# Hybrid Partitioning and Distribution of RDF Data

by

**Trupti Padiya**

A Thesis Submitted in Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy

to

Dhirubhai Ambani Institute of Information and Communication Technology

April 2018

DA-IICT

## Declaration

I hereby declare that

i) The thesis comprises my original work towards the degree of Doctor of Philosophy in Information and Communication Technology at DA-IICT and has not been submitted elsewhere for a degree,

ii) Due acknowledgement has been made in the text to all other material used.

Trupti Jayantilal Padiya

## Certificate

This is to certify that the thesis work titled "Hybrid Partitioning and Distribution of RDF Data" has been carried out by Trupti Padiya (201221002) for the degree of Doctor of Philosophy in Information and Communication Technology at Dhirubhai Ambani Institute of Information and Communication Technology under my supervision.

Prof. Minal Bhise
Thesis Advisor

To

***My Family***

# Acknowledgements

*"Showing gratitude is one of the simplest yet most powerful things human can do for each other " – Randy Pausch*

Foremost, I would like to express my sincere gratitude to my Ph.D. advisor Prof. Minal Bhise for her continuous guidance and support throughout my Ph.D. work. I am indebted to my supervisor for her constant motivation during my tough times. I deeply appreciate her kindness, patience and constant encouragement that kept me going.

I am grateful to my RPS (Research Progress Seminar) committee members: Prof. Maniklal Das and Prof. Sourish Dasgupta, Synopsis committee members: Prof. Anish Mathuria and Prof. P. M. Jat for their inputs and suggestions which helped me broaden my horizon progressively.

I sincerely thank all my colleagues at Database Research Group, DA-IICT. I thank Sandeep Vasani, Mohit Pandey, Bhavik Shah, Prashant Rajkotiya, Jai Jai Kanwar, and Anubha Jain. I am thankful to DA-IICT for lab resources. I express my heartfelt gratitude to Dixita and Tanvina for their constant assurance, love, and support. I will always cherish the moments spent together. I thank all my friends outside campus especially Disha for her best wishes and constant encouragement.

Finally, I am heartily grateful to my parents who raised me with utmost love and care. It is because of their moral support and motivation I am able to pursue my dream despite adversity. My Ph.D. became possible because of invariable support of my husband Dhrumil who constantly motivated me and helped me deal with situations assertively.

# Abstract

RDF is a standard model by W3C specifically designed for data interchange on the web. RDF was established and used for the development of the semantic web. However, nowadays RDF data is being used for diverse domains and is not limited to the semantic web. Tremendous increase is witnessed in RDF data due to its applications in various domains. With growing RDF data it is vital to manage this data efficiently. The thesis aims at efficient storage and faster querying of RDF data using various data partitioning techniques.

The thesis studies the problem of basic data partitioning techniques for RDF data storage and proposes the use of hybrid data partitioning in centralized and distributed environment as a part of the solution to store and query RDF data. The dissertation emphasizes on efficient data storage and faster query execution for stationary RDF data. It demonstrates basic data partitioning techniques like PT (Property Table), BT (Binary Table), HP (Horizontally Partitioned Table), and use of MV (Materialized Views) over BT. Even though basic data partitioning techniques outperforms TT (Triple Table) they suffer from various performances issues.

The thesis gives a detailed insight into advantages and disadvantages of basic data partitioning techniques. Consequently, it proposes hybrid solutions for data partitioning by exploiting the best of available techniques. It proposes three hybrid data partitioning techniques namely DAHP (Data-Aware Hybrid Partitioning), DASIVP (Data-Aware Structure Indexed Vertical Partitioning) and WAHP (Workload-Aware Hybrid Partitioning). DAHP and WAHP are a combination of PT and BT whereas DASIVP combines structure index partitioning with BT. DAHP and DASIVP consider a data-aware approach and WAHP considers a workload-aware approach. Data-aware approach stores RDF data based on how the data is related to each other in the dataset and workload-aware approach stores RDF data based on how the data that is queried together. The thesis demonstrates detailed evaluation of query performance and data storage for all the data partitioning techniques. Query performances for these data partitioning techniques are evaluated in terms of QET (Query Execution Time). It calculates break-even point for all the data partitioning techniques. Hybrid data partitioning techniques have shown significant improvement over basic data partitioning techniques. A set of metrics is devised which can help to consider the suitability of given data partitioning technique for a RDF dataset.

RDF data has increased to a point where it is difficult to manage this data on a single machine. It is necessary to distribute the data on different nodes and process it in parallel so that efficient query performance can be achieved. Data distribution and parallel processing of queries may generate many intermediate results which will involve communication among nodes. It becomes necessary to minimize inter-node communication among nodes in order to achieve faster execution of queries. This work presents a solution to manage RDF data in a distributed environment using a proposed hybrid technique. The solution aims at efficient RDF data storage and faster query execution by minimizing inter-node communication among nodes.

Finally, the dissertation proposes DWAHP (Workload-Aware Hybrid Partitioning and Distribution) which exploits query workload and distributes data among nodes. DWAHP has two phases: Phase 1 considers Workload-Aware Hybrid Partitioning technique which generates workload-aware clusters consisting of PT and BT. Phase 2 considers a distribution scheme that distributes data among nodes using an *n-hop Property Reachability Matrix*. DWAHP Phase 1 helps in reducing number of joins, as it keeps the data which is queried together as a separate partition. DWAHP Phase 2 helps in diminishing inter-node communication among nodes with the use of an *n-hop Property Reachability Matrix*. The thesis demonstrates DWAHP and analyzes its query performance in terms of query execution time, query cost, storage space, and inter-node communication. Queries on RDF data mostly involve star and linear query patterns. DWAHP manages joins such that it is able to answer all linear and star queries without inter-node communication. DWAHP is compared with a state-of-the-art solution. It outperforms the state-of-the-art solution with 72% of faster query execution time, 61% of reduced query cost by occupying less than one-third of storage space.

Increase in RDF data is witnessed as RDF data is being used in diverse domains. Discussed partitioning techniques can be utilized for various RDF stores. Data-aware RDF stores can be utilized for applications when data characteristics are known and workload-aware RDF stores can be utilized when data queries are known in advance.

# Table of Contents

# List of Acronyms and Symbols

## Acronyms

| | |
|---|---|
| BT | Binary Table |
| CPU | Central Processing Unit |
| DAHP | Data-Aware Hybrid Partitioning |
| DASIVP | Data-Aware Structure Indexed Vertical Partitioning |
| DBLP | Digital Bibliography & Library Project |
| DWAHP | Workload-Aware Hybrid Partitioning and Distribution |
| FOAF | Friend of A Friend |
| GB | GigaByte |
| HP | Horizontally Partitioned Table |
| JARS | Join-Aware distributed RDF Storage |
| MB | MegaByte |
| MMDB | Main Memory Database |
| MPT | Multi-Valued Property Threshold |
| MV | Materialized View |
| OPR | One Property Retrieval |
| PT | Property Table |
| QET | Query Execution Time |
| RAM | Random Access Memory |
| RDF | Resource Description Framework |
| SR | Structuredness Ratio |
| TET | Total Execution time |
| TT | Triple Table |
| URI | Uniform Resource Identifier |
| W3C | World Wide Web Consortium |
| WAHP | Workload-Aware Hybrid Partitioning |
| XML | eXtensive Markup Language |

## Symbols

| | |
|---|---|
| BEA | Break-even Analysis |
| D | Number of insert/update/delete operations |
| MC | Materialized view creation time |
| MR | Materialized view refreshment time |
| $N_{MP}$ | Number of multi-valued properties |
| $N_P$ | Number of properties |
| $N_Q$ | Number of queries |
| $N_T$ | Number of triples |
| $N_{UP}$ | Number of unique properties |
| $QET_{MV}$ | Query Execution Time for Materialized View |
| $QET_{TT}$ | Query Execution Time for Triple Table |
| $R_{D1}, R_{D2} \ldots R_{Dn}$ | RDF data stores |
| $o$ | Object |
| $p$ | Property |
| $s$ | Subject |
| c | Number of clusters (Algorithm Complexity) |
| n | Number of records (Algorithm Complexity) |
| p | Number of properties (Algorithm Complexity) |
| q | Candidates in Query Property Basket (Algorithm Complexity) |
| s | Candidates in Subject Property Bin (Algorithm Complexity) |

# List of Figures

# List of Tables

# Chapter 1

## Introduction

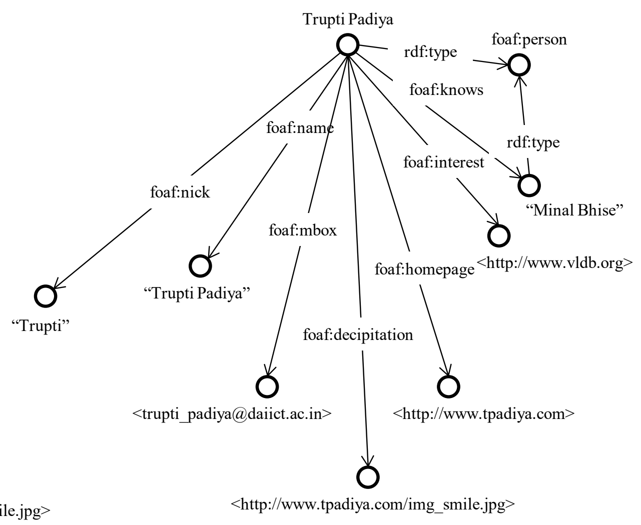
RDF (Resource Description Framework) [**1**] is a standard model for data exchange on web established by W3C (World Wide Web Consortium). It encodes semantic relationship between things. RDF data is represented as a triplet of the form subject-predicate-object symbolized as <*s,p,o*>. Subject signifies a resource, predicate is property or characteristic of the resource, and object denotes value of the property. For example, Trupti is a Student at DA-IICT which in triple form is represented as <Trupti, Student, DA-IICT>. To be more specific, RDF uses URIs to make RDF statements. URI (Uniform Resource Identifier) is a global unique id for every entity on web. Another way to represent RDF structure is using graphs. It is a labeled graph structure where subject and predicate are nodes connected by edges which represent properties.

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
<#Trupti Padiya>
   a foaf:Person ;
   foaf:name "Trupti Padiya" ;
   foaf:mbox <mailto:trupti_padiya@daiict.ac.in> ;
   foaf:homepage <http://www.tpadiya.com> ;
   foaf:nick “Trupti" ;
   foaf:depiction <http://www.tpadiya.com/img_smile.jpg> ;
   foaf:interest <http://www.vldb.org> ;
   foaf:knows [
      a foaf:Person ;
      foaf:name "Minal Bhise" ]
```

**(a) RDF Data**

