分类号 密级

U D C

学 位 论 文

基于分布式的海量XML文档关键字查询处理的研究与实现

|  |  |  |  |
| --- | --- | --- | --- |
| 作者姓名： | 占 恒 | | |
| 指导教师： | 王国仁 教授 | | |
|  | 东北大学信息科学与工程学院 | | |
| 申请学位级别： | 硕士 | 学科类别： | 工学 |
| 学科专业名称： | 计算机系统结构 | | |
| 论文提交日期： | 2016年12月 | 论文答辩日期： | 2016年12月 |
| 学位授予日期： | 2017年1月 | 答辩委员会主席： | 高福祥 教授 |
| 评 阅 人： | 王波涛 教授 张大波 教授 | | |

东 北 大 学

2015年 6月

**A Thesis in Computer System Architecture**

**Research and Implementation of Distributed Keyword Query Processing over Massive XML Documents**

By Zhan Heng

Supervisor: Professor Wang Guoren

**Northeastern University**

**November 2016**

独创性声明

本人声明，所呈交的学位论文是在导师的指导下完成的。论文中取得的研究成果除加以标注和致谢的地方外，不包含其他人己经发表或撰写过的研究成果，也不包括本人为获得其他学位而使用过的材料。与我一同工作的同志对本研究所做的任何贡献均已在论文中作了明确的说明并表示谢意。

学位论文作者签名：

日 期：

学位论文版权使用授权书

本学位论文作者和指导教师完全了解东北大学有关保留、使用学位论文的规定：即学校有权保留并向国家有关部门或机构送交论文的复印件和磁盘，允许论文被查阅和借阅。本人同意东北大学可以将学位论文的全部或部分内容编入有关数据库进行检索、交流。

作者和导师同意网上交流的时间为作者获得学位后：

半年 □ 一年□ 一年半□ 两年□

学位论文作者签名： 导师签名：

签字日期： 签字日期：

基于分布式的海量XML文档关键字查询处理的研究与实现

# 摘 要

XML已成为Web数据交换和信息表示的事实标准。随着XML数据量的急剧增长，如何对这些正在快速增长的海量XML数据有效地组织和存储，并提供高效快速的数据检索，是当今信息检索领域的一个研究重点。目前，XML数据的存储和检索一般还使用原生的XML数据库或者关系数据库，但这类系统无法满足海量XML数据的性能要求，而基于分布式的XML数据存储和检索技术也尚未成熟。MapReduce的出现在一定程度上缓解了上述问题。MapReduce是一种处理海量数据的有效解决方案，但是基于该框架处理海量XML数据查询问题的研究成果却很少，而且现有的分布式Twig查询算法在Map阶段需要做结构连接操作，大多会产生大量无用的中间结果，此外，这种算法往往还需要额外的查询分解操作。针对以上问题，本文提出了两种基于节点分发的有效查询方案：NDTH算法和DTH算法，实现海量XML数据的Twig查询处理。

本文提出的两种查询方案是基于节点分发的思想，即在Map阶段不做结构连接操作，而是将处于不同分片上但却可能构成查询解的节点分发到一台Reduce计算节点上，这样在Reduce阶段就可以根据查询模式的特点，选择适合该查询且性能最优的整体匹配算法，如选用对祖先后代关系性能最优或父子关系最优的整体匹配算法等。

本文首先基于ComMapReduce提出了NDTH算法，该算法利用ComMapReduce的协调者节点收集全局的键值，通过全局键值能够舍弃那些不能构成最终查询解的节点，进而提高查询效率，同时保证最终查询结果不丢失。其次，本文通过对XML数据结构和MapReduce工作原理的研究，分析了现有基于MapReduce的XML查询处理方法中，文档分片技术的局限性，提出了松弛分割算法（Relax-Fragment），该算法能够实现对XML文档的任意分割而不需要依赖查询信息。进而在松弛分片策略RFS的基础上，我们设计了基于松弛分片的DTH算法。该算法利用记录了分片祖先信息的松弛分片索引，能够加快查询速度和保证并行查询结果的正确性和完整性。

最后，本文采用大量真实数据设计了一组实验，并对本文所提的两种分布式Twig查询处理算法的实验结果进行了详细分析。本文实验结果表明：分布式NDTH算法和DTH算法能够减少海量XML数据查询处理时间，具有较高的查询效率和良好性能。

**关键词**：XML；关键字查询；SLCA；分布式环境；MapReduce

Research and Implementation on Distributed Holistic Twig Query Processing based on Nodes Distribution

Abstract

XML has been adopted as a de facto standard format for web data representation and exchange. With the sharply increasing scale of XML data, pursuing effective management of massive amounts of XML data as well as providing efficient and fast data retrieval have been a research focus in the field of information retrieval today. So far, most storage and retrieval systems of XML data are using Native XML database or relational database, but those systems are not effective on large-scale XML data, and the storage and retrieval of XML data technology on distributed environment is not mature. MapReduce is an effective solution to deal with huge amounts of data. However, there are few research findings aiming at processing the query over massive XML data. And the existing distributed twig query algorithm needs structural join operation in the Map phase, which may conduct a large number of useless intermediate results in most cases. In addition, this kind of algorithm often need extra operation of twig query pattern decomposition. In order to solve the above problems, in this thesis, we put forward two kinds of effective query schemes based on node distribution, i.e., algorithm NDTH and DTH, to realize twig query over massive XML data.

These two kinds of query schemes proposed in this thesis are based on nodes distribution, which distribute all the nodes that may contribute to the query solution to the same computing node in the Map phase, instead of perform structural join operation. So in the Reduce phase it can adopt any holistic matching algorithm, and choose the optimal performance of holistic matching algorithm according to the characteristics of the query, such as optimal matching algorithm of ancestor-descendent relationship or parent-child relationship.

In this thesis, we first propose NDTH algorithm, which collect global keys through a coordinator site in ComMapReduce. This algorithm can improve the efficiency of query processing and guarantee the final query solution is complete. Next, based on the study of XML data structure and MapReduce framework, we analyze the limitations of fragmentation technology in existing XML query processing methods based on MapReduce. We put forward the relax-fragment algorithm, which can realize arbitrary fragmentation of XML without dependence on query information. Then, based on the relax fragmentation strategy RFS, we propose DTH algorithm. The DTH algorithm use the RF index which stored ancestors information, to speed up the query processing and guarantee the correctness and completeness of the query results.

Finally, this thesis use a large number of real dataset to design a set of experiments, and we made an analysis on the experimental results of the two kinds of distributed twig query processing algorithms proposed in this thesis. The experimental results in this thesis show that the distributed algorithms NDTH and DTH can reduce the query processing time on massive XML data, achieving high efficiency and good performance.

**Keywords:** XML; nodes distribution; MapReduce; Twig query processing; holistic query processing

目 录

[独创性声明 I](#_Toc469438523)

[摘 要 II](#_Toc469438524)

[Abstract III](#_Toc469438525)

[第1章 绪 论 1](#_Toc469438526)

[1.1 研究背景 1](#_Toc469438527)

[1.2 研究目的与意义 2](#_Toc469438528)

[1.3 本文主要工作 3](#_Toc469438529)

[1.4 组织结构 4](#_Toc469438530)

[第2章 相关研究工作 5](#_Toc469438531)

[2.1 XML数据管理概念 5](#_Toc469438532)

[2.1.1 XML基础 5](#_Toc469438533)

[2.1.2 XML编码 7](#_Toc469438534)

[2.2 关键字查询研究现状 9](#_Toc469438535)

[2.2.1 最紧致片段的相关概念 10](#_Toc469438536)

[2.2.2 分布式环境下XML关键字查询 12](#_Toc469438537)

[2.4 本章小结 13](#_Toc469438538)

[第3章 基于ComMapReduce的NDTH算法 15](#_Toc469438539)

[3.1 云计算技术 15](#_Toc469438540)

[3.1.1 分布式文件系统 15](#_Toc469438541)

[3.1.2 MapReduce分布式计算框架 16](#_Toc469438542)

[3.1.3 ComMapReduce分布式计算框架 18](#_Toc469438543)

[3.2 问题定义和问题分析 19](#_Toc469438544)

[3.3 基于ComMapReduce的分布式Twig查询处理算法 21](#_Toc469438545)

[3.3.1 节点分发处理策略 21](#_Toc469438546)

[3.3.2 基于ComMapReduce的NDTH算法 24](#_Toc469438547)

[3.4 本章小结 26](#_Toc469438548)

[第4章 基于松弛分片的DTH算法 28](#_Toc469438549)

[4.1 ComMapReduce框架的瓶颈分析 28](#_Toc469438550)

[4.2 松弛分片策略 29](#_Toc469438551)

[4.2.1 松弛分片索引 30](#_Toc469438552)

[4.2.2 Relax-Fragment算法 33](#_Toc469438553)

[4.3 基于松弛分片的DTH查询处理算法 35](#_Toc469438554)

[4.4 本章小结 39](#_Toc469438555)

[第5章 实验及结果分析 40](#_Toc469438556)

[5.1 实验性能评估标准 40](#_Toc469438557)

[5.2 实验环境及实验设计 41](#_Toc469438558)

[5.2.1 实验环境 41](#_Toc469438559)

[5.2.2 实验数据集 42](#_Toc469438560)

[5.2.3 实验方案 42](#_Toc469438561)

[5.3 实验结果分析 43](#_Toc469438562)

[5.3.1松弛分片算法Relax-Fragment对文件上传时间影响 43](#_Toc469438563)

[5.3.2 NDTH与DTH算法的加速比性能分析 44](#_Toc469438564)

[5.3.3 NDTH与DTH算法的数据伸缩率性能分析 47](#_Toc469438565)

[5.3.4 NDTH与DTH算法的扩展率性能分析 49](#_Toc469438566)

[5.4 本章小结 50](#_Toc469438567)

[第6章 总结与展望 52](#_Toc469438568)

[6.1 论文工作总结 52](#_Toc469438569)

[6.2 未来工作展望 53](#_Toc469438570)

[参考文献 55](#_Toc469438571)

[致 谢 59](#_Toc469438572)

[攻读硕士学位期间参加的项目和发表的论文 61](#_Toc469438573)

# 第1章 绪 论

随着互联网地高速发展，XML语言已经成为互联网上重要的存储和交换标准，它涉及到各个领域，例如是电子商务，数字图书馆，Web服务等都有广泛的使用，XML类型的数据成为当前主流的数据形式。如何从XML数据中有效的获取所需的信息已成为学术界研究的热点问题。一种有效的查询XML数据的方法是使用XPath和XQuery等结构化查询语言。它们可以精确的表达用户的查询意图，但是想要正确使用它们，用户必须了解XML数据模式信息并掌握复杂的查询语法。实际应用中，对于普通用户来说，很多情况下难以顺利得到所需的数据。另一种简单便捷的查询方式是使用关键字查询。用户无需了解模式和复杂的查询语法，只需提交简单的关键字即可。因此，寻求一种支持海量XML数据关键字的有效、高性能的存储和查询方案，是人们密切关注的。

## 1.1 研究背景

XML（eXtensible Markup Language，可扩展标记语言）[1]是一种类似于HTML的标记语言，不同的是，XML是一种元标记语言，用户可以自己定义所需要的标记，能够更加灵活的存储数据。XML由万维网协会（W3C）设计，与HTML等同为SGML（Standard Generalized Markup Language，标准通用标记语言）的重要分支。可扩展标记语言可以对文档和数据进行结构化处理，从而能够在部门、客户和供应商之间进行交换，实现动态内容生成，企业集成和应用开发。可扩展标记语言可以使我们能够更准确的搜索，更方便的传送软件组件，更好的描述一些事物。

XML是目前在互联网上最流行的一种数据表示和数据格式[2]。由于越来越多的数据已经采用XML格式排版，单个XML文档的数据量也变得巨大，而且增长速度也很快。例如，维基百科提供的页面转储为一个单独的XML文档，其大小也超过40GB[3]。在科学领域这种大规模的XML文档更常见。此外，XML文档的大小还在不断增长，因为生物学家在实验中不断会有关于蛋白质的新发现。因此，在一个大型XML文档中的查询处理需求也仍在不断增长。

伴随着海量的XML文档出现，传统的关键字查询算法无法直接高效地应用在海量规模XML数据的查询处理中，所以人们使用多核处理器并行技术和传统分布式数据库技术处理海量规模XML数据。虽然这些技术在一定程度上能提高查询效率，但是当数据量持续急剧增长时，不同站点之间的中间结果传输量会变得很大，从而这些站点之间的较高通信代价使得大规模XML数据的实际查询效率下降，此外，线程或机器节点之间的通信以及一致性维护等代价也严重降低了XML数据的查询效率，所以研究者们又开始寻找更有效的方法来提高海量规模XML数据的查询效率。

近年来，云计算技术的出现，给研究者们提供了广阔的研究空间，使得云环境下的关键字查询成为了新的研究领域之一。云计算技术主要以MapReduce作为计算模型。该模型是一个处理和生成超大数据集的算法模型的相关实现，有很强的容错性和可扩展性，而且具有强大的海量数据并行处理能力， MapReduce架构的程序能够在大量的普通配置的计算机上实现并行化处理。这个系统在运行时只关心：如何分割输入数据，在大量计算机组成的集群上的调度，集群中计算机的错误处理，管理集群中计算机之间必要的通信。采用MapReduce架构可以使那些没有并行计算和分布式处理系统开发经验的程序员有效利用分布式系统的丰富资源，而且用户只需要定义Map函数和Reduce函数即可。MapReduce的本质是将问题分解为完全独立的子问题，并将局部计算发送至需要被处理的数据上进行并行的局部计算。

云环境系统这种高效的数据存储、计算及管理技术，给我们进一步研究海量XML数据关键字查询处理提供了新的技术支持和解决方案。

## 1.2 研究目的与意义

随着XML数据的海量应用的快速增多，XML数据规模也越来越庞大，从数百MB上升为数千GB，甚至为更大的TB级别，呈现出爆炸式增长的趋势。例如在科学研究上，对DNA序列的记录，历史天气的记录等，以及在互联网上，数据库历史日志记录、用户的搜索行为记录、数字图书馆系统维护的大量图书信息等。这些由XML格式记录的信息，随着时间推移和研究的深入而变得越来越大，同时新的大规模XML数据不断出现，数据量也日益增多。因此，在这种新的应用背景下，传统数据库性能已远远满足不了大数据时代对系统高性能的需求。这对XML数据的存储、查询、分析处理提的性能出了更高的要求。因此设计和实现一种高效而准确的并行查询处理海量XML数据的算法就变得由为迫切和重要。

目前在XML文档的关键字查询方面已进行了大量卓有成效的研究，提出了一系列有效的关键字查询处理算法，但这些算法还不足以满足现有需求，传统分布式数据库技术处理大规模XML数据所产生的中间结果的传输共享、线程或机器节点之间的通信以及一致性维护等代价太大，查询效率不显著，而且该环境是将XML数据全部映射到单机内存中进行集中式分片，需要看到全局的文档，而海量XML数据大到已无法装载到单机内存中。目前，虽然研究者们针对MapReduce框架也做过一些关键字查询处理的研究，但是学术成果非常稀少。所以本课题研究的目的就是为了设计一个用于提高海量XML文档的查询效率的分布式关键字查询处理算法，以解决海量XML的关键字查询处理效率低的问题。

近年来，随着云计算技术的快速发展，一些发展比较成熟的开源云计算平台，如Hadoop，凭借其可靠、高效、可伸缩的特性不仅在互连网领域得到了广泛的应用，同时还得到了学术界的普遍关注[6]。基于MapReduce框架的分布式关键字查询处理算法的学术研究成果还不多，相关的研究也还不深入。因此本课题对基于MapReduce的分布式关键字查询处理技术进行深入研究是很具有学术意义的，而且本课题的研究成果能解决已有的分布式关键字查询处理中的问题，在XML数据分片方面，能够弥补已有的分片技术的不足，满足对不从根到叶的路径进行分割需求也能正确执行查询操作。

如何设计一种有效的分片后的结构信息保存策略，使其能够满足任意情况的分片结果都能执行关键字查询处理算法，是本课题在MapReduce框架下研究海量规模XML关键字查询处理的一个关键问题。此外，如何设计一个能提高查询速度、减少中间结果、并能够保证最终解的完整性和正确性的分布式关键字查询处理算法也是本课题需要重点研究的问题，相关的研究成果也具有理论意义和实际应用价值。

