A Survey of Methods for Distributed RDF-store: How to Distribute and Query
分布式RDF数据库综述：如何分布和查询

詹才韬
中国地质大学计算机学院
武汉，中国
caitaozhan@163.com

***Abstract*—In this survey, methods for querying data in a distributed RDF-store will be reviewed.**

***Index Terms*—RDF-store; distributed; query;**

I. Introduction

A. 语义网和*RDF*

万维网发明者Tim Berners-Lee创造了语义网这个概念，他认为未来的互联网是一个web of data[1]，即一张连接数据的网络，或者说是数据互相连接构成了网络。依据W3C，语义网提供了一个框架，让数据在不同的应用、企业、和社区之间得到共享和重复利用[2]。正如AJAX技术是Web2.0的基石，语义网可能会大力推动Web3.0的发展。混搭网站(Mashup)从某种意义上实现了将不同地方的数据结合在一起的功能，然而不同的混搭网站之间没有统一的标准，而语义网的正是实现这个web of data的标准。

万维网上面拥有海量的数据，然而很多数据机器难以理解：对于一大段文字，人类可以轻松理解文字的内容，但是机器不可以。让数据以RDF的方式存储起来，解决了机器难以理解这些数据的难题，可以说RDF是语义网的基石。RDF[3]是一个数据模型：包括资源(resource)，属性(property)，陈述(statement)。资源可视为一个对象具体，一个资源拥有一个URI；属性是一类特殊的资源，用来描述资源之间的关系，属性也拥有URI；陈述用于描述资源所具有的属性，用三元组{Subject, Property, Object}来表示。

B. *RDF*数据库管理：从表结构到图结构

语义网让机器可以理解数据以及数据之间的连接，这样就可以使万维网能够智能地处理用户的请求。从某种意义上说，语义网可以将万维网进化成一个巨大而智能的“在线数据库”。数据终究是要存放在数据库里面，数据库存储RDF数据分为两大类：一是在关系型数据库的基础上进行扩展；二是抛开关系型数据库，从新的思路创建一种NoSQL数据库。文献[4] 中将以关系型数据库为基础来处理RDF数据的方法分为五类：简单三列表、水平存储、属性表、二元存储、全索引。这五种方法各有各自的缺陷：比如简单三列表的查询性能差，因为存在大量的自连接操作；再比如水平存储中存在大量的列，表往往很稀疏，存在大量空值，存在难以处理多值的问题，以及表结构更新麻烦等等。总体而言，这五种方法在空间和时间两个维度上，很难找到一个好的平衡点，往往是“按下葫芦浮起瓢”。文献[9]总结道，gstore是第一个通过子图匹配来回答RDF查询的研究。

在新千年之后，Web2.0的渐渐兴起，它产生的数量巨大，以及非结构化的数据，越来越让关系型数据库难以适应。关系型数据库在处理RDF数据上，显得力不从心。那么学术界和工业界需要找到解决问题的新思路：把资源看作一个图中的顶点，把属性看作顶点之间的边，那么顶点和边构成了数据结构中的图。以图为数据基础来构建存储和查询RDF 数据的数据库，叫做RDF-Store，它是一种特殊的图数据库。db-engine[5]网站中列举了在工业界中投入使用的18款不同的RDF-store，近三年来，业界对RDF-Store的关注度不断提升。在学术界也有团队在研究更新更强的RDF数据库，比如邹（Zou）等人的团队研究并开发出了gstore系统[6]。gstore系统以图模型来存储RDF数据，创造了VS-tree 用来对经过编码的实体和顶点建立索引，开发了一套与查询系统无缝衔接的过滤规则，最后引入了T-index用来优化查询效率。总体而言，由于图结构模型可以很自然的直接映射到RDF数据，是比表结构更自然、更合理的存储解决方案。

II. Preliminary

A. 并行性

从计算机体系结构的角度看计算机的发展历史，计算机发展的历史很大程度上就是不断追求更大并行性的历史：比如在指令系统使用流水线技术；

在I/O中使用通道技术；在操作系统中使用各种作业调度算法，多进程，还有单个进程里的多线程机制；在超级计算机中使用的集群架构等等。大规模科学计算和市场需求决定了计算机需要处理越来越庞大的数据，而处理越来越庞大的数据有两类解决方案：一是提高单个系统的处理能力(scale-up)，二是将处理需求分散在多个系统中并行处理(scale-out)。一般而言，业界会先绞尽脑汁提高单个系统的能力，在提高单个系统能力遇到瓶颈的时候，转换思路，去想办法利用多个系统的力量合并在一起来解决问题。这样的规律适合于计算机的很多领域，比如英特尔公司，它们在做奔腾芯片的单核时代，绞尽脑汁想提高单个处理器的能力，在2004年左右遇到瓶颈难以突破。最终从2006年开始开创了酷睿的多核时代。在追求更大并行性的思路之下，学术界和工业界做过很多并行数据库系统的尝试。大体上来说，并行数据库系统是建立在MPP和集群等多处理器系统基础上的数据库系统，在这篇文章中不做详细讲述。