```
<Trupti Padiya> <foaf:name>  “Trupti Padiya”
<Trupti Padiya> < foaf:mbox> <mailto:trupti_padiya@daiict.ac.in>
<Trupti Padiya> <foaf:homepage> <http://www.tpadiya.com>
<Trupti Padiya> <foaf:nick>  “Trupti”
<Trupti Padiya> <foaf:depiction> <http://www.tpadiya.com/img_smile.jpg>
<Trupti Padiya> <foaf:interest> <http://www.vldb.org>
<Trupti Padiya> <foaf:knows> "Minal Bhise"
```

**(b) RDF Data Triples**



**(c) RDF Data Graph**

**Figure 1.1: RDF Data Representations**

Collection of RDF statements essentially characterizes a set of concepts or a particular kind of knowledge that give you an idea about things, and the relationship between things, which is formally referred to as an ontology. FOAF [**2**] is an ontology describing person and their details, their activities, their relationship with other persons and so on. It basically describes a social network which is expressed in RDF. Figure 1.1 depicts various ways to represent FOAF (Friend Of A Friend) data. Figure 1.1 (a) is a RDF XML representation, Figure 1.1 (b) represents RDF data as set of triples and Figure 1.1 (c) represents RDF graph.

RDF initially was introduced for the development of the semantic web. However, RDF is now used in a broader spectrum. For e.g. DBpedia [**3**] and Yago [**4**] extracts information from Wikipedia and stores them in RDF format in order to support structural queries over Wikipedia. RDF is also used in the field of biology which has generated many large RDF collections like Bio2RDF and UniProt [**5**]. RDF is nowadays being used in diverse domains. It is a building block for the intelligent web, which is used as a machine-readable standard to exchange heterogeneous data on the web. Thus storing, retrieving and managing this data remains an important aspect of developing interactive applications.

## 1.1.RDF Data Storage

RDF data storage is of vital importance to materialize intelligent web. Several research efforts are made for efficient storage and management of RDF data. Data management has been put into execution either by centralized approach or distributed approach. Researchers have viewed RDF data storage and management with different viewpoints, irrespective of these approaches. RDF data storage is accomplished using these viewpoints: 1) RDF data is stored in a relational database system where the data is stored as relational tables. 2) RDF data is stored in its native graph pattern. When data is stored as a graph, it maintains its original representation of RDF data. However querying these graphs is complex due to tree traversal and graph-pattern matching, as the cost of subgraph matching is NP-complete [**6**]. It is intricate to extract frequent patterns from graph databases. It is much more difficult to scale graph data in a distributed network compared to scaling simpler data models [**7**]. On the contrary, traditional database techniques such as indexing, and query processing/optimization can be utilized to address this issue. Therefore with increasing RDF data, we feel classic relational database techniques can be used to store the data, so that one can take advantage of years long research on efficient storage and querying, transactions support, locking, indexing, query optimization, security and other features of database management systems. This dissertation hence discusses RDF data storage considering a relational perspective.

## 1.2. Motivation

RDF is a pillar of the intelligent web which makes the data understandable not only to humans but also to machines. It is a linked structure that has gained wide acceptance and is used for real-life applications. Lots of tools have been developed for purposes like triple stores, inference engines, converters, search engines, middleware, semantic web browsers, development environments and semantic wikis to name a few [**3**][**8**][**9**][**10**]. Linked Data [**11**] connects structured data on the web. It connects data, information, and knowledge on the intelligent web using RDF. RDF is also used as a standard for information exchange for IoT (Internet of Things) [**12**][**13**]. IoT is an environment where every participating object, say a person, place or a thing are provided with a unique identifier in order to endow with the ability to transfer information. It is a heterogeneous data originating from various devices with different standards. RDF can assist in merging data even if the original schemas differ, and it particularly supports the development of schemas over the period of time. RDF allows the data to be shared across different applications [**1**].

Because of higher utilization of RDF data, a rapid increase in RDF data is observed over the period of time. W3C Wikipedia page of RDF dumps [**14**] is flourished with trillions of triples. Linked Open Data is linking more than 9960 datasets out of which, some datasets contain more than 50 billion triples [**15**]. This rapid increase in RDF data makes RDF data management an open research issue. Storing, retrieving and managing this proliferating RDF data is of major concern. Efficient management of this RDF data is essential in order to fabricate robust RDF storage systems. With regard to building highly interactive applications, it is crucial to manage this RDF data competently, especially in terms of data storage and query execution time. This dissertation aspires to solve some of the major issues in RDF data management. Especially efficient RDF data storage and faster query execution using a set of proposed techniques, which can help in the development of interactive applications.

## 1.3. Objectives

This thesis deals with stationary RDF data and uses relational perspective to efficiently store and manage the data. The research work addressed in this dissertation essentially focuses on leveraging query performance using various partitioning strategies. It has following objectives:

- Explore basic RDF data partitioning techniques
- Develop appropriate techniques to store RDF data efficiently for faster query execution
- Develop solution for distributed systems using proposed approaches

## 1.4.Goals and Contributions

The comprehensive goal of this dissertation is to advance research into the area of RDF data management by enhancing previous attempts to manage RDF data. It proposes new techniques to manage RDF data efficiently. Particularly it concentrates on efficient storage and faster querying of RDF data. Fundamentally this thesis gives minute details for a set of proposed techniques for RDF data management. First, it details on basic RDF data partitioning techniques. Second, it examines the tradeoffs of basic partitioning methods and proposes three hybrid data partitioning techniques. Third, it proposes a method for partitioning RDF data and then distributes this partitioned data on different nodes for efficient query execution. Our proposed method executes queries such that inter-node communication among nodes is minimized. This dissertation reflects an empirical research for efficiently storing and querying RDF data in both centralized and distributed environment using proposed techniques.

### 1.4.1. Basic RDF Data Partitioning

This section discusses available partitioning strategies. A set of experiments are performed for existing partitioning techniques, which let us get a hands-on and detailed insight into their working mechanisms. Basic partitioning methods include triple table [**16**], properly tables [**17**], binary tables [**18**], and materialized views [**19**] . Experimental work using basic partitioning methods and their break-even analysis are reported in [g] [h] [i]. Experimental work done during this thesis for basic RDF data partitioning is discussed in Chapter 3 in detail.

### 1.4.2. Hybrid RDF Data Partitioning

A series of experiments using basic RDF data partitioning helped us gain a detailed insight into advantages and disadvantages of existing approaches, and lead us to propose hybrid mechanisms to partition RDF data. We combine existing approaches by considering their advantages and eliminating their disadvantages to a certain extent. Proposed hybrid RDF partitioning techniques include: 1) DAHP (Data-Aware Hybrid Partitioning) [e] 2) DASIVP (Data-Aware Structure Indexed Vertical Partitioning) [f] and 3) WAHP (Workload-Aware Hybrid Partitioning) [c] [d]. All hybrid partitioning mechanisms are discussed in Chapter 4.

### 1.4.3. Distribution of Partitioned RDF Data

RDF data is increasing rapidly and it has reached a point where processing and managing this data on a single machine has become difficult. We propose DWAHP (Workload-Aware Hybrid Partitioning and Distribution) [a] which is discussed in Chapter 5. The technique performs hybrid RDF data partitioning and manages query joins, such that queries can be executed faster. On top of that, it distributes hybrid partitioned data across nodes on a network such that, inter-node communication can be minimized and efficient query performance can be gained. In short, the proposed technique DWAHP partitions and distributes RDF data such that, not only it manages joins but also reduces inter-node communication among nodes to execute queries efficiently.

### 1.4.4. Comparison with State-of-the-art

We compare our system with JARS [**20**], a state-of-the-art mechanism to manage RDF data in distributed environment. JARS eliminates inter-node communication for star query patterns and minimizes inter-node communication for linear queries. DWAHP is compared with JARS in terms of query performance parameters: query execution time, query cost, storage space, and inter-node communication. The proposed system DWAHP has demonstrated on an average 72% of better query execution time with 61% of reduced query cost by occupying less than one third storage space compared to the state-of-the-art solution. DWAHP manages joins efficiently, and it is able to answer frequent query patterns like star and linear queries without inter-node communication.

## 1.5. Dissertation Outline

This dissertation focuses on using partitioning techniques to store stationary RDF data efficiently in a relational system. It proposes three hybrid partitioning techniques namely DAHP, DASIVP, and WAHP. It utilizes these techniques and using a proposed distribution scheme, the thesis proposes a technique called DWAHP to manage RDF data efficiently in a distributed environment. The thesis is organized as follows. Chapter 2 highlights related work. Chapter 3 demonstrates basic data partitioning techniques to manage RDF data. Chapter 4 discusses proposed data-aware and workload-aware based partitioning techniques. Chapter 5 demonstrates a solution to manage RDF data in a distributed environment and compares it with a state-of-the-art solution followed by conclusions and future work.

# Chapter 2

# Related Work

RDF has been extensively used for the development of the semantic web. However, RDF nowadays is not limited to semantic web, as it is being used for various diverse domains. With rapid increase in RDF data, efficient management of this data has always remained a primary challenge. Researchers have provided numerous solutions to manage this data efficiently over the period of time. Relational databases are thought-out as one of the prevalent solutions for managing different kinds of data. In 1998, Tim Berners Lee discussed RDF and E-R model for their distinct characteristics [**21**]. This work suggested RDF model to relational model by mapping record to a RDF node, a field to a RDF property and table cell as object-value. There are several tools that have been developed to bridge the gap between RDF and relational representation [**22**]. Continuous attempts are made to find solutions to manage RDF data efficiently using a relational system [**23**]. This chapter discusses some of the major contributions in RDF data management.

## 2.1.Basic RDF Data Partitioning

This section discusses basic RDF data partitioning schemes proposed over years. Almost every research work uses some of these techniques at the base to partition data in a RDF store.

### 2.1.1. Triple Table

RDF data description generally has a triple format representation, which includes a triple of the form <subject, predicate, object> or <*s,p,o*>. Triple Table (TT) is a flat representation of these triples in a three column relational table. Systems like SesameSQL92SAIL[**16**], Oracle [**24**], 3Store [**25**] used this triple table approach. Figure 2.1 illustrates triple table. Figure 2.1 (a) depicts a triple table prepared for a set of RDF triples using row strings and Figure 2.1 (b) depicts an example of a triple table using a generic representation where S1, S2 ... Sn represents subjects, P1, P2 ... Pn represents predicates and O1, O2 … On represents object values. All the examples in this thesis are demonstrated using this generic representation.

| Subject | Predicate | Object |
|---|---|---|
| Trupti Padiya | First name | “Trupti” |
| Trupti Padiya | Interest | “databases“ |
| Trupti Padiya | Institute | “DA-IICT“ |
| Disha Pathak | First name | “Disha” |
| Disha Pathak | Interest | “Photography“ |
| Dhrumil Shah | First name | “Dhrumil” |
| Dhrumil Shah | Institute | “CISCO |

**(a) Triple table example using row strings**

| Subject | Property | Object |
|---|---|---|
| S1 | P1 | O1 |
| S1 | P2 | O2 |
| S1 | P3 | O3 |
| S2 | P1 | O4 |
| S2 | P2 | O5 |
| S3 | P1 | O6 |
| S3 | P3 | O7 |

**(b) Triple table example using generic representation**

**Figure 2.1 Triple Table**

The main advantage of TT is, it has a very general representation and it is flexible too. It is easier to map a triple directly to a three column table. However, storing a large number of triples can result in performance issues. Storing a large number of triples in a single table will make the table huge. Moreover, real-life queries will require many self-joins which will result in slower query execution and there is a possibility to run out of memory capacity [**18**]. Due to these serious performance issues triple table doesn’t scale well.

### 2.1.2. Indexing

Researchers proposed solutions to use indexing mechanism to overcome these issues of a triple table in order to execute queries faster. These alternative solutions made use of extensive indexing on triple tables [**26**]. For every such system, there exists only a single triple table but it is indexed extensively. For example systems like Virtuoso [**27**], Hexastore [**28**] and RDF-3X [**29**] use extensive indexing techniques. Virtuoso stores quads by binding a graph *g* with every triple. Conceptually it is a four-column table indexed by *g,s,p,o* and *o,g,p,s*. Hexastore uses a sextuple indexing mechanism. This six indexing scheme includes *spo, sop, pso, pos, ops,* and *osp*. On top of that they use dictionary encoding techniques and instead of storing entire URIs it stores shorten version of keys. Hexastore needs five times storage space than the actual triple table. RDF-3X also considers extensive indexing using *spo, sop, pso, pos, ops*, and *osp*. It stores triples in compressed B+ trees which are sorted lexicographically by values of six different permutations. Indexing mechanisms although perform efficiently, because of required excess storage space they limit the scalability of data. TripleBit [**30**] tries to reduce storage space to some extent. These extensive indexing mechanisms offer the advantage of indexed based faster query processing. However, its major drawback is overhead of excess storage space and updating multiple indexes.

## 2.1.3. Property Tables

Property table scheme has been proposed by Jena [**17**]. Systems like DB2RDF [**31**] and 4store [**32**] also uses property table technique. Property table is a wider, flat representation which is similar to traditional relational schemas. Property tables can be of two types: Clustered property table and Property class table [**33**]. Clustered property table contains a group of properties that tend to be defined together. Property class table exploits "type" property and group subjects in the same table which are associated with similar type. Figure 2.2 depicts a clustered property table.

| **Subject** | **Property** | **Object** |
|---|---|---|
| S1 | P1 | O1 |
| S1 | P2 | O2 |
| S1 | P3 | O3 |
| S2 | P1 | O4 |
| S2 | P2 | O5 |
| S3 | P1 | O6 |
| S3 | P3 | O7 |

**(a) Triple Table**

| **Subject** | **P1** | **P2** | **P3** |
|---|---|---|---|
| S1 | O1 | O2 | O3 |
| S2 | O4 | O5 | NULL |
| S3 | O6 | NULL | O7 |

**(b) Property Table**

**Figure 2.2: Property Table**

The advantage of using property table approach is subject-subject joins are diminished. Hence a query will have fewer joins or in some cases, it can be just a single table scan. Besides this advantage, it suffers from many serious drawbacks. There could be a significant number of null values in the table and can result in a sparse table for wider tables which will result in space overhead. RDF data use multi-valued attributes and it is awkward to express multi-valued attributes in property tables. Clustering similar properties will need some clustering algorithm and it may require re-clustering as there can be alterations in data characteristics over time. Apart from the subject-subject join, property tables may not help other join types.

### 2.1.4. Data Partitioning

These limitations of discussed approaches were addressed by using partitioning technique for RDF stores. SW-Store [**18**] introduced use of vertical partitioning for RDF data. C-store [**34**] and IBM DB2 [**35**] implements this approach for managing RDF data. It is a fully decomposed model [**36**]. A triple table is partitioned into *n* two column tables where *n* is number of properties. Each table contains two columns (*s*, *o)* for every property. Hence it is called binary table.

| **Subject** | **Property** | **Object** |
|---|---|---|
| S1 | P1 | O1 |
| S1 | P2 | O2 |
| S1 | P3 | O3 |
| S2 | P1 | O4 |
| S2 | P2 | O5 |
| S3 | P1 | O6 |
| S3 | P3 | O7 |

**(a) Triple Table**

**P1**

| **Subject** | **Object** |
|---|---|
| S1 | O1 |
| S2 | O4 |
| S3 | O6 |

**P2**

| **Subject** | **Object** |
|---|---|
| S1 | O2 |
| S2 | O5 |

**P3**

| **Subject** | **Object** |
|---|---|
| S1 | O3 |
| S3 | O7 |

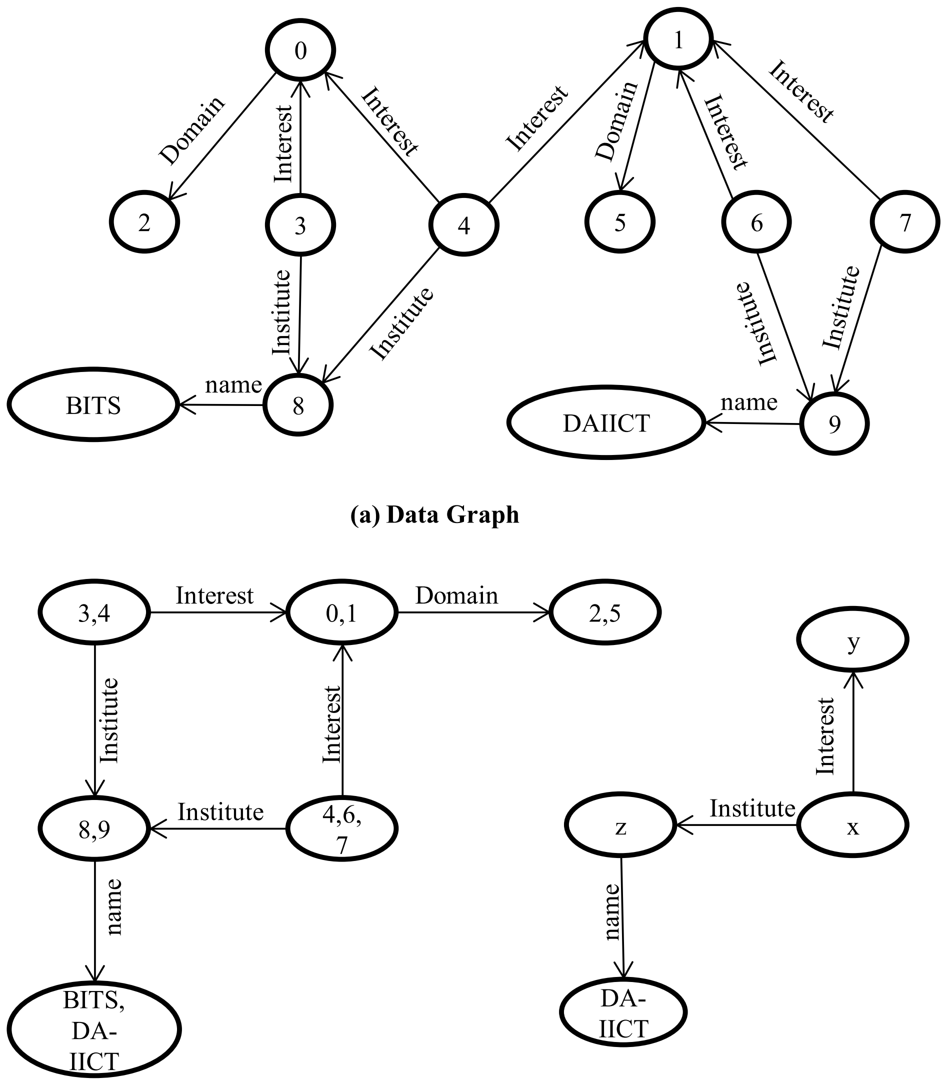
**(b) Binary Tables**

**Figure 2.3: Binary Tables**

Figure 2.3 shows binary tables formed from sample triple table. This approach offers many advantages. Subjects that do not define a particular property are simply eliminated and hence it solves the problem of null value. Compared to property tables it will not need any clustering algorithm. All data related to a particular property is residing in the same table. So it may result in fewer union clauses. Besides these advantages, this approach suffers from few drawbacks. It requires increased number of joins. If multiple properties are queried, one needs to access and join all the required binary tables for the queried properties. Insert operations can be slow for binary tables as insert operations for same subject need to access different partitions. In a dynamic scenario where data keeps on changing data insertions can be tricky.

## 2.2.Hybrid RDF Data Partitioning

Continuous research efforts lead to Hybrid solutions to manage RDF data. A technique is proposed which uses PIG (Parameterized Structure Index) for data partitioning and query processing that exploited underlying structure to improve management of RDF data [**37**] [**38**] [**39**].



**(a) Data Graph**

**(b) Index Graph** **(c) Query Graph**

**Figure 2.4 Structure Index Partitioning**

It uses structure index for RDF. At querying time, structure index is used for structure-level query processing to identify data that matches the query structure. And then this structure level processing is combined with data-level processing for join operations and retrieval of data. Basically, an index graph is prepared using the basic data graph and the query graph is mapped on the index graph to retrieve data. Figure 2.4 shows an example of Structure Index Partitioning approach. It shows the data graph, index graph and a query graph for query "Retrieve people who belong to DA-IICT and are interested in Database". Structure Index Partitioning maps query graph on index graph, followed by query pruning in order to get a final answer of the query. Though it outperforms few of the discussed basic RDF storage techniques, it suffers from the issue of scalability as it is a graph-based method. Moreover, in dynamic environment query structure may change and which may result in increased complexity of this solution.

A RDF data-centric approach [**40**] is proposed to manage RDF efficiently. It uses a combination of binary tables and property tables to store RDF data. It basically decreases the need for number of joins in the query plan and also tries to keep the number of null values at minimum. It offers a simple solution but in some scenarios, it has shown moderate performance as data-centric tables contained some redundant values.

## 2.3. Distribution of RDF Data

RDF data has increased to a point where it is difficult to manage it on a single system. For few datasets, it has accelerated to a level where it becomes necessary to partition the data across multiple nodes to manage this data proficiently. Researchers have contributed many solutions to partition and distribute RDF data. This section briefs some of the prominent solutions to manage RDF data in a distributed environment.

### 2.3.1. Federated Systems

Federated systems are proposed to manage RDF data. FedX [**41**] offers one such federated query processing mechanism for heterogeneous linked data sources. It targets to minimize the number of intermediate requests among federated nodes and proposes a mechanism to group triples such that they can be evaluated at a single endpoint. It employs an optimization technique for efficient query processing. DARQ [**42**] is also a federated system to query RDF data and it also discusses use of semi-join same as FedX to compute join between intermediate results.

### 2.3.2. Adaptive RDF Data Storage Systems

Though this dissertation deals with stationary data, it will be suitable to highlight some work that has been carried out to manage RDF data in a dynamic environment, where data keeps on changing over time. Cerise [**43**] is an adaptive RDF data store for a distributed environment. It dynamically adapts to underlying storage and query execution based on query history. It tries to co-locate data on the same node that are accessed together so that disk and network latency can be reduced. Though it outperforms few of the techniques to manage RDF data in a distributed environment, it has a certain cost involved for co-locating data for read-only workloads. In some cases, this cost is compensated with performance improvement of queries. However, this approach is more suitable for query patterns which are repeated over time. DREAM [**44**] is another adaptive query planner for RDF data management in a distributed environment. DREAM doesn't partition data like other techniques, instead it partitions only queries. It offers a general paradigm for various query patterns and its adaptive scheme automatically runs queries on different machines depending on query complexities. Although it outperforms Scalable RDF Storage [**45**], RDF-3X [**29**] and other distributed systems, it can certainly accelerate its performance by considering parallelizing queries at different granularities. PHD-store [**46**] doesn't assume any pre-placement of data and does not require pre-partitioning of data. It answers queries using distributed semi-join algorithms and adapts dynamically to query load by progressively redistributing frequently accessed data. It uses propagating hash distribution approach to solve the problem.

### 2.3.3. Data-Aware RDF Data Partitioning Systems

Various RDF data partitioning mechanisms for distributed systems are also proposed by many researchers to efficiently handle this data. TriAD [**47**] is a shared-nothing RDF engine which is based on asynchronous message passing. It combines join ahead pruning with a locality based, horizontal partitioning of RDF triples. It uses METIS [**48**] to partition RDF graph. Each partition is distributed among nodes and every node maintains in-memory vectors of triples. All vectors are permuted for subject, predicate, and object and hence it has six such vectors. It also maintains a summary graph which has partition information. Partout [**49**] extends the notion of minterm predicates and use it as a fragmentation unit to partition data.

Scalable RDF Storage [**45**] has proposed a scalable querying technique to leverage single node data management. It is a horizontally scalable RDF storage system that makes use of METIS partitioner [**48**]. It introduces data partitioning and placement technique on a Hadoop framework where query execution is parallelized between nodes. They introduce a parameter PWOC (Parallelization WithOut Communication) to check for a query is parallelizable and can be executed without the need of message passing. Another RDF partitioning approach for scalable RDF store [**50**] is also a Hadoop based framework which automatically partitions RDF data and approximates a solution to place partitions on different nodes such that redundancy can be reduced. It actually proposes this framework to make a good tradeoff between data redundancy and efficient query execution. It combines advantages of both RDF-3X [**29**] and MapReduce framework. However, it needs considerable data preprocessing and additional data structures to perform data distribution among nodes.

JARS [**20**] pursue a join aware RDF data distribution approach. It is a relational system based approach to distribute RDF data and hence doesn't hold expensive MapReduce jobs. It uses dual-hash partitioning and dual indexing on the triple table. It uses MD5 message digest algorithm to generate a hash value for placement of a triple on its respective node. This algorithm is applied for both subject and object. After triple placement on their respective node using the MD5 algorithm, it indexes subject table and object table. The subject table is indexed using *pos, pso, osp,* and *spo* while object table is indexed by *pos, pso, sop,* and *ops*. With this strategy, it eliminates inter-node communication for queries having subject-subject joins and it is minimizing inter-node communication for queries having subject-object joins. Even though it achieves significant performance improvement over few distribution schemes, it suffers from several issues. It stores triple twice and on top of that, it indexes them using four permutations of subject, predicate, and object. Consequently, it consumes more than double storage space than it is required to store the actual data. While updating triples, one need to re-calculate its MD5 hash value for both subject [**51**] and object, place it on respective node and then re-index on their corresponding subject and object table. This complex set of operations makes it difficult to handle updates on such systems.

### 2.3.4. Workload-Aware RDF Data Partitioning Systems

In parallel, researchers also have utilized workload information in order to answer frequent queries efficiently for a distributed environment. For example system like WARP [**52**] exploit workload information to replicate and distribute data among nodes. It avoids expensive MapReduce jobs and performs cost-aware optimization of arbitrary queries. Another workload based RDF data fragmentation and allocation [**53**] offers a solution by considering frequent access patterns of the queries. It proposes a local pattern-based fragmentation strategy by considering the entire workload and satisfying their storage constraint. Researchers have worked on systems that separate hot and cold data based on the frequency of query workload and then manages hot data in main memory databases [**54**] [**51**] [**55**]. ClusterRDF [**56**] [**57**] also exploits workload information and generates RDF templates. These templates are frequent query patterns identified for the system. It allocates these templates such that the data that is queried together can appear on the same node in order to reduce messaging passing between nodes. While performing data distribution among nodes, it fragments these templates by keeping a check on structure query patterns to cluster them solely. At the time of clustering these fragments, it groups fragments with strong affinities considering a storage threshold. Though ClusterRDF outperforms other storage systems, there can be significant issues that can arise while using this approach. Depending on number of nodes and number of multi-valued relationships in RDF dataset, it is possible that more number of fragments may be required. With increase in fragments, the idea of reduced message passing gradually dissolves. It is necessary to use an acceptable threshold value to reduce data redundancy at the time of clustering fragments. To cluster fragments, it needs to traverse through all triples so that fragments with strong affinity can be kept together. This in itself is a complex process and on top of that changes in RDF data may make it even more difficult to allocate data on nodes. Query graph pattern needs to be defined at an earlier stage to use this approach. So it may be difficult to run ad-hoc queries on this system.

## 2.4. Research Issues

Researchers have proposed numerous solutions with different perspectives to manage RDF data. However, few research issues are still open. This section discusses some of the open research issues in the area of RDF data management followed by the research issues addressed in the dissertation and some recent work.

### 2.4.1. Open Research Issues

Researchers have put in abundant efforts to manage RDF data in a relational system and a lot of issues have been solved adequately. However following research issues are still open:

- **Strategies for RDF data storage and query processing**

Basic RDF data stores like triple table [**58**], property table [**17**], binary tables [**18**], use of materialized views [**18**] have been proposed. Triple table though being simple suffers from issues of a large number of self-joins. Property table contains potential null values in large number making it a sparse matrix, and vertical partitioning face problem of more number of joins and update issues. On the other hand, various indexing mechanisms [**27**][**28**][**29**] are also proposed for efficient handling of RDF data. However indexing RDF store faces the problem of complex operations like updating indexes and usage of extra memory space to store those indexes. Researchers have experimentally evaluated RDF storage systems [**59**] and have proposed workload-aware storage and query processing strategies [**60**] on centralized systems. Many RDF storage systems like [**17**][**61**][**62**] rely on MMDB (Main Memory Data Bases) [**63**] for faster retrieval of data. However managing RDF data using in-memory systems may not be feasible in the scenarios where there is no strong hardware support. Besides storage mechanisms, researchers also have worked on various query processing strategies which make use of different query optimization techniques [**64**] to efficiently process RDF queries.

Although numerous solutions have been proposed to address the issue of RDF data storage and query processing, they suffer from some of the performance issues as discussed in section 2.1 and section 2.2, which makes a room for further research in this domain and find solutions to overcome some of the issues addressed.

- **RDF data management in distributed environment**

With rapid increase in RDF data, it has become difficult to manage this data on a centralized system. Hence researchers have proposed solutions to manage RDF data in a distributed environment. Various RDF data partitioning systems [**47**][**45**][**50**][**20**][**57**][**65**] have been devised for distributed systems. Mostly these systems provide solutions for minimum message passing in order to achieve faster query execution. Scalable RDF storage system

[**45**] is able to minimize inter-node communication and parallelize queries using a metric called PWOC. It is able to perform better than other hash partitioning systems like SHARD. However, it doesn't consider query workload for partitioning RDF data. JARS [**20**] is another RDF data storage system that achieves significant performance improvement. However it stores triples twice and on top of that, it indexes them using four permutations of subject, predicate, and object. This eventually results in consumption of more than double storage space than it is required to store the actual data. It also involves a huge complexity during update operation due to its data distribution strategy. ClusterRDF [**57**] exploits workload information and distributes RDF data using generated templates. All of these discussed systems have managed minimum message passing between nodes by either using certain partitioning strategies or by managing joins for query processing. However, these techniques do suffer from some issues as discussed in detail in section 2.3 which motivates to carry out further research in this domain. RDF systems generally deal with queries having subject-subject and subject-object joins. There is a necessity for such systems that can not only minimize inter-node communication but also manage both subject-subject and subject-object join efficiently.

- **RDF data management and applications**

RDF has been used increasingly in real life applications like triple stores, inference engines, converters, search engines, middleware, semantic web browsers, development environments and semantic wikis. Various new technologies are looking forward to using RDF as a standard because of its flexible representation and its capability for interoperable systems. For example, researchers feel that RDF is important for the successful Internet of Things [**66**] [**9**][**67**]. IoT is an environment where every participating object, say a person, place or a thing are provided with a unique identifier in order to endow with the ability to transfer and process information. This data is heterogeneous in nature and there is a growing need to manage this data together using standard protocols. Although there is no shortage of standards, these standards remain disconnected and hence researchers propose utilization of RDF to offer connectivity. RDF has features that assist in data merging even though the underlying schema differ, and it particularly supports the evolution of schemas over time without requiring all the data patrons to be changed. Use of RDF allows structured and semi-structured data to be mixed, exposed, and shared across different applications [**68**].

Furthermore, RDF data management techniques can be applied to IoT data in order to make IoT applications interactive [**69**] [**70**] [**71**].

### 2.4.2. Issues Addressed in Dissertation

This dissertation provides solution to some of the issues discussed in section 2.4.1. Following are the highlights of research issues addressed:

- **Proposed hybrid approaches to store RDF data for faster query processing**

We propose three hybrid approaches to leverage query performance and offer a solution for a relational based RDF data storage system. DAHP (Data-Aware Hybrid Partitioning) and DASIVP (Data-Aware Structure Indexed Vertical Partitioning) are two proposed data-aware approaches. They focus on how the data is organized in the dataset and offers a storage plan using a hybrid approach in order to execute queries faster. WAHP (Workload-Aware Hybrid Partitioning) is a workload-aware approach which offers a solution to store RDF data considering the actual query workload and aims at leveraging frequent queries. Further, we carry out an insightful comparison of all the approaches by the means of empirical evaluation. Our proposed approaches are able to outperform state-of-the-art systems and overcome some of the issues discussed in section 2.1 and 2.2.

- **Proposed method to manage RDF data in a distributed environment**

We utilize those ascertained hybrid approaches and propose DWAHP (Workload-Aware Hybrid Partitioning and Distribution) to manage RDF data in a distributed environment. Basically, we use workload-aware approach to partition RDF data. Subsequently, we make use of data-aware approach for data allocation and data distribution among nodes. This blend helps us achieve an order of magnitude better performance compared to state-of-the-art mechanisms. Our approach not only minimizes inter-node communication but also manages subject-subject and subject-object joins such that linear and star queries can be answered without inter-node communication. Further, we apply our proposed techniques for IoT data which can direct towards making IoT applications interactive.

## 2.5.Recent Work

This section highlights a few most recent work carried out in the area of RDF data management. Some new data storage strategies also have been formulated. For example system like RDF-4X [**72**] provides a scalable solution to manage RDF data. It introduces storage methods and indexing mechanisms that scale billions of quads across multiple nodes. A Google patent [**73**] proposes methods and apparatus for querying relational data store using schema-less queries. A distributed RDF store [**74**] introduces a bulk-loading algorithm for triples using MapReduce framework. This bulk-loading algorithm for loading billions for triples makes it faster to respond to queries. RDFox [**75**] is an in-memory scalable centralized RDF store for static RDF data. Researchers also have proposed in-memory RDF dictionary [**76**] for dynamic and streaming RDF data.

Indexing mechanism suffers from storage space issues and researchers are working on it to minimize storage space occupied by indexes. Double chain-star [**77**] is an indexing scheme for fast processing of RDF queries. It reduces chain star patterns as it involves multiple self-joins. Another indexing scheme [**78**] is proposed for a large scale semantic web data. It creates a subject-object index and a separate predicate index to minimize total size of the index. v-RDFCSA [**79**] is a self-indexing solution that provides version-based queries on top of compressed RDF archives. It reduces space-time up to35 - 60 times over current solutions. RIQ [**80**] proposes a solution for faster processing of queries for RDF data that is stored as quadruples. It employs a decrease-and-conquer strategy for faster query processing. Instead of indexing entire RDF dataset, it identifies similar groups of RDF data and creates an index on each group separately. Ontop [**81**] is an open-source ontology-based access system. It allows querying relational data sources through a conceptual representation of their corresponding interest domain. Along with a theoretical foundation, it utilizes query rewriting technique and extensive optimization techniques. S2RDF [**82**] proposes a mechanism for faster query processing. It uses is relational partitioning schema which employs semi-join based preprocessing to efficiently minimize query input size. Cliquesquare [**83**] is an optimization approach for evaluation of RDF queries in massively parallel environment. It minimizes the number of joins encountered on the root to leaf plan. It actually proposes optimization algorithms to build flat query plans.

## 2.6.Summary

This chapter of literature survey discusses some of the major contributions in the area of RDF data management. It details about various available data partitioning solutions to manage RDF data using data and workload information. It discusses proposed solutions for storing and querying RDF data efficiently in a centralized and distributed environment. It talks about open research issues and issues addressed in this dissertation.

# Chapter 3

## Basic Partitioning for RDF Data

RDF data maintains its original representation when stored in its native graph pattern. However querying these graphs is complex due to tree traversal and graph-pattern matching. It is difficult to deal with large data graphs as graphs may be ideal if nodes are fairly balanced [**7**]. There can be super-nodes with many edges and there can be nodes with very few edges which make it challenging to scale. On the other hand, scaling data on simpler models like relational models is quite a worked out solution. On top of that, relational database techniques such as indexing, and query optimization can be utilized to handle scaled data efficiently. Therefore with increasing RDF data, we feel classic relational database techniques can be used to store the data. So that one can take advantage of years-long research on efficient storage and querying, transactions support, locking, indexing, query optimization, security and other features of database management systems. This chapter discusses basic RDF data storage techniques using relational systems. It further discusses the experiments performed for basic RDF data partitioning techniques. An empirical evaluation is carried out for a triple table, property tables, binary tables, horizontally partitioned tables, and materialized views over binary tables. Additionally, these basic storage techniques are also evaluated for scaled data. This experimental study has helped to gain hands-on to manage RDF data in a relational database system.

### 3.1. Triple Table (TT)

Structurally RDF is in the triple format as <subject, predicate, object>. These triples can then be stored in a relational database with a flat representation of a three-column schema, known as TT (Triple Table) [**16**]. The advantage of this approach is that it has a very simple and flexible data representation. However, it can suffer from serious performance issues, as there is only a single RDF table. Almost all interesting queries will involve many self-joins over this table. Hence it becomes important to improve RDF query performance as data is represented in RDF format on the web.

## 3.2. Partitioned Tables

Data partitioning deals with the logical arrangement of data in the database. There exist mainly two types of data partitioning techniques: Vertical and Horizontal. Vertical partitioning divides a table into multiple tables that contain fewer columns. Horizontal partitioning is a horizontal division of a table which contains fewer rows. Vertical Partition for RDF data may result in property tables and binary tables. Horizontal partition on RDF data results in horizontally partitioned tables.

### 3.2.1. Property Tables (PT)

PT can be visualized as a vertical partitioning of RDF data. It is a de-normalized RDF table, which is physically stored in a wider, flattened representation [**17**]. It is quite similar to traditional relational schemas which have subject and its required predicates as columns. Now if there are queries which have more joins, then this query will be very slow to execute, because as the number of triples in the collection scales, the RDF table may well exceed the size of memory, and each of these filters and joins will require a scan or index lookup. Real world queries involve much more joins, which complicates selectivity estimation and query optimization, and limits the benefit of indices [**33**]. Even though the representation is very simple and flexible, there are several issues with PT which includes a large number of nulls, difficulty of storing multi-valued attributes, and complex joins. Besides this, a clustering algorithm must be used to find a group of properties to form a property table. These clustering algorithms may increase complexity and on top of that, if the data characteristics change over time, reclustering might become necessary.

### 3.2.2. Binary Tables (BT)

Using vertical partitioning, another way to store RDF data is to create a table for every property. There will be as many tables as number of properties in the dataset. Each table will have two columns, one for the subject and other for the object for their corresponding property [**18**]. As it has two columns it is referred to as BT (Binary Tables). Subjects that do not have properties defined for them will not appear in BT and hence BT solves the problem of null value. Compared to property tables it will not need any clustering algorithm. Besides these advantages, this approach suffers from few drawbacks. Some of the queries may require larger number of joins. If multiple properties are queried, one needs to access and join all the required binary tables for the queried properties.

### 3.2.3. Horizontally Partitioned Tables (HP)

Horizontal partitioning divides a table into multiple tables where each table contains the same number of columns, but fewer rows. Horizontal partitioning can be applied on a triple table, property table, and binary tables. In this chapter horizontal partitioning is applied on the triple table. Basically, they are smaller triple tables considering number of rows. Every HP (Horizontally Partitioned table) contains different rows.

### 3.2.4. Materialized Views (MV)

A view is a database object which contains the query result. For example, it may be a local copy of data which is located remotely, or it can be a subset of the rows and/or columns of a table or join result, or it can be a summary based on aggregations of a table's data [**84**]. In short, a view is a virtual table representing the result of a database query. An MV (Materialized View) [**19**] takes a different approach, in which the query result is cached as a concrete table that may be updated from the original base tables from time to time. Materialized views are most often used in data warehousing or business intelligence applications where large fact tables with thousands of millions of rows are queried. For the work represented here, materialized views are considered in conjunction with binary tables, which is expected to improve query performance.

## 3.3. Experiments

The presented work experimentally study and evaluate query performance for RDF data using various basic RDF storage techniques. Initially, RDF data is converted to triple form. This RDF data in the triple form is mapped to a three-column table called triple table. The dataset is partitioned vertically and horizontally. Vertical partitioning results in property tables and binary tables. A set of clustered property tables are prepared and there exists a binary table for each property in the dataset. Horizontally partitioned tables are created based on subjects, in order to evaluate query performance for subject-specific queries. These horizontally partitioned tables are created for triple tables. Materialized views are applied on binary tables in order to gain better performance as materialized views store partial results beforehand. A query set is then executed over all the data stores namely triple table, property tables, binary tables, horizontally partitioned tables, and materialized views over binary tables.

Experiments are performed for all basic RDF data storage techniques in order to evaluate their query performance and scalability in terms of QET (Query Execution Time). FOAF [**85**] dataset is used to compare TT, PT, BT, HP, and MV. A query set is executed on all the data storage techniques described. Cold and hot runs are observed for the queries. A cold run is a run of the query right after a database is started and no data is preloaded into the system's main memory, neither by the database nor in file system caches. Such a clean state can be achieved via a system reboot or by running an application that accesses sufficient data to flush file system caches, main memory, and CPU caches. On the other hand, a hot run is defined as repeated runs of the same query without stopping the database, ignoring the initial cold run [**86**]. Hot runs demonstrate a real-life scenario of data access and hence hot runs are considered as actual query runs. Hot runs are represented as query runs throughout the thesis. Query runs are averaged over three runs and are reported in terms of QET.

### 3.3.1. Algorithms

This section describes algorithms for basic RDF data storage techniques like TT, PT, BT, and HP. PT and HP algorithms are customized based on the dataset used for the experiment. All the given algorithms converge in either linear or polynomial time.

TT is a flat representation of RDF triples in the relational model which has three columns subject, predicate, and object. It can be seen in the Triple Table algorithm that .n3 file is given as input and all triples are read line by line and stored in a database in a three column table.

```
Algorithm: TT
Input: .n3 file
Output: TT
1. For all r in RDFTriples
2.    TT ← s,p,o
3. End For
4. Return TT
```

Once a TT is formed, clusters are created for storing data in clustered property tables. Two property tables are broadly clustered for relationship details and personal details which are customized for the data set used in the experiment. Rest of the data is stored as leftover triples.

