**《计算机科学发展前沿》**

**课程报告**

姓名 王淑军

学号 2018216134

专业 计算机技术

班级 4班

天津大学计算机科学与技术学院

2018年 5 月

用于RDF数据的SPARQL 引擎综述

摘 要 SPARQL单机版与分布式版本的查询引擎都承诺可以在大规模RDF数据上快速获得所需的结果。 有些分布式引擎基于一些分布式框架，如 MapReduce; 其他人实施专有的分布式处理; 有些人依靠昂贵的数据预处理以及数据分区。 这些系统表现出各种各样的特性，对于用户来说是难以选择和理解的。 在这篇综述中我们对几十种最先进SPARQL引擎进行了调查，研究范围覆盖整个分布式RDF数据流程，并通过几个特征对它们进行分类。 然后， 根据论文中给出的结果，我们进行了权衡与对比。我们共从论文中选择了12个代表性系统并进行了关于预处理成本的实验评估， 查询性能，可伸缩性和工作负载适应性。

关键词 SPARQL, RDF, Survey

# 简介

近年来，随着大数据与人工智能的迅速发展, 基于“互联网+人工智能”模式，知识图谱（Knowledge Graph）作为一种智能、高效的知识组织方式，凭借能够迅速、准确地帮助用户从网络资源中查询到自己需要的信息等特点而受到广泛关注。知识图谱是语义网（Semantic Web[1]）在大数据环境下的延续，当前主要涉及语义网中的四个层次模块：RDF[9]，SPARQL[17]，RDFS和OWL。

资源描述框架（RDF）[2]作为知识图谱中一种重要的知识表达方式，以其结构简单、概念明确、表达力强的特点，受到了学术和工业界前所未有的重视。绝大多数开放知识图谱，例如DBpedia[10]、PubChemRDF[4]、Bio2RDF[5]和UniProt[6]等，都以RDF的格式呈现了上亿条关系数据，并且随着时间推移不断扩大。由于对于RDF的需求急剧增长，2014年W3C公布了RDF 1.1[7]作为用于Web上读写关联开放数据的框架。

随着RDF数据量的增长，计算量也在增加，索引和查询大型数据集的复杂性变得复杂并且极具挑战性。 单机RDF系统，如RDF-3X[3] 和gStore [17]，不能很好地扩展到复杂的查询网络规模的RDF数据。 为了克服这个问题，许多分布式SPARQL查引擎为此诞生。 他们利用无共享计算集群和 或者建立在分布式数据处理框架之上工作，如MapReduce，或实施专有的分配计算方法。 他们通过使用并行化查询执行来减少运行时间。查询通常涉及到 在每个Worker处理本地数据，与数据交错 Worker之间的交流。 分布式的主要挑战为系统正在向外扩展，受各种因素的影响 包括数据分区，负载均衡，索引，查询 优化和通信开销。 尽管有大量的分布式RDF系统，但是，实际应用中，由于信息有限很难比较他们的表现。使得用户对这些SPARQL查询引擎不能得到一个很直观的印象。不能直接理解他们，因此我选择写一篇来介绍SPARQL查询引擎以及其中包含的技术。

# 单机版SPARQL引擎存储方式

首先我们讨论单机版RDF系统的存储方法，因为事实上很多分布式版查询引擎依旧依赖单机版查询引擎。

**Triple Table:**用户只使用一个表，其中三列与主题，谓词和对象相对应，以存储RDF数据。每列创建一个索引，以便更快地进行连接评估。具有多个谓词的查询对应于大三元组上的一组自连接。由于昂贵的自连接，这种方法很难扩展。 RDF-3X[3]和Hexastore[5]通过使用一组涵盖S，P和O的所有可能排列的索引来降低此成本。这些索引存储为聚类B +树，并使用严格的字节级技术进行压缩。聚合索引，选择性直方图和有关频繁访问路径的统计信息用于选择成本最低的执行计划。对于大多数操作，它们的优化器使用保持顺序的合并连接，对于最后几个操作，它们使用散列连接。

