面向分布式的SPARQL查询优化算法

汪璟玢 方知立 张燕琴
(福州大学数学与计算机科学学院 福州 350108)

摘 要 采用分布式来实现SPARQL(Simple Protocol and RDF Query Language)查询是解决海量RDF(Resource Description Framework)查询的一种新思路。目前实现的基于Hadoop的RDF查询都要启用多个MapReduce来完成任务,浪费时间。为了克服此缺点,提出MRQJ(using MapReduce to query and join)算法,用以实现SPARQL的分布式查询。该算法分为连接计划生成与SPARQL查询执行两个部分:连接计划生成采用贪心策略,生成最优的连接方案;在SPARQL查询执行中只需结合一次MapReduce计算即可得到查询结果。在LUBM数据集上进行的测试实验表明:在查询语句较为复杂的情况下,MRQJ方法的查询效率具有明显的优势。

关键词 RDF,Hadoop,SPARQL查询,MapReduce
中图法分类号 TP391 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2014.07.047

Distributed Optimized Query Algorithm Based on SPARQL

WANG Jing-bin FANG Zhi-li ZHANG Yan-qin
(College of Mathematics and Computer Science, Fuzhou University, Fuzhou 350108, China)

Abstract It's a new way of solving the large amount of RDF(Resource Description Framework) query problem to use distributed technique to realize the SPARQL(Simple Protocol and RDF Query Language) Query. At present, most of the RDF queries based on Hadoop have to use multiple MapReduce jobs to complete the task, resulting in waste of time. In order to overcome this drawback, this paper proposed MRQJ(using MapReduce to the query and the join) algorithm to perform distributed SPARQL query. The algorithm can be divided into join plan generation and SPARQL query execution two parts: join plan generation uses greedy strategy to generate the most optimal join scheme, and only one MapReduce job should be done to get the query results in SPARQL query execution. The experiment on the LUBM test data set was made. The experimental results show that MRQJ method query efficiency is higher when the case of a query is more complicated.

Keywords RDF, Hadoop, SPARQL query, MapReduce

1 引言

RDF作为语义网的基础,以三元组形式表示信息并交换万维网上的知识和数据[1]。早期RDF存储系统采用单一机器集中式架构模型,大多通过关系数据库来存储和查询RDF数据[2]。随着语义Web技术的发展,RDF数据格式得到越来越多的应用,RDF数据集的规模几乎以每年翻一番的速度在不断扩大。在这样爆炸式增长的存储压力之下,原始的简易架构模型无可避免地遭遇了性能瓶颈,因此海量RDF数据在存储能力和查询响应两方面都不能满足日益增长的需求。目前有不少工作在讨论海量分布式RDF存储与查询问题,RDF分布式存储主要包括HDFS和HBase两种方案[2-8]。在存储的基础上研究者也各自提出分布式RDF查询策略,文献[5-8]中主要是利用MapReduce的框架实现查询算法。Mohammad Farhan Husain[5]等人提出了一个贪心的MapReduce任务生成算法,即多个MapReduce任务迭代处理SPARQL BGP连接操作。Jaeseok Myung[6]等人提出的查询策略中首先需要创建一个MapReduce任务,从HDFS中的RDF数据文件中查找对应的RDF三元组,随后创建多个MapReduce任务迭代处理SPARQL BGP连接操作。Mohammad Farhan Husain[7]等人在文献[5]的基础上提出查询计划优化算法,并用多个MapReduce任务迭代处理SPARQL BGP连接操作。Jieru Cheng[8]等人提出的算法中采用PredicateLead进行数据预处理后,并采用JobPartitioner算法生成多个MapReduce任务进行查询连接处理得到查询结果。Liu L[9]等人提出使用多重连接过滤的方法替代传统的SQL连接查询,然后在工作流中合并不同的连接计划。以上多种方案都要在SPARQL BGP连接过程中创建多个MapReduce来完成任务,由于MapReduce Job的启动相对耗时,因此十分浪费查询时间。

为了解决上述查询效率不高的问题,本文提出了MRQJ

到稿日期:2013-09-29 返修日期:2013-12-30 本文受福州大学科技发展基金资助项目(2013-XQ-32),空间数据挖掘与信息共享教育部重点实验室开放研究基金项目(201006),2011年福建省科技拥军基金项目(JG2011005),福建省自然科学基金项目(2012J01168)资助。
汪璟玢(1973—),女,硕士,副教授,主要研究领域为海量数据管理、网络数据库和智能技术,E-mail:wjbcc@263.net;方知立(1990—),男,硕士生,主要研究领域为海量数据管理、智能技术;张燕琴(1989—),女,硕士生,主要研究领域为海量数据管理、智能技术。