```
Algorithm: PT
Input: TT
Output: PT
1. For all r in RDFTriples
2. if p in r belongs to relationship
3.    create property cluster $p_{c1}$
4. Else if p in r belongs to personal
5.    create property cluster $p_{c2}$
6. Else
7.    Consider as leftover triples
8. End For
9. For all r ∊ RDFTriples
10.      For all $p_i$ in cluster $c_i$
11.         $pc_i \leftarrow$  s, p1,p2..pn
12.      End For
13.   End For
14.   Return PT
```

```
Algorithm: BT
Input: TT
Output: BT
1. For all r in RDF Triples
2.    create binary table for every unique property
3. End For
4. For all r in RDF Triples
5.    For all p in RDF Triples
6.       $BT_i \leftarrow$  s,o
7.    End For
8. End For
9. Return BT
```

```
Algorithm: HP
Input: TT
Output: HP
1. For all r in RDF Triples
2.    For all n ∊ {0 to 9} ∨ {a-z} ∨ {A-Z}
3.       HP ← n
4.       n ←  s,p,o
5.    End For
6. End For
7. Return HP
```

Binary tables are created for every distinct property. All unique properties are identified for the dataset and a two column table is created for each property. Relevant data is then inserted into each binary table.  The dataset is horizontally partitioned based on subjects. A horizontally partitioned table is created which is a three column triple table, specifically based on the first character of the subject which is customized for the dataset used in the experiment.

Figure 3.1depicts execution flow for experimenting basic RDF data storage techniques. The data set is in RDF/XML which is converted to triples using a Jena parser [**87**] and stored in a relational database. Basic RDF data stores like TT, PT, BT, and HP are created using algorithms described in section 3.3.1. Dataset is scaled in terms of number of triples and query performance is reevaluated for all the basic RDF data stores. Query performance is measured in terms of QET.

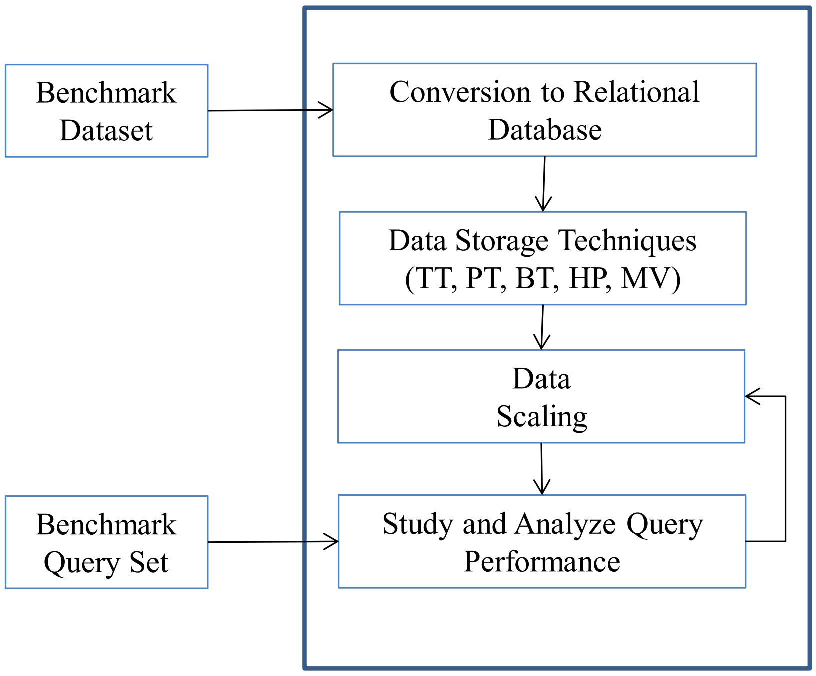

Benchmark Dataset
Conversion to Relational Database
Data Storage Techniques (TT, PT, BT, HP, MV)
Data Scaling
Benchmark Query Set
Study and Analyze Query Performance

**Figure 3.1 Execution Flow for Basic RDF Data Stores**

### 3.3.2. Dataset

Basic RDF data storage techniques are experimented on a benchmark dataset, Friend Of A Friend (FOAF) [**85**] from the University of Maryland with 406540 triples and 380961 distinct triples. It has 234 unique properties and occupies 65 MB of disk space. Details of FOAF dataset are given in Table 3.1.

**Table 3.1 FOAF Dataset Specifications**

| **Specifications** | **FOAF Dataset** |
|---|---:|
| No of Triples | 406540 |
| No of distinct Triples | 380961 |
| No of unique properties | 234 |
| Size on disk | 65 MB |

### 3.3.3. Query Set

In general, there are four kinds of queries fired on RDF data, which includes: star queries, linear queries, administrative or range queries, and snowflake queries. Hence RDF queries are categorized into four different query types. Type 1 queries are linear queries which retrieve predetermined set of properties from the RDF dataset. Type 2 queries are star queries that involve selection of properties defined for a particular subject. Type 3 queries are administrative or range queries and Type 4 queries are snowflake queries that retrieve predetermined properties or subjects. Query set contains 15 queries for FOAF dataset and is classified into four different query types. QET is averaged over query types to gain correct insight into query performance for a specific query type. All the SQL queries for FOAF dataset are given in Appendix 1.

### 3.3.4. Hardwares and Softwares

A test bed is created to implement all RDF data storage techniques and execute queries for FOAF dataset. The dataset is in RDF/XML format which is converted to triples by using Jena parser [**87**]. Eclipse 3.5 and Java 1.8 are used to implement the parser and other algorithms of the experiment. PostgreSQL is used as a database tool to store RDF data using various basic RDF data storage techniques. These tools are installed on a Linux based machine with hard disk storage space of 500 GB and 2 GB RAM. Figure 3.2 gives a glimpse of the test bed for all the experiments discussed in the thesis. Corresponding algorithms for the experiment are executed to generate RDF store. This RDF store is stored using a relational database. For Chapter 3 and Chapter 4 PostgreSQL and for Chapter 5 PostgreSQL-XL is used as a relational database.

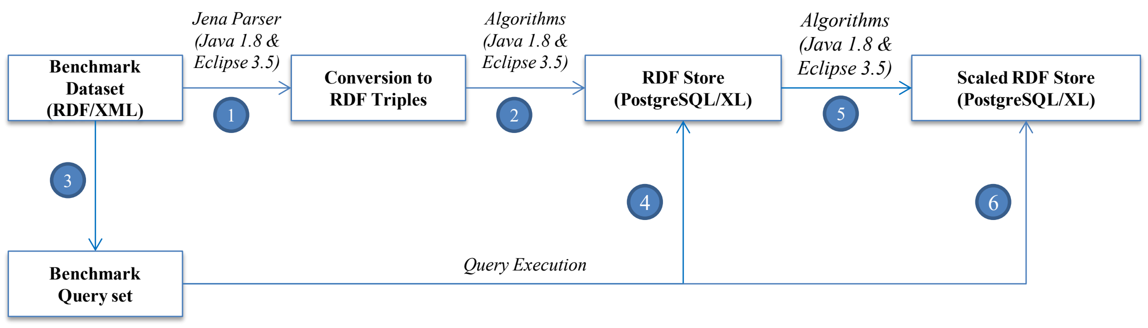

Jena Parser (Java 1.8 & Eclipse 3.5)
Algorithms (Java 1.8 & Eclipse 3.5)
Algorithms (Java 1.8 & Eclipse 3.5)
Benchmark Dataset (RDF/XML)
Conversion to RDF Triples
RDF Store (PostgreSQL/XL)
Scaled RDF Store (PostgreSQL/XL)
1
2
3
4
5
6
Benchmark Query set
Query Execution

**Figure 3.2 Testbed**

## 3.4. Results and Discussions

This section discusses analysis and comparison of query performance for various RDF data storage techniques in terms of QET. TT algorithm takes .n3 file as input and stores RDF data in a database in a three column table. TT consists of 406540 records as there are 406540 triples. Once a TT is formed, clusters are created for storing FOAF data in clustered property tables. As FOAF data is a social network data, information like relationship and personal details appeared frequently in the dataset. Hence, two PTs are broadly clustered for relationship details and personal details. Relationship details table consists of 9 properties and has 186572 records and personal details consist of 5 properties with 50264 records. Rest of the data is stored as leftover triples with 169704 records. Binary tables are created for every distinct property. FOAF dataset has 234 distinct properties and hence there are 234 binary tables. FOAF dataset is horizontally partitioned based on subjects. It contains total 36 tables, out of which 26 tables are for all alphabets (a-z) and 10 tables for all numeric from (0-9). It denotes subject starting with a corresponding character.

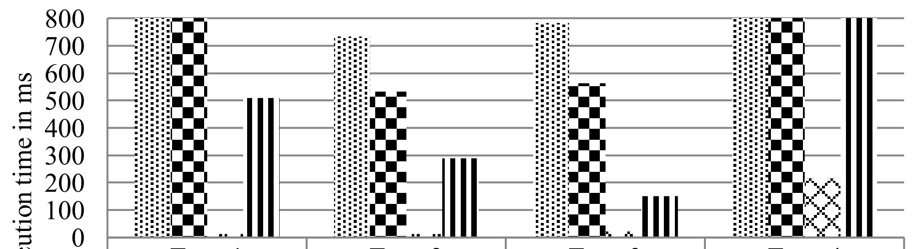
### 3.4.1. Query Execution for Basic RDF Data Stores

Query sets are executed on TT, PT, BT, and HP on PostgreSQL. All the queries are executed and averaged over three query runs. QET is recorded in milliseconds and it is averaged over query types. Figure 3.3 depicts query execution for FOAF dataset for basic data storage techniques discussed. It can be clearly seen from Figure 3.3 that partitioning techniques outperform TT for all query types for this experiment.

**Query Type 1:** Query Type 1 retrieves data for a fixed set of properties. It selects all records for which a set of properties are defined. BT, PT, and HP show 99%, 73% and 94% on an average improvement over TT respectively. Binary table is having far less number of records than a TT and hence join cost comes to minimum. For a sample query, BT contains one subject-subject join and one subject-object join within three tables whereas the same query in TT consists of 3 self-joins. Time taken by self-join increases due to scan operation for all records as TT is huge compared to BT. PT has five attributes based on the query, and it contains no joins but the scan operation for all the records costs higher than BT. In case of

HP, it is a union of 36 tables which makes the operation time consuming compared to BT using only three tables.

**Query Type 2:** Query Type 2 queries data for a predetermined subject. It selects all records defined for a subject. On an average, BT, PT, and HP perform 98%, 28% and 61% better compared to TT for this query type. For a sample query, BT contains no joins but a union of nine binary tables. These binary tables are much smaller in size compared to TT, and hence scanning cost is lesser for BT compared to TT. In case of PT, there is only one table with 9 attributes. However, it takes longer time as it has records for every subject. It also suffers from space overhead due to lots of nulls. HP uses a single table which has subjects starting with initial of the queried subject. HP takes longer execution time is due to 9 filters for queried properties. It is simpler to perform union operation of 9 small tables than to filter 9 properties out of a horizontally partitioned triple table.



| | Type 1 | Type 2 | Type 3 | Type 4 |
|---|---|---|---|---|
| ░ TT | 7952 | 734 | 783 | 2929 |
| ▦ PT | 2185 | 532 | 564 | 1767 |
| ⋯ BT | 13 | 11 | 22 | 216 |
| ‖ HP | 509 | 289 | 149 | 1287 |

**Figure 3.3 Query Execution for Basic RDF Stores**

**Query Type 3:** Query Type 3 falls in the category of administrative queries which are generally range queries. It selects all records which satisfy the given range criterion. Query performance for BT, PT, and HP is seen to be 97%, 28%, and 81% better than TT. For a sample query, BT contains no joins and it queries records from a single binary table and hence takes lesser time compared to TT. The clustered property table contains 5 attributes and hence takes longer time in scanning the whole table. HP needs to scan a single horizontally partitioned triple table. However, size of HP is more than BT and hence it takes longer for scan operation.

**Query Type 4:** Query Type 4 asks for queries with a specific subject and specific properties. It selects records with predetermined subject and property. It is observed that on an average BT, PT, and HP perform 93%, 40% and 56% better compared to TT. For a sample query, BT contains one subject-object join and joins two binary tables. Whereas TT contains two self-joins and hence takes longer QET than BT because of table size and cost of self-join. Clustered PT has no joins but has a larger table with 5 attributes which take longer to execute the same query compared to BT.

As expected, partitioned data occupies less space compared to other RDF data storage techniques. FOAF data of initial size for TT occupied 65 MB, for PT it occupied 60 MB, for HP and BT it occupied 53 MB and 35 MB respectively.

### 3.4.2. Query Execution for Materialized Views

Queries like "suggest friend of a friend" are very common. In such inference queries, one first needs to find friends of the subject from the triple table, and then join this data with the triple table to get friend of a friend. In general, these queries are evaluated using subject-object joins. The process can be accelerated by storing the result beforehand in the form of views and avoiding self-joins. One way to address this issue is using query modification technique [**88**]. However, query modification technique creates view on-the-fly and it can slow down query execution. An alternative to query modification is to pre-compute the view definition and store the result. When the query is posed on the view, the original query is executed directly on the pre-computed result. This approach, called View Materialization [**19**] is likely to be much faster than the query modification approach. The drawback is that one must maintain the consistency of the materialized view whenever the underlying tables are updated. However, it is necessary to deduce how much preprocessing one can afford and from when one can start getting the benefit of better query performance. This can be worked out using break-even analysis.

QET for normal queries and queries using materialized views for a triple table and binary table are compared. It is observed that QET for the triple table is 99% more than the materialized view based approach for the query "Suggest friend of a friend". This happens because the number of triples to be scanned is decreased from 406538 to 66914, which is approximately 16% of actual data size and there is no subject-object join in the materialized view base scheme. A break-even analysis is also carried out for this experiment. Two cases have been considered for break-even analysis for triple table and binary table. They are explained for Query 3 and Query 6.

### *3.4.2.1. Break-even analysis for triple table*

This section describes break-even analysis for triple table considering both cases where the base tables are updated frequently and are not updated frequently.

*Case 1: The base tables are not updated frequently.*

This means that the view creation time can be considered as the biggest cost. For Query 3 and Query 6 the following costs are observed as given in Table 3.2. The break-even point will be reached when $N_Q$* $QET_{TT}$ = MC + $N_Q$* $QET_{MV}$. Here $N_Q$ is the no. of user queries to be executed to reach the break-even point after calculation, the value of $N_Q$ as 1.14 and 18.59 for Query 3 and query 6 respectively.

**Table 3.2 MV cost v/s Normal cost for TT (Base tables not updated frequently)**

| Query | Materialized view Costs | Normal Costs |
|---|---|---|
| Query 3 | MV Creation Time MC = 3325ms<br>Query Execution Time $QET_{MV}$ = 4 ms | Query Execution Time for TT<br>$QET_{TT}$ = 2929 ms |
| Query 6 | MV Creation Time MC = 13591ms<br>Query Execution Time $QET_{MV}$ = 2.88 ms | Query Execution Time for TT<br>$QET_{TT}$ = 734 ms |

*Case 2: The base tables are updated frequently.*

Here the system will take more time to reach the break-even point as the view refreshment will have to be done frequently and this is costly. For Query 3 and Query 6 the following costs were observed as depicted in Table 3.3. The break-even point will be reached when $N_Q$ * $QET_{TT}$ = D*MR + $N_Q$ * $QET_{MV}$. Here $N_Q$ is the no. of user queries to be executed to reach the break-even point. D is the no of insert/update/delete queries. After calculation, the ratio $N_Q$:D is found to be 2.62, signifying that for 1 insert/delete/update query one needs at least 3 user queries and for query 6 ratio $N_Q$:D as 18.37 signifying that for 1 insert/delete/update query one needs at least 18.37 user queries.

**Table 3.3 MV cost v/s Normal cost for TT (Base tables updated frequently)**

| Query | Materialized view Costs | Normal Costs |
|---|---|---|
| Query 3 | MV Refreshment Time MR = 7672 ms<br>Query Execution Time $QET_{MV}$ = 4 ms | Query Execution Time for TT<br>$QET_{TT}$ = 2929 ms |
| Query 6 | MV Refreshment Time MR =13429 ms<br>Query Execution Time $QET_{MV}$ = 2.88 ms | Query Execution Time for TT<br>$QET_{TT}$ = 734 ms |

### *3.4.2.2. Break-even analysis for binary tables*

This section describes break-even analysis for triple table considering both cases where the base tables are updated frequently and are not updated frequently.

*Case 1: The base tables are not updated frequently.*

This means that the view creation time can be considered as the biggest cost. For Query 3 and Query 6, the following costs are observed as given in Table 3.4. The break-even point will be reached when $N_Q$ * $QET_{TT}$ = MC + $N_Q$ * $QET_{MV}$. Here $N_Q$ is the no. of user queries to be executed to reach the break-even point after calculation, the value of $N_Q$ as 16.05 and 181.09 for query 3 and query 6 respectively.

**Table 3.4 MV cost v/s Normal cost for BT (Base tables not updated frequently)**

| Query | Materialized view Costs | Normal Costs |
|---|---|---|
| Query 3 | MV Creation Time MC = 3325 ms<br>Query Execution Time $QET_{MV}$ = 8.76 ms | Query Execution Time for BT<br>$QET_{TT}$ = 216 ms |
| Query 6 | MV Creation Time MC = 297 ms<br>Query Execution Time $QET_{MV}$ = 9.36 ms | Query Execution Time for BT<br>$QET_{TT}$ = 11 ms |

*Case 2: The base tables are updated frequently.*

Here the system will take more time to reach the break-even point as the view refreshment will have to be done frequently and this is costly. For Query 3 and Query 6, the following costs are observed as given in Table 3.5. The break-even point will be reached when $N_Q$ * $QET_{TT}$ = D*MR + $N_Q$ * $QET_{MV}$. Here $N_Q$ is the no. of user queries to be executed to reach the break-even point. D is the no of insert/update/delete queries. After calculation, the ratio $N_Q$:D is found to be 37.02, signifying that for 1 insert/delete/update query one needs at least 38 user queries. For Query 6, the ratio $N_Q$:D is found to be 96.95, signifying that for 1 insert/delete/update query one needs at least 97 user queries.

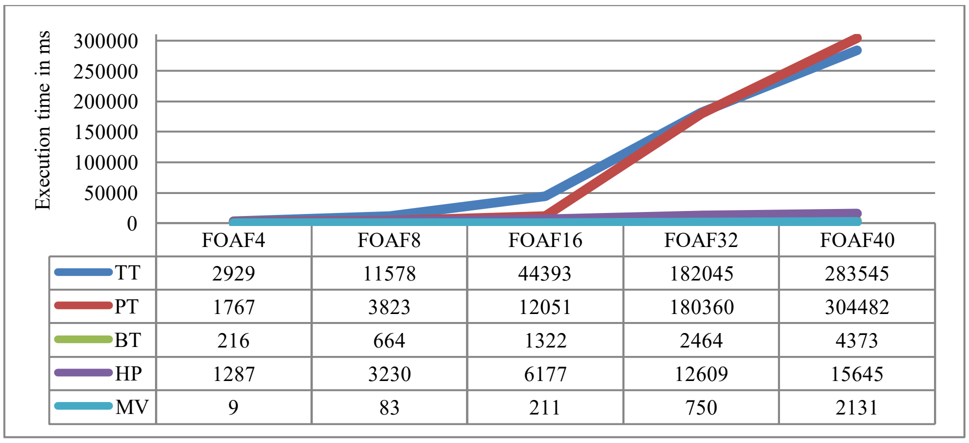
**Table 3.5 MV cost v/s Normal cost for BT (Base tables updated frequently)**

| Query | Materialised view Costs | Normal Costs |
|---|---|---|
| Query 3 | MV Refreshment Time MR = 7672 ms<br>Query Execution Time $QET_{MV}$ = 8.76 ms | Query Execution Time for BT<br>$QET_{TT}$ = 216 ms |
| Query 6 | MV Refreshment Time MR = 159 ms<br>Query Execution Time $QET_{MV}$ = 9.36 ms | Query Execution Time for BT<br>$QET_{TT}$ = 11 ms |

It can be inferred from the discussed cases that break-even analysis mainly depends on materialized view creation and refreshment times which in turn depend on the number of records and the number of join operations that need to be performed. When the materialized view creation and/or refreshment time are very high then the materialized view based scheme is beneficial if there are frequent queries. When the materialized view creation and/or refreshment time are low then the materialized view based scheme is beneficial in any case.

### 3.4.3. Query Execution for Scaled Data

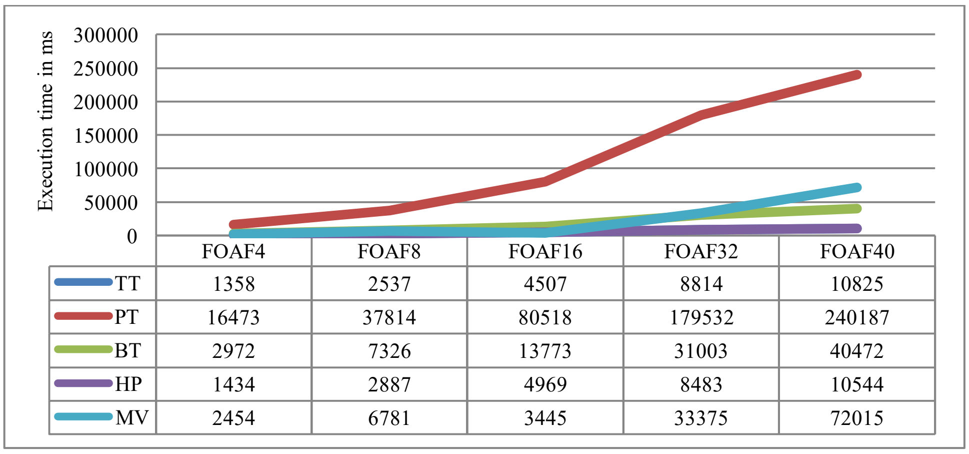
RDF data is proliferating and hence it is essential to ensure the query performance with increasing RDF data. Data scaling is performed to evaluate query performance for basic RDF data storage techniques. It is performed by increasing number of triples to 2 times, 4 times, 8 times and 10 times of actual data size. Data is scaled maximum up till 10 times considering other hardware constraints. The data size of resultant data stores is 813080, 1626160, 3252320 and 4065400 triples. The data scaling experiment and analysis is performed for each data size on all basic RDF data stores. It is observed that partitioning techniques performed an order of magnitude better compared to all other data stores for scaled data.



| | FOAF4 | FOAF8 | FOAF16 | FOAF32 | FOAF40 |
|---|---|---|---|---|---|
| TT | 2929 | 11578 | 44393 | 182045 | 283545 |
| PT | 1767 | 3823 | 12051 | 180360 | 304482 |
| BT | 216 | 664 | 1322 | 2464 | 4373 |
| HP | 1287 | 3230 | 6177 | 12609 | 15645 |
| MV | 9 | 83 | 211 | 750 | 2131 |

**Figure 3.4 Query Execution and Scaling for Query 3**

Real-life queries on RDF data generally have more number of subject-object joins. Apart from QET, query performance is further analyzed for scaled data based on joins. For FOAF query set, query 1,2,3,4 and 11 contains one subject-object join, query 7, 12, and 15 contains two subject-object joins, and Query 14 contains three subject-object joins. Query performance is analyzed based on number of joins against scaled data. It is found that queries having one subject-object join, when fired on binary tables, shows 43 times average improvement over the triple table. Queries having two subject-object join shows 36 times average improvement and query having three subject-object joins show 28 times performance improvement over the triple table.



| | FOAF4 | FOAF8 | FOAF16 | FOAF32 | FOAF40 |
|---|---|---|---|---|---|
| TT | 1358 | 2537 | 4507 | 8814 | 10825 |
| PT | 16473 | 37814 | 80518 | 179532 | 240187 |
| BT | 2972 | 7326 | 13773 | 31003 | 40472 |
| HP | 1434 | 2887 | 4969 | 8483 | 10544 |
| MV | 2454 | 6781 | 3445 | 33375 | 72015 |

**Figure 3.5 Query Execution and Scaling for Query 6**

It is seen that binary tables outperforms other data storage techniques and hence materialized view over partitioned data are implemented to gain even better performance. Materialized views are created for every query listed in the query set. Based on the kind of joins, queries and their performances with various data size on various data storage techniques are studied. Figure 3.4 plots a chart for the scaled data from 406540 triples to 4065400 triples. It shows the data for query 3 and 6 with its execution time in milliseconds. Query 3 has one subject-object join, whereas Query 6 has no joins. It can be seen that binary tables give the best-case performance compared to all other storage techniques. The performance got enhanced after using materialized views for binary tables. Figure 3.5 show that horizontal partitioning gives a best-case performance for query 6. Since query 6 has no subject-object joins materialized views have not shown improvement in QET. It is clearly visible that BT and HP scale linearly which shows that even with increases of data, partitioning technique leads to better performance compared to other discussed storage techniques.

## 3.5.Summary

The chapter discusses basic RDF data storage techniques. An experiment is carried out where basic RDF data storage techniques are implemented on a benchmark RDF dataset. Query set is fired on the basic RDF data stores and the presented work evaluates query performance in terms of QET. The data is scaled in terms of number of triples and QET is reevaluated to further study issues related to data scaling. It demonstrates that partitioning techniques have shown better performance over TT. There are cases when BT has outperformed TT especially when a single property is queried and on the other hand there are cases when PT has outperformed TT when set of properties queried together lay in the same property table.  Data scaling experiment also have shown that partitioning techniques scales linearly with increasing data and leads to better query performance compared to TT.

# Chapter 4

# Hybrid Partitioning for RDF Data

Basic data partitioning techniques and their experimental evaluations have been discussed in Chapter 3. It is observed that all the basic data partitioning techniques have their own advantages and disadvantages. The idea is to exploit best of different partitioning techniques and combine them together, in order to overcome their disadvantages. This leads to hybrid data partitioning techniques. This chapter proposes three different hybrid partitioning techniques for RDF data: 1) DAHP (Data-Aware Hybrid Partitioning), which combines binary tables with property tables 2) DASIVP (Data-Aware Structure Indexed Vertical Partitioning), which combines binary tables with structure indexing 3)WAHP (Workload-Aware Hybrid Partitioning), it combines binary tables and property tables based on query frequencies.

## 4.1.Data-Aware Approaches for Partitioning RDF Data

RDF data can be stored and configured using two ways: Data-aware approach and Workload-aware approach. Data-aware approach stores RDF data based on how the data is related to each other in the dataset. Workload-aware approach stores RDF data based on the data that is queried together.

The data-aware approach helps in designing a customized schema, where a set of data that appears together in the dataset can be kept together. It helps in reducing number of joins and retrieve data efficiently. This sub-section presents two data-aware approaches: 1) Data-Aware Hybrid Partitioning 2) Data-Aware Structure Indexed Vertical Partitioning.

### 4.1.1. Data-Aware Hybrid Partitioning (DAHP)

DAHP combines property tables and binary tables. Property table helps in minimizing number of joins. However, it has few issues, such as storage of multi-valued attributes. It also suffers from sparsity as it may contain a lot of nulls. Due to these problems, although property tables are able to minimize joins, join operation becomes complex. On the other hand, binary tables can execute join queries faster. However, it may suffer from the problem of increased number of joins when data is queried from multiple binary tables. Binary tables can encourage faster query execution when a query contains a single property, and property table can possibly give better performance when queried properties are found in the same table. Considering these advantages and disadvantages of both the RDF data storage techniques this chapter proposes DAHP, a technique that partitions data using a combination of binary tables and property tables.

The dataset is analyzed based on the properties that tend to appear together. The set of properties that have a tendency to be defined together for a subject in the dataset is stored as a separate group in form of a property table. Properties which are not a part of a group or they don't occur with a set of properties are stored as binary tables. DAHP hence generate two kinds of tables: property tables and binary tables. The advantage of using DAHP over property tables is that it will contain property tables which will have fewer nulls, as set of properties which tend to be together are grouped in a single table. The advantage of using DAHP over binary tables is, it will consist of less number of binary tables means less number of joins and hence faster query execution can be achieved.

### 4.1.2. Data-Aware Structure Indexed Vertical Partitioning (DASIVP)

DASIVP combines structure index partitioning and binary tables. Structure Index Partitioning [**38**] is a data partitioning technique that partitions RDF triples based on the characteristics of data. A data graph of the dataset is primed where subjects and objects are nodes and properties are edges between them. The purpose behind creating the data graph is to recognize different structures based on incoming and outgoing edges. Data nodes having same incoming and outgoing edges in terms of property values are associated with the same index known as structure index. Data having the same index are positioned together in one table and are accessed via its index. When a query is executed, the query structure is matched with structure index graph and the output is generated.

In structure index partitioning, the data is analyzed considering properties associated with it. Subjects which are associated with same properties are kept together and triples related to those subjects are grouped together, which is considered as a structure. It uses a *Lookup Table* to keep the record of different structures. *Lookup Table* holds property list and index to structures. Individual access to particular intended structure facilitates in saving a lot of time spent on scanning and joining unrelated data while executing queries on basic RDF stores.

The advantage of using structure index partitioning over binary tables is that it includes relevant data that can cover entire query structure, whereas a binary table contains data for a single property that can cover up only one query. However, structure index partitioning suffers from the problem of redundant self-joins like TT because structure index partitioning is also in the form of RDF triples. Further, if query structure matches with more than one structure indexes, join overhead will be augmented.

Considering the pros and cons of these partitioning techniques, it will be useful to put them together in a suitable way so as to improve QET. The chapter introduces integration of binary tables and structure index partitioning. Although structure index partitioning requires redundant self-joins, it can be compensated by binary tables. Moreover, binary tables may be inefficient for queries targeted towards related data, as it may contain some data that is not relevant to the query and this problem can be addressed by using structure index partitioning. This integration is called DASIVP, where structure index partitioning is followed by binary table.

Algorithms, data structures, and execution flows for both the data-aware approaches are discussed in section 4.3.1.

## 4.2. Workload-Aware Approach for Partitioning RDF Data

A workload-aware approach, as the name implies configures data based on query workload. Data which is queried together is kept together in order to retrieve it efficiently. Several applications where queries are known in advance can use workload-aware approach to configure and partition RDF data stores. This section describes one such workload-aware technique: Workload-Aware Hybrid Partitioning.

### 4.2.1. Workload-Aware Hybrid Partitioning (WAHP)

WAHP is a workload-aware approach for RDF data partitioning. This approach is useful in applications where queries are known well in advance. Based on query knowledge and its frequencies, data can be partitioned such that faster query execution can be achieved. Like DAHP, WAHP also combines binary tables and property tables. However, the approach is different. WAHP follows a workload-aware approach in contrast to DAHP which follows a data-aware approach.

WAHP in advance requires query information like actual data queries and their frequencies. Based on the query frequency, a set of properties that are queried together are clustered in a table. Frequently queried properties that satisfy null and support thresholds are kept together and clustered as a property table and other frequently queried properties that do not satisfy these thresholds culminate in binary tables. The actual query workload for these properties is then mapped to the table and the table is populated with relevant workload data. Likewise, many such property tables and binary tables are generated based on query workload and that result in workload-aware clusters. Algorithms, data structures, and execution flow for WAHP is discussed in section 4.3.1.

## 4.3. Experiments

The experiments section discusses experimental details like data structures, algorithms, dataset, query set and implementation details for all the hybrid RDF data partitioning techniques.

### 4.3.1. Data Structures and Algorithms

This sub-section explains relevant data structures and algorithms for all the three hybrid RDF data partitioning techniques namely DAHP, DASIVP, and WAHP.

#### *4.3.1.1. DAHP Data Structures and Algorithms*

DAHP is implemented using two data structures: *Property Use Listing* and *Subject Property Bin*. *Property Use Listing* is prepared by finding the number of subjects associated with that property which reflects the occurring frequency of the property in the RDF dataset. *Subject Property Bin* contains set of properties associated with a particular subject. It is sorted in descending order of number of properties associated with a subject, giving highest number of associated properties first. Figure 4.1 depicts DAHP data structures.

Apart from these two data structures, occurrence of similar property list is calculated, which gives a metric called support threshold. Support threshold helps to reduce number of joins. Another metric used is a null threshold which indicates permitted number of nulls. Null threshold helps in restricting null storage. Another metric called Multi-valued Property Threshold (*MPT*) is also used which checks the number of multi-valued properties in the dataset. DAHP algorithm utilizes *Subject Property Bin*, *Property Use Listing*, and null and support threshold. DAHP returns a set of binary tables and property tables using Apriori Algorithm [**89**]. All data structures require O (n) scans. DAHP algorithm needs O (n) + O ($p^2$ * s) operations where n is the number of records in the dataset, p is the number of properties and s denotes candidates in *Subject Property Bin*.

| Subject | Property List |
|---|---|
| S1 | P1, P2, P3, P4 |
| S2 | P1, P2, P5 |
| S3 | P1, P2 |
| S4 | P1,P3 |
| S5 | P1,P4 |

**(a) Subject Property Bin**

| Property | Use |
|---|---|
| P1 | 5 |
| P2 | 3 |
| P3 | 2 |
| P4 | 2 |
| p5 | 1 |

**(b) Property Use Listing**

**Figure 4.1 Data Structures for DAHP**

