWRL 对比实验中基于 LUBM 的本体测试数据集

Table 6.3 LUBM benchmark-based testing dataset for CloudSWRL

数据集 ID	OWL 文件数	数据集大小
Dataset 1	90	50MB
Dataset 2	140	75MB
Dataset 3	190	100MB
Dataset 4	240	125MB
Dataset 5	290	150MB

6.5.2 实验效果及分析

首先，在对比实验中，对测试规则 Rule 1 和 Rule 2，各测试推理工具均生成相同推理结果。统计由本体加载用时、规则加载用时、推理用时和输出用时组成

的计算总用时。图 6.2 和图 6.3 分别描述了实验中各个推理工具在对 Rule 1 和 Rule 2 进行推理时，完成各个输入数据集的计算用时统计，其中 X 轴为各个测试推理引擎名，Y 轴为计算用时，单位为秒。

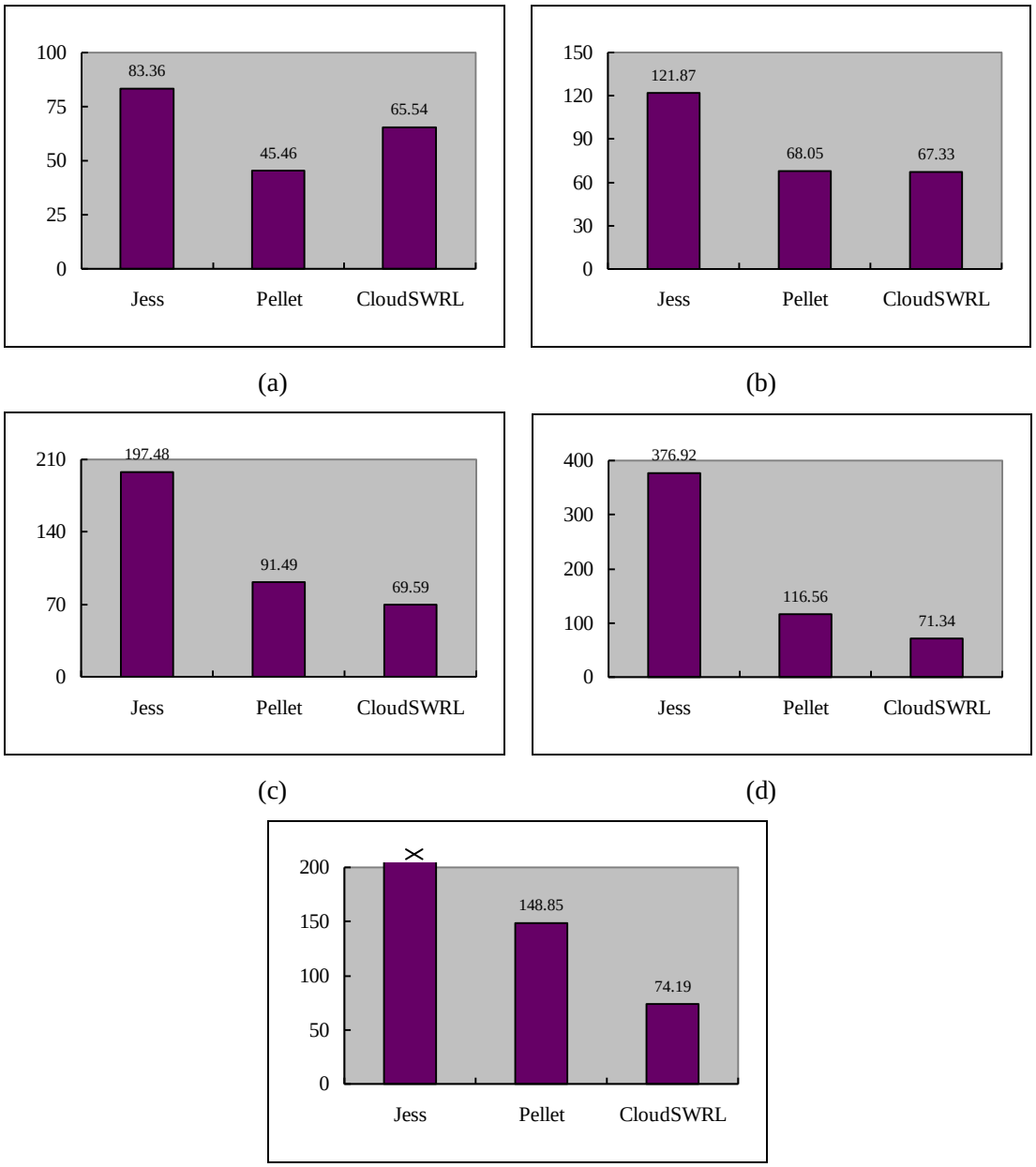


图 6.2 CloudSWRL, Pellet 和 Jess 的 Rule 1 对比实验结果

(a) Dataset 1 计算用时；(b) Dataset 2 计算用时；

(c) Dataset 3 计算用时；(d) Dataset 4 计算用时；(e) Dataset 5 计算用时

Figure 6.2 Rule 1 comparision experimental results among CloudSWRL, Pellet and Jess.

(a) runtime for Dataset 1; (b) runtime for Dataset 2;

(c) runtime for Dataset 3; (d) runtime for Dataset 4; (e) runtime for Dataset 5.