算法，该算法只需要一个 MapReduce 任务即可完成 SPARQL 查询。首先将 RDF 文件解析成三元组形式存储在 HDFS 文件系统中，并将 SPARQL 查询语言用贪心策略生成连接计划后利用 MapReduce 并行计算得到查询结果。文章最后通过对比实验验证了 MRQJ 方法在查询语句较为复杂的情况下查询效率较高。

本文首先介绍 RDF 数据模型、SPARQL 查询语言与 MapReduce 计算框架；第 3 节详细论述 MRQJ 算法，包括采用贪心策略生成连接计划的算法与 SPARQL 查询执行算法；第 4 节给出实验结果并加以分析；最后对本文工作进行总结。

2 基本概念

2.1 RDF 数据模型

RDF 的基本思想是通过统一资源标识符(Uniform Resource Idetifier，URI)来标识 Web 上的资源，用简单的属性(property)以及属性值(value)来描述资源。RDF 模型可以采用不同的语法形式来描述。大多情况下都是用 XML 来刻画 RDF，其特点是简单，易于掌握和使用[10]。

三元组是 RDF 模型的另外一种常见描述方式。每一个三元组包括一个主语(subject)、一个谓语(predicate)和一个宾语(object)。陈述的主语(subject)通常是指向相应资源(resource)的 URI，谓语(predicate)是表示某一属性(property)的 URI，而宾语(object)既可以是指向某一个资源的 URI，也可以单纯地以一个文字(literal)作为属性值(property value)。除此之外，主语和宾语也可以是没有 URI 表示的匿名资源，亦称空白节点(blank node)[11]。本文采用以三元组形式表示的 RDF 数据。

除了以上两种方式之外，RDF 还可以用有向图的形式描述。一个 RDF 图的结点就是它包含的所有三元组的主语和宾语，而边的方向总是指向宾语[12]。在 RDF 图建立完成之后便可通过图中内容来了解 RDF 所描述的资源的信息。基本的 RDF 图的表示形式如图 1 所示。

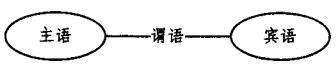

图 1 包含一个三元组的 RDF 图

2.2 SPARQL 查询语言

SPARQL(Simple Protocol and RDF Query Language)是目前 W3C(World Wide Web Consortium)针对 RDF 查询语言的推荐标准，它定义了 RDF 查询语言的语法和语义，它的语法同关系数据库查询语言 SQL 接近[2]。Triple Pattern 作为 SPARQL 中最基本的匹配单元，其构成与 RDF 三元组表示形式相对应。RDF 数据以(S,P,O)三元组表示，Triple Pattern 也由这 3 部分构成。Triple Pattern 对应位置可以是已绑定的值或是未绑定的变量，未绑定变量由问号加变量名组成。例如，以下 SPARQL 语句表示查询“工作于 University1 并且是 Department1 成员的教授”。

```
SELECT ? x
Where {
? x rdf:type Professor.
? x worksFor University1.
? x memberOf Department1.
}
```

2.3 MapReduce 计算框架

MapReduce 是一个编程模式，充分利用了“分而治之”的理念，使得在处理海量数据时可以对任务并行处理，这样通过协作能够轻松地完成更多任务[13]。MapReduce 并行计算框架将任务处理划分为 Map 阶段与 Reduce 阶段。Map 与 Reduce 函数的输入都是(key，value)值，因此也可以看出 MapReduce框架的局限性。

用户指定一个 map 函数，通过这个 map 函数处理 key/value(键/值)对，产生一系列的中间 key/value 对，并且使用 reduce 函数来合并所有相同 key 值的 value 部分后得出最终结果。