```
Algorithm: DAHP
Input:  Null Threshold, Support Threshold, Subject Property Bin, Property
Use Listing, Triple Table
Output: Property Tables, Binary Tables
1. //merge sort
2. Sort descending Property Use Listing on property-use
3.    For all properties $p_i$ in Property Use Listing
4.       If $p_i$<support Threshold
5.         BT  ← $p_i$
6.       Else
7.          $C_p$ ← Consider $p_i$ for generating property tables
8.    End for
9. //Find PT and BT using Apriori Algorithm
10.   $sp_c$  ← Subject Property Bin
11.   // Cartesian product
12.   for all $p_i$ in Subject Property Bin
13.      $C_{p+1}$ = candidate generated from Subject Property Bin
14.       For all occurrences in Subject Property Bin
15.          Increment the count of all candidates in $c_{p+1}$ that are
   contained in Subject Property Bin
16.       End for
17.       $sp_c$ ← candidates in $c_{p+1}$ with support threshold and null threshold
18.       PT ← $sp_c$
19.   End For
20.   Return Property Tables, Binary Tables
```

Figure 4.2 depicts DAHP execution flow. It takes benchmark dataset as input and generates binary tables and property tables. DAHP considers null and support threshold defined for underlying properties to generate binary and property tables.

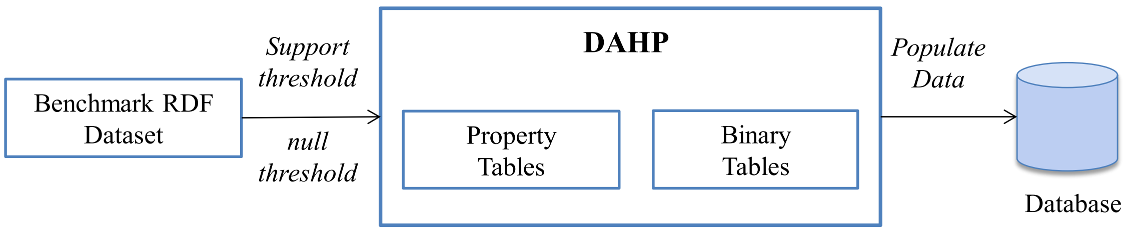


**Figure 4.2 Execution Flow for DAHP**

### *4.3.1.2. DASIVP Data Structures and Algorithms*

DASIVP uses three data structures: (i) *Subject Property Bin* (ii) *Structure Index Table* and (iii) *Lookup Table*. They are depicted in Figure 4.3. *Subject Property Bin* is a list of subject with their associated corresponding properties in the dataset. This is same data structure as used in DAHP. *Structure Index Table* contains set of associated subjects and their corresponding index. A *Lookup Table* is also created where similar structures are given an index name.

| Subject | Property List |
|---|---|
| S1 | P1, P2, P3, P4 |
| S2 | P1, P2, P3, P4 |
| S3 | P1, P2,P3 |
| S4 | P1,P3 |
| S5 | P1,P3 |

**(a) Subject Property bin**

| Associated Subjects | Corresponding Structure Index |
|---|---|
| S1,S2 | Index 1 |
| S3 | Index 2 |
| S4,S5 | Index 3 |

**(b) Structure Index Table**

| Index | Property List Structure |
|---|---|
| Index1 | P1, P2, P3, P4 |
| Index2 | P1, P2,P3 |
| Index n | P1,P3 |

**(c) Look up Table**

**Figure 4.3 Data Structures for DASIVP**

```
Algorithm: DASIVP1
Input: RDF Triples
Output: Subject Property Bin
1. For all s in RDF Triples
2.     SPB ← p ∨ o
3. End For
4. Return Subject Property Bin
```

```
Algorithm: DASIVP2
Input: RDF Triples, Subject Property Bin
Output: Structure Index Table
1. For all p in Subject Property Bin
2.    Lookup Table ← {Property list, location index}
3.       For all  s in Subject Property Bin
4.          SIT ← Identified Triple at location index
5.       End For
6. End For
7. Return Structure Index Table
```

```
Algorithm: DASIVP3
Input: Structure Index Table
Output: Structure Index Vertical Partitioned Data
1. For all Structure Index Table
2.    For all t in triples
3.       Fetch property p
4.       If   BT for p exists
5.         BT ←  (s,o)
6.       Else
7.         Create BT
8.         BT ←  (s,o)
9.    End For;
10.   End For;
11.   Return Structure Indexed Vertical Partitioned Data
```

Three algorithms are developed for DASIVP namely DASIVP1, DASIVP2, and DASIVP3. DASIVP1 prepares data graph as Subject Property Bin. DASIVP2 prepares index graph as *Lookup Table*, and DASIVP3 matches query structure with partitioned structure. The main reason behind preparing data graph is to identify properties associated with each subject based on its appearance in triples. Subject Property Bin gathers all properties associated with each subject and stores in the table. Two different indexes are created for subject and object column in TT so that searching time can be reduced. Moreover, the properties are sorted to make matching of properties easier. Each distinct property list value in Subject Property Bin is a different structure. For each value, subject entries are found. Corresponding triples are moved to the identified structure location from triple table. For each structure, property list value and reference index are recorded in *Lookup Table*. DASIVP2 prepares *Lookup Table* and partitions data using Structure index partitioning. Every structure indexed partitioned triples are converted into binary tables for each property. For every triple, *(s,o)* value is inserted into associated property table. Algorithm DASIVP3 results into DASIVP binary tables for each structure. All data structures and algorithms converge in linear time. Figure

4.4 depicts DASIVP execution flow. DASIVP takes benchmark dataset as input and creates binary tables from structure indexed partitioned tables.

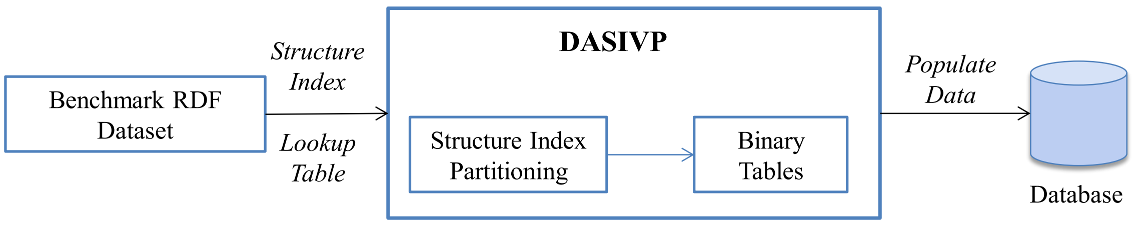


**Figure 4.4 Execution Flow for DASIVP**

### *4.3.1.3. WAHP Data Structures and Algorithms*

WAHP uses four data structures to identify workload-aware clusters. (i) *Query Property Basket*, which contains a list of queries with its associated queried properties. (ii) *Query Frequency List* holds total frequency count. (iii) *Query Frequency Property Basket* stores query frequency and their corresponding properties (iv) *Property Frequency List* is primed using *Query Property Basket* and *Property Frequency List*. It includes frequency count of each property. WAHP algorithm makes use of a frequency threshold, which is a count for a property to be a contender of the hot schema. Figure 4.5 depicts WAHP data structures.

| Q | Property List |
|---|---|
| Q1 | P1, P2, P3,P4 |
| Q2 | P2, P5 |
| Q3 | P1, P2, P5 |

**(a) Query Property Basket**

| Q | Site1 | Site2 | Site3 | *f* |
|---|---|---|---|---|
| Q1 | 100 | 200 | 300 | 600 |
| Q2 | 150 | 250 | 100 | 500 |
| Q3 | 400 | 350 | 250 | 1000 |

**(b) Query Frequency List**

| *f* | Property List |
|---|---|
| 100 | P1, P2, P3,P4 |
| 200 | P1, P2, P3,P4 |
| 300 | P1, P2, P3,P4 |
| 150 | P2, P5 |
| 250 | P2, P5 |
| 100 | P2, P5 |
| 400 | P1, P2, P5 |
| 350 | P1, P2, P5 |
| 250 | P1, P2, P5 |

**(c) Query Frequency Property Basket**

| Properties | *f* |
|---|---|
| P1 | 1600 |
| P2 | 1100 |
| P3 | 600 |
| P4 | 600 |
| P5 | 1500 |

**(d) Property Frequency List**

**Figure 4.5 Data Structures for WAHP**