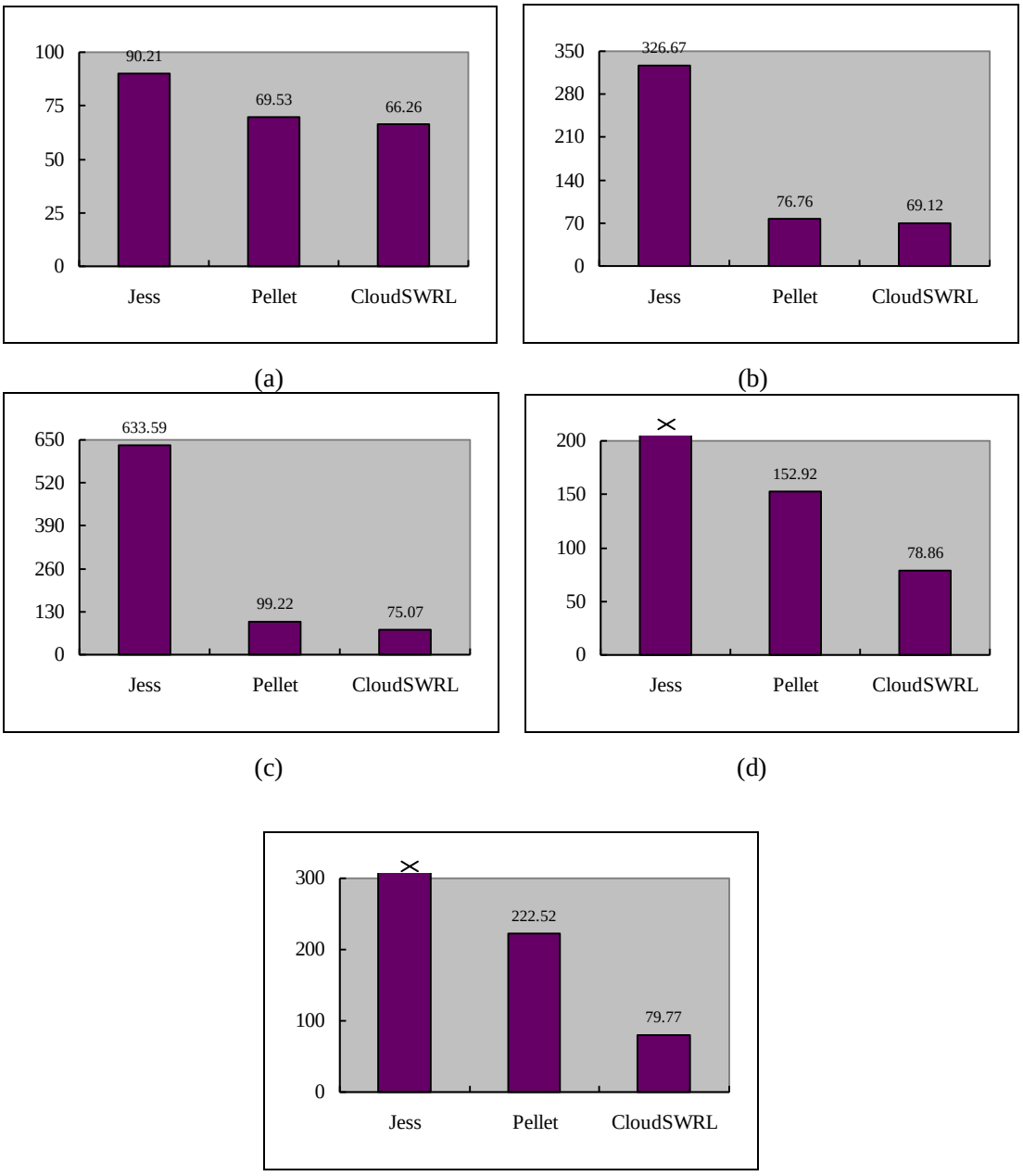


图 6.3 CloudSWRL, Pellet 和 Jess 的 Rule 2 对比实验结果

(a) Dataset 1 计算用时；(b) Dataset 2 计算用时；

(c) Dataset 3 计算用时；(d) Dataset 4 计算用时；(e) Dataset 5 计算用时

Figure 6.3 Rule 2 comparision experimental results among CloudSWRL, Pellet and Jess.

(a) runtime for Dataset 1; (b) runtime for Dataset 2;

(c) runtime for Dataset 3; (d) runtime for Dataset 4; (e) runtime for Dataset 5.

图 6.2 和图 6.3 的对比实验结果表明，在单机环境和分布式环境的硬件配置总体相当的前提下，当输入本体数据量较小时，Pellet 推理引擎具有较好的计算性能，而 Jess 规则引擎由于需要进行本体和规则的映射，在占用大量内存空间的同时推

理性能与 Jess 和本章方法存在差距。但随着输入本体数据量不断增加，本章方法较 Jess 和 Pellet 推理引擎在处理大规模本体数据时，计算性能上存在明显优势。例如，在对数据量仅有 50MB 的 Dataset 1 进行 Rule 1 规则推理时，Pellet 用时为 45.46 秒，优于本章方法的 65.54 秒，但当输入数据增加至 Dataset 2 时，本章方法的计算用时已少于 Pellet。而整个对比实验中，本章方法均优于 Jess 工具。

为验证本章方法在处理大规模本体数据的 SWRL 规则推理时的可扩展性，还针对 Rule 1 和 Rule 2 进行了通过增加参与推理的计算结点数的可扩展性实验。以 1GB 的 LUBM 本体数据集及 Rule 1 和 Rule 2 测试规则作为输入，实验硬件环境对比实验相同。计算结点数分别设置为 2、4、6 和 8 台，其实验结果如图 6.4 所示，其中横轴表示计算结点数，纵轴表示推理总用时，单位为秒。

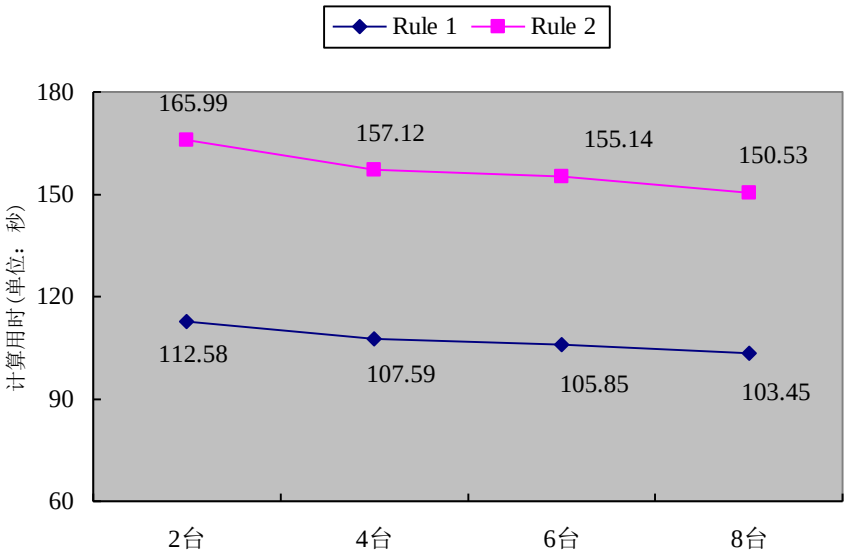


图 6.4 CloudSWRL 递增计算结点数可扩展性实验结果

Figure 6.4 CloudSWRL experimental results for increasing the number of computing nodes

如图 6.4 所示的扩展性实验结果表明，对相同输入数据，当 MapReduce 环境中计算节点数逐步递增时，本章方法具备更强的计算能力，推理用时逐步降低。即当需要处理海量本体输入数据时，可通过扩展计算节点数来获得更佳的推理性能。

最后，生成了 LUBM(100)、LUBM(200)、LUBM(300)和 LUBM(400)的本体数据集来验证本章方法在对更大规模本体数据 SWRL 规则推理时的计算性能和可扩展性。图 6.5 和图 6.6 分别描述了本章方法对 Rule 1 和 Rule 2 的各个数据集推理用时，实验中参与计算的 Mapper 和 Reducer 计算结点数为 8 台。