3 MRQJ 算法

MRQJ 算法中主要包含连接计划生成与 SPARQL 查询执行两个部分，算法框架如图 2 所示。

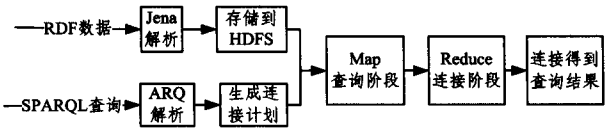

图 2 MRQJ 算法框架图

首先使用 Jena 作为 RDF 数据解析工具，将 RDF XML 格式解析成三元组格式并存放于 HDFS 文件系统中。在连接计划生成之前选用开源工具 ARQ 来进行 SPARQL 查询语句解析。SPARQL 查询执行阶段分为 Map 阶段和 Reduce 阶段来完成查询连接任务。若查询语句中存在多变量的情况，则需要对 Reduce 输出连接得到最终查询结果。

3.1 连接计划

图 3 为采用 LUBM 开源查询语句中的 Q12，其 ARQ 的解析结果如图 4 所示，展示了 Triple Pattern 子句的详细信息。

```
SELECT ?X?Y WHERE {
?X rdf:type ub:Chair. //1
?Y rdf:type ub:Department. //2
?X ub:worksFor ?Y. //3
?Y ub:subOrganizationOf  University0. edu} //4
```

图 3 LUBM Q12

```
(Project(?X ?Y)
(bgp
(triple ?X rdf:type ub:Chair)
(triple ?Y rdf:type ub:Department)
(triple ?X ub:worksFor ?Y)
(triple ?Y ub:subOrganizationOfUniversity0. edu)
))
```

图 4 Q12 的 ARQ 解析结果

Triple Pattern 间的关联性通过它们之间是否存在共同的未绑定变量判定。LUBM Q12 共有 4 个 Triple Pattern 子句，其中 1 与 3 拥有共同变量 x，故可连接得到(x,y)；2 与 3 拥有共同变量 y，也可连接得到(x,y)。由此可见，同一个查询语句存在多种不同的连接方案。如 LUBMQ12 连接方案可以是 1→3→2→4 或者 2→3→4→1 或者 4→2→3→1 等多种方案。然而连接方案的优劣很大程度上决定了 SPARQL 查询的效率。为了提高 RDF 查询效率，本文提出的 MRQJ 算法中给出连接计划生成策略。

定义 1(连接计划) 给定 SPARQL 查询 Q，经过 ARQ

解析后生成 Triple Pattern 子句集 $T=\{T_1,T_2,T_3,\cdots,T_n\}$。分析所有 Triple Pattern 子句集中变量间的关系，得到一个最优的连接方案，称为该 SPARQL 查询的连接计划。

连接计划生成采用贪心选择策略，共包含 4 个步骤，具体策略如下所述：

步骤 1　遍历各 Triple Pattern，得到每个变量的句子集合；

步骤 2　贪心选择包含句子集合个数最多的变量，先连接该变量的所有 Triple Pattern 句子，连接过程中优先连接结果集少的句子；

步骤 3　连接完一个变量后，更新其他变量中的句子集合，删除已连接的 Triple Pattern 句子；

步骤 4　重复步骤 2 和 3，直到所有变量遍历结束。

假设查询变量集为 $X=\{X_1,X_2,\cdots,X_t\}$。首先遍历 T 得到每个查询变量 X_i 所在 Triple Pattern 子句序号集，如 $X_i=\{T_p,T_q\}$，其中 $p,q\in\{1,2,3,\cdots,n\}$。连接计划算法描述如下。

算法 1　生成连接计划的算法

```
输入:查询语句 Q
输出:连接计划 JoinPlan
1. Begin
2. // VI 为一个对象,VI. v 为查询变量,VI. listIndex 为该查询变量对应的子句集
3. // listVI 为所有变量对应 VI 的 list
4. // T={T1,T2,T3,…,Tn}为 Triple Pattern 子句集
5. // JoinPlan[n]为连接计划,JoinPlan[i]为第 i 个 Triple Pattern 子句用于连接的变量 v
6. isContain=False;
7. for i←0 to n do
8. v[t]←Ti的变量
9.    for j←0 to t do
10.      for k←0 to |listVI. size| do
11.        if(v[j]==listVI[k]. v)
12.        listVI[k]. listIndex. add(i)
13.        isContain←True
14.   if(isContain==False)
15.      VI. v←Ti 的变量
16.      VI. listIndex. add(i)
17.      listVI. add(VI)
18. While |listVI. size|≠0 do
19.   sort(listVI)
20.   VI=listVI. get(0)
21.   for i←0 to |VI. listIndex. size|
22.      JoinPlan[VI. listIndex[i]] ← VI. v
23.   listVI. remove(0);
24.   for j←0 to |listVI. size| do
25.      update(listVI[j])
26. End
```