```
Algorithm: WAHP
Input:  Null Threshold, Support Threshold, Property Frequency List,
Query Property Basket, Triple Table
Output: Property Tables, Binary Tables
1. //merge sort
2. Sort descending Property Frequency List on property-frequency
3. //merge sort
4. Sort descending Query Property Basket on property count
5. For all properties $p_i$ in Property Frequency List
6.    If $p_i$<support Threshold
7.       BT  ← $p_i$
8.    Else
9.       $C_p$ ← Consider $p_i$ for generating property tables
10.   End for
11.   //Find PT and BT using Apriori Algorithm
12.   $QP_c$  ← Query Property Basket
13.   // Cartesian product
14.   For all $p_i$ in Query Property Basket
15.      $C_{p+1}$ = candidate generated from Query Property Basket
16.      For all occurrences in Query Property Basket
17.         Increment the count of all candidates in $c_{p+1}$ that are
   contained in Query Property Basket
18.      End for
19.      $QP_c$ ← candidates in $c_{p+1}$ with support threshold and null
   threshold
20.      PT ← $QP_c$
21.   End for
22.   Return Property Tables, Binary Tables
```

All data structures require O (n) scans and WAHP algorithm needs O (n) + O ($p^2$ * q) operations where n is the number of records in the dataset, p is the number of properties and q denotes candidates in *Query Property Basket*.

WAHP uses queries and their frequencies to identify the actual workload. WAHP algorithm further uses a hybrid approach of partitioning as mentioned for DAHP. The only difference between DAHP and WAHP is DAHP focuses on how the data is defined together in the dataset and WAHP focuses on how the data is queried together. WAHP algorithm identifies hot schema which is a combination of property tables and binary tables. Hot schemas are populated with actual query workload. This results in workload-aware clusters. Figure 4.6 depicts WAHP execution flow. Benchmark dataset, query set and query frequencies are considered to generate hot schema. This hot schema is set of binary tables and property tables based on query workload.

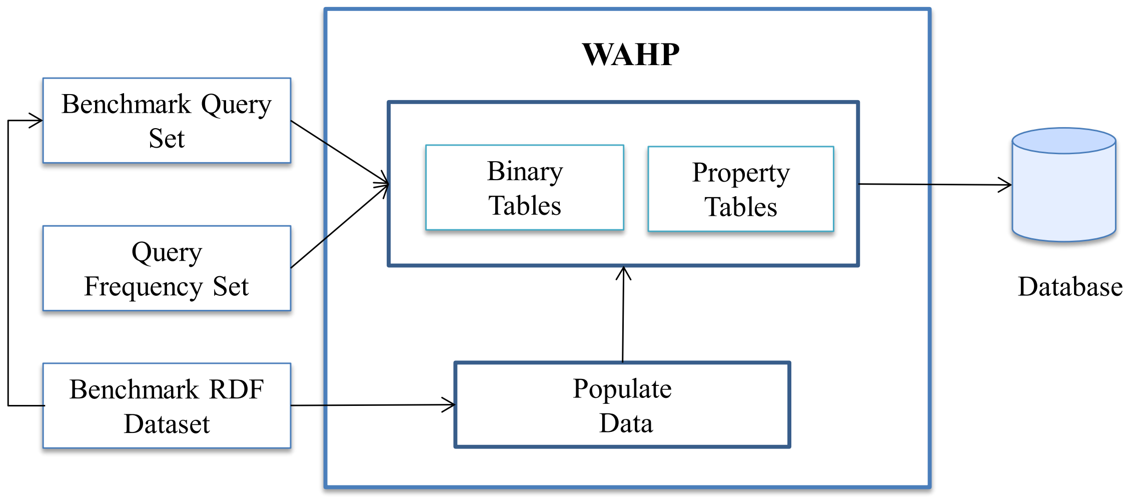


**Figure 4.6 Execution Flow for WAHP**

### 4.3.2. Metrics for RDF Data Stores

A set of metrics is devised to evaluate performance and suitability of each RDF data partitioning technique. These metrics can help to consider the suitability of a RDF store for a dataset. The following four metrics are devised in order to consider the appropriateness of a RDF data store.

#### *4.3.2.1. Structuredness Ratio (SR)*

Structuredness is categorized as well-structured and semi-structured. Well-structured datasets are denser in structure and have higher number of properties defined per subject. Semi-structured dataset is less dense comparatively and has a lower number of properties defined per subject. Structuredness ratio of the dataset measures usefulness of a particular kind of RDF data store. The more the data is structured and more there are subjects per structure, DASIVP may be the best suitable RDF store for such data compared to other mentioned RDF stores. Structuredness of the dataset can be derived using the following equation:

$$SR = N_T/N_{UP}$$

Where $N_T$ is the number of triples in dataset and $N_{UP}$ is the number of unique properties in the dataset. Datasets with higher structured ratio signify well-structured dataset and datasets with a lower value of structuredness ratio means they are semi-structured datasets.

#### *4.3.2.2. One Property Retrieval (OPR)*

One Property Retrieval implies a query retrieves only one property. It means only a single property needs to be fetched. In such situation, binary table gives the best performance. Therefore queries using binary tables excel in performance and execution time compared to other RDF data stores.

#### *4.3.2.3. Multi-Valued Property Threshold (MPT)*

Metric *MPT* finds the multi-valued property threshold which gives the percentage of multi-valued properties. *MPT* needs to be met in order to qualify to be a part of property table. *MPT* can help in controlling redundancy. It is given using following equation:

$$MPT = N_{MP} / N_P$$

Where $N_{MP}$ is number of multi-valued properties and $N_P$ indicates total number of properties in the dataset.

#### *4.3.2.4. Break-even Analysis (BEA)*

Break-even analysis is carried out for discussed RDF data stores, in order to know from when one can start getting the benefit of a particular RDF data storage technique. In general following equation is defined to calculate a break-even point, where $N_Q$ is the total number of queries. $R_{D1}$ and $R_{D2}$ define a comparison between different data stores and TET is Total Execution Time, which is a summation of the execution time of all phases for $R_{D1}$.

$$N_Q * R_{D2} > N_Q * R_{D1} + TET$$

### 4.3.3. Dataset

All the hybrid RDF data partitioning technique is implemented on a benchmark dataset SWetoDBLP [**90**] for our experiment. SWetoDBLP- Semantic Web Technology Ontology Digital Bibliography Library Project is a large size ontology focused on the bibliography of computer science publications. The primary data source of SWetoDBLP is DBLP [**91**]. It also includes additional relationships from other data sources such as FOAF and Dublin Core [**92**]. Basic RDF data partitioning experiment uses FOAF dataset. However, SWetoDBLP dataset is used for implementation of hybrid RDF data stores as it is a bigger and more structured dataset compared to FOAF. This structuredness of dataset can be measured using *SR* metric as discussed in section 4.3.2. Details of datasets including number of triples, number of distinct triples, number of unique properties and data size is provided in Table 4.1

**Table 4.1 SWetoDBLP Dataset Specifications**

| Specifications | SWetoDBLP dataset |
|---|---:|
| No of Triples | 14932830 |
| No of distinct Triples | 14932603 |
| No of unique properties | 29 |
| Size on disk | 2.16 G.B |

### 4.3.4. Query set

RDF queries are categorized into four different query types. Type 1 retrieve predetermined set of properties from the RDF dataset. Type 2 queries involve selection of properties defined for a particular subject. Type 3 queries are range queries and Type 4 includes queries that retrieve predetermined properties or subjects. QET is averaged over query types to gain correct insight into query performance for a specific query type. All SQL queries for the SWetoDBLP dataset are listed in Appendix 2.

### 4.3.5. Hardwares and Softwares

A test bed is created to implement all RDF data storage techniques and execute queries for the SWetoDBLP dataset. It is same as the one depicted in Figure 3.2. The dataset is in RDF/XML format which is converted to triples by using Jena parser [**87**]. Eclipse 3.5 and Java 1.8 are used to implement the parser and other algorithms of the experiment. PostgreSQL is used as a database tool to store RDF data using various basic RDF data storage techniques. These tools are installed on a Linux based machine with hard disk storage space of 500 GB and 2 GB RAM.

## 4.4.Results and Discussions

This section discusses results for data-aware and workload-aware hybrid approaches. DASIVP is a data-aware approach which is a combination of structure indexing and binary tables. DAHP and WAHP are hybrid of property tables and binary tables, however, DAHP is data-aware and WAHP is a workload-aware approach. Both the data-aware approaches are compared with BT as it is known that BT outperformed other basic RDF data stores. The query set is executed on various data storage techniques and averaged over three query runs. QET is recorded in milliseconds and it is averaged over query types. All implementation details of both data-aware approaches and workload-aware approaches along with their query results are discussed here.

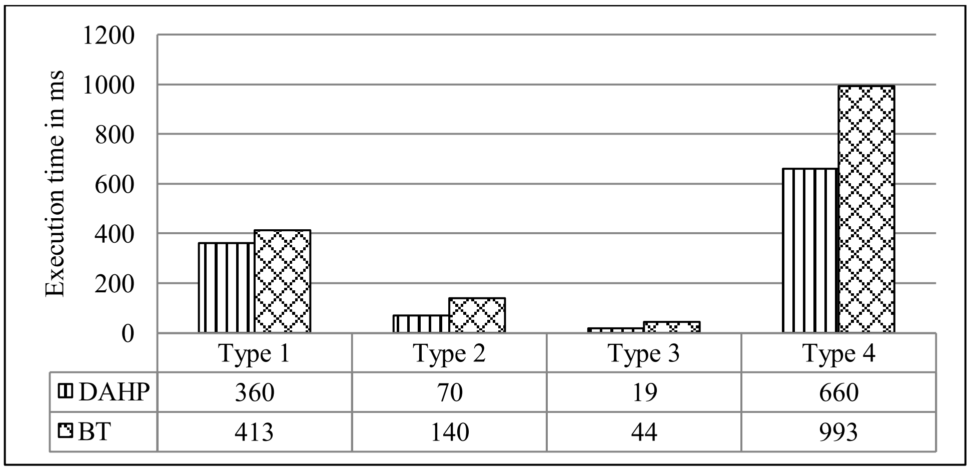
### 4.4.1. Query Execution for DAHP

DAHP is implemented for SWetoDBLP. DAHP technique gives 6 property tables and 11 binary tables which are formulated using *Property Use Listing* and *Subject Property Bin*. *Subject Property Bin* is sorted in descending order of number of properties associated with a subject, giving highest number of associated properties first. Occurrence of similar property list is calculated which helps to find support threshold. One by one all bins is scanned and checked for its support threshold. If it satisfies the threshold they are checked for the null threshold. Null storage is restricted below the provided null threshold. Satisfying the null threshold criterion they are considered as a final set of property tables. Next, *Property Use Listing* is pruned and if property exists in any of the bin then it is dropped else it is stored in as a binary table. All property tables are then populated with RDF data.

BT is also implemented for a SWetoDBLP dataset to compare BT with DAHP. As the dataset has 29 unique properties, BT contains 29 binary tables. Every binary table contains a list of subjects and associated objects in their respective table. All types of queries are fired on the database of 29 tables and QET is recorded. Figure 4.7 depicts a comparison between BT and DAHP.

**Query Type 1**: It retrieves a set of predetermined properties. As each and every subject needs to be checked against given properties it becomes an expensive operation. The properties asked in the queries are related with other because most of the times related data are retrieved together. Query 1 in this type retrieves 5 properties and hence BT uses 5 tables to get the query result. For DAHP all these properties are found in the same property table. Even after handling the overhead of null values, DAHP is still able to perform 13% better than BT.

**Query Type 2:** It asks for a specific subject. Properties are clustered based on relatedness of the properties and when it comes to subject-specific queries, it is not necessary to get all the subjects in the same property table. For query type 2, 50% of improvement over BT is observed as accidentally subjects were lying in the same property table. In other cases, it is possible that BT can perform better when subjects lay in different property tables and there are many property tables.



| | Type 1 | Type 2 | Type 3 | Type 4 |
|---|---|---|---|---|
| ▥ DAHP | 360 | 70 | 19 | 660 |
| ▨ BT | 413 | 140 | 44 | 993 |

**Figure 4.7 Query Execution for DAHP and BT**

**Query Type 3:** These are administrative queries where a range is queried from a table. Generally, such types of queries have related properties and are found in the same property table. In a sample query, it retrieves 3 properties which are laying in the same property table gives an improvement of 57% compared to BT.

**Query Type 4:** These queries retrieve specific properties and specific subject. They are generally constraint-based queries. Such queries perform better for DAHP because filtration can be carried out easily on a single property table compared to filtering from many tables in BT. In the experiment, DAHP is performing 34% better than BT.

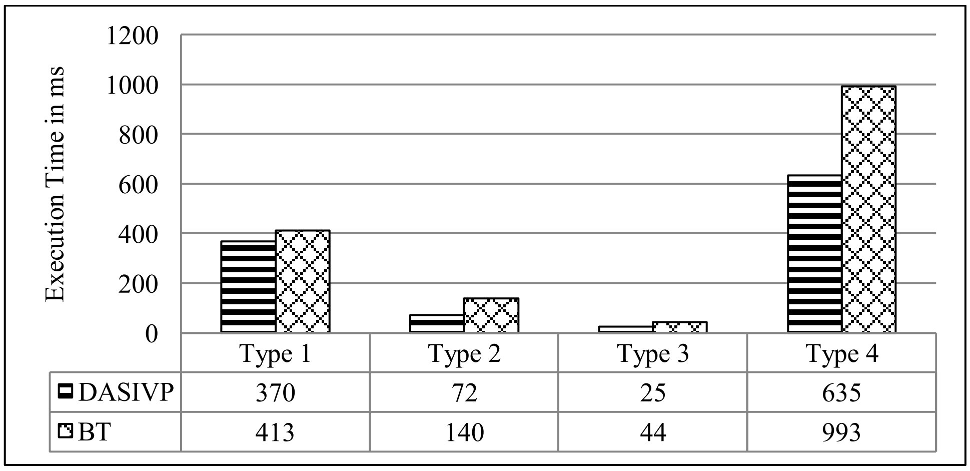
DAHP shows an average improvement of nearly 40% for all query types over BT. The average break-even point is 5 when compared to BT. However, queries which support *OPR* shows that BT performs better compared to DAHP.

### 4.4.2. Query Execution for DASIVP

DASVIP is implemented for the SWetoDBLP dataset and it is compared with BT. Further, both the data-aware approaches DASIVP and DAHP are also compared with each other in terms of their QET and break-even point. The query set is fired on the SWetoDBLP dataset for its BT, DASIVP and DAHP implementation. BT contains 29 binary tables as

SWetoDBLP dataset consists of 29 properties. Using Structure Index Partitioning, DASIVP consists of 39 structures, which are further followed by binary tables resulting in total 166 binary tables. DAHP implementation is same as discussed in section 4.4.1. Figure 4.8 shows a comparison between DASIVP and BT for each query type.

**Query Type 1**: This kind of queries retrieve records with few predetermined properties. For a sample query for query type 1, BT executes in 537 ms and DASIVP executes the same query in 307ms using 9 target structures. However, there is a preprocessing onetime cost that includes lookup cost and merges cost. Lookup time is 40ms and merge time is 3018ms. The average break-even point is found to be 49 for this query type. Query type 1 has shown an average of 10% of improvement over BT.



| | Type 1 | Type 2 | Type 3 | Type 4 |
|---|---|---|---|---|
| ▤ DASIVP | 370 | 72 | 25 | 635 |
| ▨ BT | 413 | 140 | 44 | 993 |

**Figure 4.8 Query Execution for DASIVP and BT**

**Query Type 2:** Type 2 asks for records specific to a subject and can retrieve all properties. For a sample query of query type 2, BT takes an execution time of 1647ms and DASIVP for the same query takes 954 ms and uses 27 target structures. However, there is a preprocessing onetime cost that includes lookup cost and merges cost. Lookup time is 39ms and merge time is 6667ms. The average break-even point is found to be 71 for this query type. On an average, query type 2 has shown 49 % of improvement over BT.

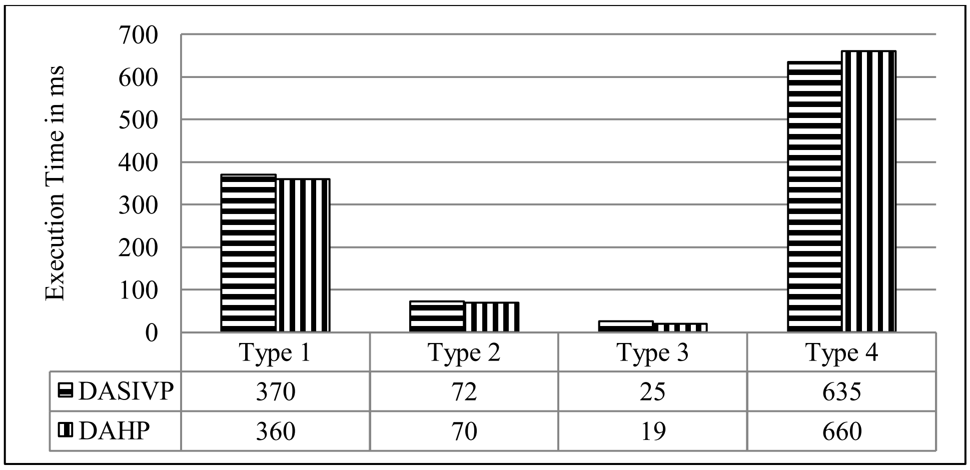
**Query Type 3:** Query Type 3 retrieves queries specified for a typical range. They are generally administrative queries. For a sample query of query type 3, BT executes the query in 3151 ms and DASIVP in 1957 ms and makes use of 8 target structures. However, there is a preprocessing onetime cost that includes lookup cost and merges cost. Lookup time is 20ms and merge time is 58867 ms. The average break-even point is found to be 173 for this query type. Query type 3 has shown 43% of average improvement over BT.

**Query Type 4:** These queries retrieve records for a specific subject and specific properties. For a sample query of query type 4, BT executes the same query in 1317 ms and DASIVP executes in 598 ms and uses 12 target structures. However, there is a preprocessing onetime cost that includes lookup cost and merges cost. Lookup time is 10ms and merge time is 2932ms. The average break-even point is found to be 46 for this query type. Query type 4, on an average, shows 36% of improvement over BT.

When averaging over query types, DASIVP results in 35% better performance compared to BT. However, it is also observed that queries that satisfy *OPR* metric, are more suitable for BT than DASIVP as DASIVP requires extra lookup and merge time, unlike BT. The reason for the overall better performance of DASIVP is, the merged binary tables for DASIVP are either same or less dense compared to binary tables. On top of that, DASIVP covers only related data, whereas BT covers all data in the table which is the main advantage of using DASIVP over BT.

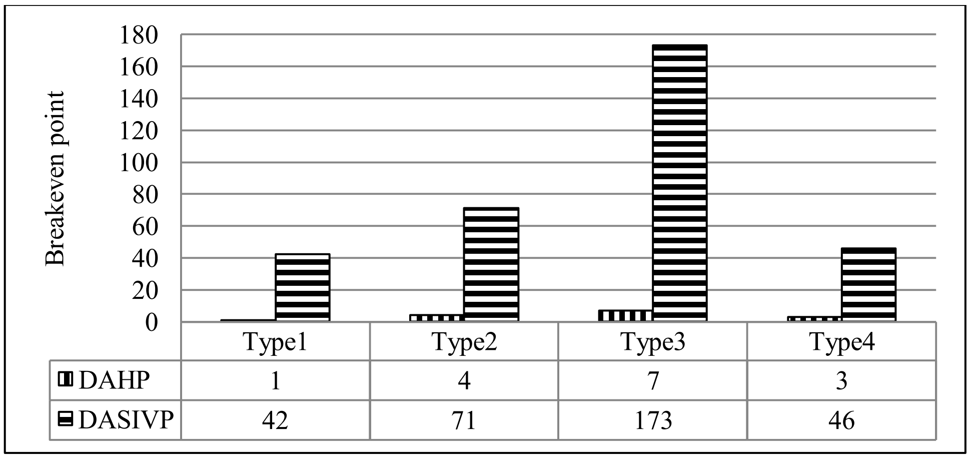
Break-even point is also calculated as break-even point specifies the minimum number of queries from when DASIVP can become beneficial. For example, to cover up the extra time lookup time and merge time in DASIVP, A sample query needs to be submitted at least 124 times. The average break-even point frequency for all the queries is 97.

Figure 4.9 depicts a comparison between both hybrid techniques. It can be seen that DASIVP and DAHP perform nearly equivalent when compared to BT. However, BT may perform better for the queries that satisfy *OPR* metric. Figure 4.10 depicts break-even point comparison between DASIVP and DAHP. It can be seen that break-even point for DASIVP is almost 95% more than DAHP. Even though DAHP and DASIVP perform almost equivalently, due to the huge difference in break-even point, the philosophy of DAHP is used for workload-aware approach. Detailed evaluation for all the discussed RDF data stores is given in section 4.5.



| | Type 1 | Type 2 | Type 3 | Type 4 |
|---|---|---|---|---|
| ▤ DASIVP | 370 | 72 | 25 | 635 |
| ▥ DAHP | 360 | 70 | 19 | 660 |

**Figure 4.9 Query Execution time for DAHP and DASIVP**



| | Type1 | Type2 | Type3 | Type4 |
|---|---|---|---|---|
| ▥ DAHP | 1 | 4 | 7 | 3 |
| ▤ DASIVP | 42 | 71 | 173 | 46 |

**Figure 4.10 Break-even point for DASIVP and DAHP**

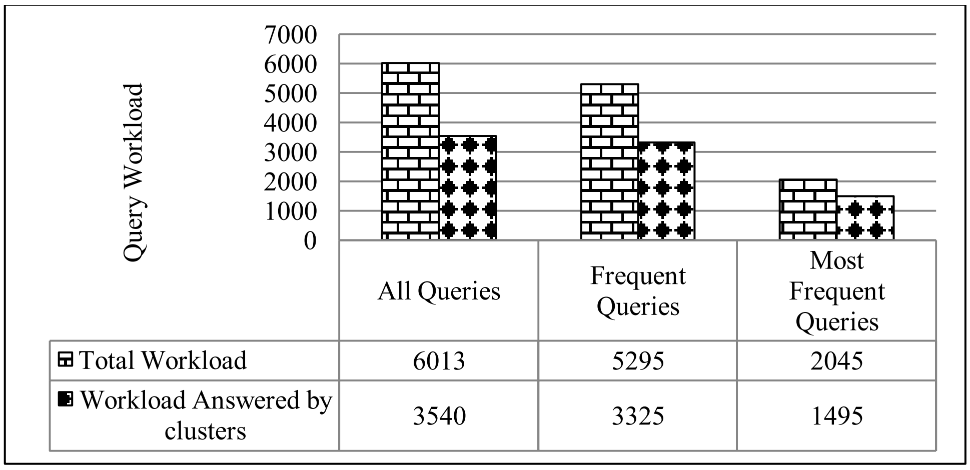
### 4.4.3. Query Execution for WAHP

WAHP is a workload-aware hybrid partitioning technique which generates a set of binary tables and property tables based on query workload. WAHP is a combination of BT and PT. Workload-aware clusters are constructed using WAHP algorithm which uses hot schemas and relevant workload. Query performance is analyzed for workload-aware clusters. Queries are classified as *all queries*, *frequent queries,* and *most frequent queries* where *all queries* mean all the queries designed for the dataset, *frequent queries* are top 50% of all queries based on query frequency and *most frequent queries* are top 10% of frequent queries based on query frequency.

In case of *all queries*, the average QET is estimated to be 7 seconds for original data and 6 seconds is estimated for workload-aware clusters. All the queries running on workload-aware clusters provide an average percentage time gain of 22%.

The average QET for *frequent queries* when running on original data is observed to be 10 seconds and 9 seconds on workload-aware clusters. *Frequent queries* when running on workload-aware clusters provide a time gain of 19%.

For *most frequent queries*, the average QET for original data is 19 seconds and for workload-aware clusters, it is estimated to be 14 seconds. Time gain of 5 seconds is observed when queries were executed on workload-aware clusters. It is seen that workload-aware clusters can answer 73% of the amount of most frequent query workload by using just 9% of the actual data.



| | All Queries | Frequent Queries | Most Frequent Queries |
|---|---|---|---|
| ▦ Total Workload | 6013 | 5295 | 2045 |
| ■ Workload Answered by clusters | 3540 | 3325 | 1495 |

**Figure 4.11 Workload Answered by Clusters**

It is observed that there is an improvement in QET for queries when they are executed on workload-aware clusters as compared to original data. Query 9 provides 98% time gain when submitted to workload-aware clusters. Query 10 shows 70% of time gain. Likewise, Query 2 and Query 5 shows a gain of 49% and 50% respectively. Query10 gives the best performance using workload-aware clusters with a time gain of 96.8%. On an average 37% of time gain is observed for query runs for all queries. The total time saved by *most frequent queries* when submitted to workload-aware clusters is observed to be 16 minutes i.e. on an average 1.7 minutes per query. Figure 4.11 depicts statistics for workload answered by workload-aware clusters in comparison to total workload. Workload is summation of query frequencies. Total workload for all the queries means summation of query frequencies of all the queries. Total workload for frequent queries means summation of query frequencies of all the frequent queries. Total workload for most frequent queries means summation of query frequencies of all the most frequent queries. Likewise, Workload answered by workload aware clusters means summation of those query frequencies which are answered by workload aware clusters.

## 4.5. Comparing RDF Data Stores

Various RDF data storage techniques and their query performances are evaluated using the set of metrics discussed in section 4.3.2.

**Query Type 1:** It retrieves records for specific properties. In this case, if query retrieves a single property then BT will be the best suitable technique as it is evident in *OPR* metric. If a query asks for multiple properties, it is necessary to see if there is a tendency of this group of properties to occur together in a table. If the properties tend to occur together then PT will give best case performance provided table is not wide and contains number of nulls below the threshold value. In other cases, hybrid approaches will perform better. DAHP will perform best when there are fewer unions between PT and BT. DASIVP may perform best as there are index graph and query graph already prepared for the specific query. However, there can be overhead of lookup and merge cost based on the query. Based on break-even analysis, on an average break-even frequency for DASIVP turns out to be 98 and for DAHP break-even frequency is 4.

**Query Type 2:** These types of queries ask for information of a specific subject. If it's specific to a subject and specific to a single property then BT will be best as there will be only one record from the BT provided it is not a multi-valued property. Even if, it is a multi-valued property, it is easier and faster to retrieve such query from BT as it scans only a single table out of the huge database. These details are clear from *OPR* metric for such query types. If a query needs multiple properties for a specific subject and BT is used to store data, there will be a large number of unions based on the number of tables accessed which may affect query performance. When PT is used to store data for such queries, PT may contain a huge number of nulls resulting in a sparse matrix. There can be a lot of overhead handling nulls and complex joins. It should also be noted that if data is well structured there will be fewer nulls which can be derived from *SR* metric. Hybrid approaches help in eliminating the problem of nulls and complex joins. In case of DAHP, properties are clustered based on relatedness of the properties and when it comes to subject-specific queries, it is not necessary that all the subjects belong to the same cluster. However, problem of nulls is almost eliminated due to the null threshold used in DAHP and hence a less sparse matrix is seen. Also, there are less BT tables used in DAHP which means fewer unions. Break-even point for using DAHP data storage technique is 6 and for DASIVP it is 97.

**Query Type 3:** Range queries fall in this category. These queries are administrative in nature. It can ask for a range of subjects or properties. Generally, a range of properties is queried more compared to a range of subjects. If such query asks for a range of a particular property, BT can be used to gain query performance as evident from *OPR* metric. If the query asks for a range of a subject, it will be hard to perform the union of all tables in BT. For property table, if PT is not wide it can perform well based on structuredness of data which can be achieved using *SR* metric. Structured data will have fewer nulls in PT. If a range of subject is asked, then in case of PT, union of several PT will be needed. However, performing union operation in DAHP is preferred than performing unions in PT. Hybrid approaches work well for such kind of queries. If it is a range of properties which is queried, DAHP gains better performance than PT due to less number of nulls as related properties are tied together in DAHP. And if it is a range of subjects which is queried, then it will result into fewer unions compared to PT. Break-even point of 4 is achieved for DAHP. In case of DASIVP, a respective query graph and an index graph is going to be there which may help to achieve best query performance. However, in DASIVP, there is an overhead of lookup and merge time. Break-even point for DASIVP is 95.

**Query Type 4:** Such queries ask for data for specific properties or specific subjects. They are a combination of type 1 and type 2 queries. If such queries ask for a specific property for a specific subject or a specific subject for a specific property, BT is going to give best case performance which is obvious from *OPR* metric. Discussions for type 1 and type 2 queries are applicable to all other cases and hence are not repeated here.

## 4.6.Summary

This chapter proposes and demonstrates data-aware and workload-aware hybrid approaches to store RDF data. It discusses the methodology, data structures, algorithms, execution flow and all the experimental details. The chapter also proposes a set of metrics to measure the suitability of a RDF store for a particular kind of dataset. Based on the metrics, it compares all the RDF data stores, which include basic RDF data stores and hybrid RDF data stores.

# Chapter 5

## DWAHP: Workload-Aware Hybrid Partitioning and Distribution for RDF Data

A lot of work has been carried out to efficiently manage RDF data on a single machine. However, RDF data has increased to a point where managing RDF data on a single node is not adequate. Efforts are being made to achieve high-performance data management, by distributing data on multiple nodes, so that the data can be processed in parallel and can eventually speed up query processing. In this scenario, message passing across nodes is of vital importance for a query to execute. It is obvious that increased number of inter-node communication will become a performance bottleneck. So it is necessary to partition and distribute data such that message passing can be diminished for faster query execution.

This chapter presents a solution for faster query execution for RDF queries, in a distributed environment. The chapter proposes DWAHP, a hybrid partitioning approach to distribute RDF data based on query workload. The proposed approach has two phases: Phase 1 is Hybrid Partitioning, which partitions data considering actual query workload and generates workload-aware clusters consisting of binary tables and property tables. Property tables contain properties that frequently tend to get queried together above some support threshold and binary tables are prepared for the properties that have lower support threshold taking into consideration the query frequencies. Phase 2 is a Distribution Scheme, which distributes RDF data using an *n-hop Property Reachability Matrix*. This matrix helps in proper placements of the clusters on respective nodes as it is acquainted with the underlying relationship between properties in the dataset. DWAHP offers two-fold advantages. First, Hybrid Partitioning phase keeps the data which is queried together as a partition and hence reduces number of joins at the time of query processing. Second, Distribution Scheme phase uses an *n-hop Property Reachability Matrix* to distribute data, which helps in diminishing message passing across nodes such that maximum number of queries can be answered with minimum number of message passing.

## 5.1.DWAHP

DWAHP is a workload-aware approach for hybrid partitioning of data in a distributed environment. DWAHP consist of two phases: Phase 1 is Hybrid Partitioning which generates clusters based on query workload. These workload-aware clusters contain property tables and binary tables. Phase 2 is a Distribution Scheme which distributes data by positioning workload-aware related clusters on the same node such that, the data that is queried together is generally found to be stored on the same node. The workload-aware related cluster is found using Property Reachability Matrix. It is a fact that real life queries involve many joins. To answer a query with joins, one needs to know how these properties are related to each other. Hence the presented work not only studies query workload to find which properties are queried together but also look into the actual structural relationship between properties in the dataset.

### 5.1.1. Phase 1: Hybrid Partitioning

DWAHP Phase 1 considers queries and query workload in order to achieve suitable partitioning method. It generates workload-aware clusters which contain a set of binary tables and property tables. Workload-aware clusters are generated for the properties that are queried together. Query workload helps us derive property frequencies and their occurrences with other properties. Apriori algorithm [**89**] is used to determine all possible occurrences of frequent properties, as it uses join and prune property for execution and generate all possible candidates of frequent property sets. These all possible frequent property sets are found based on null and support threshold. Null threshold indicates the acceptable percentage of null values for a property table and support threshold specifies the acceptable percentage of occurrence of properties together in a query set. It results in a set of binary tables and property tables based on the query workload. Properties that are queried together and have acceptable null and support threshold becomes the candidate of a property table and properties that have lesser support threshold or higher null threshold becomes separate binary tables. This results in vertical partitioning of RDF data. This combination of property tables and binary tables helps in reducing the number of traversals in the RDF data and query the required properties faster. DWAHP Phase 1 is depicted in Figure 5.1.

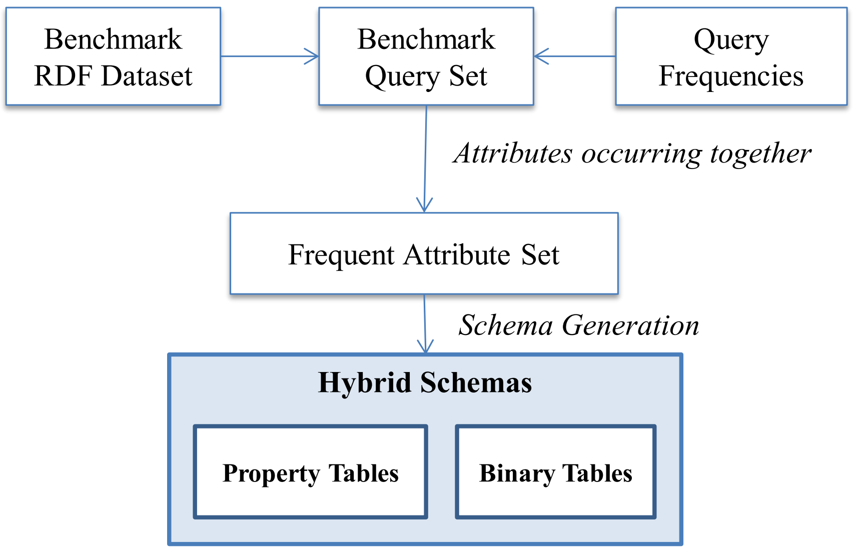


**Figure 5.1 DWAHP Phase 1: Hybrid Partitioning**

### 5.1.2. Phase 2: Distribution Scheme

The more connected RDF data is the more it is difficult to partition and distribute this data. RDF data and queries have certain characteristics which can be exploited to partition the data and distribute it over several nodes. RDF data contains rdf:type predicate that generally exists in a large number, as it connects all types in the RDF data. This makes data partitioning little complex. The idea is to exploit this fact and to make a separate binary table of rdf:type and replicate it across multiple nodes. It has three fold advantages. 1) Had rdf:type been a part of property table, it may contain a lot of nulls and will ultimately result in space overhead. Making rdf:type a binary table, helps in minimizing space overhead. 2) Considering rdf:type as a part of property table will also result in a huge table with a lot of null storage space. This will eventually deteriorate performance for complex operations like joins and unions, because of additional time consumption in the scan and join operations between such huge tables. Preparing a separate binary table for rdf:type, will reduce complexity in join and union operations. 3) Mostly rdf:type joins other properties for the queries, so making it a separate binary table helps in the faster scan as it can be indexed and compressed.

Distribution of these generated tables can be done in many ways. The easiest way is to distribute the same schema over all nodes. This type of design may result in more number of message passing between nodes to answer a query. Researchers have worked on this issue and have distributed data in various manners that can reduce message passing [**45**][**57**][**20**].

DWAHP focuses on distributing data such that related clusters are allotted to the same node so that messaging passing can be decreased. DWAHP Distribution Scheme is represented in Figure 5.2. Most of the real-life queries involve multiple joins, especially subject-subject and subject-object joins which are common in RDF queries. *n-hop Property Reachability Matrix* is prepared that identifies how a property is related to another property in the dataset. A single join indicates 1 hop and hence a query having n joins need to have information maximum up to n hop from that property. Property Reachability Matrix is found using the concept of domain and range for RDF data [**93**]. It is well known that rdf:subject and rdf:object is an instance of rdf:property. By definition rdfs:domain is an instance of rdf:property that is used to state that any resource that has a given property is an instance of one or more classes and rdfs:range is an instance of rdf:property that states the value of the property are an instance of one or more classes. An obvious overlap is found in between domain and range for certain properties. This overlap is identified between subject–subject and subject–object for the properties, and a Property Reachability Matrix is prepared. This matrix is hence able to reveal join types between properties. This information is exploited and used to distribute data among nodes.

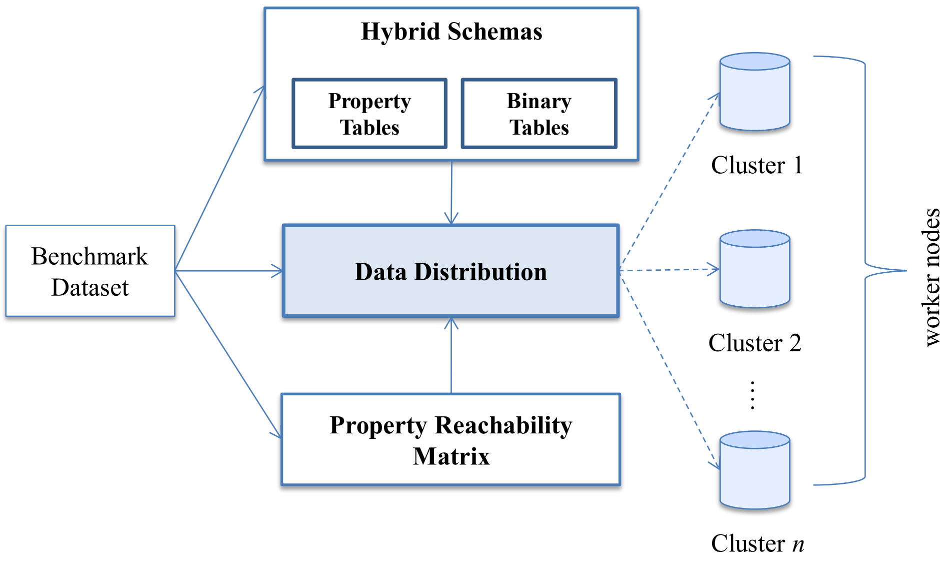


**Figure 5.2 DWAHP Phase 2: Distribution Scheme**

DWAHP knows number of joins in a query, and it also knows the underlying relationship between properties which is addressed using *n-hop Property Reachability Matrix*. For example, there exists a query which finds a location of a place in terms of latitude and longitude with reference to Linked Sensor Data [94] [95]. This query can be answered using these properties: longitude, latitude, and process location. To answer this query one needs to traverse 2 hops maximum from a property to reach another property. It is obvious it will require 2 joins and both are subject-object joins. If it is a frequent query, one can use 2-hop Reachability Matrix, and position these properties on nodes such that message passing can be avoided or minimized. This scheme will provide following advantages. 1) There is an increased possibility of answering frequent queries using a single node as the distribution is based on prior knowledge of queries and their properties. 2) When related clusters are there on the same node, there is less possibility for message passing and hence complex operations like joins become efficient. 3) The data is partitioned vertically considering query frequency and then it is distributed among nodes which help manage data efficiently.

### 5.1.3. Architecture

DWAHP architecture is depicted in Figure 5.3. RDF triples are partitioned using Hybrid partitioning phase and clusters are created considering query workload. These clusters are a combination of binary tables and property tables. Once the data is partitioned, Distribution phase is responsible to distribute clusters on different nodes.

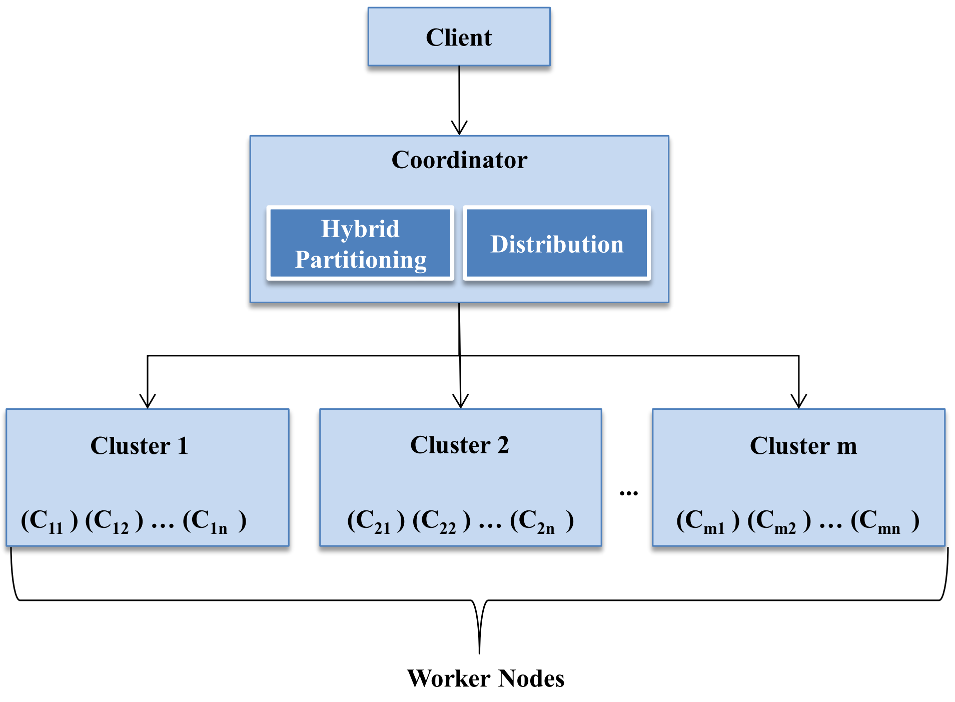


**Figure 5.3 DWAHP Architecture**

Worker nodes contain actual data inside clusters and coordinator contains information related to data residing on different nodes. Coordinator checks if the query is parallelizable or not. If the query is parallelizable, it is executed on relevant nodes in parallel and results of the local nodes are computed on the node itself. On the other hand, if a coordinator finds that a query needs inter-node communication, it generates a query plan where a data node having a lesser amount of data ships its intermediate results to other nodes. At the end of query processing, coordinator returns results to the client.

## 5.2. Experiments

This section discusses all experimental details which include dataset, query set, data structures, test bed, and implementations for DWAHP.

### 5.2.1. Data Structures and Algorithms

DWAHP uses five data structures to identify workload-aware clusters. (a) *Query Property Basket*, which contains a list of queries with its associated queried properties. (b) *Query Frequency List* holds total frequency count. (c) *Query Frequency Property Basket* stores query frequency and their corresponding properties (d) *Property Frequency List* is primed using *Query Property Basket* and *Query Frequency List* (e) *n-hop Property Reachability Matrix* which is a *n*n* matrix of properties.

| Q | Property List |
|---|---|
| Q1 | P1, P2, P3,P4 |
| Q2 | P2, P5 |
| Q3 | P1, P2, P5 |

**(a) Query Property Basket**

| Q | Site1 | Site2 | Site3 | *f* |
|---|---|---|---|---|
| Q1 | 100 | 200 | 300 | 600 |
| Q2 | 150 | 250 | 100 | 500 |
| Q3 | 400 | 350 | 250 | 1000 |

**(b) Query Frequency List**

| *f* | Property List |
|---|---|
| 100 | P1, P2, P3,P4 |
| 200 | P1, P2, P3,P4 |
| 300 | P1, P2, P3,P4 |
| 150 | P2, P5 |
| 250 | P2, P5 |
| 100 | P2, P5 |
| 400 | P1, P2, P5 |
| 350 | P1, P2, P5 |
| 250 | P1, P2, P5 |

**(c) Query Frequency Property Basket**

| Property | *f* |
|---|---|
| P1 | 1600 |
| P2 | 1100 |
| P3 | 600 |
| P4 | 600 |
| P5 | 1500 |

**(d) Property Frequency List**

|  | P1 | P2 | P3 | P4 | P5 |
|---|---|---|---|---|---|
| P1 | -2 | -1 | 2 | -1 | -1 |
| P2 | -1 | -2 | -1 | -1 | 1 |
| P3 | 2 | -1 | -2 | 2 | **0** |
| P4 | -1 | -1 | 2 | -2 | -1 |
| P5 | -1 | 1 | **0** | -1 | -2 |

**(e) Property Reachability Matrix**

**Figure 5.4 DWAHP Data Structures**

It can contain either of these values in the matrix: -2,-1, *m* or 0 where -2 indicates the property itself, -1 indicates both properties are directly reachable, *m* indicates property number through which it is reachable and 0 indicates they are not reachable for the value of *n*. Figure 5.4 represents DWAHP data structures. All data structures execute in either linear or polynomial time. DWAHP phase 1: hybrid partitioning uses algorithm same as WAHP which is described in Chapter 4. DWAHP Phase 2: distribution scheme uses two algorithms 1) n-hop Property Reachability and 2) cluster creation and allocation.