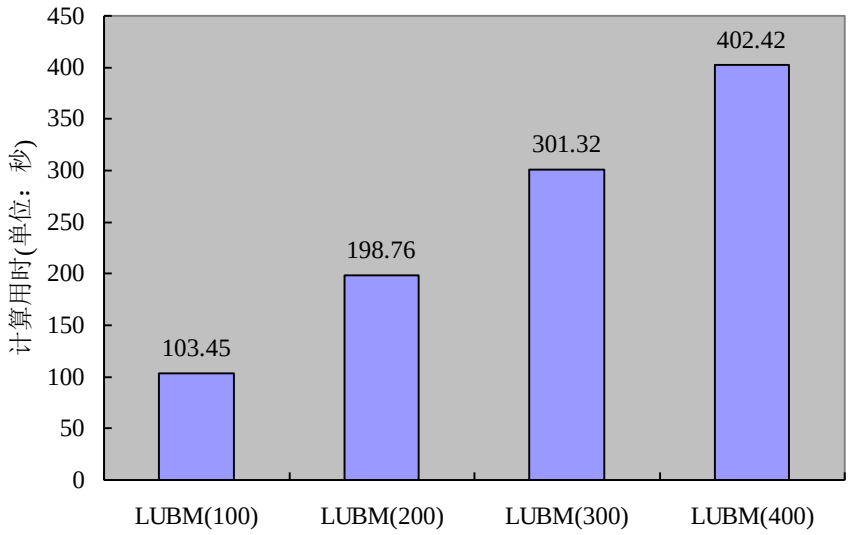


图 6.5 CloudSWRL 递增输入本体数据量的 Rule 1 可扩展性实验结果

Figure 6.5 CloudSWRL experimental results for increasing the number of OWL ontologies based on Rule 1.

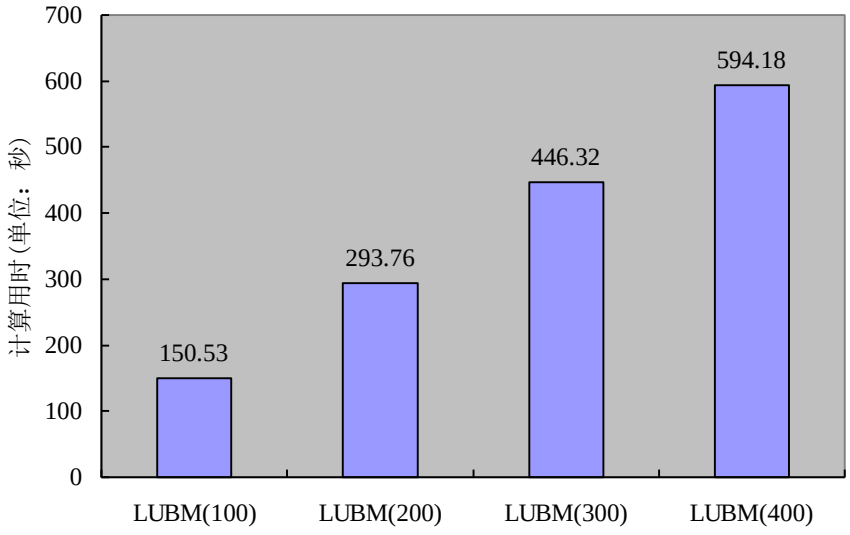


图 6.6 CloudSWRL 递增输入本体数据量的 Rule 2 可扩展性实验结果

Figure 6.6 CloudSWRL experimental results for increasing the number of OWL ontologies based on Rule 2.

实验结果表明，随着输入数据量的增加，本章方法推理用时也随之增长，但推理用时的增长幅度小于数据量的增长。例如，对 Rule 1，当输入 LUBM(100)时本章方法用时为 103.45 秒，而当输入 LUBM(200)、LUBM(300)和 LUBM(400)时，数据量分别增长 2 倍、3 倍和 4 倍，而推理用时仅增加约 1.92、2.91 和 3.89 倍。

综合以上实验结果，可以得出结论，Pellet 推理引擎在对输入数据量较小的 OWL 本体数据进行 SWRL 规则推理时比本章方法具有稍好的计算性能，而 Jess 推理引擎的计算性能不如本章方法。在对大规模 OWL 本体数据的 SWRL 规则进行推理时，传统单机环境下的 Pellet 和 Jess 均存在内存限制，无法完成推理任务，而本章方法均能在合理时间内完成计算，因此本章方法在可扩展性方面优于上述传统规则推理引擎。

6.6 本章小结

本章面向语义 Web 中大规模 OWL 本体的 SWRL 规则推理问题，提出了基于 MapReduce 的 SWRL 规则分布式推理方法。

首先，以 SWRL 规则形式化语义为基础，定义了 SWRL 在 MapReduce 模型下的相关解析模型，提出了基于 MapReduce 的推理计划生成算法和在 DL-safe 限制下 SWRL 规则分布式推理算法。对比实验和可扩展性实验结果表明，本章方法较传统的 Jess 和 Pellet 推理引擎在处理大规模语义 Web 本体的 SWRL 规则推理时，具备更好的计算性能和可扩展性。

7 总结和展望

7.1 研究工作总结

语义 Web 是 Tim Berners-Lee 提出的下一代互联网远景，是实现 Web 信息计算可理解，以及计算机之间语义信息交换的关键。在 W3C 的语义 Web 体系结构中，基于 SPARQL 的 RDF 数据查询、基于描述逻辑的 OWL 本体推理和基于 SWRL 规则 OWL 本体规则推理是语义 Web 数据管理研究领域的核心。

近年来，随着语义 Web 技术的不断发展和推广应用，已出现了海量语义 Web 本体，且该数据存在规模性、高速增长性、多样性等大数据特性，而传统单机环境下的本体数据查询与推理工具不可避免地存在计算性能和可扩展性不足的问题。另一方面，云计算 Hadoop 技术已成为学术界在进行大数据计算研究领域的事实标准。目前，已有研究人员将云计算 Hadoop 技术与语义 Web 技术相结合，研究面向大规模本体数据的查询和推理并行化方法，但研究仍处于起步阶段。

本文以改进本领域研究工作存在的不足为总体目标，研究并提出了新的针对于大规模语义 Web 本体数据分布式查询和推理方法。本文主要的研究工作和创新性成果包括以下五个方面：

① 提出了一种大规模语义 Web 数据查询与推理云计算体系结构

本文以 W3C 提出并推荐的语义 Web 体系结构为基础，结合云计算研究领域中的 MapReduce 分布式计算模型和 HBase 分布式数据技术特性，设计了由物理层、存储层、数据层、逻辑层、接口层、网络层和应用层组成的语义 Web 数据查询与推理云计算框架体系结构，并设计了逻辑层中核心的功能模块：数据预处理器、数据适配器、查询与推理分析器、查询与推理计划生成器、MapReduce SPARQL 查询引擎、MapReduce SWRL 规则推理引擎和 MapReduce Tableau 推理引擎。为实现高性能、易扩展的语义 Web 数据管理云服务提供体系结构支持，为进行进一步的关键技术理论研究奠定了基础。