**Property Table:**是RDF数据的更宽和平坦的表示[14]。该表的维度由主题数和不同谓词决定。表中的每个单元格都包含相应主题和谓词的对象值。由于稀疏表示，当唯一谓词的数量很大时，此表示具有高存储开销。而且，它不能代表多值属性，即使用相同谓词连接到不同对象的主题。 Jena2 [15]通过引入两个替表

示来解决这个问题，即聚类属性表和属性类表。 BitMat [20]提出了使用压缩的3维位矩阵的属性表的替代表示。位矩阵的每个维度对应于RDF三元组的一部分; S，P和O.每个单元表示存在由S，P，O位置定义的RDF三元组。对压缩数据执行查询，而不对中间连接结果进行处理。

**垂直分区:**是SW-Store提出的RDF数据的另一种表示。三元组表被分为n个表，其中n是不同谓词的数量。为每个谓词创建一个两列表，其中一行是通过谓词连接的一对主题 - 对象值。表在主题上排序，以更快地呈现主题查找和合并连接。此方法将多值属性存储为连续行，并且不存储NULL值。它为具有有界谓词的查询提供了良好的性能，但是，它需要扫描多个表来重建与单个实体相关的信息

# 分布式SPARQL引擎优化方法

## 分布式SPARQL查询处理

分布式环境下的SPARQL查询处理存在一些挑战，比如怎样分解复杂查询使计算本地化，怎样将高级查询转变为低级查询使其更快等等。而比起单机系统来说，分布式中需要额外考虑通信开销，并行策略以及分块和复制。中国人民大学杜小勇、陈晋川教授等人采用Hadoop和HDFS，这样使得SPARQL查询处理既可以受益于传统的分布式数据库技术，如分布式连接、流水线、负载均衡等，又从分布式框架中获得了良好的容错能力和高可扩展性。东北大学王国仁、袁野教授等人将SPARQL查询分解为多个子查询，然后进行子图相似匹配，利用了分布式环境的并行能力，并减少了通信。北京大学邹磊教授课题组则从查询日志中挖掘出频繁被查询的模式，从而在数据划分时保证RDF数据完整性，降低数据存储的冗余度。华中科技大学袁平鹏教授研究组的方法首先找出了覆盖全图的末端到末端路径集合，然后将该集合分成k份，每份作为一个分块存储到一台机器上。北京航空航天大学马帅教授等人利用建立虚拟网络的方法处理大量的映射，帮助图模式的匹配。

我们将近年流行的SPARQL分布式查询引擎根据查询策略（计算方式）大致分为以下三类：基于关系的SPARQL分布式查询引擎、基于图的SPARQL分布式查询引擎和基于混合方法的SPARQL分布式查询引擎。

## 分布式查询优化

查询优化是整个查询处理过程中关键的一步，在这个过程中，引擎生成了代价最小、最优的查询执行计划（Query Evaluation Plan,简称QEP）。例如对于I/Ocost + CPUcost + Communicationcost这样的代价模型，目的是要令其结果最小化。一般地，查询优化模块基本由三个方面结合而成：搜索空间、代价模型和搜索策略。

搜索空间是通过由查询语法树抽取得到的查询计划集合。从搜索空间中选取最优的查询计划，引擎可以获得物理操作的执行顺序。对于一条给定查询，它的搜索空间被定义为一个等价查询语法树。

因为不同构子的代价不同，所以连结顺序在查询优化中具有重要地位。对于高级查询，可替代的低级查询或者最优查询可以通过应用某些交换性和关联性规则而获得。但是对于另外一些高级查询，当查询优化器从一个较大的搜索空间中试图找到解时，优化带来的成本可能比不优化的情况更高。因此，查询优化器一般会采用减小搜索空间的策略。因为决定查询优化计划算法复杂度的两个主要因素是：（1）连结变量的度数;（2）三元组的总数[46]。[46]分别基于这两个因素提出了两个启发式方法来：（1）剪掉连结变量度数高的查询计划;（2）减少连结图的三元组数。另外，分布式中连结树的两种类型——线性树（Linear Tree）和矮树（Bushy Tree）的选取也需要考虑，明显并行矮树更有优势。