```
Algorithm: n-hop Property Reachability
Input: properties
Output: n-hop property reachability
1. N ←n
2. For all $p_i$ ∈ properties
3.    For all $p_j$ ∈ properties
4.       If i==j
5.          P (i,j) ← -2
6.       Else If   s.s join ∨ s.o join
7.          P (i,j) ← -1
8.       Else
9.          P (i,j)  ← 0
10.         End if
11.      End for
12.   End for
13.   // Dijkstra's Algorithm
14.   For every hop in property reachability matrix till N
15.         If p (i,j)  == 0
16.            p (i,j)  = $p_k$ //minimum distance from $p_i$ to $p_j$ via $p_k$
17.         End if
18.   End for
19.   Return n-hop property reachability
```

DWAHP algorithms converge in either linear or polynomial time. Hybrid partitioning requires O (n) + O ($p^2$ * q) operations where n is total number of records in dataset, p is number of properties in the dataset and q denotes candidates in *Query Property Basket*. DWAHP distribution algorithm requires O ($p^2$) + O ($c^2$) operations where p denotes the number of properties in datasets and c denotes the number of clusters. n-hop Property Reachability algorithm needs O ($p^2$) operations.

```
Algorithm: Cluster Creation and Allocation
Input: property table, binary table, n-hop property reachability matrix,
support threshold, Nodes
Output: Distributed Clusters
1. Together ← False
2. For $C_i$ consider every combination of $p_i$ in $T_1$ with $p_j$ in $T_2$
3.   For all  $p_k$ in n-hop property reachability matrix
4.      If  $p_k$ is Reachable $p_i$,$p_j$   ∧  supportThreshold $p_i$,$p_j$
5.         Together ← True
6.      End if
```

```
7.    End for
8.    If  Together
9.          $C_i \leftarrow C_i \cup \{T_1, T_2\}$
10.      End if
11.    Together ← False
12.    End for
13.    For all $C_i \in$ Clusters
14.         For all $C_j \in$ Clusters
15.            If all properties is Reachable $C_i$, $C_j$
16.               Together ←True
17.            Else
18.               Together=False
19.               Exit
20.         End for
21.         If  Together
22.               $N_k \leftarrow N_k \cup C_j \cup C_i$
23.         Else
24.               $N_k \leftarrow N_k \cup C_j$
25.         End if
26.    End for
27.    Return Distributed Clusters
```

### 5.2.2. Dataset

Linked Observation Data [**96**] is a benchmark RDF dataset which contains descriptions of blizzard and hurricane observations. Linked Sensor Data [**94**] is a part of Linked Observation Data which measures temperature, precipitation, wind speed, wind direction, humidity etc through sensors located at weather stations. The dataset used consists of maximum up to 8 GB of data and contains more than 35 million triples. Details of the Linked Observation Data is given in Table 5.1

**Table 5.1 Linked Observation Dataset Specifications**

| Specifications | Linked Observation Data |
|---|---:|
| No of Triples | 35105649 |
| No of distinct Triples | 35105649 |
| No of unique properties | 20 |
| Size on disk | 8 G.B |

### 5.2.3. Query set

The query set contains 20 queries. These RDF queries are categorized into four different query types as represented in Figure 5.5. Type 1 queries are linear queries which retrieve predetermined set of properties from the RDF dataset. Type 2 contains star queries which involve selection of properties defined for a particular subject. Type 3 queries are range queries or administrative queries, and Type 4 are snowflake queries that retrieve predetermined properties or subjects. Out of these 20 queries, 5 queries are of Type1, 6 queries belong to Type2, 4 queries belong to Type3 and 5 queries are of Type4. All the SQL queries for Linked Observation dataset are listed in Appendix 3.

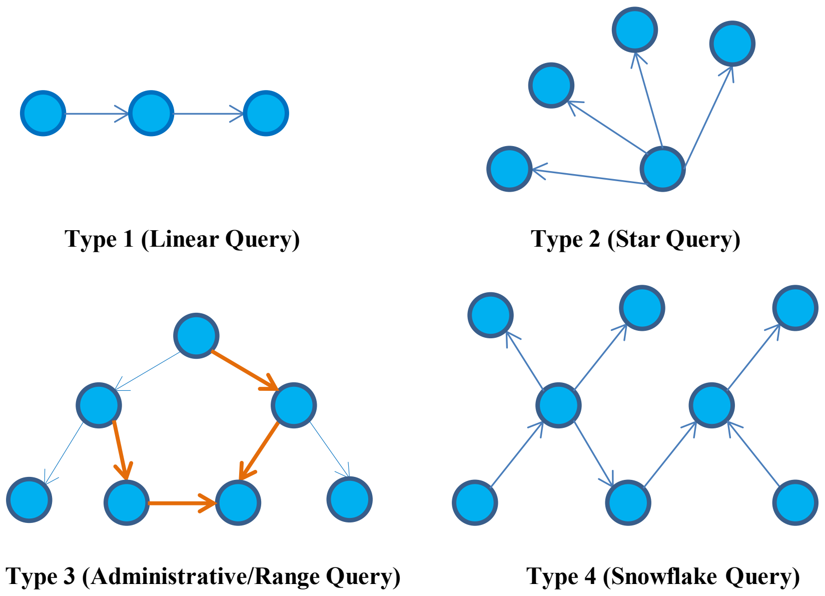


**Figure 5.5 Query Types**

### 5.2.4. Harwares and Softwares

A test bed is created to implement DWAHP and execute queries for the Linked Observation dataset. The dataset is in RDF/XML format which is converted to triples by using Jena parser [**87**]. Eclipse 3.5 and Java 1.8 are used to implement the parser and other algorithms of the experiment. The experiment uses Postgres-XL [**97**] which is a Postgresql based database cluster tool installed on a Linux based machine. Deployed worker nodes and coordinator, each is configured with 1.7 GHz Intel Core i5-4210U processor and 8GB of RAM.

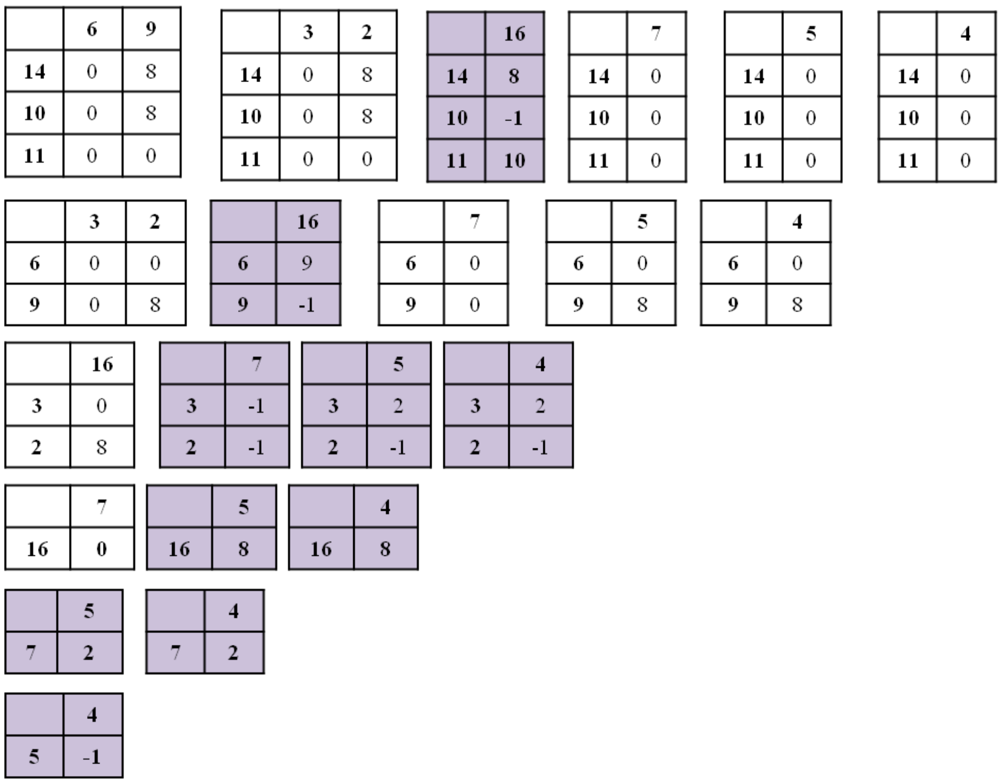
### 5.2.5. Implementation

Benchmark Linked Sensor Dataset is used to implement and evaluate storage and query performance for the proposed system. Phase 1 generates workload-aware clusters and populates them and Phase 2 distributes clusters among nodes by using *n-hop Property Reachability Matrix*. For this experiment, n = 2. *n-hop Property Reachability Matrix* for n=2 is shown in Figure 5.6. The dataset contains 20 unique properties and all properties are numbered from 1 to 20. Let M be the n-hop reachability matrix and i and j be the rows and columns respectively. In Figure 5.6, M(i,j) = -1 in the cell denotes property $P_i$ is directly reachable from $P_j$, M(i,j) = -2 denote the property itself, M(i,j) = 0 denotes a property is not reachable in n hops and M(i,j) = $P_x$, denotes that $P_i$ is reachable by $P_j$ via property $P_x$. DWAHP has generated 7 tables, out of which 3 are property tables: (14,10,11), (6,9), and (3,2). There are four binary tables: (16), (7), (5), and (4).

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **1** | -2 | -1 | 2 | -1 | -1 | 0 | 2 | -1 | 8 | 8 | 0 | 2 | 2 | 8 | 8 | 8 | 0 | 8 | 8 | 0 |
| **2** | -1 | -2 | -1 | -1 | 1 | 0 | -1 | -1 | 8 | 8 | 0 | -1 | -1 | 8 | 7 | 8 | 7 | 8 | 8 | 0 |
| **3** | 2 | -1 | -2 | 2 | 0 | 0 | -1 | 2 | 0 | 0 | 0 | -1 | -1 | 0 | 7 | 0 | 7 | 0 | 0 | 0 |
| **4** | -1 | -1 | 2 | -2 | -1 | 0 | 2 | -1 | 8 | 8 | 0 | 2 | 2 | 8 | 8 | 8 | 0 | 8 | 8 | 0 |
| **5** | -1 | 1 | 0 | -1 | -2 | 0 | 0 | -1 | 8 | 8 | 0 | 0 | 0 | 8 | 8 | 8 | 0 | 8 | 8 | 0 |
| **6** | 0 | 0 | 0 | 0 | 0 | -2 | 0 | 9 | -1 | 0 | 0 | 0 | 0 | 0 | 0 | 9 | 0 | 9 | 9 | 0 |
| **7** | 2 | -1 | -1 | 2 | 0 | 0 | -2 | 2 | 0 | 0 | 0 | 2 | 2 | 0 | -1 | 0 | -1 | 0 | 0 | 0 |
| **8** | -1 | -1 | 2 | -1 | -1 | 9 | 2 | -2 | -1 | -1 | 10 | 2 | 2 | -1 | -1 | -1 | 15 | -1 | -1 | 10 |
| **9** | 8 | 8 | 0 | 8 | 8 | -1 | 0 | -1 | -2 | 8 | 0 | 0 | 0 | 8 | 8 | -1 | 0 | -1 | -1 | 0 |
| **10** | 8 | 8 | 0 | 8 | 8 | 0 | 0 | -1 | 8 | -2 | -1 | 0 | 0 | 8 | 8 | -1 | 0 | 8 | 8 | -1 |
| **11** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 10 | 0 | -1 | -2 | 0 | 0 | -1 | 0 | 10 | 0 | 0 | 0 | -2 |
| **12** | 2 | -1 | -1 | 2 | 0 | 0 | 2 | 2 | 0 | 0 | 0 | -2 | -1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **13** | 2 | -1 | -1 | 2 | 0 | 0 | 2 | 2 | 0 | 0 | 0 | -1 | -2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **14** | 8 | 8 | 0 | 8 | 8 | 0 | 0 | -1 | 8 | 8 | -1 | 0 | 0 | -2 | 8 | 8 | 0 | 8 | 8 | -1 |
| **15** | 8 | 7 | 7 | 8 | 8 | 0 | -1 | -1 | 8 | 8 | 0 | 0 | 0 | 8 | -2 | 8 | -1 | 8 | 8 | 0 |
| **16** | 8 | 8 | 0 | 8 | 8 | 9 | 0 | -1 | -1 | -1 | 10 | 0 | 0 | 8 | 8 | -2 | 0 | -1 | -1 | 10 |
| **17** | 0 | 7 | 7 | 0 | 0 | 0 | -1 | 15 | 0 | 0 | 0 | 0 | 0 | 0 | -1 | 0 | -2 | 0 | 0 | 0 |
| **18** | 8 | 8 | 0 | 8 | 8 | 9 | 0 | -1 | -1 | 8 | 0 | 0 | 0 | 8 | 8 | -1 | 0 | -2 | -1 | 0 |
| **19** | 8 | 8 | 0 | 8 | 8 | 9 | 0 | -1 | -1 | 8 | 0 | 0 | 0 | 8 | 8 | -1 | 0 | -1 | -2 | 0 |
| **20** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 10 | 0 | -1 | -2 | 0 | 0 | -1 | 0 | 10 | 0 | 0 | 0 | -2 |

**Figure 5.6 n-hop Property Reachability Matrix for n=2**

rdf:type is a binary table that gets replicated on all nodes as discussed in section 5.1.2. The number assigned to rdf:type is (8). Besides rdf:type, DWAHP made another binary table to become a candidate to be replicated on two nodes. This property is numbered (16). It is due to its association with other tables on the nodes based on query workload. All tables are indexed on the subject. Default B-tree indexing of Postgres-XL is used to index tables. Figure 5.7 shows cluster allocation using *n-hop Property Reachability Matrix*. The highlighted matrix in Figure 5.7 shows the possible set of properties to be candidate of workload-aware clusters. Cluster Creation and Allocation algorithm is used to find the clusters and its distribution scheme for various nodes. Figure 5.8 depicts the actual cluster distribution for a given query workload.



| | 6 | 9 |
|---|---|---|
| 14 | 0 | 8 |
| 10 | 0 | 8 |
| 11 | 0 | 0 |

| | 3 | 2 |
|---|---|---|
| 14 | 0 | 8 |
| 10 | 0 | 8 |
| 11 | 0 | 0 |

| | 16 |
|---|---|
| 14 | 8 |
| 10 | -1 |
| 11 | 10 |

| | 7 |
|---|---|
| 14 | 0 |
| 10 | 0 |
| 11 | 0 |

| | 5 |
|---|---|
| 14 | 0 |
| 10 | 0 |
| 11 | 0 |

| | 4 |
|---|---|
| 14 | 0 |
| 10 | 0 |
| 11 | 0 |

| | 3 | 2 |
|---|---|---|
| 6 | 0 | 0 |
| 9 | 0 | 8 |

| | 16 |
|---|---|
| 6 | 9 |
| 9 | -1 |

| | 7 |
|---|---|
| 6 | 0 |
| 9 | 0 |

| | 5 |
|---|---|
| 6 | 0 |
| 9 | 8 |

| | 4 |
|---|---|
| 6 | 0 |
| 9 | 8 |

| | 16 |
|---|---|
| 3 | 0 |
| 2 | 8 |

| | 7 |
|---|---|
| 3 | -1 |
| 2 | -1 |

| | 5 |
|---|---|
| 3 | 2 |
| 2 | -1 |

| | 4 |
|---|---|
| 3 | 2 |
| 2 | -1 |

| | 7 |
|---|---|
| 16 | 0 |

| | 5 |
|---|---|
| 16 | 8 |

| | 4 |
|---|---|
| 16 | 8 |

| | 5 |
|---|---|
| 7 | 2 |

| | 4 |
|---|---|
| 7 | 2 |

| | 4 |
|---|---|
| 5 | -1 |

**Figure 5.7 Clusters and Allocation**

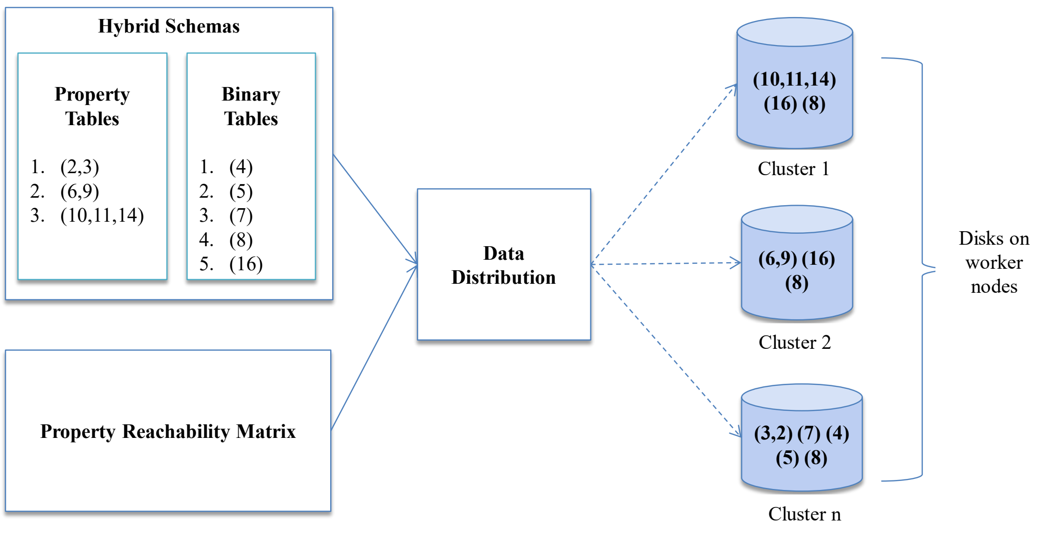


**Figure 5.8 Cluster Distribution using DWAHP**

***JARS implementation**:* JARS [**20**] is a join aware distributed RDF storage system. It uses two layered clustered indexes with dual hashing for distribution of triples. Triples are hashed twice based on subject and object. The resulting hash value is divided by the number of nodes and the remainder becomes the target node for the triple. The triple is then allotted to the target node.

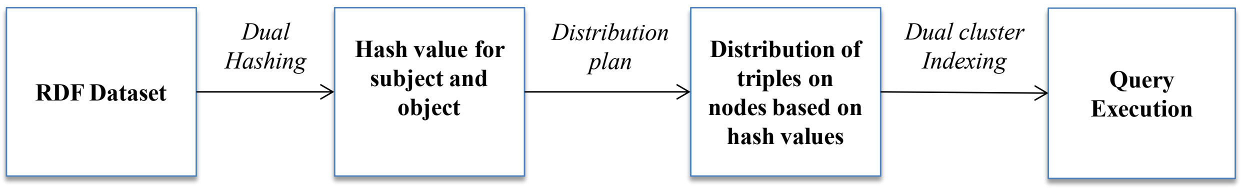

RDF Dataset
Dual Hashing
Hash value for subject and object
Distribution plan
Distribution of triples on nodes based on hash values
Dual cluster Indexing
Query Execution

**Figure 5.9 Execution Flow using JARS**

This experiment is performed on the same configuration as it is done for DWAHP. Every node has one subject triple table and one object triple table. JARS store triples twice so the data is almost doubled. All subject triple tables are cluster indexed like: *pos*, *pso*, *osp*, and *spo*. All object triple tables are indexed like: *pos, pso, sop*, and *ops*. Figure 5.9 depicts execution flow for JARS. Figure 5.10 depicts an actual implementation of JARS on the test bed.

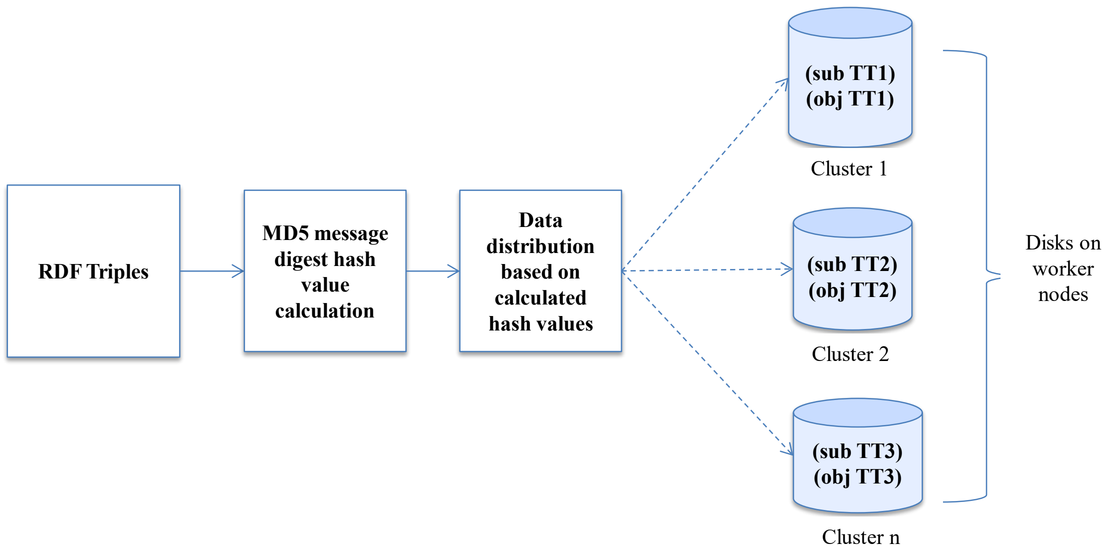


**Figure 5.10 Cluster Distribution using JARS**

## 5.3. Results and Discussions

This section discusses DWAHP results and its performance comparison with JARS. Data scaling is performed for DWAHP and JARS both. Data is scaled from 2GB, 4GB, 6GB, till 8GB. This section discusses results for 8GB of data. Query performance for DWAHP is evaluated using following parameters:

a. **Query Execution Time** is the time taken by a query to execute.
b. **Query Cost** is the cost of the query execution plan, which is a summation of costs of different database operations like joining, scanning, indexing, and sorting.
c. **Storage Space** is the space occupied by the database on the disk.
d. **Inter-node Communication** parameter identifies message passing between nodes to answer a query.

The RDF query set is executed on both the systems and query execution time is recorded in milliseconds. Query execution time is averaged over three query runs for both the systems to minimize fluctuations. Real life RDF queries often include linear and star queries. In the query set, queries which belong to Type1 and Type2 are linear and star queries respectively. With the blend of workload-aware hybrid partitioning and proposed distribution scheme, star and linear queries are able to get answered without inter-node communication, which naturally results in faster query execution of these queries.

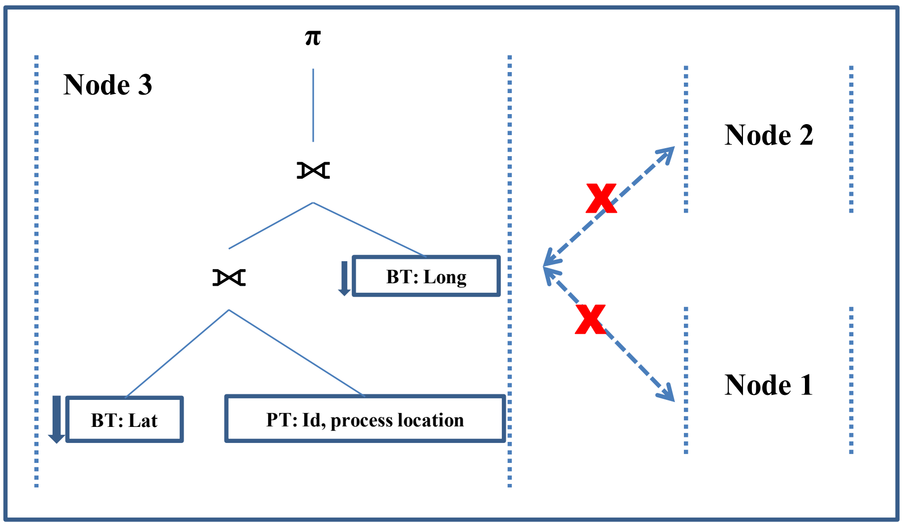


**Figure 5.11 Example Execution Plan for Query Type2**

In the query set, out of 20 queries 12 queries i.e. 60% of the total queries are answered without inter-node communication. When the queries are looked at in descending order of query frequency, Out of top 10 frequent queries, 9 queries are answered without inter-node communication, which indicates that 90% of the most frequent queries are answered without inter-node communication. Query workload analysis is also carried out for the amount of workload answered by the proposed system. Different queries have different occurrence frequencies. The overall workload of the system is based on the summation of frequencies of all the queries targeted for the system. DWAHP is able to answer 67% of the total query workload and 83% of frequent query workload without inter-node communication. Queries which needed inter-node communication were executed in parallel using the default mechanism of Postgres-XL.

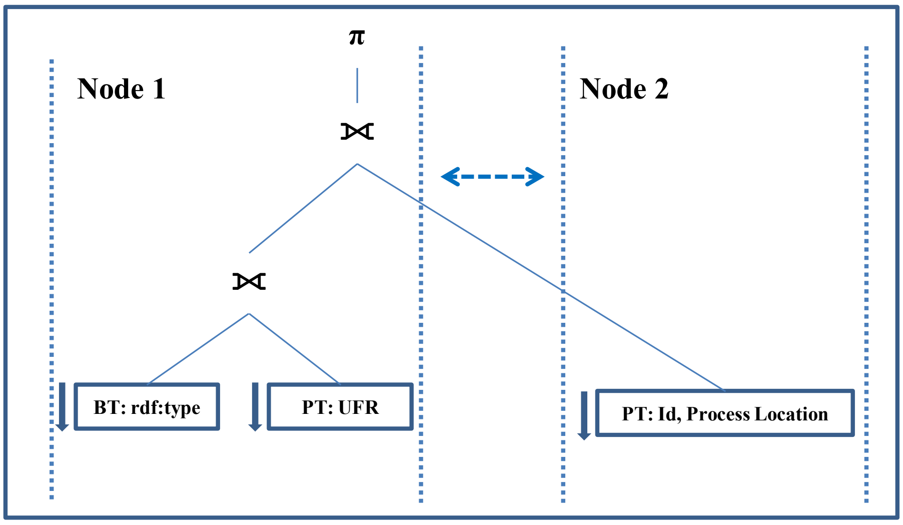


**Figure 5.12 Example Execution Plan for Query Type3**

The Query set in the experiment contains queries with different number of joins. In order to execute these queries using triple tables, one needs minimum two and maximum up to six joins. However, DWAHP generates clusters consisting set of property tables and binary tables considering query workload. As a result, it minimizes these joins and requires maximum up to three joins. It can be noted that 1) Property table reduces subject-subject joins. 2) subject-object joins and inter-node communications are managed by keeping related binary tables and property tables together on the same node which is derived from *n-hop Property Reachability Matrix*. For example, during this experiment, DWAHP replicated one such binary table due to its association with other nodes. Because of these reasons, a significant reduction in inter-node communication and number of query joins is observed. Considering these two points, similar kind of results are expected for any dataset or query workload because *n* in the *n-hop Property Reachability Matrix* can be tweaked accordingly. Setting *n* to a point, where at least one property is reachable from every property can result in a situation, where communication among nodes can be eliminated for the star and linear queries. Value of *n* may vary from dataset to dataset. If *n* needs to be set at some higher value, which indicates that a longer path is needed to reach at least one property from all other properties. Clearly, this tells that the number of joins in the query may also increase. Hence, query performance parameters like query cost and storage-space may get affected. This trade-off may be dealt with compromising on inter-node communication to achieve efficient query execution. However, parameter like inter-node communication, not only depend on number of joins but also on the hardware capacity of worker nodes. Worker nodes with sufficient hardware capacity may also help to deal with this trade-off. Overall, it can be said that DWAHP can reduce complex operations like joins to a certain extent that results in efficient query execution for RDF data.

Real life applications access a small amount of data from the entire database. These applications display highly skewed access patterns as they have certain kinds of queries frequently pose on their systems. For such applications, if query workload is known, DWAHP can be utilized for data storage and distribution for efficient query processing, as it exploits workload information. For example, IoT applications produce a lot of data periodically. These applications generally observe skewed access patterns for their queries. DWAHP can be utilized for such applications with skewed query workload and growing data.

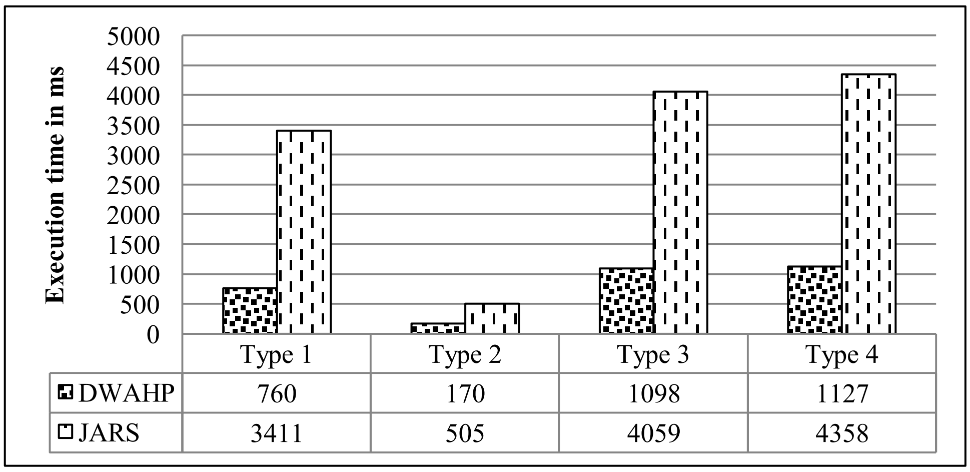
## 5.4.Comparison with state-of-the-art systems

A scalable RDF data storage system [**45**] tries to minimize inter-node communication and uses a parameter called PWOC to find out that it can parallelize queries without inter-node communication. However, it uses data-aware approach, whereas DWAHP uses workload-aware approach and it minimizes inter-node communication by managing joins using *n-hop Property Reachability Matrix*. DWAHP is also able to answer star and linear queries without inter-node communication. Besides this, DWAHP needs less replication compared to this scalable storage system as DWAHP replicates "rdf:type" property on different nodes. WARP [**52**] is quite similar to the scalable RDF data storage system. However, it follows a workload-aware approach and tries to use lesser replication compared to scalable RDF data storage system. WARP uses horizontal partitioning and on the other hand, DWAHP uses vertical partitioning by using a combination of binary tables and property tables. DWAHP is hence able to minimize inter-node communication among nodes. ClusterRDF [**57**] partitions data based on workload and uses both horizontal and vertical partitioning. It distributes same schema on different machines and tries to minimize inter-node communication. DWAHP performs only vertical partitioning and distributes different schemas on different machines and hence able to eliminate inter-node communication for the star and linear queries.

JARS [**20**] is a Join-Aware distributed RDF Storage system. It uses two layered clustered indexes with dual hashing for distribution of triples. JARS manages joins and tries to minimize inter-node communication. DWAHP also manages joins and tries to minimize inter-node communication. JARS uses a data-aware approach and on the other hand, DWAHP uses a workload approach. DWAHP and JARS both use relational based system to manage RDF data. DWAHP is closely comparable to JARS and hence this section presents a comparison between DWAHP and JARS using performance parameters like QET, query cost, storage space and inter-node communication.

### 5.4.1. Comparing DWAHP and JARS

Query performance is compared for DWAHP and JARS. Figure 5.13 shows execution time for Linked Observation dataset queries on DWAHP and JARS.



| | Type 1 | Type 2 | Type 3 | Type 4 |
|---|---|---|---|---|
| ▨ DWAHP | 760 | 170 | 1098 | 1127 |
| ☐ JARS | 3411 | 505 | 4059 | 4358 |

**Figure 5.13 Query Execution for DWAHP and JARS**

It is clearly visible that DWAHP outperforms JARS. Out of 20 queries, DWAHP is able to answer 12 queries without inter-node communication. It includes all linear and star queries. Star queries contain subject-subject joins and linear queries contain subject-object joins.

DWAHP is able to answer all star and linear queries without inter-node communication because it distributes clusters based on *n-hop Property Reachability Matrix*, which covers subject-subject and subject-object joins between properties. JARS is able to answer 6 queries all of which are star queries. JARS manage joins by distributing triples based on subject/object hash values. So subjects with same hash values appear on same node. As star queries consist of subject-subject joins, JARS is able to answer all star queries without inter-node communication.

DWAHP utilizes query workload information to partition data and also manages joins by partitioning it into property tables and binary tables. On top of that, data is distributed considering underlying property relationships which not only reduces number of joins but also reduces inter-node communication. Comparing DWAHP query performance based on query types with JARS, it can be noticed that Type1 queries executes with an average time gain of 78%, Type2 queries executes with an average time gain of 66%, Type3 queries executes with an average time gain of 73% and Type4 queries executes with an average time gain of 74%. DWAHP on an average gives 72% better query performance than JARS for the entire query set. Besides query execution time gain query costs are also calculated for all the queries. DWAHP costs 20%, 50%, 90%, and 85% lesser for Type1, Type2, Type3, and Type4 queries respectively. On an average DWAHP costs 61% lesser than JARS query costs for the entire query set.

There is a significant difference between storage space occupied by DWAHP and JARS. DWAHP uses almost same storage space as the actual database as the triple table is partitioned in a group of binary tables and property tables. However, it occupies additional storage space for tables that are indexed and replicated over different nodes. On the other hand, JARS stores triple twice. So it is obvious that data storage space is doubled. Additionally, it uses dual indexing technique which also adds to storage space. DWAHP occupies 71% less total storage space than JARS. Specifically, DWAHP occupies 64% less index space and 36% less data storage space. So in all DWAHP occupies a lot less space than JARS. The total performance gain in summarized in Table 5.2.

DWAHP minimizes inter-node communication for frequent queries and manages joins efficiently. However, during Phase 1, it needs to partition triple table into a set of property tables and binary tables based on query workload. Whereas JARS does not require such pre-processing of data as it simply uses a hash function and distributes data among nodes.

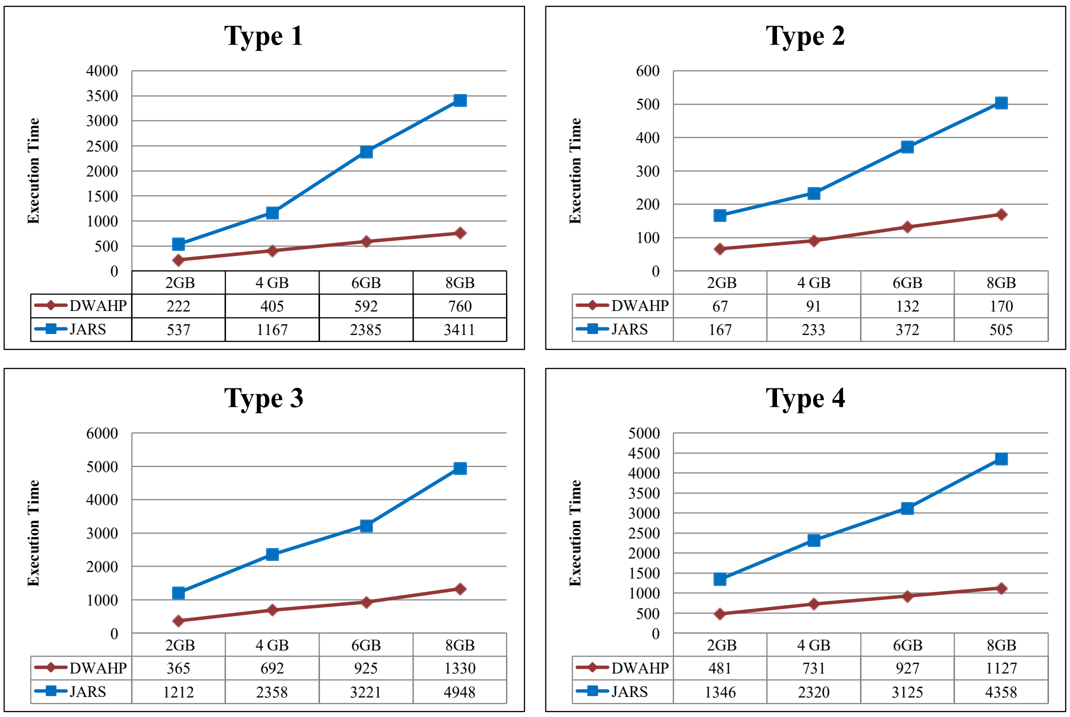
DWAHP uses workload-aware clusters which are created at a certain point of time considering certain query workload. Therefore DWAHP may not be able to execute ad-hoc queries that efficiently. On the other hand, JARS doesn't consider any query workload and may execute ad-hoc queries. Additionally, JARS uses some preliminary RDF compression techniques for disk space reduction.

**Table 5.2 Summary of Query Performance Comparison between DWAHP and JARS**

| **Performance Parameters** | **% Gain using DAWHP over JARS** |
|---|---|
| Query Execution Time | 72% |
| Query Cost | 61% |
| Storage Space | 71% |
| Inter-node Communication | 50% |

### 5.4.2. Comparing DWAHP and JARS using Scaled Data

Data scaling for Linked Observation Data is performed from 2 GB, 4 GB, 6 GB, and 8 GB. It is observed that query performances for all the query types for DWAHP and JARS increase linearly. However, it can be seen from Figure 5.14 that QET slopes for JARS is steeper compared to DWAHP which clearly indicates better performance of DWAHP with increasing data compared to JARS.



Type 1

| | 2GB | 4 GB | 6GB | 8GB |
|---|---|---|---|---|
| DWAHP | 222 | 405 | 592 | 760 |
| JARS | 537 | 1167 | 2385 | 3411 |

Type 2

| | 2GB | 4 GB | 6GB | 8GB |
|---|---|---|---|---|
| DWAHP | 67 | 91 | 132 | 170 |
| JARS | 167 | 233 | 372 | 505 |

Type 3

| | 2GB | 4 GB | 6GB | 8GB |
|---|---|---|---|---|
| DWAHP | 365 | 692 | 925 | 1330 |
| JARS | 1212 | 2358 | 3221 | 4948 |

Type 4

| | 2GB | 4 GB | 6GB | 8GB |
|---|---|---|---|---|
| DWAHP | 481 | 731 | 927 | 1127 |
| JARS | 1346 | 2320 | 3125 | 4358 |

**Figure 5.14 Query Execution for DWAHP and JARS with Scaled Data**

## 5.5.Summary

DWAHP focuses on creating workload-aware clusters and managing joins in order to minimize message passing across nodes. DWAHP create workload-aware clusters which can answer 90% of the most frequent queries without inter-node communication. Star and linear queries are the most frequent query types in real life. It manages joins such that it is able to answer all linear and star queries without inter-node communication. DWAHP and JARS are compared in terms of query execution time, query costs, storage space, and message passing parameters. DWAHP has shown on an average 72% of better query performance than JARS with 61% of reduced query cost. Considering storage space, DWAHP occupies less than one-third of storage space compared to JARS. JARS eliminates inter-node communication for star queries and minimizes inter-node communication for linear queries, whereas DWAHP is able to get rid of inter-node communication for both linear and star queries.

# Chapter 6

## Conclusions and Future Work

The thesis aims at managing RDF data efficiently in terms of storage and query execution time. It initially explores basic RDF data partitioning techniques. It demonstrates use of various data partitioning techniques for efficient RDF data storage and faster query execution. It demonstrates TT (Triple Table), PT (Property Table), BT (Binary Table), HP (Horizontally Partitioned Table), and MV (Materialized Views). It gives detailed study of these basic data partitioning techniques.

Chapter 3 demonstrates query execution for basic RDF data stores. It measures query performance in terms of QET which are averaged over query types. It finds that partitioned tables (PT, BT, and HP) outperform TT and even with scaled RDF data in terms of triples, it is observed that partitioned tables outperform TT. HP can be applied over TT, BT, and PT. BT may perform better when a query involves less number of joins. Moreover, MV can be used over BT to achieve faster execution for queries involving subject-object joins. PT may perform better when the queried set of properties appear in the same table.

The dissertation proposes three hybrid data partitioning techniques namely DAHP (Data-Aware Hybrid Partitioning), DASIVP (Data-Aware Structure Indexed Partitioning), and WAHP (Workload-Aware Hybrid Partitioning) to manage stationary RDF data for a centralized relational database system.

Chapter 4 discusses advantages and disadvantages of the basic data partitioning techniques and proposes three hybrid partitioning techniques which either uses a data-aware approach or a workload-aware approach. DAHP and DASIVP use a data-aware approach. WAHP uses a workload-aware approach. DAHP combines binary tables and property tables. It gives 40% of an average improvement over binary tables. DASIVP combines structure index partitioning and binary tables. It gives 38% of an average improvement over binary tables. Both the data-aware approaches have performed nearly equivalent. However, the break-even point of DAHP is 95% lesser than DASIVP. Therefore DAHP is carried further and looked at using a workload-aware approach which resulted in WAHP. WAHP combines binary tables and property tables alike DAHP. WAHP creates workload-aware clusters, which are able to answer 73% of query workload by using just 9% of the actual data with 37% of time gain. This chapter proposes a set of metrics which helps in considering the suitability of a data partitioning technique for a RDF dataset and compares all the data partitioning techniques using the derived set of metrics.

The thesis further demonstrates use of WAHP to manage RDF data in a distributed environment and proposes a distribution scheme. This proposed solution DWAHP (Workload-Aware Hybrid Partitioning and Distribution) is then compared with a state-of-the-art system called JARS (Join-Aware distributed RDF Storage). DWAHP is conceptually compared with a scalable RDF storage, WARP and ClusterRDF.

Chapter 5 proposes DWAHP, a workload-aware hybrid partitioning approach for a distributed environment. DWAHP uses two phases: hybrid partitioning and distribution scheme. DWAHP architecture is presented and it is compared with a state-of-the-art system called JARS. DWAHP and JARS are compared in terms of query execution time, query costs, storage space, and message passing parameters. DWAHP create workload-aware clusters which can answer 90% of the most frequent queries without inter-node communication. Generally, it is observed that queries for RDF data mostly involve either star or linear query patterns. DWAHP manages joins such that it is able to answer all linear and star queries without inter-node communication. DWAHP uses *n-hop Property Reachability Matrix* which considers subject-subject and subject-object joins between properties in the dataset, and sets *n* to such a value where at least one property is reachable from all other properties. Value of *n* may vary from dataset to dataset and setting *n* to such a value will result in similar query performances. However there is a trade-off and this trade-off can be dealt by considering data characteristics and hardware requirements. DWAHP has shown on an average 72% of faster query execution time than JARS with 61% of reduced query cost. Considering storage space, DWAHP occupies less than one-third of storage space compared to JARS. For our experiment, JARS eliminates inter-node communication for star queries and minimizes inter-node communication for linear queries and DWAHP is able to get rid of inter-node communication for both linear and star queries.

DWAHP is compared with other state-of-the-art solutions like a scalable RDF data storage system which tries to minimize inter-node communication and uses a parameter called PWOC to find out that it can parallelize queries without inter-node communication. However, it uses data-aware approach, whereas DWAHP uses workload-aware approach and it minimizes inter-node communication by managing joins using *n-hop Property Reachability Matrix*. DWAHP is also able to answer star and linear queries without inter-node communication. Besides this, DWAHP needs less replication compared to this scalable storage system as DWAHP replicates “rdf:type” property on different nodes. DWAHP is also compared with WARP, which is quite similar to the scalable RDF data storage system. However, it follows a workload-aware approach and tries to use lesser replication compared to scalable RDF data storage system. WARP uses horizontal partitioning and on the other hand, DWAHP uses vertical partitioning by using a combination of binary tables and property tables. DWAHP is hence able to minimize inter-node communication among nodes. DWAHP is then compared with ClusterRDF, which partitions data based on workload and uses both horizontal and vertical partitioning. It distributes same schema on different machines and tries to minimize inter-node communication. DWAHP performs only vertical partitioning and distributes different schemas on different machines and hence able to eliminate inter-node communication for the star and linear queries.

Applications in various domains use RDF data as a standard for information exchange. Now-a-days application’s data, its characteristics, data queries, and nature of the queries are generally known well in advance. Workload-aware techniques like WAHP and DWAHP can be utilized for such applications at the data storage level to alleviate query performances. In some cases if data queries or nature of the queries are not known or queries change frequently, data characteristics are generally inevitable and data-aware RDF stores can be utilized for efficient data storage and execution of queries.

## 6.1. Future Work

Although the discussed hybrid data partitioning techniques have shown improvement over other techniques, there is a room for further enhancement.

- Discussed partitioning techniques, whether they are for a centralized system or a distributed system, deals with static RDF data. Therefore they might be suitable only for read intensive applications. However, in a dynamic scenario, these techniques need to be looked at with a completely different perspective:
    - When considering a data-aware approach, there may be change in data and data characteristics. Techniques need to be devised to manage database operations along with change in data and data characteristics of a RDF dataset.
    - When considering a workload-aware approach, not only data and data characteristics are going to change but also the data queries may change over time. All of these changes need to be addressed in order to manage RDF data.
- Discussed techniques try to separate frequent and non-frequent data and store it on the disk. However main memory databases can be utilized to further improve query performance.

# Appendix 1

Below are the SQL queries as implemented for a triple table for the FOAF dataset. Note that for clarity of presentation the properties are shown as row strings instead of actual URI.

**Query 1** Type 4