② 提出了一种新的基于 HBase 的语义 Web 本体数据分布式存储策略

本文基于语义 Web 本体 RDF 三元组数据特性和基于描述逻辑的 OWL 本体描述语言形式化语义，结合 HBase 分布式数据库基于列的数据存储模式，研究并提出了由三个 HBase 数据表 T_OS_P、T_PO_S 和 T_SP_O 构成的语义 Web 本体数据分布式存储策略，分析了在进行 MapReduce 查询和推理任务时的数据检索机制，并通过与传统基于关系型数据库的本体数据存储策略进行分析和对比，证明了本文提出的存储策略能够在本体数据存储空间开销和检索性能方面达到良好的平衡。

③ 提出了一种基于 MapReduce 的 SPARQL 复杂组图模式分布式查询方法

本文基于 SPARQL 语法和形式化语义，结合 MapReduce 基于键值对的数据计算特性，研究并提出了基于 MapReduce 的 SPARQL 复杂组图模式查询任务生成算法，及其在 map 和 reduce 函数中的分布式查询算法。然后，通过使用语义 Web 领域广泛采用的 SP2Bench 测试数据集和测试查询语句，证明了本文所提出方法在对大规模 RDF 数据进行查询计算时，在计算性能和可扩展性方面均优于传统的单机环境下运行的查询引擎。

④ 提出了一种基于 MapReduce 的 OWL 本体分布式一致性检测方法

本文基于 OWL Lite 本体所对应的 SHIF 描述逻辑语义及其 Tableau 算法，研究并提出了相关的解析模型定义以及基于 MapReduce 的 OWL 本体数据划分方法和分布式 Tableau 推理算法。然后，通过使用语义 Web 领域广泛采用的 LUBM 测试数据集，证明了本章方法在对大规模 OWL 本体数据进行处理时，在计算性能和可扩展性方面均优于传统的单机环境下运行的推理引擎。

⑤ 提出了一种基于 MapReduce 的 SWRL 规则分布式推理方法

本文基于 SWRL 规则语法和形式化语义，并以 MapReduce 和 HBase 技术特性为基础，首先定义了框架中相关解析模型，提出了基于 MapReduce 的 SWRL 规则推理计划生成算法和在 DL-safe 限制下 SWRL 规则分布式推理算法。对比实验和可扩展性实验结果表明，本文所提出方法较传统的 Jess 和 Pellet 推理引擎在处理大规模语义 Web 本体的 SWRL 规则推理时，具备更好的计算性能和可扩展性。

7.2 进一步研究展望

本文的研究工作在大规模语义 Web 本体数据查询与推理方面取得了阶段性的成果，仍有一些问题值得改进和继续深入研究，才能真正实现大规模语义 Web 本体数据云服务的产业化应用，具体包括以下几个方面：

① 在对本文提出的基于 MapReduce 的 SPARQL 复杂组图模式分布式查询、OWL 本体一致性分布式检测推理和 SWRL 规则分布式推理的实验过程中，由于实验条件限制，在测试云计算环境中仅采用的是 100 兆每秒的交换机构建局域网，且只部署了九台计算机参与 MapReduce 计算，未能对如 32 台、64 台等的大规模计算机集群参与计算时的计算性能和可扩展性进行验证。因此在今后的研究工作中，拟通过购置更多的计算机硬件来搭建更大规模的云计算实验环境，对所提出方法进行进一步的实验验证，并探寻计算结点数与最佳计算性能之间的关系。

② 在本文第 4 章的研究工作中，仅面向基于描述逻辑 SHIF 的 OWL Lite 本体的一致性检测推理，而未涉及更复杂的 OWL DL 或 OWL 2 子语言，因此在今后的研究工作中，准备通过扩展本文提出的方法，进一步研究描述逻辑 SHOIN 甚至描

述逻辑 SROIQ 的 Tableau 推理算法在 MapReduce 环境下的并行化方法。

③ 在本文第 6 章的研究工作中发现，当前语义 Web 研究领域中尚未出现针对于 SWRL 规则推理性能测试的大规模本体测试数据集及标准测试语句，并且已有测试数据集多面向 RDF 数据或 OWL Lite 本体，尚不存在面向 OWL 2 描述语言的本体数据集。因此，下一步的研究计划拟提出一个全面包含 SPARQL 1.1 标准和 SWRL 规则的 OWL 2 本体数据集生成工具。

④ 本文的研究工作仅面向云计算环境下的大规模语义 Web 本体数据查询与推理关键技术，而没有涉及云计算研究中的安全性、服务模式等方面，而上述问题均是云计算实际产业化应用的关键。因此，在今后的研究工作中，拟将云计算安全性研究与本文研究工作相结合，以加快实现本文提出方法的实际应用。

致　谢

时光荏苒，在重庆大学三年多的博士研究生学习生活即将在山城的初夏时节结束。当离别之际终将到来时，回首在重庆大学十年来的成长历程，有许多师长、亲朋和好友给予了我巨大的帮助和支持，往日学习和科研过程中的愉悦和艰辛也历历在目，这一切值得我用一生去感恩和回味。

首先要向我的导师杨丹教授致以最诚挚的谢意和最崇高的敬意。在我攻读博士学位的学习生涯中，杨老师不仅在百忙之中为我指明了学术研究方向，还在平日的科研工作和生活中给予了我悉心的指导和帮助。杨老师严谨的治学态度、渊博的知识、缜密的思维方式及儒雅的学者风范深深地感染着我，不仅使我领悟到学术研究的精髓，还激励着我在今后的科研工作中继续努力奋进。

其次要衷心感谢实验室的张小洪教授、洪明坚副教授、徐玲副教授和杨梦宁老师。他们不仅在我论文选题和研究过程中给予了耐心指导，提出了宝贵的意见，还在平时生活中给予了无私的帮助。老师们严谨的科研思维和对科学研究一丝不苟的治学作风是我永远值得学习的榜样。

特别感谢实验室的胡海波副教授。胡老师在语义 Web 和软件工程研究领域渊博的知识和敏锐的洞察力，帮助我开启了适合自己的科研之路，是我博士阶段学习和研究的引路人。感谢实验室的硕士研究生罗建华和谢娟在论文研究实验过程中给予的帮助和支持。

感谢实验室的博士师兄弟方蔚涛、葛永新、李博、冯欣、杨娟、付春雷、徐传运、吕建斌、吴云松、黄晟等，大家在一起共同营造了团结协作、共同进步的学术氛围，让我在攻读博士学位期间感受到实验室科研团队的支持和力量。

感谢尊敬的重庆大学软件学院傅鹂教授、向宏教授、付云清教授、文俊浩教授和我的硕士导师王成良教授多年以来的关心、指导和帮助。

感谢澳大利亚悉尼科技大学博士生李牧、日本九州大学博士生曾骏。在与你们的学术交流和关于人生的探讨中，给予了我许多有价值的启发和思路。

最后衷心感谢我的妻子熊婉利女士，以及我的父母和岳父母。在我从事研究期间给予我生活上的支持和精神上的鼓励，正是他们在我身后默默的奉献、悉心的关怀，才能使我完成博士论文。

谨以此篇献给我即将出生的孩子。

李　韧

二O一三年四月 于重庆

参考文献

[1] T. Berners-Lee, J. Hendler, and O. Lassila. The Semantic Web [J]. Scientific American, 2001.