## 基于图遍历算法的查询优化

基于图遍历算法的查询优化以其突出的特点备受关注，它将整个查询表示为一个图，例如BGP被表示为以主语和宾语为结点，谓语为边，从主语到宾语的有向图。每条边的权重可以根据执行相应三元组模式得到的代价分析得出。[]首先生成一个静态查询执行计划，然后利用普里姆算法（Prim）或者埃德蒙兹算法（Edmonds）优化查询。[]中的方法对于不同的迭代不能保持分离的数据副本，新的绑定查询过程不能产生正确的结果。

[17]中SPARQL查询语句被表示为有向图，并使用遍历算法抓取结果。第一步进行查询执行，然后进行查询分析，排除掉对查询结果无用的类。第二步形成一个上下文图（Context Graph），作为执行模型。使用启发式方法分析更多在运行时就发现的模式，进一步减少从Web抓取的数据，从而在一定程度上减少查询执行时间，提高查询性能。

[24]提出了查询优化的混合方法，它结合了自顶向下和自底向上[28]的策略。与分开使用这两种策略相比，混合方法体现出了运行时间上的优势。它提炼旧的信息，为与查询相关的节点的排序提供参考。混合方法缓解了循环和基于流的带有同步哈希连结操作符的方法中的等待问题。

在开发分布式查询引擎时大多数研究者没有将研究重点放在查询优化上，学术界也没有提出一个系统的分布式查询优化方法，所以未来在分布式SPARQL查询优化方面仍然有较大的改进空间。其中一方面就是对于每个可能进行独立查询的节点，使用并行算法。这一点在现有的SPARQL引擎研究中还没有被提到过。

在采用了查询优化技术的SPARQL分布式查询引擎中，基于图的AdPart-NA[31]和TriAD[26]的性能尤为突出。其中AdPart-NA将数据分布作为工作负载，实现动态调控，实验表明它是减少端到端工作负载运行时间的最佳选择。

# 分布式SPARQL引擎简介

**SHARD**是在MapRecece之上实现的三重存储。整个RDF数据集存储在HDFS内的单个文件中，其中每一行代表一个主题的所有三元组。输入数据集在工作者之间进行散列分区，使得每个工作人员负责一组不同的三元组。 SHARD不使用索引;因此，在查询评估期间，它会扫描整个数据集。 SPARQL查询作为MapReduce迭代序列执行。每次迭代都负责一个子查询，而结果则连续地与后续迭代相结合。最后一次迭代负责过滤有界变量并删除多余的结果。

**HadoopRDF[16]**也使用HDFS将RDF数据存储为平面文件;复制和分发留给HDFS。与SHARD不同，HadoopRDF在HDFS上使用多个文件，每个谓词的文件类似于SW-Store的[24]垂直分区。 HadoopRDF还会根据显式和隐式类型信息将每个谓词文件拆分为多个较小的文件。最初，它将rdf：type文件划分为与不同对象数量一样多的文件。然后，为每个类型类型对象创建一组文件。

**TriAD[21]**采用基于主题和对象的轻量级散列分区。由于分区信息被编码到三元组中，因此TriAD具有对数据的完全位置感知并且在没有通信的情况下处理大量并发连接。它在每台机器上创建六个内存中的表格，每个机构对主题，谓词，对象进行排列。六个SPO排列分为两组;主题密钥索引（SPO，SOP，PSO）和对象密钥索引（OSP，OPS，POS）。然后，在不同的机器之间对这些索引中的每一个进行散列分区，并按字典顺序在每台机器内进行分类。这使TriAD能够在不同的SPO索引上执行高效的分布式合并连接。所有工作者同时执行多个连接操作符，这些操作符通过异步消息传递进行通信。在每个计算节点上，TriAD使用多线程并行地评估查询计划中的多个运算符。在评估分布式合并（散列）连接时，TriAD会分割一个（两个）关系，这不会保留中间结果的局部性。如果前一个连接的分片列不是当前连接列，这会导致TriAD重新分片中间结果。对于具有多个属性的大型中间结果，此成本很重要