```
Select t2.subject,t2.object,t3.object from triples t2 , triples t3, triples t4
where t2.subject =t4.object
and t4.predicate= 'text'
and t4.subject= 'text '
and  t2.predicate = 'text '
and  t3.predicate = 'text '
and t2.subject=t3.subject;
```

**Query 2** Type 2

```
select t1.subject,t1.object from triples t1
where t1.subject= ‘text’
and t1.predicate= ‘text’
intersect
select t2.subject,t2.object from triples t2,triples t3
where t2.predicate= ‘text’
and t2.subject = t3.object
and   t3.predicate = ' text '
and t3.subject= ‘text’;
```

**Query 3** Type 4

```
select distinct t1.subject,t2.subject,t2.object from triples t2 ,triples t1
where t2.subject =t1.object
and   t1.predicate = ' text '
and t1.subject= ‘text’
and   t2.predicate = ' text '  ;
```

**Query 4** Type 2

```
select distinct t1.subject,t1.object from triples t1,triples t3
where  t1.predicate = ' text '
and t1.object like ' text '
and t1.subject = t3.object
and   t3.predicate = ' text '
and t3.subject= ‘text’;
```

**Query 5** Type 4

```
select t1.subject,count t1.subject  from triples t1
where   t1.predicate = ' text '
and t1.subject= ‘text’
group by t1.subject;
```

**Query 6** Type 2

```
select t1.subject,count (t1.subject) from triples t1
where   t1.predicate = ' text '
group by t1.subject order by count t1.subject  asc;
```

**Query 7** Type 2

```
select  distinct  t2.subject,t2.object,t3.object  from  triples  t2,triples t3,triples t4
where t2.subject =t4.object
and   t4.predicate = ' text '
and t4.subject= ‘text’
and   t2.predicate = ' text '
and   t3.predicate = ' text '
and t2.object=t3.subject;
```

**Query 8** Type 4

```
select t1.object from triples t1
where   t1.predicate = ' text '
and t1.subject= ‘text’
intersect
select t1.object from triples t1
where   t1.predicate = ' text '
and t1.subject= ‘text’;
```

**Query 9** Type 2

```
select distinct t1.subject,t1.predicate,t1.object from triples t1
where   t1.predicate = ' text '
or  t1.predicate = ' text '
or  t1.predicate = ' text '
.
.
or  t1.predicate = ' text '
and t1.subject= ‘text’;
```

**Query 10** Type 4

```
select t1.object from triples t1
where t1.subject= ‘text’
and   t1.predicate = ' text '
intersect
select t2.object from triples t2,triples t1
where   t2.predicate = ' text '
and t2.subject = t1.object
and   t1.predicate = ' text '
and t1.subject= ‘text’;
```

**Query 11** Type 1

```
select t1.object from triples t1,triples t2,triples t3
where   t1.predicate = ' text '
and   t2.predicate = ' text '
and   t3.predicate = ' text '
and t3.object like ' text '
and t1.object=t3.subject
and t2.subject=t3.subject;
```

**Query 12** Type 2

```
select t1.object,t2.object from triples t1,triples t2,triples t3
where   t1.predicate = ' text '
and   t2.predicate = ' text '
and t1.object=t2.subject
and   t3.predicate = ' text '
and t3.object in
(select t4.object from triples t4,triples t5
where   t4.predicate = ' text '
and   t5.predicate = ' text '
and t4.subject=t5.subject )
intersect
select t6.object from triples t6,triples t7
where   t6.predicate = ' text '
and   t7.predicate = ' text '
and t6.subject=t7.object
```

**Query 13** Type 1

```
select t1.subject from triples t1,triples t2,triples t3
where   t1.predicate = ' text '
```

```
and   t2.predicate = ' text '
and   t3.predicate = ' text '
and t1.object = t2.object
and t2.subject=t3.subject
intersect
select t1.subject from triples t1,triples t2,triples t3
where   t1.predicate = ' text '
and   t2.predicate = ' text '
and   t3.predicate = ' text '
and t2.subject=t3.object
and t1.object = t2.object
```

**Query 14** Type 3

```
select t1.object,t2.object from triples t1,triples t2,triples t3,triples
t4,triples t5
where   t1.predicate = ' text '
and t1.subject= 'text'
and   t2.predicate = ' text '
and   t2.predicate = ' text '
and   t4.predicate = ' text '
and   t5.predicate = ' text '
and t4.subject=t5.object
and t1.object=t2.subject
and t1.object=t3.subject
and t3.object = t4.object
```

**Query 15** Type 1

```
select t1.object,t2.object from triples t1,triples t2
where   t1.predicate = ' text '
and   t2.predicate = ' text '
and t1.subject=t2.object
and t2.object in
select t3.object from  triples t3,triples t4
where   t3.predicate = ' text '
and   t4.predicate = ' text '
and t3.subject=t4.subject
intersect
select t5.object from triples t5,triples t6
where   t5.predicate = ' text '
and   t6.predicate = ' text '
and t5.subject=t6.object
```

## Appendix 2

Below are the SQL queries as implemented for a triple table for the SWetoDBLP dataset. Note that for clarity of presentation the properties are shown as row strings instead of actual URI.

**Query 1** Type 1

```
Select subject from triples t1
where t1.predicate = 'text'
and t1.obj = 'text';
```

**Query 2** Type 1

```
select t1.subject,t1.object from triples t1, triples t2
where t1.predicate= ‘text’
and t2.predicate= ‘text’
and t2.subject = t1.object
```

**Query 3** Type 4

```
select t2.object from  triples t1,  triples t2,  triples t3
where
t1.object = 'text '
and t1.predicate = 'text '
and t2.subject = t1.subject
and t2.predicate = 'text '
and t3.subject = t2.object
and t3.predicate = 'text '
and t3.object like 'text';
```

**Query 4** Type 4

```
select count(t1.subject) from  triples t1,  triples t2,  triples t3
where t1.subject = 'text'
and t1.predicate = ' text '
and t2.subject = t1.object
and t2.predicate = ' text '
and t2.object = ' text '
and t3.subject = t2.subject
and t3.predicate = ' text ';
```

**Query 5** Type 3

```
select t2.subject from  triples t1,  triples t2
where t1.object = 'text '
and t1.predicate = 'text '
and t2.predicate = 'text '
and t2.subject = t1.subject;
```

**Query 6** Type 3

```
select t4.subject from  triples t1,  triples t2,  triples t3,  triples t4
where t1.object = 'text '
and t1.predicate = 'text '
and t2.subject = t1.subject
and t2.predicate = 'text '
and t3.subject = t2.object
and t3.predicate = 'text '
and t3.object = 'text '
and t4.subject = t3.subject
and t4.predicate = 'text ';
```

**Query 7** Type 4

```
select t1.object from  triples t1,  triples t2
where t1.object = 'text '
and t1.predicate = 'text '
and t2.predicate = 'text '
and t2.object = t1.subject
```

**Query 8** Type 3

```
select t1.object from t1subject t1, t1subject t2
where t1.subject like ‘text’
and t1.object like 'text '
and t1.predicate like 'text '
and t1.subject like ‘text’
and t2.predicate like 'text '
and t2.subject = t1.subject
```

**Query 9** Type 2

```
select t1.object from  triples t1,  triples t2,  triples t3,  triples t4
where t1.object = 'text '
and t1.predicate = 'text '
and t2.subject = t1.subject
and t2.predicate = 'text '
and t3.subject = t2.object
and t3.predicate = 'text '
and t3.object = 'text '
and t4.subject = t3.subject
and t4.predicate = 'text '
```

**Query 10** Type 2

```
select t4.object from  triples t1,  triples t2,  triples t3,  triples t4
where t1.object = 'text '
and t1.predicate = 'text '
and t2.predicate = 'text '
and t2.subject = t1.subject
and t3.predicate = 'text '
and t3.object = 'text '
and t3.subject = t2.object
and t4.predicate = 'text '
and t4.subject = t3.subject
```

# Appendix 3

Below are the SQL queries as implemented for a triple table for the Linked Observation dataset. Note that for clarity of presentation the properties are shown as row strings instead of actual URI.

**Query 1** Type 3

```
select avg (t5.object) from  triples t1,  triples t2,  triples t3,  triples t4,
triples t5
where t1.subject = 'text'
and t1.predicate = ' text '
and t2.subject = t1.object
and t2.predicate = ' text '
and t2.object = ' text '
and t3.subject = t2.subject
and t3.predicate = ' text '
and t4.object = ' text '
and t4.predicate = ' text '
and t4.subject = t3.object
and t5.subject = t4.subject
and t5.predicate = ' text ';
```

**Query 2** Type 1

```
select t4.object from  triples t1,  triples t2,  triples t3,  triples t4
where t1.object = 'text '
and t1.predicate = 'text '
and t2.predicate = 'text '
and t2.subject = t1.subject
and t3.predicate = 'text '
and t3.object = 'text '
and t3.subject = t2.object
and t4.predicate = 'text '
and t4.subject = t3.subject;
```

**Query 3** Type 3

```
select max(t6.object) from  triples t1,  triples t2,  triples t3,  triples t4,
triples t5,   triples t6
where t1.object = 'text '
and t1.predicate = 'text '
and t2.subject = t1.subject
and t2.predicate = 'text '
and t3.subject = t2.object
and t3.predicate = 'text '
and t3.object = 'text '
and t4.subject = t3.subject
and t4.predicate = 'text '
and t5.object = 'text '
and t5.predicate = 'text '
and t5.subject = t4.object
and t6.subject = t5.subject
and t6.predicate = 'text ';
```

**Query 4** Type 1

```
select t4.object from  triples t1,  triples t2,  triples t3,  triples t4,  triples
t5
where
```