B. 分布式

分布式和并行性这两个名词背后代表的领域有很多相通和交叉的地方，但是也有区别。并行(parallel)的反义词是串行(serial)，并行性强调的是在同一时刻或者同一时间间隔内做多件事情。分布式(distributed)的反义词是集中式(centralized)，分布式强调的是在不同的地点(可以相隔很远，不是在同一个机房内)做多件事情。并行数据库往往在同一个机房内，通过高速局部网络或者交换机连接，而分布式数据库往往通过公有的互联网连接，一般而言，分布式处理中往往包含并行处理的技术，但是反之不亦然[7]。文献[8]中提到早在1981年，System R^*[Williams et al. 1981]，SDD-1[Bernstein et al. 1981]，Distributed Ingres [Stonebraker 1985]这三个数据库系统已经在做分布式查询的尝试了。然而分布式查询数据库系统过于领先于那个年代，它们没有取得商业上的成功，原因有两方面：一是当时的通信技术无法给分布式查询足够的支撑；二是当时的企业可以在不使用分布式数据库的情况下解决现实需求，比如通过寄送磁带，磁盘，甚至直接人工交换纸质文档，还句话说，上个世纪八九十年代初并没有使用分布式数据库的强烈需求和动力。然而计算机行业变化剧烈，从上个世纪末开始，分布式查询的巨大需求已经成为现实，通信和网络技术的快速发展也给分布式查询提供了物理基础。分布式数据库和集中式数据库相比，从性价比、可扩展性、软件集成、遗留软件集成、适应新型应用等方面均有优势[8]，最终体现在提供分布式数据库管理解决方案的厂商在许多方面拥有更强的市场竞争力。究竟为什么需要分布式数据库系统？毕竟数据库诞生的重要初衷，就是为了集成企业的数据，并对数据提供集中的、可控制的存取和查询功能。教材[10]中做出回答：分布式处理更好的和今天分散企业的组织结构相对应；更为重要的是，现在的计算机的许多应用本身就是分布式的，典型的例子是基于互联网的应用；根本原因在于使用一种分而治之的办法对应现在所面临的大规模数据管理问题。下面的文章主要探讨分布式的RDF查询。

III. 分布式RDF-STORE：如何分布和查询

以下将分布式RDF-store分为3类，分别介绍这3类系统是如何做到分布式的，还有如何进行分布式查询。每一个分类都对一个典型代表做的介绍。

A. Partition-Based

1) 基于分区的方法*:* 这种方法的思想来自于关系型数据库中的水平分区。在关系型数据库的中，假设对一张表100万行的大表进行水平分区，可以将之拆分成十张10万行的小表，小表和大表的属性一样。水平分区(horizontal partitioning)和分片(shard)十分接近，不过也有区别，资料[12]中提到，水平分区是对在一个关系模式的在一个数据库服务器里的一张表进行拆分，而分片可以是是对一个关系模式的在多个数据库服务器里的多个表进行拆分。在基于分区的方法中，一般要把大的RDF图分为多个分块(fragment)，不同的分块在不同的节点中。在查询的时候，往往把查询分解为多个子查询，在节点中进行子查询然后将子查询的结果汇集起来。GraphPartition[Huang et al. 2011], WARP[Hose and Schenkel, 2013], Vertex-block[Lee and Liu, 2013], 和Partout[Galarraga et al., 2012]这四个系统均采用了基于分区的思想进行设计。接下来将以Partout[11]为典型的例子，介绍如何进行分区，然后再分区之后进行查询。

*2) Partout*架构*:* 如图1，它拥有一个中央调停者(coordinator)，和若干个分布的主机(host)。调停者计算如何对RDF数据进行分区，不同的分区存放在不同的主机里面。调停者处理SPARQL查询，并把查询任务分配到各个主机上，然后对各个主机的结果进行汇总。

*3) Partout*分区*:* a）初始化：需要有一个查询负载(Query Load)，这可以是来自RDF数据库系统的若干实时SPARQL查询语句，也可以是依据从访问RDF数据的应用中预测模拟出来的SPARQL查询语句; b）预处理：抽取查询负载中查询语句里的三元组，对抽取的结果进行规范化和匿名化; c）分块：对预处理过后的查询负载的结果做进一步处理抽取，得到对RDF数据库里面的所有三元组的若干