算法第 1－17 行将 SPARQL 查询语句转换成 listVI 存放每个查询变量对应的子句集。这个过程中，遍历 n 个 Triple Pattern 子句，若子句中有两个查询变量，则同时添加两个变量中对应的子句集。

算法第 18－26 行，对上一步生成的 listVI 按贪心策略生成连接计划。每次对 listVI 按 listIndex 的长度进行排序，即按查询子句集个数从大到小排序。选择查询子句集个数多的变量，用以更新剩余变量的句子集，删除已经连接过的句子集，并同步生成连接计划中对应子句的连接变量。

通过上述算法描述可知，该算法的复杂度为 $O(n)+O(t^2)$，其中 n 为 Triple Pattern 子句数，t 为查询变量个数。

例 1　LUBM Q12 的连接计划生成

首先遍历各 Triple Pattern 得到每个变量的句子集合，以 Q12 为例，将得到$\{x:1,3\}$、$\{y:2,3,4\}$。贪心选择连接句子集合数量最多的变量，故选择变量 y，先连接 2,3,4 Triple Pattern 子句，且更新 x 变量的连接句子集合，去掉已连接的 Triple Pattern 子句，故得到$\{x:1\}$。最终得到连接计划$\{y:2,3,4\}$、$\{x:1\}$。连接计划的生成过程如图 5 所示。

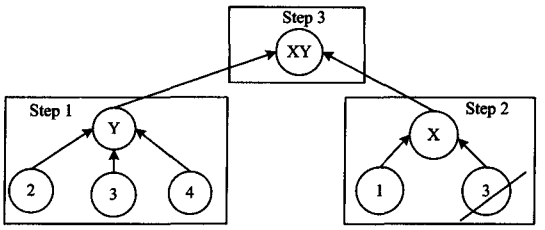

图 5　连接计划示意图

3.2　SPARQL 查询执行

由于 RDF 数据量很大，查询连接的计算量巨大，因此借助 MapReduce 分布式计算完成 SPARQL 查询。在 MapReduce 之前，首先按上节介绍进行 SPARQL 查询解析以及得到连接计划。本文只需采用一个 MapReduceJob 即可完成 SPARQL 查询任务，处理过程如图 6 所示。

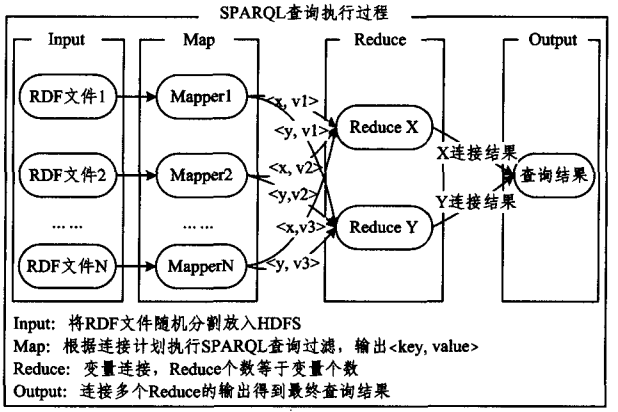

图 6　SPARQL 查询执行过程示例图

Map 阶段：输入查询的 RDF 三元组文件，遍历每个三元组，过滤所有 Triple Pattern 子句，若三元组满足查询子句的条件，则生成一对(key,value)。其中 key 为根据生成的连接计划匹配该 Triple Pattern 子句对应的连接变量，value 为过滤的查询结果。

Reduce 阶段：完成同一个变量对应的多个 Triple Pattern 子句的连接，每个连接变量得到一个连接结果集。

最后将每个变量 Reduce 输出的结果进行最后的连接，得到最终查询结果。

例 2　LUBM Q12 的 SPARQL 查询执行过程

Map 阶段：若三元组(zhangsan,rdf:type,ub:Chair)满足 Triple Pattern(?x rdf:type ub:Chair)，则 key=x,value=[x] zhangsan；若三元组(zhangsan,ub:worksFor,Baidu)满足 Triple Pattern(?X ub:worksFor ?Y)，因为该子句的查询变量为

xy，则查找子句 3 在变量 x 的句子集合中还是 y 的句子集合中。在该例子中 Triple Pattern(?X ub:worksFor ?Y)句子 3 连接变量为 y，故生成 key=y，value=[x]zhangsan[y]Baidu。