**gStoreD[24]**是一个分布式分区不可知系统。它不会分解按原样发送给所有工作者的输入查询。 gStoreD通过计算每个worker的部分本地匹配来启动查询评估。这个过程取决于gStore [37]的修订版本，这是一个基于单机图的RDF引擎。然后，gStoreD组装部分匹配以构建交叉分区结果。 gStoreD允许两种装配模式;集中和分布式。集中式装配模式将部分结果发送到集中式站点，而分布式模式将它们并行组装在多个站点中

# 实验与论证

主要通过阅读论文来获得不同SPARQL查询引擎的表现。

AdPart-NA和TriAD在专业系统类别中表现最佳。AdPart-NA利用其哈希分布来解决L4和L5而无需通信。它也解决了L2没有通信但由于L2的低选择性而更慢。对于L2，AdPart-NA比TriAD和TriAD-SG快一个数量级，因为它通过单次扫描报告结果，然后通过利用哈希索引和右深度树计划进行哈希查找。与AdPart-NA相比，TriAD通过使用两个分布式索引扫描（每个基本子查询一个）来解决L2，然后是基于二进制搜索的合并连接，该连接仅对选择性查询有效。由于其修剪技术消除了通信，TriAD-SG在L6中优于TriAD和AdPart-NA。

在DREAM中，逐个查询地收集统计信息

基础，然后缓存以供将来查询。这导致第一次运行DREAM与其后续运行之间存在巨大的性能差异。

SHAPE和H-RDF-3X的性能优于MapReduce-Based系统，因为它们不需要通信。

RDF-3X（单机）在简单查询中表现良好

虽然复杂查询的速度要慢得多。 SHARD，H2RDF +和CliqueSquare遭受了MapReduce连接的昂贵开销。

# 总结

在本文中，我们提供了对最先进的RDF系统的实验评估。首先，我们对每个类别的系统进行分类和简要概述。然后使用大规模的真实和合成数据集，我们通过各种SPARQL查询对现有系统进行了大量评估，考虑了不同的性能因素，包括启动开销，发生的复制，查询性能和可伸缩性等…