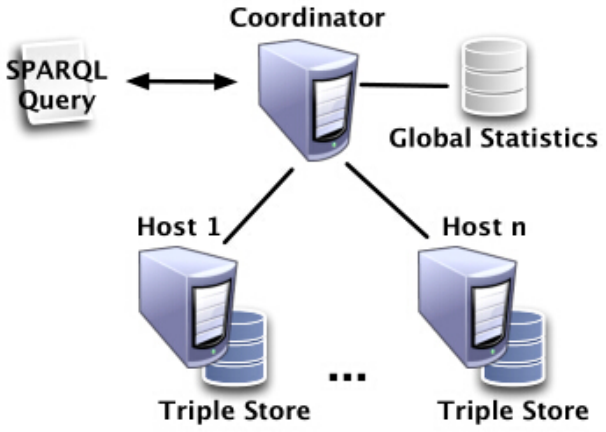

Fig. 1. Partout系统结构，来自文献[11]

约束，使用简单谓词对RDF数据分成多个分块; d）分配：把多个分块分配到各个主机中。

进一步讲解：a）查询负载（很多SPARQL查询语句）是分块的核心依据，即RDF数据库中真实的SPARQL查询语言的特点和应用程序访问的RDF数据的特点决定了Partout如何对数据库里面存储RDF数据进行分块；b）对查询负载进行规范化和匿名化即从中抽取出有用、相关的信息，把无用或者价值低的信息剔除掉。具体做法是查询语句中出现频率很低的URL和文字转换成变量；c）利用经过预处理的查询负载产生对三元组的约束，即简单谓词(simple predicate)，比如三元组和常量之间的大小比较关系等。这些简单谓词相当于是构成了许多规则，利用这些规则对数据库中的RDF数据进行分块；d）把产生的分块进行分配，分配的两个目标是：使在同一个SPARQL查询语句里涉及到的分块在同一个主机里面，使不同主机之间的负载均衡。具体分配的算法采用了贪心算法的思想。

*4) Partout*查询*:* RDF数据存储在各个主机的RDF数据库里面，里面索引的建立基于RDF-3X[13]。调停者的作用是接受用户的查询请求，将查询分解为若干子查询，然后分配到到各个主机中。分解和分配请求主要依赖于调停者拥有的一个全局统计文件(相当于是元数据，即描述数据的数据)，该文件包含了各个主机的信息，比如哪个主机分配到那个分块，分块里面有哪一些数据，再比如一些特殊用途的字符字典和各个主机的关于负载和分区的统计信息。Partout创建了一个评估模型，用来评估查询花费的代价，并且会依据评估结果，使用一些启发式策略对查询的分解和分配策略进行优化。在对各个主机的子查询的结果进行汇总的时候，大量使用了归并连接。

B. Hadoop-Ecosystem-Based

1) 基于*Hadoop*生态系统的方法*:* Apache Hadoop [14]软件库是一个框架，允许在集群服务器上使用简单的编程模型对海量数据进行分布式处理。Hadoop项目中有两个著名的基石性模块，分别是HDFS和Hadoop MapReduce。HDFS是一个分布式的、可扩展的、可移植的文件系统[15]，它的容错性很高，并提供了高吞吐量的数据访问。MapReduce[17,18,19] 是一个编程模型，能实现在集群上面通过并行和分布式算法处理和产生海量数据。SHARD [Rohloff and Schantz, 2010] , HadoopRDF [Husain et al., 2011] , EAGRE [Zhang et al., 2013] and JenaHBase [Khadilkar et al., 2012] 均使用了基于Hadoop生态系统的方法。以下以SHARD为典例，介绍这类方法如何工作。

*2) SHARD*分布式*:* SHARD系统直接使用了HDFS进行管理数据，RDF数据分布在计算节点上，直接利用HDFS现成、成熟的技术对分布的数据进行管理。RDF数据存储的时候，不采用经典的基于图的数据结构，比如邻接表，而是使用平面文件(flat file)格式：每一行是由包含相同subject 的三元组{subject, property, object}构成。subject不重复出现，只出现这一行的最前面，不同的行拥有不同的subject，行与行之间没有结构化关系，这样是为了配合适应HDFS文件系统。不对三元组做索引，这样是为了配合适应MapReduce并行计算框架。