Reduce 阶段：如该例子中 key=x，将连接 Triple Pattern 句子 1 得到 x，key=y 将连接 Triple Pattern 句子 2、3、4 得到 (x,y)。

最后将变量 x 的连接结果与变量 y 的连接结果进行连接得到最后结果(x,y)。具体算法如表 1 所列。

表 1　Map 与 Reduce 方法

Map 阶段	Reduce 阶段

function map {
读入三元组 // example triple:s1 p3o1
for each(triple Pattern in a query){
// BGP example={
//1:? x p1 o1. 2:? y p2 o2.
//3:? x p3 ? y. 4:? y p4 o3.
// }
if(输入的三元组满足至少一个查询子句){
key=y,value =3&[x]s1[y]o1
output(key,value)
//key =满足的查询子句在连接计划中对应的连接变量，该例子中连接计划为 1-x,2-y,3-y,4-y
//value =查询子句的序号以及结果值
}
}
	function reduce {
	//相同的 key 在同一个 reduce 中做连接，假设该 reduce 处理 y 变量的连接
	//将同一个查询子句得到的结果值放在一起
	for each(value in values){
	List1=查询子句 2 的所有查询结果值
	List2=查询子句 3 的所有查询结果值
	List3=查询子句 4 的所有查询结果值
	}
	//不同查询子句的结果做连接
	Join(List1,List2,List3)
	key=y
	value=[x]s1[y]o1
	output(key,value)
	//key =连接变量
	//value =连接结果
	}

4　实验分析

实验所使用的硬件环境为 Intel(R) Core(TM) i5-3570，CPU 3.40GHz 双核双线程，内存 2GB，磁盘 80 GB，转速 7200 rpm。软件环境为操作系统 Linux Ubuntu，采用 Java 作为编程语言，开发环境为 eclipse。在实验环境中，用表 2 所列配置作为本系统 Hadoop 集群的配置，共计 5 台，其中 1 台作为 HDFS 的名称节点兼 MapReduce 的主节点，4 台为 HDFS 的数据节点兼 MapReduce 的从节点。集群工作站的基本配置如表 2 所列。

表 2　集群工作站的基本配置

CPU	Intel(R) Core(TM) i3-2310M
内存	2G
硬盘	500 GB SATA
Java	JDK1.6
Hadoop	Hadoop 2.0.4

本文采用 LUBM 工具分别生成了 10、50、100 所大学的测试。数据集的三元组数目、XML 文件大小以及解析后的三元组文件大小如表 3 所列。

表 3　LUBM 文件大小说明

LUBM 学校个数	三元组个数	XML 文件大小	解析后文件大小
10	140 万	116MB	228M
50	720 万	570MB	1.1GB
100	1340 万	1.1GB	2.3GB

在此数据集上，将本文算法与文献[5]中提出的 Mapreduce 迭代查询法进行了对比实验。在 RDF 数据的存储性能上，由于两种算法都是采用 Jena 解析成三元组格式后存储于 HDFS 上，文献[5]M. F. Husain 算法采用按 P 分割，只存储 S、O 数据，因此它们能够节省一定的空间。不过分割过程也消耗了一定的时间，而 MRQJ 算法解析后直接存储，会更节省存储时间。表 4 显示了两种算法存储时间消耗的实验结果对比。

表 4　存储时间对比(单位:s)

LUBM	M. F. Husain	MRQJ
10	57	45
50	145	93
100	215	197

由上面对比结果可知，MRQJ 在存储性能上略优于 M. F. Husain 算法。同时，本实验重点对比了两种算法的查询性能。在 10 所、50 所、100 所学校的数据集上，分别从 LUBM14 个标准查询中选取了 8 个查询语句进行测试实验，对比的实验结果如表 5－表 7 所列，查询时间单位为秒(s)。

表 5　LUBM(10) 对比实验

LUBM(10)	Q1	Q2	Q4	Q6	Q7	Q8	Q9	Q10	Q14
M. F. Husain	49	66	112	39	95	123	144	54	38
MRQJ	40	50	87	35	70	80	119	50	29