## 1.3 本文主要工作

本文主要分析和研究了云计算平台上海量XML数据中关键字查询处理的各个阶段，包括对海量XML文档的预处理、海量XML文档分片、分布式关键字查询处理等。在此基础上，采用节点分发思想提出了两种有效的查询方案，包括基于XXX算法和YYYY算法。

本文主要围绕以下内容展开研究和讨论：

（1）本文是第一篇讨论对海量XML数据进行任意分割，并且能够在任意分片的情况下，对进行海量XML数据执行整体小枝模式匹配算法的文章。

（2）本文首先提出了基于ComMapReduce的NDTH算法。该算法在任意分片阶段不需要特殊处理，按照Hadoop的默认分割方式处理即可。在节点分发处理阶段，引入了协调者节点收集全局的键值，保证了各分片上的解不丢失，并且在查询处理阶段能够应用整体结构连接算法提高查询效率。

（3）本文提出了具有普适性的松弛分片算法和基于松弛分片的DTH算法。本文设计了能够满足任意分片需求，又能够维持分片与原始文档间结构信息的松弛分片算法。在此基础上，研究了如何在节点分发阶段将相关XML节点分发到同一个计算节点做查询处理，提出了基于节点分发的Twig整体匹配查询算法。

（4）在不同数据集上，对本文所提出的两种算法从加速比、数据伸缩率和扩展率三个方面进行了实验和评估，实验结果验证了本文所提算法的正确性，同时表明了在海量XML数据的Twig整体匹配查询问题上，本文所提算法具有高效性。

## 1.4 组织结构

本文主要研究云环境下海量XML数据的关键字查询处理。首先对XML文档及文档编码技术进行了综合性阐述。然后介绍了单机上Twig查询算法和分布式环境的XML查询算法。为了解决当前海量XML数据的查询处理问题，我们研究了MapReduce计算框架，并提出了两种基于节点分发思想的查询算法，即NDHT和DHT算法。最后，对NDTH算法和DTH算法进行实验评估，并给出了实验结果分析。

本文后续章节的组织结构安排如下：

第2章是相关研究工作。首先是介绍了XML数据和几种比较重要的XML编码技术，并对各种编码技术做了对比；然后介绍了集中环境下关键字查询算法；最后介绍并总结了分布式环境下的XML的关键字查询处理算法。

第3章是基于ComMapReduce的NDTH算法。本章针对海量XML数据的Twig查询问题，首先介绍了MapReduce计算模型，包括ComMapReduce的工作原理；然后介绍了NDTH算法，针对各文档分片进行节点分发时引起查询结果丢失的问题，NDTH算法借助协调者节点来处理全局信息，以保证查询处理阶段不会丢失查询结果。

第4章是基于松弛分片的DTH算法。为了避免ComMapReduce中由大量通信代价及时间代价引起的瓶颈问题，DTH算法是结合了MapReduce框架特性，重新考虑了文档分片的处理。第一，首先分析了DTH算法的瓶颈问题和现有分片技术存在的问题，设计了可以实现任意分割XML文档的松弛分片策略，然后介绍了松弛分片索引和松弛分片算法。第二，针对如何将不同分片上的相关节点分发到一个计算节点上进行查询处理的问题，本章在松弛分片算法的基础上，提出了基于松弛分片的DTH算法。

第5章是实验及结果分析。针对本文提出的两个分布式Twig查询处理算法，NDHT算法和DHT算法，我们主要从算法的加速比（Speedup），规模增长性（Sizeup）和可扩展性（Scaleup）三个方面进行了实验比较，并对实验结果进行详细的分析。

第6章是总结与展望。首先对全文的研究工作进行了总结，然后在本文的研究工作和实验结论的基础上，对未来进一步的研究内容和方向提出展望。

# 第2章 相关研究工作

现如今互联网上大量的数据以XML作为输入和交换的标准，高效查询XML数据一直成为人们研究关注的焦点。本章首先全面的介绍XML文档的相关定义，其中包括XML标准、编码、关键字查询等；再详细介绍现有的XML关键字查询技术，然后定义文章中出现的技术和概念；紧接着讨论了分布式环境下的关键字查询，包括Hadoop系统、MapReduce技术框架等。

## 2.1 XML数据管理概念

本节主要介绍XML文档的相关定义，其中包括XML标准、编码、关键字查询相关特点的描述。

XML已经广泛的应用到各个领域，相关的技术也十分丰富和成熟。XML的查询研究主要分为以下两个方面，首先是XML查询语言，如XQuery[12]、XPath[13]等结构化查询语言，得到的是精确逻辑结构的查询结果；另一方面是关键字查询，如XRANK[14]、Multiway SLCA[15]，最终获得是基于某种语义下包含所有关键字的XML节点集。

### 2.1.1 XML基础

XML（eXtensible Markup Language）是由是一种可扩展标记语言[1]，是一种介于SGML和HTML之间。XML通常被视为一种特殊的半结构化数据，它可以创建自己的标记。相比与HTML语言，XML更加灵活和开放。

另外，自描述性和半结构化是XML数据最基本的两个特点，XML数据本身就已经包含了元数据，而且其可以创建自己的标记，如元素、属性等。所谓的半结构化，即XML数据不同于传统关系数据，传统的数据都有一定的数据模型，可以根据模型来具体描述特定的数据，而XML数据的结构限制并不严格，表现为语义单元相互嵌套的层次关系。如图2.1所示的XML文档，它由子元素组成，每个元素都是由起始标记（如<conf>）和结束标记（</conf>）以及它们之间的内容来标识，元素与元素之间还可以嵌套，每个标签元素按照一定顺序排列，最终可以形成一颗完整的文档树。但元素之间嵌套必须正确，例如<p><w>info</w></p>w是p的子元素，那么子元素w的起始标记<w>和结束标记</w>需要被父级元素p包含。

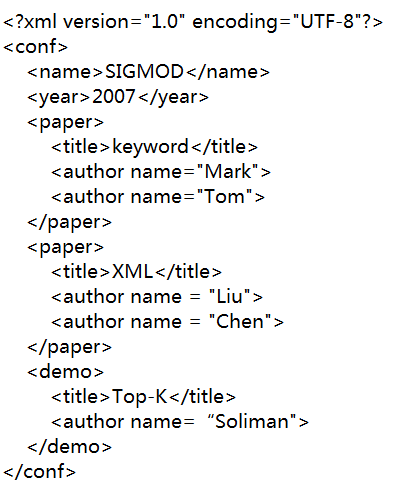


图2.1 XML文档

上图中的XML数据是是采用数据形式来描述定义的。我们可以从另一个数据模型角度来描述它，实际上XML数据可以看成是由节点和边组成的包含文档结构和内容信息的文档树。具体的内容信息是由节点来描述，而XML数据中的祖先-后代的结构信息则存储在边上。根据这种节点和边的结构关系可以将上图中的XML文档改写成一颗XML文档树，如图2.2所示，一颗结构完整的XML文档树，其中节点<conf>为此XML文档树的根节点。

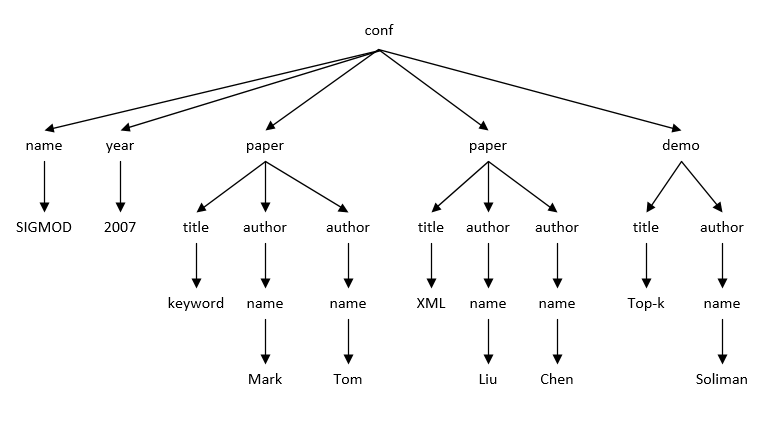


图2.2 XML文档树

**定义2.1 XML文档树** 给定一个XML文档D ，节点的集合和边的集合分别是*N*和*E*，一颗有序的结构文档树T*D*­=（*N,E*）{*v*∈N,*e*∈E}，其中*v*表示XML文档树中的任意节点，*e*表示任意一条边。

XML文档树在加强XML表达结构能力的同时，也为编码和查询奠定了基础。在XML文档树中每个节点的位置，都可以由根节点出发和指向它的边组成一条唯一的路径来表示。树结构是用来描述半结构化数据的一种高效的数据模型。

### 2.1.2 XML编码

XML编码是指按照某种规则，统一的为XML文档树中的每一个节点指定唯一标识。XML文档树统一编码之后，每个节点之间的包含关系就能确定，在XML数据查询的时候，就能够根据编码迅速的确定节点间的祖先-后代关系。

#### 2.1.2.1 位向量编码

XML文档树上的每个节点，位向量编码是由一个*n*位向量构成，*n*是文档树的节点总数，该*n*位向量上的第*i*位标示文档树上第*i*个节点是否存在，“1”为存在，“0”为不存在。在一个自底向上（或自顶向下）的编码方案中，每一个节点的编码继承它祖先节点（或后代节点）所有位置上为1的编码，保证了每个节点都包含了祖先（或后代）节点信息。若一个位向量编码继承了其祖先，用二元位与运算AND（&）能快速地检测节点*u*是否为节点*v*的祖先。如果c(*u*)&c(*v*)=c(*u*)，则说明*u*是*v*的祖先。针对继承后裔的位向量编码，用二元位或运算OR（|）能快速地检测出节点*u*是否是节点*v*的后代。如果c(*u*)|c(*v*)=c(*v*)，则说明*u*是*v*的后裔。因此位向量编码能够很好的支持包含关系。如图2.3所示，一个基于位向量编码的XML文档树实例。另外位向量编码的缺点在于无法准确的表示出兄弟节点之间的关系。

#### 2.1.2.2 区间编码

根据XML文档有序的特点提出了区间编码，每个节点存有一对数字。数字形成的区间则表示节点覆盖的范围，那么节点间的包含关系和位置关系的判断问题，可以转化为节点覆盖区间的包含关系或不相交关系的判断问题。用区间编码来描述节点之间的结构关系，就可以把问题具体分成两种情况：1）区间编码之间完全不相交；2）其中一个节点的区间编码完全包含另一个节点的区间编码。即如果某一节点*u*的区间编码与另一个节点*v*的区间编码不相交，那么节点*u*或者在节点*v*的左侧，或者在节点*v*的右侧；如果某一节点*u*表示的区间编码完全包含另一节点*v*的区间编码，那么节点*u*是节点*v*的祖先节点。鉴于这个思想，人们在研究过程中提出了很多基于区间的编码方案，其中就包括起止编码和前序后序编码。

#### 2.1.2.3 Dewey编码

Dewey编码是一种基于路径的前缀编码方法，针对某一个节点，其双亲节点的编码将作为该节点编码的前缀。换句话说，若一个节点的Dewey编码是另一个节点Dewey编码的前缀，那么可以判断该节点是另一个节点的祖先节点。假设一颗XML文档树T中的一个节点*u*，令其编码为*D(u)*，其孩子节点*v*的编码是D(*v*)= *D(u)*.*n*，其中*n*是节点*u*的孩子节点中v所在的孩子节点的序号，“．”代表分隔符。Dewey编码是最著名的前缀编码方法，它能够有效地支持前缀编码的思想。

**定义2.1 Dewey编码**：给定一个XML文档对应的标签有向树G=(*V,E,r,A*)，其中*V*为树中节点的集合；*E*为树中边的集合；*r*为有向树的根节点；*A*是节点标签的集合。则节点*v*的Dewey编码是这样规定的：

1. 根节点*r*的Dewey编码规定为0；
2. 对G进行宽度优先遍历的过程中，若节点*v*是节点*u*的第*i*个子节点，那么节点*v*的Dewey码为*D*(*v*)=*D*(*u*).(*i*-1)，其中*D*(*u*)表示节点*u*的Dewey编码。

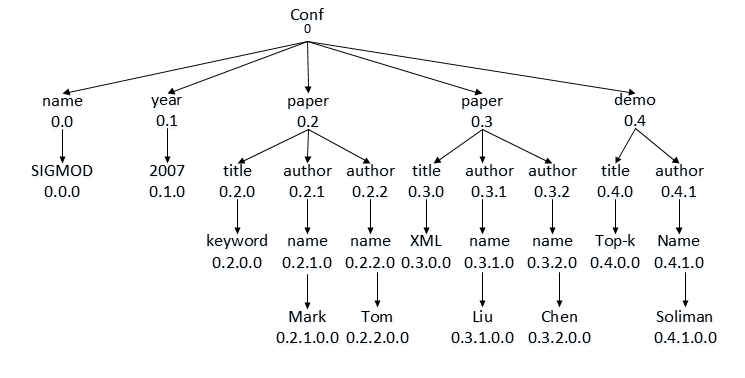


图2.3 基于Dewey编码的XML文档树

如图2.3所示，这是将图2.1中的XML文档进行Dewey编码所形成的一颗XML文档树，每个节点下面的数字就是该节点的Dewey编码。一个节点v的Dewey编码记为*D*(v)，其长度就是被“.”分割开数字的数字的个数，记为*lv*。例如图2.3中的SIGMOD节点的Dewey编码长度为3，Soliman节点的编码长度为5。

如何判断两个节点之间的父子关系是Dewey编码的优势所在，给定两个节点*u*和*v*，若满足*D*(*u*)是*D*(*v*)的前缀，那么*u*是*v*的祖先节点。特别地，当*u*是*v*的祖先节点，且*D*(*v*)比*D*(*u*)只多出一个分隔符“.”，那么*u*是*v*的父亲节点。例如图2.3中Mark的父节点是name，对应的Dewey编码0.2.1.0是0.2.1.0.0的前缀且仅少一个分隔符，同理可知paper(0.2)节点是Mark节点的祖先节点，0.2是0.2.1.0.0的前缀。

*D*(*u*)和*D*(*v*)分别表示节点*u*和*v*的Dewey编码，长度分别记为*lu*和*lv*，u(i)代表D(u)中第i层的整数，pu(i)记为第i层的前缀，即第1层到第i层整数构成的Dewey编码。

性质2.1 （相等关系）：若*lu*=*lv*，然后比较*D*(*u*)和*D*(*v*)，依次从左往右逐层比较，若每层的整数都相等，则称*D*(*u*)和*D*(*v*)相等，表示成*D*(*u*)=*D*(*v*)。

性质2.2 （包含关系）：若*lu*<*lv*，且pv(*lu*)=*D*(*u*)，则称*D*(*v*)包含*D*(*u*)。

性质2.3 （判断大小）：

1. 若*D*(*v*)包含*D*(*u*)，则称为*D*(*v*)>*D*(*u*)。
2. 从左至右依次比较*D*(*v*)和*D*(*u*)，若第i层上的整数不相等若u(i)<v(i)，则称*D*(*v*)>*D*(*u*)。

性质2.4 （后代关系）表示为*descendent*(*u*,*v*)，运算如下:

1. 如果参数*D*(*v*)、*D*(*u*)中某一个为空，则返回另一个非空的Dewey编码;
2. 如果*v*包含*u*，则返回*v*。

性质2.5 （公共前缀）表示为*lca*(*u*,*v*)

1. 若二者具有包含关系，则返回被包含的Dewey码。
2. 从左至右顺序比较*D*(*v*)和*D*(*u*)各层的整数，仅当第*i*层时二者的整数不同，

那么，pu (i-1)即为二者的公共前缀。

由上图可见，Dewey编码能够准确的包含路径信息，为XML文档查询提供了很好的支持，能够在线性时间内判断某两个节点的祖先后代关系，这也是为什么Dewey编码能够被广泛使用的其中一个原因；其缺点在于浪费了很大的存储空间，一颗XML文档树的深度越大，Dewey编码的长度就越大，每个节点的存储代价就会越大。