*3) SHARD*查询*:* 整个查询过程基于MapReduce计算框架。MapReduce 自带并行计算属性，适合分布式应用场景，然而MapReduce只能提供简单粗暴的数据操作技术：先把数据拆分成键-值对，然后对拥有相同键的值做操作。SHARD为了能在MapReduce的基础上回答复杂的SPARQL查询，做了如下工作[16]：查询的时候，不断对SPARQL语句中的查询子句（WHERE里面的查询条件）进行迭代，渐进式的一个接着一个满足所有查询子句。每一次迭代使用一个MapReduce操作来满足一个查询子句。每次迭代满足一个查询子句之后会产生中间结果，和之前一次次迭代的累积的中间结果连接合并在一起。图-2展示了这个过程。假设一个SPARQL语句有n个查询子句，那么就会迭代n次MapReduce

i_{th}-Map：识别出三元组数据中所有满足i_{th}查询子句，并且在前$i-1$个查询子句中出现过的变量，这些变量构成Key；而对于前$i-1$个查询子句中没有出现过的变量构成Value。这样Map阶段产生了许多Key-Value对构成中间结果。

i_{th}-Reduce：对中间结果进行连接操作。对拥有相同Key的Key-Value对进行连接操作。重复n次MapReduce操作直到所有查询子句都被执行，所有变量都得到赋值并且满足查询条件

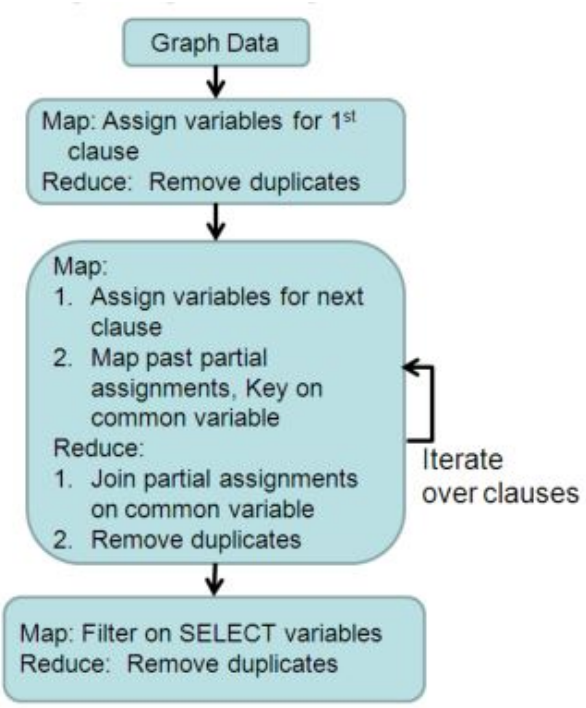

Fig. 2. SHARD的迭代式MapReduce查询，来自文献[16]

迭代完成之后，最后一次MapReduce操作负责过滤出SELECT子句所需要的结果。

C. Federated-Based

1) 基于联邦制思想的方法*:* 在政治领域，联邦制(federalism)是由两个以上的政治实体（共和国、州、邦）结合而成的一种国家结构形式。以美国为例，每个州拥有自己独立的法律，拥有部分管理内部事务的权利，但是不拥有独立的主权，部分权利让给美国联邦政府。受到这种思想的启发，业界创造出了联邦数据库，它和一般的分布式数据库有一点重要的区别是：联邦数据库的每一个成员是自治的[20]。这说明这种类型的数据库并没有一个强悍有力的统一调度的中央节点，而是把各个自治的（而且很有可能是异构的）数据库虚拟地集成在一起。这种类型有DARQ [Quilitz and Leser, 2008], Q-Tree [Harth et al., 2010], FedX [Schwarte et al., 2011]。以下以FEDX为典例做介绍。

*2) FEDX*查询*:* 图-3是FEDX联邦制查询的架构。

第一，解析。原始的SPARQL查询语句将被解析，转换成FEDX系统需要的内部表达。

第二，选择数据源。对于一次查询，有可能只需要访问联邦数据库系统中的一部分自治数据库端点，那么就需选择到底是哪一部分自治端点。为了确定是哪一些端点，FEDX不需要使用类似于Partout中使用的全局统计文件，即元数据，描述数据的数据，而是对每一个查询子句（WHERE里面

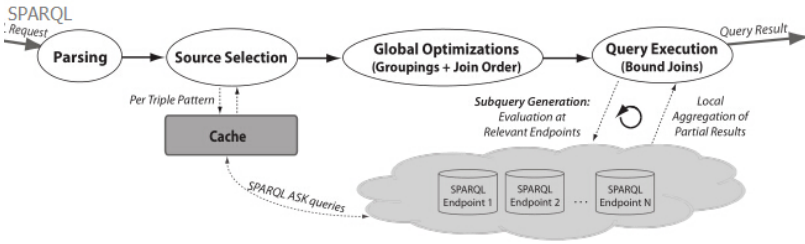

Fig. 3. FEDX的Federated Query Processing Model，来自文献[21]