表 6　LUBM(50) 对比实验

LUBM(50)	Q1	Q2	Q4	Q6	Q7	Q8	Q9	Q10	Q14
M. F. Husain	86	166	217	80	189	477	535	93	74
MRQJ	73	144	194	78	153	391	413	86	67

表 7　LUBM(100) 对比实验

LUBM(100)	Q1	Q2	Q4	Q6	Q7	Q8	Q9	Q10	Q14
M. F. Husain	175	611	434	147	356	894	1087	234	213
MRQJ	157	580	359	134	285	791	885	228	208

从实验对比结果可见，在选取的 8 个查询语句中 MRQL 算法比文献[5]中 M. F. Husain 等人所提方法的查询效率普遍更高。其中对于 Q1、Q6、Q10、Q14 4 个查询语句，两个算法的查询时间相差不大，由于这 4 个语句都只有一个查询变量，M. F. Husain 算法也只需要启动一个 MapReduce 任务即可完成查询。Q2、Q4、Q8、Q9 中查询变量都不止一个，查询子句至少也有 4 个，故这些查询语句相对复杂，采用 M. F. Husain 方法需要启动多个 MapReduce 任务迭代完成查询。MapReduce Job 启动相对耗时，会损耗一定的查询时间。下面以 Q2 为例进行分析。

如图 7 所示，Q2 查询中共有 6 个查询子句、3 个查询变量。MRQL 算法只需要 1 个 MapReduce 任务用 3 个 Reduce 分别完成 x、y、z 变量对应语句的连接。而 M. F. Husain 方法中需要 3 次 MapReduce 任务迭代完成。因此 MRQL 算法查询时间会优于 M. F. Husain 提出的方案。

```
SELECT ?X,?Y,?Z
WHERE
{ ?X rdf:type ub:GraduateStudent.
  ?Y rdf:type ub:University.
  ?Z rdf:type ub:Department.
  ?X ub:memberOf ?Z.
  ?Z ub:subOrganizationOf ?Y.
  ?X ub:undergraduateDegreeFrom ?Y}
```

图 7　LUBM Q2

结束语　本文主要提出了一种全新的在 Hadoop 平台上进行大规模 RDF 数据 SPARQL 查询的方法，通过对比实验

验证了本文提出的 MRQJ 算法在 SPARQL 查询语句查询变量多、连接复杂的情况下具备一定的优越性。本文在存储过程中尚未考虑 RDF 文件分割策略，在后续研究过程中重点研究文件分割方法，希望将关系紧密的 RDF 数据分割存储于同一个文件而提高检索效率。

参 考 文 献

[1] 李慧颖，瞿裕忠. 基于关键词的语义网数据查询研究综述[J]. 计算机科学，2011，38(7)：18-23

[2] 金强. 基于 Hase 的 RDF 存储系统的研究与设计[D]. 杭州：浙江大学，2011

[3] Li L, Song Y. Distributed Storage of Massive RDF Data Using HBase[J]. Journal of Communication and Computer, 2011, 8(5):325-328

[4] Sun J, Jin Q. Scalable rdf store based on hbase and mapreduce [C]// 2010 3rd International Conference on Advanced Computer Theory and Engineering(ICACTE). IEEE, 2010:633-636

[5] Husain M F, Doshi P, Khan L, et al. Storage and retrieval of large rdf graph using hadoop and mapreduce[M]// Cloud Computing. Springer Berlin Heidelberg, 2009:680-686

[6] Myung J, Yeon J, Lee S G. SPARQL Basie Graph Pattern Processing with Iterative MapReduce [C] // Proceedings of the Workshop on Massive Data Analytics on the Cloud (MDAC'10). 2010:6-12

[7] Husain M, McGlothlin J, Masud M M, et al. Heuristics-Based Query Processing for Large RDF Graphs Using Cloud Computing[J]. IEEE Transactions on Knowledge and Data Engineering, 2011, 23(9):1312-1327

[8] Cheng J, Wang W, Gao R. Massive RDF Data Complicated Query Optimization Based on MapReduce [J]. Physics Procedia, 2012, 25:1414-1419

[9] Liu L, Yin J, Gao L. Efficient Social Network Data Query Processing on MapReduce[C]// Proc of the 5th ACM workshop. New York: ACM, 2013:27-32

[10] 张伟奇，张坤龙. 基于关系型数据库的 RDF 存储引擎[D]. 天津：天津大学，2011

[11] 吴刚. RDF 图数据管理的关键技术研究[D]. 北京：清华大学，2008

[12] 刘翔宇，吴刚. 基于 Prüfer 序列的 RDF 数据索引与查询[J]. 计算机学报，2011，34(10)