## 2.2 关键字查询研究现状

在XML数据上进行关键字查询，就是根据给定的查询关键字集合，在目标XML文档树*T*上进行基于某种语义进行查询，最终得到正确完整的查询结果。

### 最紧致片段的相关概念

最紧致片段是用来定义根据查询的关键字集合返回的查询结果，顾名思义最紧致片段是XML文档树中的一个最小树片段，并且该片段必须同时满足所有查询关键字的组合语义，前人提出的关键字语义包括：文献[20]最早提出了LCA(Lowest Common Ancestor)，LCA所表示的节点包含所有的关键字的最近的公共祖先，该节点的任意子节点都不再包含所有关键字节点[23]，后来又有人提出了Smallest LCA(SLCA)[21]、Meaningful LCA(MLCA)[22]等相关语义并最终得到查询结果，本文主要围绕SLCA语义进行相关研究。

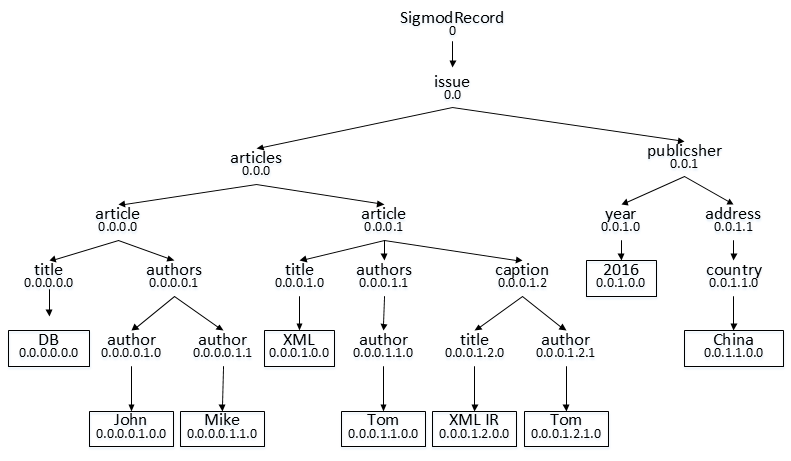


图2.4 XML文档树基于Dewey编码

**定义2.2 关键字匹配集合**：给定一个XML文档*D*和查询关键字*k*，用表示在*D*中匹配查询关键字*k*的节点集合，。

如图2.4中，，。

**定义2.3 最低共祖先（LCA）**：假设给定*m*个节点，存在节点*v*是它们的LCA，当且仅当：

1. *v*是的祖先节点，且
2. 不存在*u*，*u*是*v*的后代，并且*u*是的祖先节点，那么记。

例如图2.4中，已知查询的关键字集合为{Tom,XML}，那么返回的LCA包括article(0.0.0.1)和caption(0.0.0.1.2)，article(0.0.0.1)是XML(0.0.0.1.0.0)和Tom(0.0.0.1.1.0.0)的LCA，caption(0.0.0.1.2)是XML(0.0.0.1.2.0.0)和Tom(0.0.0.1.2.1.0)的LCA。

随着XML关键字查询研究和发展，为了提高XML关键字查询的性能和准确率，在LCA语义的基础上，又提出了许多新语义。其中最具有代表性的是SLCA，它也是目前研究XML关键字查询最紧致片段。

**定义2.4 最小最低共祖先（SLCA）**：在XML文档*D*中，给定*m*个查询关键字集合，节点集合。若不存在节点*u*，*u*是*v*的后代，集合，则称*v*是关于查询集合*K*的一个SLCA。

简单来讲，最终SLCA查询结果集合就是将LCA集合中满足祖先-后代关系的祖先节点删除后得到的结果。例如图2.4中，已知查询的关键字集合，DB(0.0.0.0.0.0)和Mike(0.0.0.0.1.1.0)的最长公共前缀是(0.0.0.0)，那么节点article(0.0.0.0)为SLCA。假设查询的关键字集合，先分别得到它们的关键字匹配集合，，，无法直接通过最长前缀匹配直接获得结果，那么可以先计算出LCA集合，即article(0.0.0.1)和caption(0.0.0.1.2)，然后通过Dewey编码可以判断出article(0.0.0.1)是caption(0.0.0.1.2)的祖先节点，那么caption(0.0.0.1.2)则为SLCA，那么基于SLCA语义，最终查询结果集将返回caption(0.0.0.1.2)唯一解。

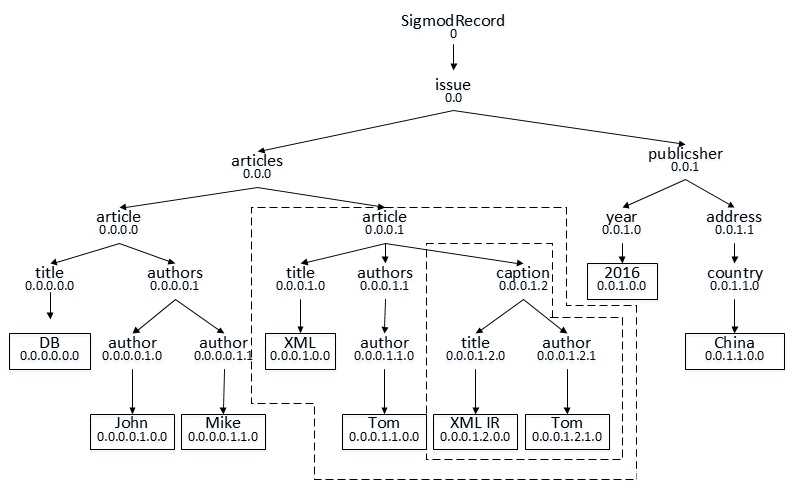


图2.5 求解SLCA示例图

### 分布式环境下XML关键字查询

随着互联网的高速发展，XML文档数据规模增长迅速，给现有的关键字检索技术带来新的性能挑战。在单个计算节点上处理大规模XML数据面临许多困难，比如CPU功耗有限、硬盘容量和内存等因素的限制，传统的XML关键字查询算法已无法满足人们的查询需求。

由于密集型计算模式下需要处理的数据对象主要是海量的网络资源，而关键字查询是网络用户最常用的查询方式，具有用户友好、便捷和效率高的特点。因此本文在设计大规模集群下的XML查询时，选择XML关键字查询作为核心数据检索处理技术。本节主要介绍分布式环境下XML关键字查询的几个相关研究成果和技术。目前为止，基于MapReduce的关键字查询的学术研究还不是很多。

文献[36-38]利用专门的MapReduce语言来支持云环境中处理各种大规模数据的应用开发。ChuQL[36]是一种XQuery在MapReduce上的扩展，允许开发者在Hadoop中处理大量XML数据，它针对MapReduce特性对XML查询做了特殊的优化。文献[37]中也提出了一种类似SQL的查询语言，称为MRQL语言，和其他已知的XPath和Xquery查询语言不同。MRQL语言支持在MapReduce编程环境中处理大规模数据，尤其是XML数据。它提供了一些新的语法规则，具有更多灵活性，能够处理多种格式的数据。MRQL的执行，需要的MapReduce任务的总数量与查询相关。文献[38]从减少数据规模角度出发，来提高海量XML查询处理中的查询效率。采用了XPath投影对XML数据分块进行预处理，减少XML数据分块的数据规模，然后对XML数据分块使用基于MRQL语言的查询引擎执行并行查询。

## 2.3 本章小结

本章先对XML文档数据管理模型进行了分析并引入了XML的相关概念，紧接着介绍了XML编码方案，包括位向量编码、区间编码、Dewey编码。然后介绍了XML数据关键字查询的最紧致片段，分别介绍了LCA和SLCA的基本概念，最后分析了海量XML数据关键字查询处理的研究现状。

# 第3章 基于SLCA的求解算法

SLCA是XML关键字查询中最紧致片段语义，目前对SLCA的求解已经有较多研究成果。SLCA问题的求解方法可概括如为:第一步获取XML文档树中包含相应关键字的叶子节点集合，标记为，即为包含关键字的所有叶节点的集合。之后可以根据节点路径和结构信息(例如Dewey编码)求解SLCA。

目前，求解SLCA的经典算法包括：基于堆栈的算法(Stack)[xx] 、ILE算法(Indexed Lookup Eager) [xx]、SE算法(Scan Eager) [xx]和LISA算法(Layered Intersection Scan Algorithm) [xx] ，上述算法均基于Dewey编码实现，其中SE算法是ILE算法的不同版本，LISA算法分为LISA 和LISA。下面将依次介绍SLCA的求解算法的思想和实现。

## 3.1 Stack算法

### 3.1.1 Stack算法基本思想

Stack算法是求解SLCA的经典算法之一，算法中引入一种数据结构堆栈，堆栈中的每个条目都是一个二元组(*id,Keywords*)，*id*从前到后依次为，即表示堆栈中的每个条目Dewey编码为；*Keywords*是一个数组，数组里存储*k*个布尔值。*Keywords* [*i*]=*T*意味着当前栈中Dewey编码对应的节点以其为根的子树直接包含了关键字*ki*。用此堆栈算法处理如图3.1(b)所示，

1. NameNode，DataNode和客户端。NameNode可以看作是分布式文件系统中的管理者，主要负责管理文件系统的命名空间、集群配置信息和存储块的复制等。DataNode是文件存储的基本单元，它将块存储在本地文件系统中，保存了块的元数据，同时周期性地将所有存在的块信息发送给NameNode。客户端就是需要获取分布式文件系统的文件应用程序。
2. 文件写入。客户端向NameNode发起文件写入的请求，NameNode根据文件大小和文件块配置情况，将它管理的DataNode的信息返回给客户端。客户端将文件划分为多个块，然后根据DataNode的地址信息，按顺序写入到每一个DataNode块中。
3. 文件读取。客户端向NameNode发起文件读取的请求，然后NameNode返回文件存储的DataNode的信息，客户端读取文件信息。



图3.1 HDFS总体结构示意图

Fig. 3.1 Diagrammatic sketch of HDFS

### 3.1.2 MapReduce分布式计算框架

MapReduce[46-51]是Google提出的一种软件编程模型，具有分布可靠的特点。该模型使用计算机集群并行的处理海量数据，其基本思想就是将并行计算简化为Map和Reduce两个阶段。编程人员就可以只关注Map阶段函数和Reduce阶段函数的编写，并且能够很容易的将自己的程序运行在分布式系统上而不必考虑在并行编程中的诸如分布式存储、工作调度、负载均衡、容错处理、网络通信等复杂问题。MapReduce模型是基于<Key,Value>键值对进行运作的，输入的数据和生成的结果都是一批键值对。而后系统会自动的通过大规模的集群并行执行这些计算，系统会控制机器故障，安排内部机器的相互协作来有效的利用网络和硬盘。MapReduce计算过程的数据处理函数如下表示：Map(*k*1,*v*1) → list(*k*2,*v*2)；Reduce(*k*2,list(*v*2)) → list(*k*3,*v*3)。

MapReduce计算是从Map函数开始处理输入数据*v*1，然后产生键值对集合list(*k*2,*v*2)。Reduce函数接收并处理Map阶段形成的中间结果键值对集合，最后形成结果键值对list(*k*3,*v*3)，这个过程就是图3.2中MapReduce处理流程的简要概括。

用户可以只需要把他们算法的逻辑转换成Map处理函数和Reduce处理函数。然后，由MapReduce框架来自动执行Map函数和Reduce函数。这样以来，复杂的并行编程被简化为只对Map函数和Reduce函数的编程。非专业的程序员需要做的仅仅是简单指定Map和Reduce函数的功能，就可以很容易地部署应用程序，而任务调度，数据分布，负载均衡和容错处理不需要用户干预，都由MapReduce框架自动处理。

图3.2中详细说明了MapReduce计算框架的任务执行过程：



图3.2 MapReduce原理图

Fig. 3.2 Schematic Diagram of MapReduce

1. 用户调用MapReduce库函数将存储在分布式文件系统（Hadoop Distributed File System，HDFS）[52]，输入文件切割成多个文件分片，每块分片的大小一般是16M到64M。然后MapReduce启动复制操作，将用户编写的Map和Reduce函数复制到各个集群节点上。
2. 在这些分派的任务中，主控程序Master负责程序的调度与监控，是全局的统治者。其它剩下的执行程序都是作为Master分派的工作机。Master选择空闲的工作机来分配任务。
3. 被指派了Map任务的工作机，首先预处理输入的数据获得键值对<Key,Value>，然后将获得的键值对<Key,Value>传递给用户定义的Map函数，最后Map函数处理键值对产生的新的中间结果键值对<Key,Value>暂时缓存到内存。
4. 第三步缓存到内存的键值对<Key,Value>会周期性的自动写到本地磁盘，这些数据通过Partition函数分成R个区。并且需要将键值对的存储地址定时发送给主机程序Master，接着主机Master负责把这些存储地址信息传送给Reduce主机。
5. 当Master通知被指派了Reduce任务的工作机关于中间键值对的存储地址时，它调用远程程序从Map工作机的本地磁盘上读取的键值对信息。当Reduce工作机读取到所有中间数据后，它就使用键值对的键值进行排序，把相同键值的信息汇总到一起。
6. Reduce工作机根据键值来遍历所有的排序后的键值对信息，并且把键值和对应的中间结果值集合发送给用户自定义的Reduce函数，Reduce函数处理接收到键值对信息最终组合成一个输出文件，以便整体输出任务结果。
7. 当所有Map和Reduce任务完成的时候，Master主机将调用用户程序，激活程序，调用Reduce工作机返回用户程序的调用位置。

通过整个程序的运行过程，我们可以看出用户程序仅仅需要编写Map函数和Reduce函数即可，MapReduce库首先通过调用用户自定义的Map函数，将输入文件分割，如果数据处理完成，将调用Reduce函数将结果合并起来。

MapReduce的特性主要有以下三个方面：

1. 能用于处理大规模数据，而且能将很多繁琐的细节隐藏起来，比如，自动并行化、负载均衡等，这样将极大地简化编程人员的开发工作；
2. 伸缩性非常好，每增加一台服务器，其就能将差不多的计算能力接入到集群中，而过去的大多数分布式处理框架，在伸缩性方面都与MapReduce相差甚远；
3. MapReduce能够满足海量数据处理需求，且Map节点之间不能相互通信。

### 3.1.3 ComMapReduce分布式计算框架

MapReduce作为处理大规模数据的并行编程框架，Mapper和Reducer的执行过程彼此相互独立，无论是在Mapper之间还是Reducer之间都是没有通信的。当最终结果远少于原始数据时，处理那些无用的中间数据非常浪费时间。文献[53]提出的基于传统的MapReduce框架的一个新的具有简单的轻量级通信功能的MapReduce框架，称为ComMapReduce。ComMapReduce框架通过增加轻量级的通信机制产生共享信息，继而有效地过滤Map阶段的无用数据，从而减少Reduce阶段的输入数据量。在不影响MapReduce框架原有的特性的前提下，ComMapReduce框架大幅度地提升了传统MapReduce框架的处理性能。总之，ComMapReduce是可以通过全局共享信息来过滤无用数据的并行编程框架。

图3.3是ComMapReduce框架原理图。在ComMapReduce框架中，主控节点和从节点与原MapReduce扮演相同的角色，所以主控节点在图3.3中不显示了。正如前面所讨论的，ComMapReduce的简单轻量级通信机制可以获得共享信息，缩短大数据应用的时间延迟。为了避免在主控节点上施加额外的压力，该框架新部署一个协调者节点（Coordinator）在ComMapReduce中用于生成共享信息。协调者节点可以与Mapper和Reducer进行轻量级通信，还可以接收和存储一些临时变量，然后生成一个消息包和共享信息的应用程序。共享信息可以是任何格式的消息包，比如数据值、数组、表格等。每个Mapper从协调器节点获得共享信息，然后过滤一些无用的中间数据，这样Mapper节点的输出和Reducer节点的输入量都显著降低。



图3.3 ComMapReduce原理图

Fig. 3.3 Schematic diagram of ComMapReduce

## 3.2 问题定义和问题分析

在XML数据查询中经常出现多个相关的结构连接，形成一种带分支的树型查询模式，这种XML查询的模式被称为小枝（Twig）模式[11]，这种查询称为小枝模式查询，又被称作树模式查询（Tree Pattern Query，TPQ）。Twig查询模式是XML查询处理中一个新兴的重要的查询模式，是对XML文档中的多个节点作选择谓词运算，进而找出所有满足树状结构的查询模式的匹配。小枝模式查询上的每一个节点都有对应的谓词，节点之间满足一定的结构关系（双亲/孩子关系和祖先/后代关系）。