参 考 文 献

1. A. Sch¨atzle, M. Zablocki, S. Skilevic and G. Lausen. S2RDF: RDF Querying with SPARQL on Spark. PVLDB, 9(10):804–815, 2016.
2. I. Abdelaziz, M. R. Al-Harbi, S. Salihoglu, and P. Kalnis. Combining Vertex-centric Graph Processing with SPARQL for Large-scale RDF Data Analytics. IEEE Transactions on Parallel and Distributed Systems (TPDS), 2017.
3. I. Abdelaziz, R. Harbi, S. Salihoglu, P. Kalnis, and N. Mamoulis. Spartex: a Vertex-centric Framework for RDF Data Analytics. PVLDB, 8(12):1880–1883, 2015.
4. M. Armbrust, R. S. Xin, Lian, et al. Spark Sql: Relational Data Processing in Spark. In Proc. of SIGMOD, 2015.
5. Z. Kaoudi and I. Manolescu. Rdf in the clouds: a survey. The VLDB Journal, 24(1):67–91, 2015.
6. P. Peng, L. Zou, M. T. ¨Ozsu, L. Chen, and D. Zhao. Processing SPARQL Queries over Distributed RDF Graphs. The VLDB Journal, 25(2):243–268, 2016.
7. R. Harbi, I. Abdelaziz, P. Kalnis, N. Mamoulis, Y. Ebrahim and M. Sahli. Accelerating SPARQL Queries by Exploiting Hash-based Locality and Adaptive Partitioning. The VLDB Journal, 25(3):355–380, 2016.
8. Bernerslee T, Hendler J and Lassila O. The Semantic Web [J]. Scientific American, 2001, 284(5):34-43.
9. RDF Primer: https://www.w3.org/TR/rdf-primer/. 2018-03-02.
10. DBpedia: http://dbpedia.org/. 2018-03-02
11. PubChemRDF: http://pubchem.ncbi.nlm.nih.gov/rdf/. 2018-03-02
12. Bio2RDF: http://bio2rdf.org/. 2018-03-02
13. UniProt: http://www.uniprot.org/. 2018-03-02
14. RDF 1.1: https://www.w3.org/TR/rdf11-new/. 2018-03-02
15. Mcguinness D L. OWL Web ontology language overview [J]. February, 2004, 63(45):990–996.
16. E. Prud'hommeaux and A. Seaborne. SPARQL query language for RDF. W3C recommendation, World Wide Web Consortium, 2008.
17. SPARQL1.1: https://www.w3.org/TR/sparql11-query/ 2018-03-02
18. Kaminski M, Kostylev E V and Grau B C. Semantics and Expressive Power of Subqueries and Aggregates in SPARQL 1.1 [C]. WWW 2016:227-238.
19. Kostylev E V, Reutter J L, Romero M and Vrgoc D. SPARQL with property paths [C] ISWC 2015: 3-18.
20. Goasdoue F, Kaoudi Z, Manolescu I, Quianeruiz J and Zampetakis S. CliqueSquare: Flat plans for massively parallel RDF queries [C]. ICDE 2015:771-782.
21. Hammoud M, Rabbou D A, Nouri R, Beheshti S and Sakr S. DREAM: Distributed RDF engine with adaptive query planner and minimal communication [J]. VLDB Endowment, 2015, 8(6):654-665.
22. Schätzle A, Przyjaciel-Zablocki M, Berberich T and Lausen G. S2X: Graph-parallel querying of RDF with GraphX [C]. VLDB Workshop 2015.
23. Harbi R, Abdelaziz I, Kalnis P, Mamoulis N, Ebrahim Y and Sahli M. Accelerating SPARQL queries by exploiting hash-based locality and adaptive partitioning [J]. VLDB J. 2016, 25(3):355-380.
24. Peng P, Zou L, Chen L and Zhao D. Processing SPARQL queries over distributed RDF graphs [J]. VLDB J., 2016, 25(2):243-268.
25. Schätzle A, Przyjaciel-Zablocki M, Skilevic S and Lausen G. S2RDF: RDF Querying with SPARQL on Spark [J]. VLDB Endowment, 2015, 9(10):804-815.
26. Damien Graux, Louis Jachiet, Pierre Genevès and Nabil Layaïda. SPARQLGX: Efficient distributed evaluation of SPARQL with Apache Spark. International Semantic Web Conference, 2016: 80-87.
27. Gombos G and Kiss A. P-Spar(k)ql: SPARQL Evaluation Method on Spark GraphX with Parallel Query Plan [C]. FiCloud. 2017.
28. [36] Bae M, Eum J, Kim D and Oh S. High Performance Query Processing for Web Scale RDF Data using BSP Style Communication and Balanced Distribution[C]. ICPP. 2017:201-210.
29. Mahmudul H and Srividya K B. RDF Data Storage Techniques for Efficient SPARQL Query Processing using Distributed Computation Engines [C]. IRI. 2018:323-330.
30. Matteo C, Michael F and Georg L. PRoST: Distributed Execution of SPARQL eries Using Mixed Partitioning Strategies. 2018:469-472.
31. Leng Y , Chen Z , Wang H , et al. A Partitioning and Index Algorithm for RDF Data of Cloud-based Robotic Systems[J]. IEEE Access, 2018,6:29836-29845.
32. Qiang X, Xin W, Li J, et al. StarMR: An Efficient Star-Decomposition Based Query Processor for SPARQL Basic Graph Patterns Using MapReduce[C]. Asia-pacific Web. 2018.