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

(上接第 226 页)

参 考 文 献

[1] 郑啸，罗军周，曹玖新，等. 基于发布/订阅机制的 Web 服务 QoS 信息分发模型[J]. 计算机研究与发展，2010，47(6)：1088-1097

[2] Al-Masri E, Mahmoud Q H. Investigating Web services on the World Wide Web [C]// Proc of the 17th Int Conf on World Wide Web. New York: ACM, 2008:795-803

[3] Wang Li-jun, Bai Xiao-ying, Zhou Li-zhu, et al. A Hierarchical Reliability Model of Service-Based Software System[C]// Proc of 33rd Annual International Computer Software and Applications Conference. Seattle, Washington, USA, 2009:199-208

[4] Ren Ying-xin, Gu Qing, Qi Jing-xian, et al. Reliability Prediction of Web Service Composition Based on DTMC[C]// Proc of third IEEE International Conference on Secure Software Integration and Reliability Improvement. Shanghai, China, 2009:369-375

[5] Ding Zuo-hua, Jiang Ming-yue, Abraham K. Port-Based Reliability Computing for Service Composition[J]. IEEE Transactions on Services Computing, 2012, 5(3):422-436

[6] Tsai T, Zhang D, Chen Y, et al. A Software Reliability Model for Web Services[C]// Proc of 8th IASTED International Conference on Software Engineering and Applications. Cambridge, MA, 2004:144-149

[7] Li Bi-xin, Fan Xiao-cong, Zhou Ying, et al. Evaluating the Reliability of Web Services Based on BPEL Code Structure Analysis and Runtime Information Capture[C]// Proc of of 17th Asia Pacific Software Engineering Conference. Sydney, Australia, 2010: 206-215

[8] Zheng Zi-bin, Wu Xin-miao, Zhang Yi-lei, et al. QoS Ranking Prediction for Cloud Services[J]. IEEE Transactions on Parallel and Distributed Systems, 2013, 24(6):1213-1222

[9] Zheng Zi-bin, Lyu Michael R. Personalized Reliability Prediction of Web Services[J]. ACM Transaction on Software Engineering Methodol, 2013, 22(2):1-25

[10] Zheng Zi-bin, Ma Hao, Lyu Michael R, et al. Collaborative Web Service QoS Prediction via Neighborhood Integrated Matrix Factorization[J]. IEEE Transaction on Services Computing, 2013, 6(3):289-299

[11] Silic M, Delac G, Srbljic S. Prediction of atomic web services reliability based on K-means clustering [C]// Proc of 2013 9th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering. Saint Petersburg, Russian Federation, 2013:70-80

[12] Bloomberg J. Web Services Testing: Beyond SOAP, Zap Think LLC, Sep[OL]. http://www.zapthink.com, 2002

[13] Chaari S, Badr Y, Biennier F, et al. Framework for Web Service Selection Based on Non-Functional Properties[J]. International Journal of Web Services Practices, 2008, 3(1/2):94-109

[14] Li Chang-bao, Cheng Bo, Chen Jun-liang, et al. A Discovery Approach for Web Services Composition Flows[C]// Proc of the 2010 IEEE Asia-Pacific Services Computing Conference. 2010: 700-706

[15] Chan K S M, Bishop J, Steyn J, et al. A Fault Taxonomy for Web Service Composition[C]// Proc of the 3rd International Workshop on Engineering Service Oriented Applications (WESOA'07). Springer LNCS, 2007:363-375

[16] Brüning S, Weißleder S, Malek M. A Fault Taxonomy for Service-Oriented Architecture[C]// Proc of the 10th IEEE High Assurance Systems Engineering Symposium. Plano, TX, 2007:367-368

[17] Mallick S, Kushwaha D S. LWSDM: Layered Web Service Discovery Mechanism [J]. Advanced in Information Sciences and Service Sciences, 2010, 2(3):25-31

[18] 陆文，徐峰，吕建. 一种开放环境下的软件可靠性评估方法[J]. 计算机学报，2010，33(3)：452-462

[19] Hwang S-Y, Lee C-H. Reliable Web service selection in choreographed environments[J]. Decision Support Systems, 2013, 54(3):1463-1476