**定义2.2 Twig查询** 给定一个Twig查询模式Q和XML文档D。在文档D上匹配查询Q，即找到一个从Q中查询节点到D中数据节点的映射h，并且该映射h满足以下两个条件：

1. 对查询Q中的任意一个节点*u*，文档D中的节点h(*u*)满足节点*u*的谓词约束；
2. 对查询Q中任意两个节点*u*和*v*，文档D中的节点h(*u*)和h(*v*)的结构关系满足节点*u*和*v*的结构关系。

针对海量XML文档的分布式Twig查询处理问题，单机环境因有限的内存、硬盘存储容量和CPU功耗等限制条件，无法满足大规模数据的处理，因此将海量XML文档进行分片并分布式存储是必不可少的处理过程。由于每台机器中只存储了XML文档的一部分，所以在处理Twig查询时保证查询结果的完整性和正确性就变得尤为重要。在MapReduce计算模型中设计处理海量XML数据的分布式Twig查询算法需要重点考虑以下三个方面：

1. 数据存储形式。MapReduce要求原始数据允许被任意分割为大小相等的分片，并且分布式存储到多台机器上。由于XML文档具有结构信息，在对XML文档分片时需要记录被切断的结果信息，并且在Twig算法设计时应尽量减少由于局部结构信息不完整性导致的额外传输代价和计算代价等。
2. 数据表示模式。MapReduce要求输入输出以及中间结果的传输形式为键值对<Key,Value>。由于Twig算法需要将XML文档中的节点进行编码，可以考虑使用节点的编码和标签等相关信息设计键值对。
3. 计算模式。MapReduce框架要求Mapper之间以及Reducer之间没有通信。由于XML文档具有结构信息，在判断节点匹配是否参与最终结果合成时，需要考虑与其他分片中的临时结果间的结构关系。如何在MapReduce框架内确保查询结果的完整性和正确性，是设计基于MapReduce的分布式Twig查询算法的最大挑战。

现有的分布式Twig查询处理算法除了需要对Twig查询模式进行分解，还需要在Map阶段进行结构连接操作，进而产生大量无用的中间结果。对这些无用中间结果的处理就会消耗很多额外的时间，从而降低了查询效率。目前，单机上的Twig整体匹配算法已能达到线性最优，如果能在MapReduce环境下执行单机上Twig整体匹配算法，无疑能够提高海量XML数据查询处理效率。基于上述分析，本文将采用了节点分发的新思路，即在Map阶段不进行结构连接操作，而是将处于不同文档分片上，但又可能属于同一个查询解的节点分发到一台计算节点上，这样在Reduce阶段就可以并行且高效地执行整体小枝连接查询处理。基于该思路的查询处理算法，在Reduce阶段可以执行任意的整体匹配算法，比如根据查询模式的结构关系，选择祖先后代最优的整体匹配算法或父子关系最优的整体匹配算法等。通过前面对MapReduce工作原理的分析，我们知道，处理带结构信息的XML数据的分布式Twig查询时，可能会因为不能获取其他分片上的结构信息而丢失查询解。而根据本章3.1.3小节对ComMapReduce框架的介绍，该框架最大的优势是引入了一个协调者节点，通过轻量级的通信机制可以获取全局共享信息，因此我们可通过协调者节点来获取其他分片上的结构信息，保证查询解的完整性。本章将基于ComMapReduce，利用节点分发思想对海量XML数据的分布式Twig查询处理展开讨论。

## 3.3 基于ComMapReduce的分布式Twig查询处理算法

本章提出一个基于ComMapReduce框架的XML文档的分布式Twig查询处理算法。根据本章3.2小节的对分布式Twig查询问题的讨论和分析，我们知道通过Coordinator节点可以获取其他分片上的结构信息保证查询解的完整性，因此本章提出的分布式Twig查询算法不需要考虑借助额外的空间来存储维持文档结构的相关信息，只需要重点考虑在Map阶段如何设置每个分片节点的键值对，使得能合成查询解的节点分发到同一个Reduce节点进行查询处理。

### 3.3.1 节点分发处理策略

根据3.2小节的问题分析，基于ComMapReduce的分布式Twig查询处理，各机器节点可通过协调者节点获得分片间的共享信息，因此，本节将不需考虑在XML任意分片过程对XML文档流的预处理。我们首要考虑的是共享信息的存储结构及设计一种合理的键值，以便保证将处于不同文档分片上的能合成查询解的节点分发到一台计算节点上，这样在Reduce阶段就可以并行地执行整体小枝连接算法。

海量XML数据按照分布式文件系统的默认规则分割成大小均等的片段，然后存储到集群中。如图3.4(a)所示，分片上的每个节点由二元组<lable,regionCode>表示，第一个参数是节点的标签名，第二个参数是区间编码，表示了该节点在原文档中的位置信息。为了表示方便，我们把与查询Q根节点所匹配的元素节点记为nq。例如，图3.4(b)中查询Q的根节点是a，则文档树T中对应的元素节点(a,(1,2:8,2))就可以称为nq节点。

(a)XML文档树T (b)Twig查询Q

图3.4 XML文档树和Twig查询

Fig. 3.4 XML document tree and Twig query

在节点分发阶段，我们的策略是：1）采用查询根节点的匹配节点nq的区间编码作为最终要发送的Key值，因为查询根节点是查询Q中的最高祖先，如果节点nq的所有后裔节点的Key值和节点nq的Key值保持一致，那么节点nq和其后裔节点就可以被发送到一台机器上进行查询；2）而且节点的区间编码可以判断两个节点的祖先后代关系，所以我们可以用节点的区间编码作为每个节点的Key值。为了减少通信量，我们取区间编码中的(start:end)部分即可；3）每个节点最终要发送的Key值是由全局键值确定，因为协调者节点可以获取全局信息，所以我们可以将所有分片上的查询根节点的匹配发送给协调者节点，这样每个分片节点就可以找到它在其他分片上的nq祖先。

基于节点分发处理策略生成最终键值的整体过程是：1）首先各个Map节点扫描分片节点，将分片中的nq节点的区间编码发送到Coordinator节点；2）Coordinator节点把收集到的所有nq节点的区间编码经过简单的逻辑处理存储到一个数组中形成全局键值KeyList，然后再把全局键值发送给所有的Map节点；3）Mapper接收到全局键值KeyList后，对每个分片节点设置最终要发送的Key值。如果当前分片节点是查询根节点的匹配，即nq节点，就取该节点的区间编码作为最终发送的Key值；如果是非nq节点，则扫描全局键值中是否有该节点的祖先，若有，则Key值设为祖先节点的区间编码，如果有多个祖先，则设置多个Key值，每个Key值为相应的祖先的区间编码。当所有节点的最终Key都设置完，这样就可以把具有相同祖先的节点发送到同一个Reduce任务进行处理。

由于Coordinator节点要收集并处理完所有的Map节点发送来的nq节点的区间编码后，才能获得全局键值KeyList，然后再发给每一个Map节点。因此，我们利用了ComMapReduce框架中的延迟通信策略（Lazy Communication Strategy，LCS），即Coordinator节点在所有Map任务结束后产生全局共享信息。

算法3.1是生成全局键值的伪代码。

|  |
| --- |
| **算法3.1** GGShareInfo算法  **Algorithm 3.1** GGShareInfo Algorithm |
| 1.**function** GGShareInfo ()  2. allRootCodes**=**getAllRootCodesFromMapper();  3. sendAllRootCodesToMapper(allRootCodes);  4.**end function**  5.**function** getAllRootCodesFromMapper ()  6.**while** true **do**  **7.** rootCode **=** getRootCodeFromMapper();  8. arrayCode*.*add(rootCode);  9. **end while**  10.**end function** |

前面我们给出了Coordinator节点收集和生成共享信息的算法GGShareInfo如算法3.1所示。在算法3.1中，Coordinator节点接收各个Map发来的nq节点的区间编码，并存入arrayCode数组。当接收完所有Map发来的nq节点的区间编码后，得到共享数组arrayCode，最后发送给每一个Map计算节点。

下面举例说明节点分发处理中节点Key值的设定过程。

**例3.1** 图3.4中的XML文档树被转化成文档流，然后把XML文档流分成大小相同的四个分片，如图3.5所示的分片F1，F2，F3和F4。

(1)分片F1 (2)分片F2

(3)分片F3 (4)分片F4

图3.5 文档分片

Fig. 3.5 Fragments of XML document

每个Map节点处理一个文档分片，我们假设分片F1由Mapper1处理，以此类推。Mapper1会根据查询Q，将分片F1中的无关节点(f,(1,1:23,1))过滤掉。然后扫描分片节点，查找其中的nq节点，即标签为a的节点，比如节点(a,(1,2:8,2))，所以将节点区间编码中(2:8)发送给Coordinator节点。接着处理节点(b,(1,3:6,3))，因为该节点不是nq节点，所以不需要发送给Coordinator节点。再接着处理节点(c,(1,4:4,4))，因其也不是nq节点，也不发送给Coordinator节点。其他分片节点的处理过程和分片F1的相同。Coordinator节点通过GGShareInfo算法接收完Map传递的所有信息后，最后得到一个所有Mapper共享的区间编码集KeyList=[(2:8),(9:14),(10:13),(16:21)]。Mapper通过这个共享的区间编码集合，对分片中的每个非nq节点修改Key值。每个节点都表示为一个<Key,Value>键值对，其中Key表示nq节点对应的区间编码，Value表示分片节点的标签和区间编码。对于分片F1中的非nq节点(b,(1,3:6,3))，在KeyList中存在其祖先节点编码(2:8)，所以节点b的Key设为(2:8)，这个键值就是最终要发往Reduce的Key，其他节点的处理过程相同。

最后得到分片F1各节点的键值对为：<(2:8),(a,(1,2:8,2))>、<(2:8),(b,(1,3:6,3))>、<(2:8),(c,(1,4:4,4))>；分片F2中各节点的键值对为：<(2:8),(d,(1,5:5,4))>、<(2:8),(c,(1,7:7,3))>、<(9:14),(a,(1,9:14,2))>、<(10:13),(a,(1,10:13,3))>；分片F3中各节点的键值对为：<(9:14),(b,(1,11:11,4))>、<(10:13),(b,(1,11:11,4))>、<(9:14),(c,(1,12:12,4))>、<(10:13),(c,(1,12:12,4))>、<(16:21),(a,(1,16:21,3))>，其中b、c节点在全局键值中有两个祖先，所以它们都有两个键值对。分片F4中各节点的键值对为：<(16:21),(b,(1,17:20,4))>、<(16:21),(c,(1,18:18,5))>、<(16:21),(d,(1,19:19,5))>。然后把Key值相等的节点发送到同一个Reduce进行Twig查询处理，即把Key值分别是(2:8)、(9:14)、(10:13)和(16:21)的节点分别发送到四个Reduce进行并行处理。

### 3.3.2 基于ComMapReduce的NDTH算法

基于节点分发处理策略的思想，本文提出了NDTH分布式Twig查询处理算法。该算法的基本思想是：1）在Map阶段通过Coordinator节点收集所有分片中nq节点的键值，得到所有分片的全局键值；2）各个Map根据全局键值，就可以查找到每个分片节点的祖先信息，设置节点的最终Key，这样就能把可能构成最终匹配结果的节点发送到同一个Reduce上；3）Reduce接收具有相同Key的节点后，执行性能最优的整体匹配算法，如祖先后代关系最优的TwigStack算法，以产生最终的查询匹配结果。

算法3.2中海量XML文档D作为算法的输入数据，文档D分片前，首先要把输入文档转化成文档流XmlDocumentStream，且文档流中节点是二元组结构，ComMapReduce框架自动的把XML文档流分成*n*个大小相同的分片fragment，然后把这些分片分别发送个不同的Map进行下一步处理。算法3.1的Map函数2~3行，表示Map阶段把接收到的XML文档分片转换成Twig查询q对应的节点流集合。接下来要处理的问题就是如何把Map阶段产生的节点流集合中具有相同查询根节点的设置成相同的键值，以便于把这些可能产生最终结果的节点发送的同一个Reduce上进行Twig查询处理。算法的4~14行就是对上述问题的处理，第4~6行是获取查询根节点对应的节点流，并且把该根节点流发送到Coordinator节点。

Coordinator节点收集所有的Map上发送过来的查询根节点对应的节点的区间编码，经过简单的处理，生成一个查询根节点对应的所有节点的区间编码的数组。所有的Map通过第6行从Coordinator节点获取区间编码数组，根据这个区间编码数组每个Map去设定查询节点对应的所有节点的键值，每个节点的键值都是这个Map的流元素的在从协调者节点返回的数据中的祖先节点的区间编码，在返回的数据中没有祖先节点流节点，一定不能是最终结果匹配的一部分，所以会被舍弃。然后Map阶段会把处理后的数据根据键值不同发送到不同的Reduce上，把具有相同键值的节点发送到同一个Reduce上。这些操作都是在7~14行去完成的。在Reduce阶段，对接收到的节点进行一些简单处理，然后执行整体匹配算法。这里没有固定使用某种算法，当得到查询相关的节点集以后，是可以采用任意的整体匹配算法，并且根据不同的查询模式，可以选择适合该查询模式且性能最优的整体匹配算法，比如选用对祖先后代关系性能最优的TwigStack整体匹配算法。

算法3.2给出了基于ComMapReduce的分布式Twig查询处理算法的伪代码。

|  |
| --- |
| **算法3.2**分布式NDTH算法  **Algorithm 3.2** Distributed NDTH Algorithm |
| 输入：查询Q和分片文档F  输出：查询结果FQ  1.**function** Map(key,fragment)  2. q=DistributedCache.get(“Q”);  3. qStreams=getLabelStream(fragment,q);  4. qStreamOfQueryRoot=getQueryRootStream (qStreams);  5. **if** qStreamOfQueryRoot!=null **then**  6. sendKeysToCoordinator(qStreamOfQueryRoot);  7. keyArray=getKeysFromCoordinator();  8. refineKeys(results,keyArray);  9. **for** *i*=1 to results.size() **do**  10. emit(results[*i*].key,results[*i*].value);  11. **end for**  12. **else then**  13. sendKeyToCoordinator(null)*;*  14. **end if**  15. **end function**  16. **function** Reduce(key,list[value])  17. resultsFinal= HTwigStack (list[value]);  18. **for** *i*=1 *to* resultsFinal.size() **do**  19. emit (key, resultsFinal[*i*]);  20. **end for**  21. **end function** |

图3.6是NDTH算法的执行过程，展示了XML分片在Map阶段和Reduce阶段的执行过程。图3.2中的四个分片分别被发送到四个不同的Map节点上。各个计算节点在Map阶段首先扫描分片节点，将nq节点的Key发送给Coordinator节点。如图3.6所示，第一个Map发送(2:8)，第二个Map发送(9:14)和(10:13)，最后一个Map发送NULL。最后Coordinator节点将收集到的全局Key发送给各个Map节点，各个Map节点根据全局Key设置分片节点的最终发送Key。如图3.6所示，第一个Reduce接收所有Key为(2:8)的节点，第二个Reduce接收Key为(16:21)的节点，最后一个Reduce接收Key为(9:14)的节点，然后各个Reuduce并行执行整体结构连接算法，最后在第一个Reduce上产生一个查询结果{(a,(1,2:8,2))、(b,(1,3:6,3))、(c,(1,4:4,4))、(d,(1,5:5,4))}，第二个Reduce没有产生查询结果，最后一个得到一个查询结果{(a,(1,16:21,3))、(b,(1,17:20,4))、(c,(1,18:18,5))、(d,(1,19:19,5))}。所有Reduce上的解合起来就是最终的查询结果。



图3.6 NDTH算法执行流程

Fig. 3.6 Processing flow of NDTH algorithm

## 3.4 本章小结

本章重点研究了基于ComMapReduce的分布式Twig查询处理算法。首先，本章在3.1节中介绍了分布式文件系统、MapReduce框架等云计算技术，并介绍了ComMapReduce框架的特点。其次，在3.2节对分布式Twig查询处理所面临的问题做了详细的分析和总结，并介绍了节点分发的思想。最后，本章在3.3小节介绍了使用节点分发处理策略生成最终Key的过程，基于节点分发处理策略提出了NDTH算法，并结合实例对该算法进行了说明。