[2] 李善平, 尹奇韡, 胡玉杰, 郭鸣, 付相君. 本体论研究综述 [J]. 计算机研究与发展, 2004, 41(7): 1041-1052.

[3] F. Manola and E.Miller. RDF Primer [EB/OL].W3C Recommendation, 2004, http://www.w3. org /TR/rdf-syntax/.

[4] D. Brickley and R.V. Guha. RDF Vocabulary Description Language 1.0: RDF Schema [EB/OL]. W3C Recommendation, 2004, http://www.w3. org/TR/rdf-schema/.

[5] 梅婧, 林作铨. 从 ALC 到 SHOQ(D):描述逻辑及其 Tableau 算法 [J]. 计算机科学, 2005, 32(3):1-11 .

[6] D. McGuinness and F. van Harmelen. OWL web ontology language overview [EB/OL]. W3C Recommendation, 2004, http://www.w3.org/TR/owl-features/.

[7] I. Horrocks, P.F. Patel-Schneider and F. van Harmelen. From SHIQ and RDF to OWL: The Making of a Web Ontology Language [J]. Journal of Web Semantics, 2003, 1(1):7-26.

[8] 高志强, 潘越, 马力, 谢国彤, 刘升平, 张雷. 语义 Web 原理及应用 [M], 北京: 机械工业出版社, 2009.

[9] W3C OWL Working Group. OWL 2 Web Ontology Language Document Overview (Second Edition) [EB/OL]. W3C Recommendation, 2012, http://www.w3.org/TR/owl2-overview/.

[10] 何克清, 何扬帆, 王翀, 梁鹏, 刘进. 本体元建模理论与方法及其应用[M]. 北京: 科学出版社, 2008.

[11] 张维明, 宋峻峰. 面向语义 Web 的领域本体表示、推理与集成研究 [J]. 计算机研究与发展, 2006, 43(1): 101-108.

[12] R. Fikes, P. Hayes and I. Horrocks. OWL-QL: A language for deductive query answering on the semantic web [J]. Journal of Web Semantics, 2004,2(1):19−29.

[13] G. Karvounarakis, S. Alexaki and V. Christophides. RQL: A declarative query language for RDF [C]. Proceeding of the WWW 2002, New York, USA, 2002: 592−603.

[14] A. Seaborne. RDQL: A query language for RDF [EB/OL]. W3C, 2004. http://www.w3.org/ Submission/RDQL/.

[15] E. Prud'hommeaux and A.Seaborne. SPARQL query language for RDF [EB/OL]. W3C, 2008. http://www.w3.org/TR/rdf-sparql- query/.

[16] I. Horrocks, P. F. Patel-Schneider, H. Boley, S. Tabet, B. Grosof and M. Dean. SWRL: a semantic web rule language combining OWL and RuleML [EB/OL]. http://www.w3.org/Sub mission/2004/SUBM-SWRL-20040521/.

[17] I. Horrocks, P. F. Patel-Schneider, S. Bechhofer and D. Tsarkov. OWL rules: a proposal and prototype implementation [J] . Journal of Web Semantics, 2005, 3(1): 23-40.

[18] R. B. Mishra and S. Kumar. Semantic Web Reasoners and Languages [J]. Artificial Intelligence Review, 2011, 35(4): 339–368.

[19] J. Pérez, M. Arenas and C. Gutierrez. Semantics and Complexity of SPARQL [C]. Proceedings of the 5th International Semantic Web Conference, 2006: 30–43.

[20] J.J. Carroll, I. Dickinson, C. Dollin, D.Reynolds, A. Seaborne and K. Wilkinson. Jena: Implementing the Semantic Web Recommendations [C]. Proceedings of the 13th International World Wide Web Conference, 2004: 806–815.

[21] J. Broekstra and A. Kampman. Sesame: A Generic Architecture for Storing and Querying RDF and RDF Schema [C], Proceedings of the 1st International Semantic Web Conference, 2002.

[22] T. Neumann and G. Weikum. The RDF-3X Engine for Scalable Management of RDF Data [J], VLDB Journal, 2010, 19(1): 91–113.

[23] B.C. Grau, I. Horrocks, B. Motik, B. Parsia, P. Patel-Schneider and U. Sattler. OWL 2: The next step for OWL [J]. Journal of Web Semantics, 2008,6(4):309−322.

[24] FaCT++ [EB/OL]. http://owl.man.ac.uk/factplusplus/.

[25] V. Haarslev and R. Möller. RACER system description [C]. In: Proc. of the IJCAR 2001. LNCS 2083, PP.701−705, 2001.

[26] Pellet [EB/OL]. http://pellet.owldl.com/.

[27] Hermit [EB/OL]. http://hermit-reasoner.com/.

[28] I. Horrocks and U. Sattler. A tableau decision procedure for SHOIQ [J]. Journal of Automated Reasoning, 2007, 39(3): 249−276.

[29] A. Riazanov. Implementing an efficient theorem prover [D]. Manchester: University of Manchester, 2003.

[30] 徐贵红, 张健. 语义网的一阶逻辑推理技术支持 [J]. 软件学报, 2008, 19(12): 091-3099.

[31] M. Horridge et al. A practical guide to building owl ontologies using protégé4 and CO-ODE tools editiion1.1 [EB/OL]. University Of Manchester, 2007, http://www.co-ode.org/resources/tutorials/ProtegeOWLTutorial-p4.0.pdf.

[32] SWRLLanguageFAQ [EB/OL]. http:// protege.cim3.net/cgi-bin/wiki.pl?swrllanguagefaq.

[33] B. Motik and R. Studer. KAON2: A Scalable Reasoning Tool for the Semantic Web [C]. Proceedings of the 2nd European Semantic Web Conference, Heraklion, Greece, 2005.