的三元组条件）都直接远程询问各个自治端点，问它们是否拥有该查询子句需要的数据。重复远程咨询相同的问题比较浪费，FEDX使用了缓存来缓解这个问题。

第三，连接排序。查询子句的顺序很大程度上影响查询性能。FEDX提出了一种基于启发式的代价评估模型，优化排序的顺序使用了基于变量计数的技巧。

第四，分组。把多个查询子句分为一组，如何分组FEDX提出了自己的定义[21]。一个组的多个查询子句一次性传送给数据源（而不是一个接着一个传送）做联合查询可以优化查询性能，因为可以节省数据在网络上传递的时延。这种分组的思想，有点类似于上面使用MapReduce方法的SHARD中的Map过程，Map过程有点像是基于Key进行分组。第三和第四的作用均为优化查询性能。

第五，查询执行，连接优化。在此过程中产生子查询，其在相关的端点进行评估和执行，然后在端点本地聚合，在远端的虚拟数据库系统连接合并子查询结果。FEDX提出了一种基于SPARQL UNION的分组优化方法，来减少子查询连接过程中的时间。

IV. Conclusion

本文介绍了分布式RDF数据管理中的核心问题“如何分布数据，如何在分布的环境下查询”。阐述了分布式RDF数据查询的三种方法——基于分区、基于Hadoop生态系统、和基于联邦数据库思想的方法。

Acknowledgment

感谢北京大学的教师邹磊。很荣幸可以在国科大暑期学校听他的一门课《知识图谱的数据管理》，这份综述就是这门课的作业。本人对于数据库、NoSQL、以及对数据管理的有着浓厚兴趣，很高兴从这门课上面学习了很多，在写综述的过程中也锻炼了研究的能力。

References

[1] https://www.ted.com/talks/tim_berners_lee_on_the_next_web
[2] https://en.wikipedia.org/wiki/Semantic_Web

[3] http://www.w3.org/standards/techs/rdf#w3c_all
[4] Lei Zou, Yueguo Chen, A Survey of Large-Scale RDF Data Management, Comunications of CCCF
[5] http://db-engines.com/en/ranking/rdf+store
[6] Lei Zou, Jinghui Mo, Lei Chen, M. Tamer ?zsu,Dongyan Zhao: gStore: Answering SPARQL Queries via Subgraph Matching. PVLDB (2011),4(8): 482 493
[7] http://web.cs.wpi.edu/~cs561/s12/Lectures/4-5/ParallelDBs.pdf
[8] Dodald Kossmann. The State of the Art in Distributed Query Processing, ACM Computing Surveys, Vol. 32, No. 4, December 2000, pp. 422 – 469.
[9] Charu Aggarwal. NeMa: Fast Graph Search with Label Similarity, VLDB: 181-192(2013)
[10] M.Tamer Ozsu, Patrick Valduriez. *Principles of Distributed Database Systems, Third Edition*, 2011
[11] Galarraga, L., Hose, K., and Schenkel, R. Partout: a distributed engine for efficient RDF processing.(2014)
[12] https://en.wikipedia.org/wiki/Shard_(database_architecture)#Shards_compared_to_horizontal_partitioning
[13] Thomas Neumann, Gerhard Weikum. RDF-3X: a RISC-style Engine for RDF. 2008
[14] http://hadoop.apache.org/
[15] https://en.wikipedia.org/wiki/Apache_Hadoop#Hadoop_distributed_file_system
[16] Rohloff, K. and Schantz, R. E. High-performance, massively scalable distributed systems using the mapreduce software framework: the shard triple-store. (2010)
[17] https://en.wikipedia.org/wiki/MapReduce
[18] http://www.datacenterknowledge.com/archives/2014/06/25/google-dumps-mapreduce-favor-new-hyper-scale-analytics-system/
[19] David DeWitt. MapReduce: A major step backwards. (2008)
[20] https://en.wikipedia.org/wiki/Federated_database_system#Autonomy
[21] Schwarte, A., Haase, P., Hose, K., Schenkel, R., and Schmidt, M. Fedx: Optimization techniques for federated query processing on linked data. In The Semantic Web - ISWC 2011 - 10th International Semantic Web Conference, Bonn, Germany, October 23-27, 2011, Proceedings, Part I, pages 601 – 616.