# 第4章 基于松弛分片的DTH算法

在第3章的算法中，我们采用了ComMapReduce计算框架实现分布式Twig整体匹配查询，该框架引入了Coordinator节点获取共享信息，在使分布式计算变得容易的同时，也可能因大量的通信代价和时间代价造成ComMapReduce并行计算的瓶颈。针对上述问题，本章将在节点分发思想的基础上，讨论采用传统MapReduce计算框架实现海量XML数据中的Twig整体匹配查询问题。由于MapReduce中Map计算节点互不通信的工作机制，因此我们重新考虑了XML文档分片问题，并讨论了现有研究中XML分片技术所带来的问题，从而提出了松弛分片策略。在松弛分片的基础上，我们又提出了DTH算法，实现对海量XML数据的分布式Twig整体匹配查询。

## 4.1 ComMapReduce框架的瓶颈分析

NDTH算法中是让每个Map计算节点，将分片中与查询根节点匹配的节点的区间编码信息作为键值发送给协调者节点，然后协调者节点收集完所有键值时，将这些键值作为全局信息再发送给各个Map计算节点，各个Map计算节点根据收到的全局信息设定每个元素节点的键值对，最后将相关节点再发送到Reduce计算节点进行查询匹配，得到正确完整的查询结果。虽然该算法利用协调者节点，能使查询结果的完整性得以保证，并使分布式查询计算变得简单，但是引入协调者节点后可能产生以下问题：1）当各个Map计算节点返回大量的信息给协调者节点时，可能会造成大量的网络通信代价；2）为了保证查询结果的完整性，协调者节点需要收集到所有Map计算节点发送来的数据，而协调者节点是采用阻塞方式进行通信的，当Map计算节点特别多时，协调者节点需要等待所有键值的到来，这个过程将需要花费很长时间。所以大量的通信代价和时间代价都可能造成ComMapReduce并行计算的瓶颈。因此为了避免上述问题，本章我们不依赖于协调者节点，而考虑通过一般的MapReduce框架来实现分布式Twig查询。

对于海量数据的处理，分布式存储是并行计算的基础，查询处理应遵循将计算移动到数据的原则，而不是数据移动到查询。因此，研究和实现基于MapReduce的海量XML数据的分布式查询处理方法需要重点考虑的是数据划分、数据分配、分布式查询处理，其中首要任务就是将数据分片。由于XML数据是具有结构信息的，当一个完整的XML文档被分割成若干子文档片段时，原文档的部分结构信息就会被破坏，而且计算节点在处理各自的文档分片时，由于各个Map计算节点之间是相互不通信的，所以，如果直接执行单机查询算法就可能造成查询结果的丢失。因此需要在文档分片阶段保留一些辅助信息，这些辅助信息能把与查询根节点的匹配相关的节点能汇聚到一起，保证查询处理阶段不会丢失查询解。

现有的基于MapReduce计算模型的XML分片策略需要保存原始文档根到叶节点的完整结构，这种条件限制了文档分割的灵活性。如在文献[41]提出的一种横向分片（Horizontal Fragmentation）策略，该分片策略的具有以下特性：首先将XML文档按照横向分片的方式得到一个分片集合T如图4.1中的F1，F2。集合T中的每个分片具有如下性质：1）对于每个分片F上的任意节点n(F)，在原始文档上都能找到一个节点n与n(F)有相同的到达路径；2）对于原始文档的每个节点n，能找到某个分片F中存在一个节点n(F)与n有相同的到达路径；3）集合T中的每个分片F至少包含一个XML文档树上的叶子节点，而且这个叶子节点不属于其他分片。如图4.1所示，分片F1和F2就是横向分片的结果。F1和F2是在XML文档树上的节点m7分割得到的，并且都保留了从文档树*D*的根节点到m7的相同路径。



图4.1 XML文档树和文档分片

Fig. 4.1 XML Document Tree and XML Fragmentation

虽然这种分片技术通过保存各个分片共有的路径，维持了文档树*D*的结构关系，但是Hadoop平台在分割处理时，是要求每个分片的块大小（block size）相同，而横向分割就无法在保存共有路径的前提下，确保每个分片的大小相同。此外，某些情况下，这种分片技术得到的分片规模仍然很庞大，必须继续分割，否则导致查询处理不能正常执行。再者，这种分片技术还要求分解Twig查询模式，这将会造成查询的中间结果增多，从而降低查询效率。所以，在文档分片时，如果能根据用户任意指定的块大小对原始文档进行分割，将具有更大的灵活性，并能够解决横向分割带来的问题。本章提出的松弛分片策略（Relax Fragment Strategy，RFS），就能够灵活且不依赖于查询模式对XML文档进行任意分割，满足更多的查询处理需求，具有广泛的适用性。

## 4.2 松弛分片策略

XML数据不同于传统数据，XML数据本身是具有树形结构特点的，数据分片的目的是将一个大的完整的XML文档树进行分解，并将分解后的每个文档分片单独存储，每个分片将对应一次Map任务，多个Map任务之间并行处理。为了保证查询阶段不丢失解，需要在文档分片时保留一些辅助信息，保证把能合成查询结果的相关元素节点能发送到一起，通过4.1节的分析，我们了解到对文件进行任意分割具有更好灵活性和更广泛的适用性。本节将介绍可以任意分割XML文档的松弛分片策略（Relax Fragment Strategy，RFS）和松弛分片算法Relax-Fragment。

### 4.2.1 松弛分片索引

本节将要介绍本章比较重要的松弛分片策略和松弛分片索引。本节涉及到一些重要概念，在介绍分片策略之前，先给出这些概念的定义。

**定义4.1 分片祖先标签** 给定一个XML文档T，对该文档进行分割得到若干分片，记其中一个分片为F，文档分片F缺失了在原始XML文档T中的祖先节点，那么这些缺失的祖先节点的标签序列就是分片祖先标签，记为ancLabel。

**定义4.2 全局标签序号** 在整个XML文档中，按照深度遍历访问每个元素节点时，某个元素标签在全局节点流中出现的序号就是全局标签序号，记为reCount。

**定义4.3 分片子树编号** 一个XML文档分片F可能由一个或者多个XML子树组成，如果我们按照深度遍历的顺序，给分片F中每个子树*t*从1到*n*编号，那么每个子树*ti*上的所有节点有相同的编号*i*，这个编号就是分片子树编号，记为sid。

**定义4.4 分片子树区间** 给定一个分片F中的子树集*ST*={*t*1,…,*tn*}和该分片的一个祖先节点*m*，若*m*是*ST*中{*ti*,…,*tj*}的祖先，则节点*m*的分片子树区间sidRegion为sidRegion=(sidBegin:sidEnd)。其中sidBegin是*ti*的sid值，sidEnd是*tj*的sid值，1≤*i*≤*j*≤*n*，*n*≥1。

**例4.1** 如图4.2，XML文档树T按照虚线将整个文档分割为四个XML文档分片，每个分片都丢失了与原文档的部分结构信息。由于分片F1没有缺失在原始文档中的祖先节点，所以分片F1的分片祖先标签为空。分片F2在原文档T中缺失的祖先节点的标签是f、a、b，所以其对应的分片祖先标签ancLable是f、a、b。分片F3缺失的祖先节点标签为f、a，所以对应的分片祖先标签ancLable为f、a。同理分片F4的分片祖先标签是f、d、a。

在图4.2(a)中每个节点元素左侧的数字是对应的全局标签序号，右侧是该节点的区间编码，记录了该元素节点在原XML文档中的位置信息。从图4.2中可以看到按照深度遍历访问XML文档的元素节点，标签f的全局标签序号为1；标签a和c的全局标签序号都为1、2、3、4；标签b和d的全局标签序号都为1、2、3。

在文档分片中，分片F1和F4都只有一个子树，所以该子树上所有元素节点的sid值都为1；分片F2由三个子树组成，三个子树的编号分别为1、2、3，所以对应子树上的元素节点的sid值也分别为1、2、3。分片F3和分片F2处理相同，b节点的sid值为1，c节点的sid值为2，d、a节点的sid值都为3。



(a)XML文档树T

(b)分片F1 (c)分片F2

(d)分片F3 (e)分片F4

图4.2 XML文档树和文档分片

Fig. 4.2 XML document tree and XML fragmentation

图4.2中的分片F2的子树为ST={t1,t2,t3}，节点(a,(1,2:8,2),1,1)是子树{t1,t2}的祖先，所以该节点的分片子树区间为(1:2)，同理节点(f,(1,1:23,1),1,1)是子树{t1,t2,t3}的祖先，节点f的分片子树区间为(1:3)。

查询结果的完整性和正确性是分布式Twig查询处理必须要保证的，MapReduce计算框架作为大数据处理工具，主要采用流式处理方案，并且均等地将海量数据切割成许多小InputSplit，然而XML作为一种半结构化数据，是不能像传统数据一样直接分割，否则可能造成查询结果的不完整。为了实现对XML文档的任意分割，并且能保存文档分片间的结构关系，我们提出了松弛分片策略RFS，该策略的基本思想如下：1）在扫描并分割XML文档流的过程中，记录分片F中每个节点m的分片子树编号*i*；2）当得到一个分片F时，查找该分片F的祖先节点，对每个祖先节点n，建立该祖先节点与分片子树编号的映射关系；3）如果节点n是分片F中{ti,..,tj}子树的祖先，则记录下节点n的分片子树区间sidRegion，同时建立该节点标签ancLabel与其所有子树区间的映射关系的辅助信息，通过祖先节点标签就能获取该节点与各分片子树间的结构信息。

松弛分片是在任意分割XML文档的过程中，通过保留一些辅助信息，使每一个查询根节点及其后裔节点的匹配能汇聚到一起，保证查询处理阶段不会丢失查询解，这个辅助信息就是松弛分片索引，简称RF索引。松弛分片索引是由分片祖先节点的标签ancLabel和由祖先标签的全局序号与分片子树编号区间组成的若干二元组共同构成的键值对<ancLabel,(reCount,sidRegion)>，其中祖先节点的分片子树区间sidRegion是由其所有后裔节点中最小子树编号与最大子树编号构成的区间。

XML文档树的分割实际是按照文档流的方式来分割的，图4.3(a)表示的是图4.2中的XML文档树T在文件中对应的文档流，该文档被分成了四个分片。XML文档流节点由一个四元组<label,reginCode,reCount,sid>表示，第一个参数label是元素节点标签，第二个参数reginCode是对应元素节点的区间编码，第三个参数reCount是元素节点的全局标签序号。第四个参数sid是分片中的子树编号。

根据前面介绍的松弛分片策略，在文档分割阶段，对每个流分片建立一个松弛分片索引，然后随XML文档分片一起存储到计算集群中，图4.3(b)、(c)、(d)、(e)分别是XML文档的四个分片及其分片索引。下面举例详细说明松弛分片索引的建立过程。

**例4.2** 在图4.3(a)的XML文档流中，从节点c开始扫描文档，直到扫描到全局标签序号为1的元素节点c时，该分片大小足够64MB，进行一次分割得到分片F1。由于分片F1只包含一个XML子树，所以该分片上所有节点的sid都为1。因为该分片在原始文档中没有祖先节点，所以它的分片索引为空。继续扫描文档，直到扫描到全局标签序号为3的元素a时分割一次，得到分片F2，该分片包括了三个子树，各个子树上的节点sid值分别为1、2、3。分片F2缺失了在原文档中的祖先节点(f,(1,1:23,1),1,1)、(a,(1,2:8,2),1,1)、(b,(1,3:6,3),1,1)，所以分片F1对应的分片索引1中有三条记录。其中节点(f,(1,1:23,1),1,1)的reCount值为1，它是分片F2中d、c、a节点的祖先，它们的sid值分别为1、2、3，所以分片子树区间中最小的子树编号为sidBegin=1，最大的子树编号sidEnd=3，因此得到松弛分片索引1中的第一条记录为<f,[(1,1:3)]>；对于祖先节点(a,(1,2:8,2),1,1)，它是分片F2中d、c节点的祖先，所以分片子树区间中最小的sid值sidBegin=1，最大sid值sidEnd=2，所以得到RF索引1中的第二条记录<a,[(1,1:3)]>；第三条记录为<b,[(1,1:1)]>。分片F3的分片祖先有三个节点(f,(1,1:23,1),1,1)、(a,(1,9:14,2),2,3)、(a,(1,10:13,3),3,3)，节点的标签为f、a，所以分片索引2中有两条记录。其中(f,(1,1:23,1),1,1)是b、c、d的共同祖先，b、c、d节点的sid值分别为1、2、3，从而分片子树区间为1:3，f标签对应的Value为(1,1:3)；祖先节点(a,(1,9:14,2),2,3)和(a,(1,10:13,3),3,3)都是b、c的祖先，所以有sidBegin=1，sidEnd=2，故a标签对应的Value为(2,1:2)和(3,1:2)，所以得分片F3的分片索引为{<f,[(1,1:3)]>,<a,[(2,1:2)(3,1:2)]>}。同理，分片F4的分片索引为{<f,[(1,1:1)]>,<d,[(2,1:1)]>,<a,[(4,1:1)]>}。



(a)XML文档流

(b)分片F1和RF索引1 (c)分片F2和RF索引2

(d)分片F3和RF索引3 (e)分片F4和RF索引4

图4.3 XML文档流、各分片及其RF索引

Fig. 4.3 XML documents stream, XML fragments and RF index

### 4.2.2 Relax-Fragment算法

XML数据分片处理是云环境下处理分布式Twig查询的一个重要环节，需要考虑如何在节点分发阶段可以将分片节点和其对应的祖先节点汇聚到一起，以保证查询结果的正确性和完整性。针对该问题，本章在4.2.1小节已经介绍了松弛分片策略生成XML文档分片和分片索引的过程。本节给出了该分片策略的实现代码。

算法4.2中，海量XML数据T作为算法的输入，将文档转换成输入文档流，然后进行流式读取，对于每个流节点，调用函数setSubTreeId()设置其分片子树编号sid，然后将扫描到的节点都存入分片fragement[0]中，直到分片大小满足block大小。在继续扫描下一个分片节点前，需要先调用函数getFragmentRFIndex()建立当前分片的RF索引。然后将建立好的RF索引存入fragement[1]中，最后再将该分片及索引信息fragment存入分片结果集result中，之后再继续处理下一个分片，直至文档流结束。算法中3至6行设置分片子树编号，8至12行是建立和输出松弛分片索引及文档分片。

算法4.2给出了松弛分片算法生成文档分片和分片标签索引的伪代码。

|  |
| --- |
| **算法 4.2** 松弛分片算法  **Algorithm 4.2** Relax-Fragment Algorithm |
| 输入：海量XML文档T  输出：XML文档分片和分片标签索引  1. **function** getFragmentAndRFIndex (T)  2. XmlDocumentStream=mSax(T);  3. sid=0;k=0;  4. sLevel = XmlDocumentStream[*i*].level;  5. **for** *i*=0 to XmlDocumentStream.size **do**  6. setSubTreeId(XmlDocumentStream[*i*], sid, sLevel, subFirstNodes);  7. fragment[0].add(XmlDocumentStream[*i*]);  8. **if** fragment[0].size >= block.size **then**  9. fragment [1]= getFragmentRFIndex (XmlDocumentStream, k, subFirstNodes);  10. fragmentAndRFIndexResults.add(fragment);  11. k = i; subFirstNodes.clear();  12. **end** **if**  13. **end** **for**  14. **return** fragmentAndRFIndexResults;  15.**end function** |

函数setSubTreeId()，实现设置元素节点的分片子树编号。其中1至6行代码是判断当前节点是否属于下一个子树的节点。如果当前节点的层次level比变量sLevel（存储当前最大子树的第一个节点的level值）小，说明该节点不是上一个子树的节点，而是下一个子树的第一个节点。所以sid增加1，并且该节点的sid等于最新的sid值，sLevel存储当前子树的第一个节点的level，并且将该节点放入subFirstNodes中。subFirstNodes用于存放所有子树的第一个节点，方便建立索引时，寻找各子树的祖先节点。第7行是将分片子树的第一个节点的sid值赋值给当前节点的子树编号，因为当前节点和其前一个节点属于同一个分片子树。

函数setSubTreeId()实现设置元素节点的分片子树编号。

|  |
| --- |
| **Function** setSubTreeId(node, sid, sLevel, subFirstNodes) |
| 1. **if** node.level <= sLevel **then**  2. sid = sid +1;  3. sLevel = node.level;  4. node.sid = sid;  5. subFirstNodes.add(node);  6. **else then**  7. node.sid = sid;  8. **end if** |