[34] M. Boris, S. Ulrike and S. Rudi. Query answering for OWL-DL with rules [J]. Journal of Web Semantics, 2005, 3(1): 41-60.

[35] C. Bizer, A. Jentzsch and R. Cyganiak. State of the LOD Cloud [EB/OL], http://lod-cloud.net/state/.

[36] 孟小峰, 慈祥. 大数据管理: 概念、技术与挑战 [J]. 计算机研究与发展, 2013, 50(1): 146-169.

[37] 王珊, 王会举, 覃雄派, 周烜. 架构大数据: 挑战、现状与展望 [J]. 计算机学报, 2011, 34(10): 1741-1752.

[38] 杜小勇, 王琰, 吕彬. 语义 Web 数据管理研究进展 [J]. 软件学报, 2009, 20(11): 2950-2964.

[39] 冯登国, 张敏, 张妍, 徐震. 云计算安全研究 [J]. 软件学报, 2011, 22(1):71-83.

[40] K. Jorissen, F.D. Vila and J.J. Rehr. A high performance scientific cloud computing environment for materials simulations [J]. Computer Physics Communications, 2012, 183(9): 1911-1919.

[41] J. Z. Wang, P. Varman and C. S. Xie. Optimizing storage performance in public cloud platforms [J]. Journal of Zhengjiang University-Science C-Computer & Electronics, 2011, 12(12): 951- 964.

[42] M. Armbrust, A. Fox, R. Griffith, et al. A View of Cloud Computing [J]. Communication of the ACM, 2010, 53(4): 50-58.

[43] S. Ghemawat, H. Gobioff and S.T. Leung. The google file system [C]. Proceeding of the 19th ACM symposium on Operating systems principles, New York, USA, 2003: 29-43.

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

[45] F. Chang, J. Dean, S. Ghemawat, et al. BigTable: a distributed storage system for structured data [J]. ACM Transactions on Computer Systems, 2008, 26(2), Article 4.

[46] Apache. Hadoop[EB/OL]. http://hadoop.apache.org/.

[47] 王 鹏, 孟 丹, 詹剑锋, 涂碧波. 数据密集型计算编程模型研究进展 [J]. 计算机研究与发展, 2010, 47(11): 1993-2002.

[48] Semantic Web Challenge [EB/OL]. http://challenge.semanticweb.org/.

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

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

[51] M. Schmidt, T. Hornung, G. Lausen and C. Pinkel. SP2Bech: A SPARQL performance benchmark [C], Proceedings of the 25th IEEE International Conference on Data Engineering, 2009: 222–233.

[52] C. Bizer and A. Schultz. The Berlin SPARQL Benchmark [J]. International Journal on Semantic Web & Information Systems, 2009, 5(2): 1-24.

[53] M. Uschold and M. Gruninger. Ontologies: principles, methods, and applications [J]. Knowledge Engineering Review, 1996, 11(2):93-155.

[54] T. Gruber. A translation approach to portable Ontology specifications [J]. Knowledge Acquisition, 1993, 5(2):199-220.

[55] W. Borst. Construction of engineering ontologies for knowledge sharing and reuse [D]. PhD thesis, University of Twente, the Netherlands. 1997.

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

[57] The Unicode Consortium [EB/OL], http://www.unicode.org/.

[58] Uniform Resource Identifier [EB/OL], http://www.w3.org/Addressing/URL/URI_Overview.html.

[59] W3C. Extensible Markup Language (XML) [EB/OL], http://www.w3.org/XML/.

[60] T. Gruber. Toward principles for the design of ontologies used for knowledge sharing [J]. International Journal of Human-Computer Studies, 1995, 43(5-6):907-928.

[61] D. Gašvić, N. Kaviani and M. Milanović. Ontologies and software engineering. In Staab & Studer ed, Handbook on Ontologies (2nd edition) [M]. Springer, Berlin, 2009:593-615.

[62] G. Klyne and J. J. Carroll. Rescource Description Framework (RDF): Concepts and Abstract Syntax [EB/OL]. W3C Recommendation, 10 February, 2004, http://www.w3.org/TR/rdf-concepts/

[63] S. Elbassuoni and R. Blanco. Keyword Search over RDF Graphs [C]. Proceedings of the 20th ACM Conference on Information and Knowledge Management, CIKM' 11, October 24-28, 2011, Glasgow, Scotland, UK.

[64] W3C. N-Triples, W3C RDF Core WG Internal Working Draft [EB/OL], http://www.w3.org/2001/sw/RDFCore/ntriples/.

[65] T. Berners-Lee and D. Connolly. Notation3 (N3): A readable RDF syntax [EB/OL]. http://www. w3.org/TeamSubmission/n3/.

[66] R. Oldakowski, C. Bizer and D. Westphal. RAP: RDF API for PHP [C]. Proceedings of the International Workshop on Interpreted Languages, 2004.

[67] J. Broekstra and A. Kampman. Inferencing and Truth Maintenance in RDF Schema: exploring a naive practical approach [C]. In Workshop on Practical and Scalable Semantic Systems (PSSS), 2003.

[68] R. Cyganiak. A relational algebra for SPARQL [EB/OL], HP-Labs Technical Report,

HPL-2005-170. http://www.hpl.hp.com/techreports/2005/HPL-2005-170.html.

[69] M. Arenas and J. Pérez. Querying semantic web data with SPARQL [C]. Proceedings of the 30th ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems, 2011: 305-316.

[70] I. Kollia, B. Glimm and I. Horrocks. SPARQL Querying Answering over OWL Ontologies [J]. Lecture Notes in Computer Science, 2011, 6643 LNCS(PART 1): 382-396.

[71] M. Schmidt, M. Meier and G. Lausen. Foundations of SPARQL query optimization [C]. Proceedings of the 13th International Conference on Database Theory, 2010: 4-33.

[72] S. Harris and A. Seaborne. SPARQL 1.1 Query Language, W3C Proposed Recommendation 08 November 2012 [EB/OL]. http://www.w3.org/TR/sparql11-query/.

[73] Apache. ARQ - A SPARQL Processor for Jena [EB/OL]. http://jena.apache.org/documentation /query/index.html.

[74] S. A. Mcllraith, T. C. Son and H. L. Zeng. Semantic Web services [J]. IEEE Intelligent Systems & Their Applications, 2001, 16(2): 46-53.

[75] T. Neumann and G. Weikum. RDF-3X: a RISC-style engine for RDF [C]. Proceedings of the VLDB Endowment, 2008, 1(1): 647-659.

[76] D. Fensel, F. Van Harmelen, I. Horrocks, D. McGuinness and P. Patel-Schneider. OIL: An ontology infrastructure for the semantic web [J]. IEEE Intelligent Systems, 2001, 16(2): 38-45.