函数getFragmentRFIndex()实现建立分片索引，索引的每条记录为键值对，Key为祖先节点的标签，Value包含了多个三元组，每个三元组都记录了祖先节点的分片子树编号区间，Value中元组个数等于该祖先节点的最大全局标签序号。对于节点流中某个节点node与subFirstNodes中所有节点比较，判断是否存在节点node的后裔子树。如果节点n是subFirstNodes中某些节点的祖先，则取第一个后裔子树的sid以及最后一个后裔子树的sid（第3、4行）和节点node的全局标签序号组成三元组，并添加到Key为node.label的Value链表中。

函数getFragmentRFIndex()实现设置存储文档分片祖先信息的分片索引

|  |
| --- |
| **Function** getFragmentRFIndex(XmlDocumentStream, k, subFirstNodes) |
| 1. **for** *j*=0 to *k* **do**  2. RFKey **=** XmlDocumentStream[*j*].label;  3. RFValue.sidBegin =findStartSid(subFirstNodes);  4. RFValue.sidEnd =findEndSid(subFirstNodes);  5. RFValue.reCount = XmlDocumentStream[*j*].reCount;  6. **if** RFInfo.containsKey(RFKey) **then**  7.valueList = RFInfo.get(XmlDocumentStream[*j*].label).add(RFValue);  8. RFInfo.put(RFKey, valueList);  9. **else then**  10.RFInfo.put(RFKey, RFValue);  11. **end if**  12.**end** **for** |

## 4.3 基于松弛分片的DTH查询处理算法

在4.2小节提出的松弛分片策略和节点分发思想的基础上，本节的主要内容是研究并提出了MapReduce框架下基于松弛分片的分布式Twig查询处理算法，分布式DTH查询算法。该算法主要包括两个部分：Map阶段的节点分发处理和Reduce阶段的结构连接操作，节点分发处理是本算法的核心部分。

由于分布式计算需要将完整的XML文档分割成若干子文档分片，这就造成了原始XML文档的整体结构信息丢失。通过4.2节的分析，各个计算节点在计算本地文档分片时，相互之间没有通信，也就不能获得分片缺失的祖先节点信息，这就可能造成查询解的丢失，从而得到不完整的查询结果。因此，为了获取原文档与各分片间的结构信息，我们提出了松弛分片策略，并设计了松弛分片索引。

为了方便描述，我们把与查询Q的根节点匹配的XML元素节点称为nq节点；把Map阶段最终分发节点的Key称为Emit Key，简称EKey。

在DTH查询算法中，Map函数利用RF索引找到所有非nq节点的所有EKey，保证将所有可能是查询结果的节点分发到同一个Reduce节点上，因为每个发送到Reduce上的Ekey键值对都可能是Twig查询解的匹配节点。Mapper函数设置EKey的过程：对分片F上的每个非nq节点需要先查找分片F上的nq祖先节点，然后在RF索引表中查找nq节点。为了提高在分片F中查找祖先的效率，我们将分片F中的nq节点暂存在Ancestor数组。

归纳起来为：1）如果当前节点m属于nq节点，则该节点的EKey设置为节点m的全局标签序号，Value为二元组<m.lable,m.regionCode>，同时将节点m放入Ancestor数组。2）如果当前节点m不属于nq节点，就先在Ancestor找m的祖先节点，若找到，则将祖先节点的全局标签序号选取为该节点的EKey。然后在RF索引表中查找是否存在nq节点的记录，若存在，则查找nq节点对应的sidRegion中，是否有满足sidBegin≤m.sid≤sidEnd的元组，若有则取出这些元组的reCount作为节点m的Ekey。

值得一提的是，Ekey值的个数和的nq节点的最大全局标签序号相同，因此同一标签，不同全局标签序号的nq节点将被分发到不同的Reduce节点上。由于每个属于nq节点和它的所有后裔节点有相同的Ekey，它们将被发送同一台Reducer上进行查询匹配，所以不会丢失解，而且每个Reducer节点上产生查询解的效率也进一步提高。

在Reduce阶段，各个计算节点接收各个Map计算节点发来的节点信息，对这些节点进行一些简单处理，然后调用Reduce函数执行任意的整体匹配算法。Reduce函数中不需要固定某种查询匹配算法，因为基于节点分发思想的分布式Twig查询算法，每个Reduce计算节点上得到的是可能构成查询解的完整节点集，因此可以采用任意的整体匹配算法，并且根据不同的查询模式，可以选择适合该查询模式且性能最优的整体匹配算法，比如选用祖先后代关系性能最优或父子关系最优的整体匹配算法。

算法4.3是海量XML数据查询处理的DTH算法。

|  |
| --- |
| **算法4.3** DTH算法  **Algorithm 4.3** DTH Algorithm |
| 输入：查询Q和文档分片F及分片索引  输出：查询结果FQ  1. **function** Map(key, fragment)  2. q=DistributedCache.get(“Q”);  3. qNodes=getNodes(fragment[0],q);  4. RFIndex= fragment[1];  5. **if** not empty(qNodes) **then**  6. **for** *i*=0 to qNodes.size **do**  **7. if** qNodes[*i*] is not nq node **then**  8. findKeyInAncestorArray(qNodes[*i*], ancestorArray);  9. findKeyInRFIndex(qNodes[*i*], RFIndex);  10. **else then**  11. acestorArray.add(qNodes[*i*]);  12. setEKey(qNodes[*i*], qNodes[*i*].reCount);  13. **end if**  14. **end for**  15.  **for** *i*=1 to results.size **do**  16. emit(results[*i*].key,results[*i*].value);  17. **end for**  18. **end if**  19.**end function**  20.**function** Reduce(key, list[value])  21. resultsFinal=HTwigStack(list[value]);  22. **for** *i*=1 toresultsFinal.size() **do**  23. emit (key, resultsFinal[*i*]);  24. **end for**  25.**end function** |

在算法4.3中，Map函数首先从缓存中读取查询模式Q（第2行），从文档分片中过滤掉与查询无关的节点（第3行）。如果过滤后的节点集qNodes不为空，则调用函数findKeyInAncestorArray()和findKeyInRFIndex()完成每个分片节点的EKey设置（6至14行），然后按照EKey值发送到不同Recuce节点上（15至17行）。其中计算Ekey的主要步骤在函数findKeyInAncestorArray()和findKeyInRFIndex()在中，这两个函数分别给出了在分片F和RF索引中产生节点EKey的详细过程。其中RFValueList是个全局变量，因为索引表中ancLabel为nq标签的记录只有一条，并且一个分片中所有节点的处理，都只读取ancLabel为nq标签的记录，所以在函数findKeyInRFIndex()中，首先判断RFValueList变量是否为空，如果为空，就需要查找索引表，把ancLabel为nq标签的记录取出来（第1至3行）。否则，就遍历valueList，并判断当前节点m的分片子树编号sid是否包含在valueList中，即判断一个节点m在valueList中是否存在其祖先节点（第5行），如果存在祖先，就取包含了m.sid的元组的reCount值，并设置节点m的Ekey（第6行）。Map阶段主要处理节点分发，而不做结构连接操作。Reduce函数主要执行整体结构连接的查询操作，可以根据查询的特点，选择性能最佳的查询算法，如祖先后代最优的TwigStack算法等。

函数findKeyInAncestorArray()实现在分片F中查找节点的nq祖先并设置Ekey。

|  |
| --- |
| **Function** findKeyInAncestorArray(node, ancestorArray) |
| 1. **if** not empty (acestorArray) **then**  2. **for** *i*=0 to ancestorArray.size **do**  3. **if** ancestorArray[*i*] is ancestor of node **then**  4. setEKey(node, acestorArray[*i*].reCount);  5.  **end if**  6. **end for**  **7. end if** |

函数findKeyInRFIndex()实现在松弛分片索引中查找nq节点，并设置节点的Ekey。

|  |
| --- |
| **Function** findKeyInRFIndex(node, RFIndex) |
| 1. **if** empty(RFValueList) **then**  2. RFValueList = RFIndex.getKey(nq);  3. **end if**  3. **for** *i*=0 to RFValueList.size **do**  4. value = RFValueList[*i*];  5. **if** node.sid between value .sidBegin and value.sidEnd **then**  6. setEKey(node, value.reCount);  7. **end if**  8.**end for** |

**例4.3** 下面我们我们使用图3.4(b)的查询模式Q和图4.3的文档分片及分片索引来解释说明算法4.3。

XML文档被分为四个文档分片，每个分片F由一个Map计算节点处理，因为查询Q根节点标签为a，所以nq节点的标签也为a。假设Map1处理文档分片F1，首先依次扫描分片F1的每个节点，由于(f,(1,1:23,1),1,1)不是标签为a的节点，即不是nq节点，所以不添加到Ancestor数组中，并且此时Ancestor数组为空，即无f节点的祖先存在。然后查找RF索引表1，该索引表也为空，故f节点不设EKey，这个节点将不会被分发到任何Reduce节点上，所以也不会参与查询匹配。继续扫描分片F1的下一个节点(a,(1,2:8,2),1,1)，该节点属于nq节点，直接设置EKey，不需要扫描Ancestor数组和RF索引表，所以取a节点的reCount值设置EKey键值对，即<1,(a,(1,2:8,2))>，并且取该节点区间编码的start:end部分及reCount值组成二元组(2:8,1)存入Ancestor数组中。然后接着处理(b,(1,3:6,3),1,1)，该节点标签不为a，则先查找Ancestor数组中是否有其祖先节点，因为Ancestor数组中有(2:8,1)，是b的祖先，所以取祖先节点的reCount值1为EKey，得键值对<1,(b,(1,3:6,3))>。再查找RF索引表中有无a标签对应的记录，因为RF索引表1也为空，所以最后b节点只有一个EKey。同理，节点(c,(1,4:4,4),1,1)的处理过程与b节点一样，得一个键值对<1,(c,(1,4:4,4))>。Map2处理文档分片F2，依次扫描节点(d,(1,5:5,4),1,1)，因为该节点不是nq节点，查找Ancestor为空，继而查找RF索引2中有a标签对应的记录<a,[(1,1:2)]>，并且d节点的sid=1，所以a是d的祖先，所以取a的reCount值设置Ekey，得键值对<1,(d,(1,5:5,4))>。同理节点(c,(1,7:7,3),2,2)的键值对为<1,(c,(1,7:7,3))>，节点(a,(1,9:14,2),2,3)和(a,(1,10:13,3),3,3)都是查询根节点的匹配，直接设置键值对<2,(a,(1,9:14,2))>和<3,(a,(1,10:13,3))>，并将(9:14,2)和(10:13,3)存入Ancestor中。同理，分片F3和分片F4的处理过程和分片F1、F2类似，分片F3上b、c节点各产生两个键值对，d节点不产生键值对，a节点产生一个键值对。分片F4上b、c、d节点各产生一个键值对。

各分片节点的键值对计算完成后，将按照不同EKey分发到不同Reduce节点上。Reduce1接收<1,(a,(1,2:8,2))>、<1,(b,(1,3:6,3))>、<1,(c,(1,4:4,4))>、<1,(d,(1,5:5,4))>、<1,(c,(1,7:7,3))>，Reduce2接收<2,(a,(1,9:14,2))>、<2,(b,(1,11:11,4))>、<2,(c,(1,12:12,4))>，Reduce3接收<3,(a,(1,10:13,3))>、<3,(b,(1,11:11,4))>、<3,(c,(1,12:12,4))>，Reduce4接收<4,(a,(1,16:21,3))>、<4,(b,(1,17:20,4)>、<4,(c,(1,18:18,5))>、<4,(d,(1,19:19,5))>，然后各个Reduce节点并行执行Twig整体匹配算法。最后在Reduce1和Reduce4上得到两个查询解{(a,(1,2:8,2))、(b,(1,3:6,3))、(c,(1,4:4,4))、(d,(1,5:5,4))}和{(a,(1,16:21,3))、(b,(1,17:20,4)、(c,(1,18:18,5))、(d,(1,19:19,5))}。

## 4.4 本章小结

本章首先根据MapReduce计算框架的工作原理，分析了海量XML数据在MapReduce框架中的分割问题，然后结合节点分发思想提出了松弛分片策略，最后在松弛分片策略的基础上提出了DTH查询处理算法。本章在4.1小节分析了现有分片技术的不足，提出了松弛分片策略。本章4.2小节介绍了松弛分片策略和RF索引的建立过程。最后本章在4.3小节提出了基于松弛分片的DTH查询处理算法。

# 第5章 实验及结果分析

本文首先提出了基于ComMapReduce的NDTH算法，实现对海量XML数据的Twig查询处理。然后分析了NDTH算法中因通信代价和时间代价带来的瓶颈问题，提出了松弛分片策略，在此基础上，设计了基于松弛分片的DTH算法。这两种分布式Twig查询算法旨在利用云计算技术实现对海量XML文档的高效的查询。因此本章需要重点研究和分析的内容有以下两点：1）两种分布式Twig查询处理算法的文件上传至HDFS耗时情况；2）两种分布式Twig查询处理算法的查询执行时间对比和性能的分析。本章首先介绍了实验的评估标准和实验所用的环境以及实验数据集和具体实验方案。根据实验方案和性能评估标准，通过对比实验结果分析了本文所提出的算法的性能。

## 5.1 实验性能评估标准

本文提出的两种分布式Twig查询处理算法的性能好坏，需要通过实验结果来验证。本文的实验主要通过以下三个性能评估标准来衡量本文提出的两种分布式Twig查询处理算法的性能，它们分别是加速比（Speedup）、数据伸缩率（Sizeup）和扩展率（Scaleup），这些是设计并行算法和高性能并行机所追求的重要目标。

下面我们分别介绍加速比、数据伸缩率和扩展率的计算公式。

加速比（Speedup），是指在保持数据规模固定，不断增大计算机节点数量时，并行算法的性能。它是用并行前的执行速度和并行后的执行速度之比来表示的，反映的是在并行条件下，计算速度所能得到的加速倍数。计算机数目为*m*的Speedup计算公式为：

 (5.1)

计算公式(5.1)中：*T1*代表在一个计算节点上运行并行算法的执行时间；*Tm*代表在*m*个计算节点上运行该算法的执行时间。

数据伸缩率（Sizeup），是指在保持计算节点的数目不变，扩大数据规模，并行算法的性能。Sizeup的计算公式为：

 (5.2)

计算公式(5.2)中：*Tm\*DB*表示在*m\*DB*的规模数据上运行并行算法的执行时间；*TDB*代表*DB*数据上运行该算法的执行时间。

扩展率（Scaleup），是指成比例地增加计算节点数目与数据规模时，并行算法的性能。Scaleup的计算公式为：

 (5.3)

计算公式(5.3)中：*T1\*DB*代表使用1个计算节点在*DB*数据上运行并行算法的执行时间；*Tm\*m\*DB*代表使用*m*个计算节点在*m\*DB*的规模数据上运行该算法的执行时间。

## 5.2 实验环境及实验设计

本节首先介绍了本文实验需要使用到的实验环境以及查询数据集，然后再对实验方案进行概述。

### 5.2.1 实验环境

本次实验采用1台PC作为名称节点和服务器，8台PC作为数据节点和计算节点，具体所采用的实验环境如下：

1. CPU：Intel Quad core Q8400 2.66GHz；
2. 内存：4G Bytes；
3. 硬盘：1T Bytes，7200rpm；
4. 操作系统：CentOS 5.6；
5. 开发工具：Eclipse；
6. 开发语言：Java语言、jdk1.6.0\_45；
7. 交换机：1000MB；
8. Hadoop版本：hadoop-2.3.0-cdh5.0.1；
9. Hadoop块大小：64MB。

本文研究提出了针对海量数据的分布式Twig查询处理算法，采用的数据集大小均为GB级别。分布式NDTH算法和DTH算法的实验均在由9台机器组成的Hadoop集群上进行，由其中1台PC作为集群的名称节点（NameNode）和服务器，其余8台机器作为数据节点（DataNode）和计算节点，Hadoop的集群是基于master/slave模式，在主从结构中名称节点属于master，一般负责集群管理、任务调度和负载均衡等，而数据节点属于slave，负责并行执行来自master节点的计算任务等。这9个计算节点有相同的配置环境，每一台PC机的配置是主频为2.66GHZ的英特尔酷睿四核处理器，4GB内存，操作系统为Linux类型的CentOS 5.6，JDK的版本是1.6.0\_45，Hadoop版本为稳定2.3.0-cdh5.0.1，并且所有的计算机都通过高速的千兆网络连接。

### 5.2.2 实验数据集

为了更真实、更有效地体现本文对NDTH算法和DTH算法的实验结果，本文使用了在XML相关实验中广泛使用的两种标准数据集作为本文的实验数据。一种是DBLP[54]数据集，另一种是XMark[55]数据集。我们根据这两种数据集，从中选取了3到5个节点组合成对应的查询模式Q。数据集的大小和对应的查询路径如表5.1中所示。

表5.1 实验数据集

Table 5.1 Experimental data sets

|  |  |  |  |
| --- | --- | --- | --- |
| 数据集 | 数据集大小 | | 查询Q |
| xmark.xml | | 2GB | XQ1: /site/people/person[id=“person0”]/name |
| XQ2: //site[//regions]//location |
| dblp.xml | | 1.5GB | DQ1: /dblp/article[/title/sub]/author |
| DQ2: /dblp/article[/author/name]//ee |

如上面表5.1所示，本文选取xmark.xml和dblp.xml作为数据集，因为这两种数据集都是在进行XML实验中常用的。为测试本文的算法能够处理海量数据，同时使实验结果不失真实性，本文所选的每个数据集的大小均是GB级别。同时为了后续讨论方便，我们分别将xmark.xml数据集和dblp.xml数据集称为XMark数据集和DBLP数据集。

XMark数据集是由XMark生成工具xmlgen，设定扩展系数为18，随机生成的一个2GB的文档集。XMark工具可以指定生成文档的“扩展系数（Scalling Factor）”，生成所需文档的大小，而且XMark数据集的广度比较大，而深度不高，应用非常广泛。

DBLP数据集为真实的数据集，并根据时间有不同大小的版本。DBLP是计算机文章查询的一个很重要的集成数据库系统，它使用XML存储元数据，广泛用于XML有关的查询上，它的特点是结构简单、深度不高，但是文本信息非常丰富。

### 5.2.3 实验方案

本文采用了新颖的节点分发思想来处理海量XML数据，在文档分割方面不仅可以进行任意分割，而且文档分割独立于查询模式，另一方面还能够灵活选用Twig整体匹配查询算法。基于节点分发思想，本文提出了两种分布式Twig查询处理算法：一是基于ComMapReduce的NDTH算法；二是基于松弛分片的DTH算法。为了评估这NDTH和DTH两个算法的查询处理效率和并行性等性能，本文将从以下四个方面对实验结果进行分析：

1. 本文提出的松弛分片算法对文件上传时间影响的分析；
2. 本文提出的NDTH算法与DTH算法的加速比（Speedup）性能分析；
3. 本文提出的NDTH算法与DTH算法的数据伸缩率（Sizeup）性能分析；
4. 本文提出的NDTH算法与DTH算法的扩展率（Scaleup）性能分析。

## 5.3 实验结果分析

在本节中，通过不同的对比实验来分析NDTH算法和DTH算法的性能。因为DTH算法与Hadoop上传文件时的分割处理密切相关，所以我们针对不同大小文档，用Hadoop默认分片和任意分片算法[56]下文件上传时间与松弛分片算法下文件上传时间做了实验对比。任意分片算法我们简称为Arbitrary-Partition。此外实验还包括了用DTS[56]算法与NDTH与DTH算法做加速比性能对比、数据伸缩率性能对比、扩展率性能对比。同时我们对所有的实验结果都做了详细的分析。

### 5.3.1松弛分片算法Relax-Fragment对文件上传时间影响

本节主要是在DBLP数据集和XMark数据集上，用不同大小文档分别测试了采用Hadoop默认分割策略时的文件上传时间和采用松弛分片算法以及任意分片算法[56]这三种分片方式时的文件上传时间。我们分别按照大致20%、40%、60%、80%、100%的比例从两个数据集中分别取5个不同大小的文档作为查询数据集。因此在DBLP数据集中我们取300MB、600MB、900MB、1200MB、1500MB大小的文档，测试三种分片方式下的文件上传所耗时间，得到一组实验数据。在XMark数据集中取400MB、800MB、1200MB、1600MB、2000MB大小的文档，测试三种分片方式下的文件上传所耗时间，我们得到另一组实验数据，然后我们根据两组实验数据，绘出了如图5.1所示的两个曲线图。

图5.1(1)是两个算法在DBLP数据集上取不同大小的文档的测试结果，图5.1(2)是两个算法在XMark数据集上取不同大小的文档的测试结果。从图5.1中的两个曲线图中我们可以看到，随着文档的增大，上升三条曲线都在上升，当文档规模大于900MB时，增长幅度趋变大，而默认分片策略下曲线线性增长，任意分片算法下文件上传时间介于两者之间，但增长幅度也在变大。Hadoop默认分片下的文件上传时间最少，这是因为该方式下上传文件过程中不需要做额外的处理，而Relax-Fragment算法下文件上传过程中需要扫描大量分片上的节点获取分片间的额外的结构信息，而任意分片下的文件预处理扫描的分片节点数比松弛分片算法的要少。

我们从两个图中还可以观察到，不论文档规模的大小的变化如何，虽然Hadoop默认分割策略下的文件上传时间都比采用Relax-Fragment算法时的文件上传时间少，但是在查询处理阶段，基于该松弛分片策略的查询算法可以利用所存储的结构信息加快查询处理的速度。

(1) DBLP数据集上文档大小对文件上传时间影响

(2) XMark数据集上文档大小对文件上传时间影响

图5.1 Hadoop默认分片和Relax-Fragment、Arbitrary-Partition下文件上传时间

Fig.5.1 File upload time of Hadoop default fragment and Relax-Fragment and Arbitrary-Partition

### 5.3.2 NDTH与DTH算法的加速比性能分析

本节主要对NDTH算法和DTH算法的加速比性能进行实验和分析。评测加速比的方法是保持查询数据规模不变，增加slave节点的数目。首先分别用DBLP数据集和XMark数据集在集群中执行NDTH算法和DTH算法、DTS算法，然后使Hadoop集群slave节点数从1依次递增加至8个节点，测试slave节点数对三种算法的执行时间的影响，这样共产生两组实验数据，根据实验数据，绘出了如图5.2所示的两个曲线图。

(1) DBLP数据集上slave点数对算法执行时间的影响

(2) XMark数据集上slave点数对算法执行时间的影响

图5.2 slave节点数对NDTH和DTH、DTS执行时间影响

Fig. 5.2 Influence of slave node size on execution time of NDTH and DTH and DTS

图5.2(1)和(2)中表示NDTH算法和DTH算法、DTS算法在不同节点数的集群中匹配查询模式Q1的时间，其随着Hadoop集群的slave节点的增多，两种算法在两个数据集上的执行时间逐渐减少。从图中我们可以发现，DTS算法的执行时间最多，这是因为该算法会产生大量中间结果，对这些中间结果的处理会花费大量时间，而DTH算法利用索引信息大大减少了查询时间。并且随着Hadoop集群中slave节点的增多，DTH算法执行时间减小得比其他两个算法多。在图中可以看到随着slave节点增多到一定数量，NDTH算法的执行时间减少幅度非常小，这是因为多个节点与协调者节点的通信开销增加。总的来说，随着节点个数的增加，DTH和NDTH算法的执行时间比DTS算法要少，查询效率更高，其中DTH算法的性能最优。

根据图5.2的实验结果，我们能够进一步得到算法的加速比。对得到两组执行时间，按照Speedup计算公式进行计算处理，绘出了如图5.3所示的两个加速比曲线图。

(1) DBLP数据集上算法的加速比

(2) XMark数据集上算法的加速比

图5.3 NDTH和DTH以及DTS算法的加速比

Fig. 5.3 Speedup of NDTH and DTH and DTS

图5.3显示了在两种数据集上NDTH算法和DTH算法以及DTS算法的加速比性能。理想情况下，这三种算法的加速比应随着节点数的增加而呈线性增长。然而，由于存在计算机间通信开销、任务启动、任务调度和故障处理等时间消耗，实际加速比低于理想的加速比。从两组实验结果图5.3(1)和(2)中可以看出，NDTH算法和DTH算法的加速比都比DTS算法的好。因为随着slave节点个数的增加，NDTH算法和DTH算法的加速比增加的比DTS的快。此外，DTH算法的加速比曲线在最上方，说明DTH算法比比NDTH算法及DTS算法的性能更好。

### 5.3.3 NDTH与DTH算法的数据伸缩率性能分析

本节主要是对NDTH与DTH算法的数据伸缩率进行实验和分析。评测数据伸缩率的方法是保持slave节点的数目不变，扩大查询数据规模。我们首先测试了在XMark数据集和DBLP数据集中，数据规模大小对算法查询执行时间的影响。

(1) DBLP数据集上数据大小对算法时间的影响

(2) XMark数据集上数据大小对算法时间的影响

图5.4 数据大小对NDTH和DTH以及DTS算法执行时间的影响

Fig. 5.4 Influence of data size on execution time of NDTH and DTH and DTS

在本实验中保持Hadoop集群中8个slave节点数不变，分别在两种数据集中取大小比例约为20%、40%、60%、80%、100%规模的查询数据集，然后执行NDTH算法和DTH算法以及DTS算法，得出两组三种算法的执行时间。最后我们根据实验结果的数据，绘出了如图5.4所示的两个曲线图。

(1) DBLP数据集上算法的数据伸缩率

(2) XMark数据集上算法的数据伸缩率

图5.5 NDTH和DTH以及DTS算法的数据伸缩率

Fig. 5.5 Sizeup of NDTH and DTH and DTS

从图5.4我们可以看到，随着数据规模的不断增大，三种算法的执行时间也在不断增加。我们可以清楚的看到在每个集群中，随着各自XML查询数据的增大，NDTH和DTH算法的执行时间也越长。由于存在一定的传输代价，并且数据规模越大，数据分布越均衡，越能发挥各节点的计算能力，所以各曲线并不是随着文档大小的增长而呈线性增长的。从图中可以看出DTH算法执行时间最少，DTS需要时间最长，而NDTH介于两者之间，这也证明了本文所提的两种算法具有较好的性能。

由图5.4中的实验数据，我们可以进一步得到算法的数据伸缩率。在不同数据集中取不同大小的查询数据集，分别执行NDTH算法和DTH算法以及DTS算法，得出了两组执行时间。我们根据实验数据按照Sizeup的计算公式进行计算处理，绘出了如图5.5所示的两个Sizeup曲线图。

从图5.5中我们可以看出，随着查询数据规模的线性增长，NDTH算法和DTH算法的的数据伸缩率曲线在逐渐上升。但总的来说图中的NDTH和DTH的曲线整体上的增长趋势较为平缓，这是因为当数据规模越大负载越均衡越能发挥所有节点的计算能力，因此体现出NDTH和DTH算法有良好的数据伸缩率。同时，可以看到，当数据规模很大时，DTH算法的曲线增长的趋势比NDTH算法的曲线的增长趋势表现得更缓慢，而NDTH的曲线增长又比DTS的曲线更加缓慢，这体现出DTH算法和NDHT算法比DTS算法有更佳的数据伸缩率，而DTH算法的性能比NDTH算法的更良好一些。

### 5.3.4 NDTH与DTH算法的扩展率性能分析

本小节主要是对NDTH与DTH算法的扩展率性能进行实验和分析，扩展率是单个计算节点处理较小数据集所需时间与多个计算节点处理更大数据集所需时间之比，它衡量了当查询数据规模不断增大时，NDTH与DTH算法能否有效地利用可扩展的节点数从而扩展整体的计算能力。本实验中所使用的数据来自数据集DBLP和XMark，查询模式是数据集对应的查询模式DQ1和XQ1。为了测试NDTH和DTH这两种算法的扩展率性能，我们使用slave节点数目分别为1、2、3、4、5的Hadoop集群，每个集群对应的查询数据大小分别为两种数据集规模的20%、40%、60%、80%、100%，然后在集群上分别执行NDTH和DTH算法以及DTS算法，得到了两组数据，即算法在不同计算节点处理不同数据的执行时间。最后对实验结果的数据按照Scaleup计算公式进行计算处理，绘出了如图5.6所示的两个扩展率曲线图。

图5.6显示了在不同计算节点上的扩展性结果。在理想情况下，即不考虑传输代价等因素，合理选择查询数据规模和可利用的节点数可以使扩展率曲线保持水平，即扩展率的值是恒等于1。但实际上，传输代价是不能忽视的，由于任务启动开销以及通信开销随着任务数的增加而增大，实际情况中，扩展率曲线并不能保持水平。分析图5.6的实验结果，NDTH和DTH算法的扩展率的值随着slave节点数和查询数据规模的增大而呈现递减趋势。如图5.6(1)和(2)所示，当slave节点增多同时数据规模逐渐变大时，NDTH算法和DTH算法的扩展率的值都在缓慢减小，并且都比DTS算法减小的慢，并且本文所提的两种算法的Scaleup值随着slave节点数和数据规模的改变，一直比DTS算法的Scaleup值大，这说明本文所提的算法对数据集的大小有很好的适应性。在不同节点和不同规模数据的处理中，NDTH算法和DTH算法的可扩展性的值分别都在0.65以上，表明NDTH算法和DTH算法整体上还是具有较好的扩展率性能，因此本文算法能够有效地利用可扩展的节点数来实现整体计算能力的扩展。同时从图5.6中可以看出DTH算法的扩展率的值也高于NDTH算法的扩展率的值，说明DTH算法的可扩展性性能比NDTH算法的更好。

(1) DBLP数据集上算法的扩展率

(2) XMark数据集上算法的扩展率

图5.6 NDTH 和DTH以及DTS算法的扩展率

Fig. 5.6 Scaleup of NDTH and DTH and DTS

## 5.4 本章小结

本章首先在5.1节中介绍了实验性能的评估标准，主要包括加速比、数据伸缩率、扩展率三个方面。在5.2小节中介绍了本次实验所需要的软硬件环境、数据集，并概括性介绍了实验方案。在5.3节从多个方面对本文算法进行了实验比较，并介绍了实验的实施细节和实验方案，通过实验结果详细地分析了本文算法性能的好坏，实验结果表明了本文所提两种算法都有较好性能，且DTH算法的性能更好。首先在5.3.1节中，用DBLP和XMark两种数据集模拟海量查询数据，分别测试了本文的松弛分片算法的性能。在5.3.2节中，给出了NDTH算法和DTH算法以及DTS算法的加速比性能对比的实验结果，实验结果表明，随着slave节点的增多，NDTH算法和DTH算法的加速比都比DTS的加速比好，并且DTH算法比NDTH算法有更好的加速比。在5.3.3中用NDTH算法和DTH算法和DTS算法的进行了数据伸缩率实验对比，随着文档规模的增长，NDTH算法和DTH算法都表现出良好的数据伸缩率性能。在5.3.4中的扩展率实验中，NDTH算法和DTH算法的可扩展性值都在0.65以上，表明两者对数据集大小有良好的适应性。

# 第6章 总结与展望

本章对全文的研究工作和实验内容进行总结，包括分布式NDTH查询算法和XML文档的松弛分割算法及基于该分片算法的分布式DHT查询算法，并在理论分析和实验结果的基础上，对未来进一步的研究内容和方向提出了展望。

## 6.1 论文工作总结

本文主要针对海量XML文档的Twig整体匹配查询的问题，提出了两种基于节点分发的Twig查询算法。一是基于ComMapReduce的NDTH查询算法，二是基于松弛分片的DHT查询算法，这两种算法都只需要一遍Map和Reduce任务即可。本文所提算法都是基于节点分发的，所谓节点分发，其思想主要是以查询根节点的匹配为核心，将分片上的可能与查询根节点的匹配合成查询解的节点分发到一起，然后并行执行整体结构连接查询处理。所以本文所提算法与现有的基于MapReduce框架的XML查询算法有很大不同，本文的算法在Map阶段不做任何结构连接操作，只处理节点分发，而查询匹配的结构连接操作全在Reduce阶段做，并且Reduce阶段可以直接应用任何Twig整体匹配算法，非常具有灵活性。

首先，在XML文档分片方面，现有基于MapReduce框架的XML查询算法中，不能保证分片的大小相等，而且分片规则也比较复杂，查询处理的执行容易受分片的影响，并且这种分片算法还要求对查询模式进行分解。本文在研究了MapReduce的工作原理和XML的结构化特性的基础上，提出了松弛分片策略RFS，可以对XML文档进行任意分割。本文第3章提出的分布式NDTH查询算法，因引入了协调者节点，其可以传递全局信息，所以分片过程无需保存分片间的结构信息。为了减少通信代价，第4章在传统MapReduce模型的基础上提出分布式DTH查询算法。该算法需要在XML文档分片过程中保存结构信息，这就是分片索引，因此，本文提出了可以任意分片并能维持各分片与原始文档间结构信息的松弛分片算法，保证查询阶段解的正确性和完整性。