[77] DARRPA. The DARRPA agent markup language [EB/OL]. 2000, http://www.daml.org/.

[78] I. Horrocks and P. Patel-Schneider. Reducing OWL entailment to description logic satisfiablity [C]. International Semantic Web Conference 2003, Web Semantics, 2004, 1(4).

[79] 古华茂, 王勋, 凌云, 高济. 完全析取范式群判定 SHOIN(D)-可满足性 [J], 软件学报, 2010, 21(8): 1863-1877.

[80] I. Horrocks. OWL: A descritption logic based ontology language [C]. 11th International Conference on Principles and Practice of Constraint Programming - CP 2005, Sitges, Spain, October 1, 2005 - October 5, 2005, Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 3709 LNCS: 5-8.

[81] B. Konev, M. Ludwig, D. Walther and F. Wolter. The logical difference for the lightweight description logic EL [J]. Journal of Artificial Intelligence Research, 2012, 44: 633-708.

[82] M. Krötzsch. OWL 2 profiles: An introduction to lightweight ontology languages [C]. In 8th International Summer School on Reasoning Web, 2012, Vienna, Austria, September 3, 2012 - September 8, 2012, Lecture Notes in Computer Science (including subseries Lecture Notes in

Artificial Intelligence and Lecture Notes in Bioinformatics), 7487 LNCS: 112-183.

[83] S. T. Cao, L. A. Nguyen and A. Szaas. On the web ontology rule language OWL 2 RL [C]. In 3rd International Conference on Computational Collective Intelligence, ICCCI 2011, September 21, 2011 - September 23, 2011, Gdynia, Poland, Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 2011, 6922 LNAI, PART 1: 254-264.

[84] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. Patel-Schneider. The description logic handbook: theory, implementation, applications [D]. Cambridge University Press, Cambridge, UK, 2003.

[85] F. Baader, I. Horrocks and U.Sattler. Description Logics (book chapter). In van Harmelen F., Lifschitz V., and Porter B., eds, Handbook of Knowledge Representation [M]. Elsevier, Amsterdam, The Netherlands, 2007.

[86] A. Manuel. Frames, semantic networks, and object-oriented programming in APL2 [J]. IBM Journal of Research and Development, 1989, 33(5): 502-510.

[87] G. Meditskos and N. Bassiliades. A rule-based object-oriented OWL reasoner [J]. IEEE Transactions on Knowledge and Data Engineering, 2008, 20(3): 397-410.

[88] S. Klarman, U. Endriss and S. Scholbach. ABox abduction in the description logic ALC [J]. Journal of Automated Reasoning, 2011, 46(1): 43-80.

[89] F. Baader and U. Sattler. An overview of Tableau algorithms for description logics [J]. Studia Logica, 2001, 69(1): 5-40.

[90] 石莲, 孙吉贵. 描述逻辑综述 [J]. 计算机科学, 2006, 33(1): 194-197.

[91] K. Wu and V. Haarslev. A parallel reasoner for the description logic ALC [C]. Proceedings of the 25th International Workshop on Description Logics, DL 2012, June 7, 2012 - June 10, 2012, Rome, Italy, 2012: 378-388.

[92] H. Boley, A. Paschke and O. Shafiq. RuleML 1.0: The overarching specification of web rules [C]. 4th International Web Rule Symposium, RuleML 2010, October 21, 2010 - October 23, 2010, Washington, DC, United states. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 2010, 6403 LNCS:162-178,

[93] R. Reiter. On closed world data bases [R]. Technical Report, University of British Columbia; Vancouver, BC, Canada, 1977.

[94] V. Kolovski, B. Parsia and E. Sirin. Extending the SHOIQ(D) tableaux with DL-safe rules: First results [C]. Proceedings of the 2006 International Workshop on Description Logics, DL 2006, 189: 192-199.

[95] J. Mei and E. Paslaru Bontas. Reasoning paradigms for SWRL-enabled ontologies [C]. In Workshop of Protégé With Rules, Madrid. 2005.

[96] P. F. Patel-Schneider. Safe rules for owl 1.1 [C]. In Fourth International Workshop OWL: Experiences and Directions (OWLED 2008 DC), Washington, DC, 2008.

[97] 李乔, 郑啸. 云计算研究现状综述 [J]. 计算机科学, 2011, 38(4): 32-37.

[98] P. Mell and T.Grance. The NIST definition of cloud computing [J]. NIST special publication, 2011, 800: 145.

[99] G. Boss, P. Malladi, D. Quan, L. Legregni and H. Hall. Cloud computing [EB/OL]. IBM White Paper, 2007. http://download.boulder.ibm.com/ibmdl/pub/software/dw/wes/hipods/Cloud_computing_wp_final_8Oct.pdf.

[100] 陈康, 郑纬民. 云计算: 系统实例与研究现状 [J]. 软件学报, 2009, 20(5): 1337-1348.

[101] B. Hayes. Cloud computing [J]. Communications of the ACM, 2008, 51(7): 9-11.

[102] K. Jorissen, F. D. Vila and J. J. Rehr. A high performance scientific cloud computing environment for materials simulations [J]. Computer Physics Communications, 2012, 183(9): 1911-1919.

[103] I. Palit and C.K. Reddy. Scalable and Parallel Boosting with MapReduce [J]. IEEE Transaction on Knowledge and Data Engineering, 2012, 24(10):1904-1916.

[104] A. Srinivasan, T. A. Faruquie and S. Joshi. Data and task parallelism in ILP using MapReduce [J], Machine Learning, 2012, 86(1): 141-168.

[105] D. Zinn, S. Bowers, S. Kohler and B. Ludascher. Parallelizing XML data-streaming workflows via MapReduce [J]. Journal of Computer and System Sciences, 2010, 76(6): 447-463.

[106] C. Moretti, K. Steinhaeuser, D. Thain, and N. Chawla. Scaling Up Classifiers to Cloud Computers [C]. Proceedings of 8th IEEE International Conference on Data Mining (ICDM '08), 2008: 472-481.

[107] C.T. Chu, S.K. Kim, Y.A. Lin, Y. Yu, G. Bradski, A.Y. Ng and K.Olukotun. Map-Reduce for Machine Learning on Multicore [C]. Proceedings of the 20th Annual Conference on Neural Information Processing Systems, Neural information processing system foundation, Canada, 2007: 281-288.

[108] T.White. Hadoop: The definitive guide [M]. O'Reilly Media, 2012.

[109] V. Srinivasan, M. J. Carey. Performance of B+ tree concurrency control algorithms [J]. The VLDB Journal, 1993, 2(4): 361-406.

[110] H. Choi, J. Son, Y. H. Cho, M. K. Sung and Y. D. Chung. SPIDER: a system for scalable, parallel/distributed evaluation of large-scale RDF data [C]. Proceedings of the 18th ACM

conference on Information and knowledge management. ACM, 2009: 2087-2088.