其次，在海量XML文档的分布式Twig查询处理方面，分布式NDTH查询算法，是将各个分片中nq的节点的Key（该节点的区间编码）都发送至协调者节点，该节点收集到所有分片的Key以后形成KeyList，再发回给各个Map计算节点，这样每个元素节点就找到它所有的nq祖先节点，并设置键值，每个nq节点连同它所有的后裔节点都被分发到一个Reducer上，这样就可以并行执行Twig整体匹配算法。分布式DTH查询算法中，为了不丢失查询解，在节点分发阶段，利用松弛分片算法中建立的分片索引就可以找到分片节点的所有nq祖先节点。这样Key值相同nq祖先和各分片中的后裔节点就能汇聚到一起。然后再并行执行Twig整体匹配算法，如TwigStack算法。

最后，本文还构建了一系列实验，验证了本文提出的分布式NDTH查询处理算法和分布式DTH查询处理算法的有效性和高效性。从实验结果来看，本文提出的两种基于节点分发的分布式Twig查询算法，在海量XML查询处理方面的性能还是比较令人满意的。

## 6.2 未来工作展望

虽然从实验结果来看，使用本文提出的分布式NDTH查询处理算法和分布式DTH查询处理算法能够得到令人满意的查询处理效率，但在理论研究上，本文算法在以下两个方面还可以做进一步的改进和优化。

首先，XML文档松弛分片算法的优化。本文所提文档松弛分片算法，是通过记录了分片的子树编号，并通过子树编号建立了分片索引，来维持各个文档分片与整个XML文档间的结构关系，每个标签对应一条索引记录，不同标签对应不同记录。该索引方式可能记录信息偏多，如何使用更少量的信息来记录分片间的结构信息，或是采用更省存储空间的数据结构，需要在未来研究工作中进一步探讨和发现。

其次，节点分发阶段的键值对设计的优化。本文所提算法在Map阶段，都是以查询根节点的匹配节点来设置Key的。在各个查询处理节点上，某些查询根节点的匹配节点可能与很多节点形成多个查询解，这可能造成每个机器上产生的查询解不均衡，本文在键值的设置过程中没有做负载均衡处理。所以为了提高算法的查询效率，未来可以优化键值对的设计，如在节点键值的设置过程将查询负载纳入的查询问题的考虑范围内，以实现算法能够负载均衡的高效查询处理。

由于研究时间的有限和理论研究的复杂性，本文对上述问题并没有进行深入的研究。在将来的研究工作中，上述问题都将作为进一步研究的内容和方向。

# 参考文献

1. Extensible Markup Language (XML)[EB/OL]. <http://www.w3.org/XML/>.
2. Tim Bray, Jean Paoli, C. M. Sperberg-McQueen. Extensible Markup Language (XML)[J]. World Wide Web Journal (WWWJ), 1997, 2(4):27-66.
3. Wikipedia (2013): Database download. Wikipedia Foundation, Inc[EB/OL]. http://en.wikipedia.org/wiki/Wikipedia:Database\_download.
4. Amos Bairoch, Rolf Apweiler, Cathy H. Wu, et al. The universal protein resource (UniProt)[J]. Nucleic Acids Research (NAR), 2005, 33(Database-Issue):154-159.
5. Andrew Pavlo, Erik Paulson, Alexander Rasin, Daniel J. Abadi, David J. DeWitt, Samuel Madden, Michael Stonebraker. A comparison of approaches to large-scale data analysis [C]. SIGMOD, 2009:165-178.
6. 龚星,陈植华,孙璐.地下水环境影响评价若干关键问题探讨[J].安全与环境工程, 2013, 20(2):95-99.
7. Shu Long-Cang.Groundwater overdraft vulnerability and environmental impact assessment in Arusha [J]. Environmental Geology, 2007, 51(7):1171-1176.
8. Wirth N.Type Extensions. Transactions on Programming Language and Systems[J]. ACM, 1988, 10(2): 204-214.
9. Dewey Decimal Classification[EB/OL]. http://www.oclc.org/dewey/.
10. I. Tatarinov, S. Viglas, K. S. Beyer, J. Shanmugasundaram, E. J. Shekita, and C. Zhang. Storing and querying ordered XML using a relational database system[C]. SIGMOD, 2002, 204–215.
11. Jiaheng Lu, Tok Wang Ling, Chee Yong Chan, Ting Chen. From Region Encoding To Extended Dewey: On Efficient Processing of XML Twig Pattern Matching[C]. VLDB, 2005, 193-204.
12. C．Zhang, J．Naughton, D．DeWitt, Q．Luo, G Lohman．On Supporting Containment Queries in Relational Database Management Systems[C]. SIGMOD, 2001, 425-436.
13. Paul F. Dietz. Maintaining order in a linked list[C]. STOC, 1982, 122-127.
14. 毕鑫, 王国仁. XML数据中Twig查询处理与优化技术研究综述[J]. 计算机科学与探索, 2013, 43(8): 1361-1367.
15. Bruno N, Koudas N, Srivastava D. Holistic twig joins: optimal XML pattern matching[C]. SIGMOD, 2002, 310-321.
16. Jiang Haifeng, Wang Wei, Lu Hongjun, et al. Holistic twig joins on indexed XML documents[C]. VLDB, 2003, 273-284.
17. Lu Jiaheng, Chen Ting, Tok Wang Ling. Efficient processing of XML twig patterns with parent child edges: a look-ahead approach[C]. CIKM, 2004, 533-542.
18. Chen Songting, Li Hua-Gang, Agrawal D, et al. Twig2Stack: bottom-up processing of generalized-tree-pattern queries over XML documents[C]. VLDB, 2006, 283-294.
19. Qin Lu, Yu Xu Jeffrey, Ding Bolin. TwigList: make twig pattern matching fast[C]. DASFAA, 2007, 850-862.
20. Buneman P, Cong Gao, Fan Wenfei. Using partial evaluation in distributed query evaluation[C]. VLDB, 2006, 211-222.
21. Cong G, Fan W, Kementsietsidis A, et al. Partial evaluation for distributed XPath query processing and beyond[J]. ACM Transactions on Database Systems, 2012, 37(4):32.
22. Cong Gao, Fan Wenfei, Kementsietsidis A. Distributed query evaluation with performance guarantees[C]. SIGMOD, 2007, 509-520.
23. Kevin Lü, Yuanling Zhu, Wenjun Sun: Parallel Processing XML Documents[C]. IDEAS, 2002, 96-105.
24. K. Kido, T. Amagasa, and H. Kitagawa. Processing XPath Queries in PC-Clusters Using XML Data Partitioning[C]. ICDE Workshops, 2006, 114-119.
25. N. Tang, G. Wang, X. J. Yu, K.-F. Wong, and G. Yu. WIN.An Efficient Data Placement Strategy for Parallel XML Databases[C]. ICPADS, 2005, 249–355.
26. H. Kurita, K. Hatano, J. Miyazaki, and S. Uemura. Efficient Query Processing for Large XML Data in Distributed Environments[C]. AINA, 2007, 317-322.
27. Kling, P., Ozsu, M.T., Daudjee, K.: Generating efficient excution plans for vertically partitioned XML databases[J]. PVLDB, 2010, 4(1):1-11.
28. R. Bordawekar, L. Lim, and O. Shmueli. Parallelization of XPath queries using multi-core processors: challenges and experiences[C]. EDBT, 2009, 180-191.
29. N. Bidoit, D. Colazzo, N. Malla, and C. Sartiani. Partitioning XML documents for iterative queries[C]. IDEAS, 2012, 51-60.
30. I. Machdi, T. Amagasa, and H. Kitagawa. GMX: An XML Data Partitioning Scheme for Holistic Twig Joins[C]. iiWAS, 2008, 137-146.
31. Imam Machdi, Toshiyuki Amagasa, Hiroyuki Kitagawa. XML data partitioning strategies to improve parallelism in parallel holistic twig joins[C]. ICUIMC, 2009, 471-480.
32. Mary F. Fernández, Trevor Jim, Kristi Morton, Nicola Onose, Jérôme Siméon. DXQ: a distributed XQuery scripting language[C]. XIME-P, 2007, (3).
33. P. Strnad, O. Macek, and P. Jira. Mapping xml to key-value database[C]. DBKDA, 2013, 121–127.
34. J. Camacho-Rodr´ıguez, D. Colazzo, and I. Manolescu. Building large xml stores in the amazon cloud[C]. ICDE Workshops, 2012, 151-158.
35. Dan Suciu. Distributed query evaluation on semistricutred data[J]. ACM Trans. Database Syst. (TODS), 2002, 27(1), 1-62.
36. S. Khatchadourian, M. P. Consens, and J. Sim´eon. Having a chuql at xml on the cloud[C]. AMW, 2011.
37. L. Fegaras, C. Li, U. Gupta, and J. Philip. Xml query optimization in map-reduce[C]. WebDB, 2011.
38. 魏永山,张峰,陈欣,孙忠林. 一种基于MapReduce的XML查询的优化方法[J]. 微电子学与计算机, 2013, 30(3):108-111.
39. H. Choi, K.-H. Lee, S.-H. Kim, Y.-J. Lee, and B. Moon. Hadoopxml: a suite for parallel processing of massive xml data with multiple twig pattern queries[C]. CIKM, 2012, 2737-2739.
40. Adam Senk, Michal Valenta, Wolfgang Benn: Distributed Evaluation of XPath Axes Queries over Large XML Documents Stored in MapReduce Clusters[C]. DEXA Workshops, 2014, 253-257.
41. M. Damigos, M. Gergatsoulis, and S. Plitsos. Distributed processing of xpath queries using mapreduce[C]. ADBIS, 2013, 69-77.
42. Matthew Damigos, Manolis Gergatsoulis, Eleftherios Kalogeros. Distributed evaluation of XPath queries over large integrated XML data[C]. Panhellenic Conference on Informatics, 2014, 1-6.
43. Nicole Bidoit, Dario Colazzo, Noor Malla, Federico Ulliana, Maurizio Nolé, Carlo Sartiani. Processing XML queries and updates on map/reduce clusters[C]. EDBT, 2013, 745-748.
44. Huayu Wu. Parallelizing Structural Joins to Process Queries over Big XML Data Using MapReduce[C]. DEXA, 2014, 183-190.
45. Foto N. Afrati, Matthew Damigos, Manolis Gergatsoulis. Lower Bounds on the Communication of XPath Queries in MapReduce[C]. EDBT/ICDT Workshops, 2015, 38-41.
46. Ralf L. Google's MapReduce Programming Model - Revisited[J]. Science of Computer Programming, 2008, 70(1):1-30.
47. Yang H, Dasdan A and Hsiao R L. Map-Reduce-merge: Simplified Relational data Processing on Large Clusters[C]. SIGMOD, 2007, 1029-1040.
48. D. Jiang, A.K. Tung, G. Chen. Map-Join-Reduce: toward scalable and efficient data analysis on large clusters[J]. Knowledge and Data Engineering, 2011, 23 (9):1299–1311.
49. Rares Vernica, Michael J. Carey, Chen Li. Efficient parallel set-similarity joins using MapReduce[C]. SIGMOD, 2010, 495-506.
50. F.N. Afrati, J.D. Ullman. Optimizing joins in a map-reduce environment[C]. EDBT, 2010, 99-110.
51. A. Abouzeid, K. Bajda-Pawlikowski, D. Abadi, A. Silberschatz, A. Rasin. HadoopDB: an architectural hybrid of MapReduce and DBMS technologies for analytical workloads[C]. PVLDB, 2009, 2(1):922–933.
52. Hadoop Distributed File System[EB/OL]. <http://hadoop.apache.org/hdfs/>.
53. LinLin Ding, Guoren Wang. ComMapReduce: An improvement of MapReduce with lightweight communication mechanisms[J]. Data Knowl. Eng. (DKE), 2013, 88:224-247.
54. DBLP XML Dataset[EB/OL]. http://dblp.uni-trier.de/xml/.

1. [Albrecht Schmidt](http://www.dblp.org/pers/hc/s/Schmidt_0002:Albrecht.html),[Florian Waas](http://www.dblp.org/pers/hc/w/Waas:Florian.html),[Ioana Manolescu](http://www.dblp.org/pers/hc/m/Manolescu:Ioana.html),[Ralph Busse](http://www.dblp.org/pers/hc/b/Busse:Ralph.html). XMark: A Benchmark for XML Data Management[C].[VLDB, 2002, 974-985](http://www.dblp.org/db/conf/vldb/vldb2002.html#SchmidtWKCMB02).
2. Xin Bi, Guoren Wang, Xiangguo Zhao, Zhen Zhang, and Shuang Chen. Distributed XML Twig Query Processing using MapReduce[C]. To appear in APWeb 2015.

# 致 谢

时光荏苒，岁月如梭，两年的研究生生涯竟悄然结束。在攻读硕士学位期间，让我受益匪浅，收获良多，我的思维和意志力都得到了锻炼，知识水平也得到了提高，这些收获除了自己的努力也离不开所有给予我帮助的老师和朋友们。经过几个月的磨砺，我的硕士毕业论文现在终于完稿了。这几个月里我不停地收集整理资料，反复思考、修改论文直至最终完成。在这过程中，我得到了老师和师兄们的许多指导，以及同学朋友的帮助、支持和关怀，在此，我要向他们表达我最诚挚的谢意。

在这里最先要感谢的人是我的导师赵相国副教授。在赵老师的辛勤培育和教导下，我才有今天的收获。赵老师为人谦和、平易近人、对学生认真负责，能成为赵老师的学生，我感到无比地庆幸和荣幸。我的论文从定题、收集资料到写作定稿，都倾注了老师大量心血和精力。每当我遇到疑问时，老师都会从百忙之中抽出时间对我悉心指导，给我指点迷津，帮我打开、理清思路。赵老师严谨的学术作风和积极的生活态度及严于律己的品质都深深影响了我，他是我的学习榜样。再次感谢赵老师对我的关心和谆谆教导。

同时我还要特别感谢一直以来对我论文进行了指导的毕鑫师兄。在论文撰写过程中，毕师兄不厌其烦地指点我，还帮我初步审阅，对我的论文认真地批改，字字句句把关，提出许多中肯的指导意见，使我在研究和写作过程中不迷失方向。借此机会，我向毕师兄致以深深地谢意。

感谢实验室的王波涛、袁野、乔百友等老师。感谢他们一直以来对我的帮助，他们严格高质量的教学态度，让我更好地端正了自己的学习态度。还要感谢和我在实验室一起学习奋斗的杨洪波、陈远根、张祯等同学以及实验室里其他给予过我帮助师兄师姐以及师弟师妹们，感谢你们在生活和学习上给予我的帮助和支持。更要感谢寝室的姐妹们，张家叶子、丛林、王样，她们一起陪我度过了两年研究生生活，在生活上给予我关心和帮助，让我感受到了家的温馨。

还要特别感谢在我身后默默支持我的父母和亲人。他们是我在他乡求学路上坚强的后盾，他们给了我极大的鼓励和信心，他们无私的爱与照顾是我不断前进的动力。

最后，我要感谢参与我论文评审和答辩的各位老师，是他们给了我审视两年来学习成果的机会，让我能够明确今后的发展方向，这是一笔无价的财富。我将在今后的工作和学习中加倍努力，以期能够取得更多成果来回报他们、回报社会。再次感谢他们，祝他们一生幸福平安！同时祝福计算机系统结构研究所永远朝气蓬勃。

# 攻读硕士学位期间参加的项目和发表的论文

硕士期间参加的科研项目：

1. 横向课题：宁夏理工学院信息化管理平台建设；参与时间：2013.09-2015.06
2. 国家自然科学基金：云环境下海量XML数据查询处理关键技术研究；项目编号：61272181；参与时间：2013.09-2015.06

硕士期间发表的论文：

1. Xin Bi, Xiangguo Zhao, Guoren Wang, Zhen Zhang, and Shuang Chen. Distributed Learning over Massive XML Documents in ELM Feature Space[J]. Mathematical Problems in Engineering, Volume 2015.
2. Xin Bi, Guoren Wang, Xiangguo Zhao, Zhen Zhang, and Shuang Chen. Distributed XML Twig Query Processing using MapReduce[C]. To appear in APWeb 2015.