[111] J. Myung, J. Yeon, S. Lee. SPARQL basic graph pattern processing with iterative MapReduce [C]. Proceedings of the 2010 Workshop on Massive Data Analytics on the Cloud. ACM, 2010: 6.

[112] M. F. Husain, J. McGlothlin, M. M. Masud, L. R.Khan and B. Thuraisingham. Heuristics-based Query Processing for Large RDF Graphs using Cloud Computing [J]. IEEE Transactions on Knowledge and Data Engineering, 2011, 23(9): 1312–1327.

[113] C. Weiss, P. Karras and A. Bernstein. Hexastore: sextuple indexing for semantic web data management [J]. Proceedings of the VLDB Endowment, 2008, 1(1): 1008-1019.

[114] J. Sun and Q. Jin. Scalable RDF store based on HBase and MapReduce [C]. Advanced Computer Theory and Engineering (ICACTE), 2010 3rd International Conference on. IEEE, 2010, 1: V1-633-636.

[115] C. Franke, S. Morin, A. Chebotko, J. Abraham and P. Brazier. Distributed Semantic Web Data Management in HBase and MySQL Cluster [C]. Proceedings of 2011 IEEE 4th International Conference on Cloud Computing, 2011: 105–112.

[116] J. Urbani, S. Kotoulas, J. Maassen, F.V. Harmelen and H. Bal. WebPIE: A Web-scale Parallel Inference Engine using MapReduce [J], Journal of Web Semantics, 2012, 10: 59–75.

[117] FactForge [EB/OL]. http://www.factforge.com.

[118] Linked Life Data (LLD) [EB/OL]. http://linkedlifedata.com.

[119] R. Mutharaju, F. Maier and P. Hitzler. A MapReduce Algorithm for EL+ [C]. Proceedings of the 23rd International Workshop on Description Logics, 2010: 464–474.

[120] 潘超, 古辉. 本体推理机及应用 [J]. 计算机系统应用, 2010, 19(9): 163-167.

[121] 李乔, 郑啸. 云计算研究现状综述 [J]. 计算机科学, 2011, 38(4): 32-37.

[122] M. Li. Study on ontology repository management system [D], Beijing: Renmin University of China, 2006.

[123] A. Harth and S. Decker. Optimized index structures for querying RDF from the Web [C], Proceedings of the LA-WEB 2005: 71-80.

[124] D.J. Abadi, A. Marcus, S.R. Madden and K. Hollenbach. Scalable semantic Web data management using vertical partitioning [C]. In: Koch C, Gehrke J, Garofalakis MN, Srivastava D, Aberer K, Deshpande A, Florescu D, Chan CY, Ganti V, Kanne CC, Klas W, Neuhold EJ, eds. Proc. of the VLDB 2007. New York: ACM Press, 2007. 411−422.

[125] G. Qin. Some key issues of ontology repository managment system [D], Beijing: Renmin University of China, 2006.

[126] E. Chu, A. Baid, T. Chen, A.H. Doan and J. Naughton. A relational approach to incrementally

extracting and querying structure in unstructured data [C]. In: Koch C, Gehrke J, Garofalakis MN, Srivastava D, Aberer K, Deshpande A, Florescu D, Chan CY, Ganti V, Kanne CC, Klas W, Neuhold EJ, eds. Proc. of the VLDB 2007. New York: ACM Press, 2007. 1045−1056.

[127] Y. Shu, W. X. Pin and W. Gang. Analysis of semantic query performance for jena-based storage model [C]. Proceedings 2010 IEEE International Conference on Software Engineering and Service Sciences, ICSESS 2010, July 16, 2010 - July 18, 2010, Beijing, China: 553-556.

附　录

A. 攻读博士学位期间发表的学术论文

[1] **Ren Li**, Dan Yang, Haibo Hu, Juan Xie and Li Fu. Scalable RDF Graph Querying using Cloud Computing. Journal of Web Engineering, vol. 12, no. 1-2, pp. 159-180, 2013. (SCI: 000315305100006; EI: 20130315912814)

[2] **Ren Li**, Dan Yang, Haibo Hu, Jianhua Luo and Juan Xie. Scalable OWL Ontology Reasoning using Cloud Computing. International Journal of Advancements in Computing Technology, vol. 4, no. 21, pp. 623-631, 2012. (EI: 20130515976409)

[3] **李韧**, 杨丹, 胡海波, 谢娟, 吴云松, 傅鹂. 云计算环境下的 SWRL 规则分布式推理框架. 重庆大学学报, vol. 36, no. 2, pp. 57-63, 2013.

[4] **Ren Li**, Dan Yang, Haibo Hu and Jianhua Luo. Towards Description Logics based Cloud Service for UML Verification. International Conference on Information Engineering and Applications, IEA 2011, October 21, 2011 - October 23, 2011, Chongqing, China, , Lecture Notes in Electrical Engineering, 2012, vol. 154, pp. 1184-1191. (EI: 20114814559560)

[5] **Ren Li**, Jianhua Luo, Dan Yang, Haibo Hu, Ling Chen. A Scalable XSLT Processing Framework based on MapReduce. Journal of Computers. (EI 核心期刊，已录用)

[6] 王成良, **李韧**, 王主丁. 面向服务架构的电力分布式计算系统模型. 重庆大学学报, vol.32, no.2, pp.69-73, 2011. (EI: 20111313882128)

[7] Haibo Hu, Dan Yang, Hong Xiang, Li Fu, Chunxiao Ye, and **Ren Li**. Towards a semantic web enabled knowledge base to elicit security requirements with misuse cases. In Proceedings of the 8th International Workshop on Security in Information Systems (WOSIS'2011), in conjunction with ICEIS 2011, June 8-9, 2011, Beijing, INSTICC Press, pp. 103-112.(DBLP Indexed)

[8] Haibo Hu, Dan Yang, Hong Xiang, Li Fu and **Ren Li**. Semantic Web-based policy interaction detection method with rules in smart home for detecting interactions among user policies. IET Communications, vol. 5, no. 17, pp. 2451-2460, Nov 2011. (SCI: 000298862100004)

[9] Haibo Hu, D Yang, Chunlei Fu and **Ren Li**. Detecting interactions between behavioral requirements with OWL and SWRL. World Academy of Science Engineering and Technology, vol. 72, pp. 330-336, 2010. (EI: 20110113556034)

B. 攻读博士学位期间参加的科研项目

[1] 重庆市自然科学基金重点项目：云计算环境下基于语义技术的软件需求工程关键技术研究(CSTC-2011BA2022)，主研人员，在研项目。

基于Hadoop的大规模语义Web本体数据查询与推理关键技术研究

作者： 李韧
学位授予单位： 重庆大学