```
t1.object = 'text '
and t1.predicate = 'text '
and t2.predicate = 'text '
and t2.subject = t1.subject
and t3.predicate = 'text '
and t3.object = 'text '
and t3.subject = t2.object
and t4.predicate = 'text '
and t4.subject = t3.subject
and t5.predicate = 'text '
and t1.subject = t5.object
and t5.subject IN
(select t2.subject from  triples t2,  triples t3,  triples t4,  triples t5,
triples t6
where
t2.predicate = 'text '
and t3.subject = t2.object
and t3.predicate = 'text '
and t3.object = 'text '
and t4.subject = t3.subject
and t4.predicate = 'text '
and t5.object = 'text '
and t5.predicate = 'text '
and t5.subject = t4.object
and t6.subject = t5.subject
and t6.predicate = 'text '
and t6.object text ');
```

**Query 5** Type 1

```
select t2.object from  triples t1,  triples t2,  triples t3
where t1.object = 'text '
and t1.predicate = 'text '
and t2.subject = t1.subject
and t2.predicate = 'text '
and t3.subject = t2.object
and t3.predicate = 'text '
and t3.object like 'text';
```

**Query 6** Type 4

```
select t6.object from  triples t1,  triples t2,  triples t3,  triples t4,  triples
t5,  triples t6
where t1.object = 'text '
and t1.predicate = 'text '
and t2.subject = t1.subject
and t2.predicate = 'text '
and t3.subject = t2.object
and t3.predicate = 'text '
and t3.object = 'text '
and t4.subject = t3.subject
and t4.predicate = 'text '
and t5.object = 'text '
and t5.predicate = 'text '
and t5.subject = t4.object
and t6.subject = t5.subject
and t6.predicate = 'text ';
```

**Query 7** Type 3

```
select t4.subject from  triples t1,  triples t2,  triples t3,  triples t4,
triples t5,  triples t6
where
t1.object = 'text '
and t1.predicate = 'text '
and t2.predicate = 'text '
and t2.subject = t1.subject
and t3.predicate = 'text '
and t3.object = 'text '
and t3.subject = t2.object
and t4.predicate = 'text '
and t4.subject = t3.subject
and t6.object like 'text'
and t6.predicate = 'text '
and t5.object = t6.subject
and t5.predicate = 'text '
and t5.subject = t1.subject)
union
select t4.subject from  triples t1,  triples t2,  triples t3,  triples t4,
triples t5,  triples t6
where
t1.object = 'text '
and t1.predicate = 'text '
and t2.predicate = 'text '
and t2.subject = t1.subject
and t3.predicate = 'text '
and t3.object = 'text '
and t3.subject = t2.object
and t4.predicate = 'text '
and t4.subject = t3.subject
and t6.object like 'text'
and t6.predicate = 'text '
and t5.object = t6.subject
and t5.predicate = 'text '
and t5.subject = t1.subject;
```

**Query 8** Type 4

```
select t4.object from t3object t1, t3object t2, t3object t3, t3object t4,
t3object t5, t3object t6
where
t1.subject like ‘text’
and t1.object like 'text '
and t1.predicate like 'text '
and t1.subject like ‘text’
and t2.predicate like 'text '
and t2.subject = t1.subject
and t3.predicate like 'text '
and t3.subject = t2.object
and t4.predicate like 'text '
and t4.subject = t3.subject
and t6.object like '2004-08-12%'
and t6.predicate like 'text '
and t5.object = t6.subject
```

**Query 9** Type 4

```
select t4.object from t1subject t1, t1subject t2, t1subject t3, t1subject
t4, t1subject t5, t1subject t6
where
t1.subject like ‘text’
and t1.object like 'text '
and t1.predicate like 'text '
and t1.subject like ‘text’
and t2.predicate like 'text '
and t2.subject = t1.subject
and t3.predicate like 'text '
and t3.subject = t2.object
and t4.predicate like 'text '
and t4.subject = t3.subject
and t6.object like '2004-08-12%'
and t6.predicate like 'text '
and t5.object = t6.subject
```

**Query 10** Type 3

```
select max(t6.object) from  triples t1,  triples t2,  triples t3,  triples t4,
triples t5,  triples t6
where
t1.object = 'text '
and t1.predicate = 'text '
and t2.subject = t1.subject
and t2.predicate = 'text '
and t3.subject = t2.object
and t3.predicate = 'text '
and t3.object = 'text '
and t4.subject = t3.subject
and t4.predicate = 'text '
and t5.object = 'text '
and t5.predicate = 'text '
and t5.subject = t4.object
and t6.subject = t5.subject
and t6.predicate = 'text ';
```

**Query 11** Type 2

```
select t6.object from  triples t1,  triples t2,  triples t3,  triples t4,  triples
t5,  triples t6
where
t1.object = 'text '
and t1.predicate = 'text '
and t2.subject = t1.subject
and t2.predicate = 'text '
and t3.subject = t2.object
and t3.predicate = 'text '
and t3.object = 'text '
and t4.subject = t3.subject
and t4.predicate = 'text '
and t5.object = 'text '
and t5.predicate = 'text '
and t5.subject = t4.object
and t6.subject = t5.subject
and t6.predicate = 'text ';
```

**Query 12** Type 4

```
select t6.object from t3subject t1, t3subject  t2, t3subject  t3, t3subject
t4, t3subject  t5, t3subject  t6
where
t1.object like 'text '
and t1.predicate like 'text '
and t2.subject = t1.subject
and t3.predicate like 'text '
and t3.object like 'text '
and t4.subject = t3.subject
and t4.predicate like 'text '
and t5.predicate like 'text '
and t5.subject = t4.object
and t6.subject = t5.subject
and t6.predicate like 'text '
```

**Query 13** Type 1

```
select t4.object from t2subject t1, t2subject t2, t2subject t3, t2subject
t4
where
t1.object like 'text '
and t1.predicate like 'text '
and t1.subject like ‘text’
and t2.predicate like 'text '
and t2.subject = t1.subject
and t3.predicate like 'text '
and t3.object like 'text '
and t3.subject = t2.object
and t4.predicate like 'text '
and t4.subject = t3.subject
```

**Query 14** Type 2

```
select t1.object from  triples t1,  triples t2
where t1.subject = 'text '
and t1.predicate = 'text '
and t2.subject = t1.subject
and t2.predicate = 'text ';
```

**Query 15** Type 2

```
select t2.object, t3.object from  triples t1,  triples t2,  triples t3
where t1.subject = 'text '
and t1.predicate = 'text '
and t2.subject = t1.object
and t2.predicate = 'text '
and t3.subject = t1.object
and t3.predicate = 'text ';
```

**Query 16** Type 2

```
select distinct t2.object from  triples t1,  triples t2
where t1.object = 'text '
and t1.predicate = 'text '
and t2.subject = t1.subject
and t2.predicate = 'text ';
```

**Query 17** Type 1

```
select t1.subject from  triples t1
where t1.predicate = 'text '
and t1.object = 'text ';
```

**Query 18** Type 2

```
select t2.object from  triples t1,  triples t2
where t1.subject = 'text '
```

```
and t1.predicate = 'text '
and t2.subject = t1.subject
and t2.predicate = 'text ';
```

**Query 19** Type 4

```
select t1.subject from  triples t1,  triples t2,  triples t3,  triples t4,
triples t5,  triples t6
where
t1.object = 'text '
and t1.predicate = 'text '
and t2.subject = t1.subject
and t2.predicate = 'text '
and t3.subject = t2.object
and t3.predicate = 'text '
and t3.object = 'text '
and t4.subject = t3.subject
and t4.predicate = 'text '
and t5.object = 'text '
and t5.predicate = 'text '
and t5.subject = t4.object
and t6.subject = t5.subject
and t6.predicate = 'text '
and t6.object text ';
```

**Query 20** Type 2

```
select t1.subject from  triples t1
where t1.object = 'text '
and t1.predicate = 'text ';
```

# Appendix 4

This appendix contains all the algorithms and its complexity calculation. Variable r denotes total records, $p_c$ is property clusters, p is properties, h is number of horizontally partitioned tables, and $sp_f$ is properties occur that together. Complexities are mentioned beside the algorithm name.

**Algorithm: Triple Table O(r)**

```
Input: .n3 file
Output: Triple Table
1  for all r ϵ RDFTriples    ---- r
2     TT ← (s,p,o) ----- 1
3  end for
4  return TT
```

**Algorithm: Property Table -- r + r * $p_c$ ~ O(r * $p_c$)**

```
Input: Triple Table
Output: Property Table
1  for all r ϵ RDFTriples ---- r
2  if(p in r belongs to relationship)
3  create property cluster $p_{c1}$
4  Else if(p in r belongs to personal)
5  create property cluster $p_{c2}$
6  Else
7  Consider as left over triples
8  end for
9  for all r ϵ RDFTriples ---- r
10 for all pi in cluster ci
11 $pc_i$ ← (s, p1,p2..pn)---- $p_c$
12 end for
13 end for
14 return PT
```

**Algorithm: Binary Table --- r * p+r ~ O(r * p)**

```
Input: Triple Table
Output: Binary Tables
1  for all r ϵ RDFTriples ---- r
2  create binary table for every unique property
3  end for
4  for all r ϵ RDFTriples --- r
5  for all $p$ ϵ RDFTriples ---- p
6  $BT_i$ ← (s,o) --- r
7  end for
8  end for
9  return Binary Tables
```

**Algorithm: Horizontally Partitioned Table -- r * h ~ O(r * h)**

```
Input: Triple Table
Output: Horizontally Partitioned Tables
1  for all r ϵ RDFTriples --- r
2  for all n ϵ {0 to 9} ᐯ {a-z} ᐯ {A-Z} --- h
3  HP ← h --- 1
4  n ← (s,p,o) --- 1
5  end for
6  end for
7  return Horizontal Partitioned Table
```

**Algorithm: DAHP – plogp+slogs+ p+r+($p^2$ * $sp_f$) ~ O(r) + O($p^2$ * $sp_c$)**

```
Input:  Null Threshold, Support Threshold, Subject-property bin, Property-use listing, Triple Table
Output: Property Tables, Binary Tables
1  //merge sort
2  Sort descending property-use listing on property-use -- r+plogp
3  For all properties pi in property-use listing --- p
4  If(pi<support Threshold)
5  BT  ← $p_i$ --1
6  Else
7  $C_p$ ← Consider $p_i$ for generating property tables --1
8  End for
9  //Find PT and BT using Apriori Algorithm O($p^2$ * $sp_c$)
10 $sp_c$ ← subject-property bin
11 // Cartesian product
12 for all $p_i$ in subject-property bin
13 $C_{p+1}$ = candidate generated from subject-property bin
14 For all occurrences in subject-property bin
15 Increment the count of all candidates in $c_{p+1}$ that are contained in subject-property bin
16 end for
17 $sp_c$ ← candidates in $c_{p+1}$ with support threshold and null threshold
18 PT ← $sp_c$
19 end for
20 Return Property Tables, Binary Tables – r
```

**Algorithm: DASIVP1 O(r)**

```
Input: RDF Triples – r
Output: Property Basket
1  For ∀ s ϵ RDFTriples – r
2  PB ← p ᐯ o  -- 1
3  End For`
4  Return Property Basket Table
```

**Algorithm: DASIVP2 --- s+r ~r ~ O(r)**

```
Input: RDF Triples, Property Basket
Output: Indexed Partitioned Triples
1   For ∀ p ϵ PropertyBasket --- s
2   Lookup Table ⟵ {Property list, location index} --- 1
3   For ∀  s  ϵ PropertyBasket --- r
4   IPT ⟵ Identified Triple at location index -- 1
5   End For
6   return Indexed Partitioned Triples
```

**Algorithm: DASIVP3  --- p * r ~ O(p * r)**

```
Input: RDF Triples, Property Basket
Output: Indexed Partitioned RDF Triples
1   For ∀ property List ϵ PropertyBasket --- p
2   Lookup Table ⟵ {Property list, location index} -- 1
3   For ∀  s  ϵ PropertyBasket   --- r
4   IPRT ⟵ Identified Triple at location index --- 1
5   End For
6   Return Indexed Partitioned RDF Triples
```

**Algorithm: WAHP – plogp+slogs+ p+r+($p^2$ * $QP_c$) ~ O(r)+ O($p^2$ * $QP_c$)**

```
Input:  Null Threshold, Support Threshold, PF list, QP basket, Triple Table
Output: Property Tables, Binary Tables
1   //merge sort
2   Sort descending PF list on property-frequency -- r+plogp
3   //merge sort
4   Sort descending QP basket on property count -- r+slogs
5   For all properties pi in PF list --- p
6   If(pi<support Threshold)
7   BT  ⟵ $p_i$ --1
8   Else
9   $C_p$ ⟵ Consider $p_i$ for generating property tables --1
10  End for
11  //Find PT and BT using Apriori Algorithm O($p^2$ * $QP_c$)
12  $QP_c$  ⟵ QP basket
13  // Cartesian product
14  for all $p_i$ in QP basket
15  $C_{p+1}$ = candidate generated from QP basket
16  For all occurrences in QP basket
17  Increment the count of all candidates in $c_{p+1}$ that are contained in QP basket
18  end for
19  $QP_c$ ⟵ candidates in $c_{p+1}$ with support threshold and null threshold
20  PT ⟵ $QP_c$
21  end for
22  Return Property Tables, Binary Tables – r
```

**Algorithm: DWAHP 1(Hybrid Partitioning [PT+BT])**
**plogp+slogs+ p+r+($p^2$ * $QP_c$) ~ O(r)+ O($p^2$ * $QP_c$)**

```
Input:  Null Threshold, Support Threshold, PF list, QP basket, Triple Table
Output: Property Tables, Binary Tables
1  //merge sort
2  Sort descending PF list on property-frequency --plogp
3  //merge sort
4  Sort descending QP basket on property count --slogs
5  For all properties pi in PF list --- p
6  If(pi<support Threshold)
7  BT  ← $p_i$ --1
8  Else
9  $C_p$ ← Consider $p_i$ for generating property tables --1
10 End for
11 //Find PT and BT using Apriori Algorithm O($p^2$ * $QP_c$)
12 $QP_c$  ← QP basket
13 // Cartesian product
14 for all $p_i$ in QP basket
15 $C_{p+1}$ = candidate generated from QP basket
16 For all occurrences in QP basket
17 Increment the count of all candidates in $c_{p+1}$ that are contained in QP basket
18 end for
19 $QP_c$ ← candidates in $c_{p+1}$ with support threshold and null threshold
20 PT ← $QP_c$
21 end for
22 Return Property Tables, Binary Tables – r
```

**Algorithm: DWAHP2 (Cluster Creation and Allocation)--- $p^2$ ~ O($p^2$) + O($c^2$)**

```
Input: property table, binary table, n hop matrix, support threshold, Nodes N
Output: Clusters, Dc (Distributed Clusters)
1  Together ← False
2  For $C_i$ consider every combination of $p_i$ in $T_1$ with $p_j$ in $T_2$ -- p
3  For ∀  $p_k$ in n hop matrix -- p
4  If ($p_k$ is Reachable($p_i$,$p_j$)) ∧  supportThreshold($p_i$,$p_j$) – 1
5  Together ← True -- 1
6  End if
7  End for
8  If (Together)   -- 1
9  $C_i$ ← $C_i$ ∪  {$T_1$ ,$T_2$} --- 1
10 End if
11 Together ← False
12 End for
13 For ∀ $N_k$ ∈ N – n
14 //Matrix Multiplication-- O($c^2$)
15 For ∀ $C_i$ ∈ Clusters  -- c
16 For ∀ $C_j$ ∈ Clusters  -- c
17 If all properties is Reachable($C_i$, $C_j$)  --1
18 Together ←True
19 Else
20 Together=False
21 Exit
22 End for
```

```
23       If (Together) --1
24             $N_k \leftarrow N_k \cup C_j \cup C_i$
25       Else
26             $N_k \leftarrow N_k \cup C_j$
27       End if
28    End for
29  End For
30  Return N, Dc
```

**Algorithm: n hop reachability matrix ($p+p^2$) ~ $O(p^2)$**

```
Input: properties
Output: n hop matrix
1   N ←n -- 1
2   For $\forall\ p_i \in$ properties – $O(p^2)$
3      For $\forall\ p_j \in$ properties
4         If(i==j)
5            P(i,j)← -2
6         Else If  (s.s join $\vee$ s.o join)  --1
7            P(i,j)← -1 --1
8         Else
9            P(i,j) ← 0 --1
10        End if
11     End for
12  End for
13  // Dijkstra's Algorithm $O(p^2)$
14  For $\forall\ p_i \in$ properties
15     For $\forall\ p_j \in$ properties
16        If((pi,j) == 0)
17           p(i,j) = $p_k$ //minimum distance from $p_i$ to $p_j$ via $p_k$
18        End if
19     End for
20  End for
21  Return n hop matrix
```

## References

[1] (2014, Feb) RDF 1.1 Primer W3C Working Group Note 25 February 2014. [Online]. https://www.w3.org/TR/2014/NOTE-rdf11-primer-20140225/

[2] Li Ding, Lina Zhou, Tim Finnin, and Anupam Joshi, "How the semantic web is being used: An analysis of foaf documents," in *38th Annual Hawaii International Conference of System Sciences*, 2005, p. 113c.

[3] Christian Bizer et al., "DBpedia- A crystallization point for the Web of Data," *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 7(3), pp. 154-165, September 2009.

[4] FM Suchanek, G Kasneci, and G Weikum, "Yago: a core of semantic knowledge," in *16th international conference on World Wide Web*, 2007, pp. 697-706.

[5] (2013) Bio2RDF: Linked Data for the Life Sciences. [Online]. http://bio2rdf.org/describe/?uri=http://bio2rdf.org/uniprot:P05067

[6] Tamer Özsu, "A survey of RDF data management systems," *Frontiers of Computer Science*, pp. 1-15, May 2016.

[7] Jaroslav Pokorný, "Graph Databases: Their Power and Limitations," in *Computer Information Systems and Industrial Management (CISIM 2015)*, 2015, pp. 58-69.

[8] Dominique Hazael-Massieux. (2003) RDF in Real Life - Some examples of RDF applications. [Online]. https://www.w3.org/2003/Talks/semtour-athens-rdfapp/

[9] P Barnaghi, W Wang, C Henson, and K Taylor, "Semantics for the Internet of Things: early progress and back to the future," *International Journal on Semantic Web and Information Systems (IJSWIS)*, vol. 8(1), pp. 1-21, 2012.

[10] S Bischof et al., "Semantic modelling of smart city data," in *W3C Workshop on the Web of Things*, Berlin, Germany, 2014.

[11] Christian Bizer, Tom Heath, and Tim Berners Lee, "Linked Data - The Story So Far," *Semantic Services, Interoperability and Web Applications: Emerging Concepts*, pp. 205-227, 2009.

[12] A Gyrard, S Datta, C Bonnet, and K Boudaoud, "Standardizing generic cross-domain applications in Internet of Things," in *Globecom Workshops (GC Wkshps)*, December 2014, pp. 589-594.

[13] H Hasemann, A Kroller, and M Pagel, "RDF Provisioning for the Internet of Things.," in *3rd International Conference on the Internet of Things (IOT)*, 2012, pp. 143-150.

[14] (2009) W3C Dataset RDF Dumps. [Online]. https://www.w3.org/wiki/DataSetRDFDumps

[15] LOD Stats. [Online]. http://stats.lod2.eu

[16] Broekstra Kampman and Harmelen Van, "Sesame: A Generic Architecture for Storing and Querying RDF and RDF Schema," in *1st International Semantic Web Conference*, 2002, pp. 54–68.

[17] Wilkinson and Kevin, "Jena property table implementation," in *Second International*

*Workshop on Scalable Semantic Web Knowledge Base Systems*, 2006, pp. 35-46.

[18] Daniel Abadi, Marcus Adam, Samuel Madden, and Kate Hollenbach, "SW-Store: a vertically partitioned DBMS for Semantic Web data management," *The VLDB Journal—The International Journal on Very Large Data Bases*, pp. 385-406, 2009.

[19] A Levy, A Mendelzon, and Y Sagiv, "Answering queries using views: A Survey," *The VLDB Journal*, vol. 10 (4), pp. 270-294, 2001.

[20] Anjali Rajith, Shoji Nishimura, and Haruo Yokota, "JARS: Join-Aware Distributed RDF Storage," in *20th International Database Engineering & Applications Symposium(IDEAS '16)*, ACM, New York, NY, USA, 2016, pp. 264-271.

[21] (1998, September) Relational Databases on the Semantic Web. [Online]. https://www.w3.org/DesignIssues/RDB-RDF.html

[22] Ramanujam Sunitha, Anubha Gupta, Latifur Khan, Steven Seida, and Bhavani Thuraisingham, "R2D: A bridge between the semantic web and relational visualization tools," in *IEEE International Conference on Semantic Computing*, 2009, pp. 303-311.

[23] Sherif Sakr and Al-Naymat Ghazi, "Relational processing of RDF queries: a survey," in *ACM SIGMOD*, 2010, pp. 23-28.

[24] Chong E, Das S, Eadon G, and Srinivasan J, "An efficient SQL-based RDF Querying Scheme," in *31st International Conference on Very Large Data Bases*, 2005, pp. 1216–1227.

[25] Harris Stephen and Nicholas Gibbin, "3store: Efficient bulk RDF storage," , 2003, pp. 1-15.

[26] Y Luo, F Picalausa, G Fletcher, J Hidders, and S Vansummeren, "Storing and indexing massive RDF datasets," in *Semantic Search over the Web, Data centric Systems and Applications.*: Springer Berlin Heidelberg, 2012, pp. 31-60.

[27] ORRI ERLING and IVAN MIKHAILOV, "RDF Support in the Virtuoso DBMS," in *Networked Knowledge-Networked Media.*: Networked Knowledge-Networked Media, 2009, pp. 7-24.

[28] Cathrin Weiss, Panagiotis Karras, and Bernstein Abraham, "Hexastore: sextuple indexing for semantic web data management," in *VLDB*, 2008, pp. 1008-1019.

[29] Thomas Neumann and Weikum Gerhard, "RDF-3X: a RISC-style engine for RDF," in *VLDB*, 2008, pp. 647-659.

[30] P Yuan, P Liu, H Jin, W Zhang, and L Liu, "TripleBit: a fast and compact system for large scale RDF data," in *VLDB*, vol. 6(7), 2013, pp. 517–528.

[31] M Bornea et al., "Building an efficient RDF store over a relational database," in *2013 ACM SIGMOD International Conference on Management of Data*, Jun 22 2013, pp. 121-132.

[32] Steve Harris, Nick Lamb, and N Shadbolt, "4store: The design and implementation of a clustered RDF store," in *5th International Workshop on Scalable Semantic Web Knowledge Base Systems (SSWS2009)*, 2009, pp. 94-109.

[33] Daniel Abadi, Marcus Adam, Samuel Madden, and Kate Hollenbach, "Scalable semantic

web data management using vertical partitioning," in *VLDB Endowment*, 2007, pp. 411-422.

[34] M Stonebraker et al., "C-store: a column-oriented DBMS," in *31st international conference on Very large data bases , VLDB Endowment.*, 2005, pp. 553-564.

[35] V Raman et al., "DB2 with BLU acceleration: so much more than just a column store," in *VLDB Endowment*, vol. 6(11), 2013, pp. 1080-1091.

[36] Daniel Abadi, Samuel Madden, and Nabil Hachem, "Column-Stores vs. Row-Stores: How different are they really?," in *ACM SIGMOD*, 2008, pp. 967-980.

[37] T Tran and G Ladwig, "Structure index for RDF data," in *Workshop on Semantic Data Management, SemData@ VLDB*, vol. 2(10), 2010.

[38] T Tran, G Lagwig, and S Rudolph, "Managing structured and semistructured RDF data using structure indexes," in *Transaction on Knowledge and Data Engineering*, IEEE, vol. 25(9), 2013, pp. 2076-2089.

[39] M Yang, B Zhang, and Y Li, "RDF Data Query and Management Method Based on HBase and Structure Index in Railway Sensor Application," in *International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT)*, December 2013, pp. 36-43.

[40] J Levandoski and F Mokbel, "RDF data-centric storage," in *IEEE International Conference on Web Services 2009 (ICWS 2009)*, July 2009, pp. 911-918.

[41] A Schwarte, P Haase, K Hose, R Schenkel, and M Schmidt, "Fedx: Optimization techniques for federated query processing on linked data," in *International Semantic Web Conference (ISWC 2011)*, Oct 23 2011, pp. 601-616.

[42] B QuilitZ. (2007) DARQ- Federated Quereis with SPARQL. [Online]. http://darq.sourceforge.net/

[43] H Kobashi, N Carvalho, B Hu, and T Saeki, "Cerise: an RDF store with adaptive data reallocation," in *the 13th Workshop on Adaptive and Reflective Middleware ACM 2014*, 2014.

[44] M Hammoud, D Rabbou, R Nouri, S Beheshti, and S Sakr, "DREAM: distributed RDF engine with adaptive query planner and minimal communication," in *VLDB Endowment 2015*, vol. 8(6), 2015, pp. 654-665.

[45] J Huang, D Abadi, and Kun Ren, "Scalable SPARQL querying of large RDF graphs," in *VLDB Endowment*, vol. 4.11, 2011, pp. 1123-1134.

[46] R Al-Harbi, Y Ebrahim, and P Kalnis. (2014) PhD-Store: An adaptive SPARQL engine with dynamic partitioning for distributed RDF repositories. CoRR,abs/1405.4979.

[47] S Gurajada, S Seufert, I Miliaraki, and M Theobald, "TriAD: A Distributed Shared-Nothing RDF Engine based on Asynchronous Message Passing," in *2014 ACM SIGMOD international conference on Management of data*, 2014, pp. 289-300.

[48] (2013) METIS - Serial Graph Partitioning and Fill-reducing Matrix Ordering. [Online]. http://glaros.dtc.umn.edu/gkhome/metis/metis/overview

[49] L Galarraga, K Hose, and R Schenkel, "Partout: A Distributed Engine for Efficient RDF

Processing," in *23rd International Conference on World Wide Web*, 2014, pp. 267-268.

[50] X Wang, T Yang, J Chen, L He, and X Du, "RDF partitioning for scalable SPARQL query processing," *Frontiers of Computer Science*, vol. 9(6), pp. 919-933, 2015.

[51] M Boissier, "Optimizing main memory utilization of columnar in-memory databases using data eviction," in *VLDB PhD Workshop* , 2014, pp. 1-6.

[52] K Hose and R Schenkel, "WARP: Workload-Aware Replication and Partitioning for RDF," in *29th IEEE International Conference on Data Engineering Workshops*, 2013, pp. 1-6.

[53] P Peng, L Zou, L Chen, and D Zhao, "Query Workload-based RDF Graph Fragmentation and Allocation," in *EBDT 2016*, 2016, pp. 377-388.

[54] J Levandoski, P Larson, and R Stoica, "Identifying Hot and Cold Data in Main-Memory Databases," in *IEEE 29th international conference on Data Engineering*, 2013.

[55] C Curino, E Jones, Y Zhang, and S Madden, "Schism: a workload-driven approach to database replication and partitioning," in *VLDB Endowment*, vol. 3(1-2), 2010, pp. 48-57.

[56] Rebeca Schroeder, Raqueline Penteado, and Carmem Hara, "Partitioning RDF exploiting workload information," in *22nd International Conference on World Wide Web(WWW '13 Companion)*, ACM, New York, NY, USA, 2013, pp. 213-214.

[57] R Schroeder and C Hara, "Partitioning Templates for RDF," in *Advances in Databases and Information Systems (ADBIS 2015)*, September 2015, pp. 305-319.

[58] Yan Ying et al., "Efficiently querying rdf data in triple stores," in *World Wide Web*, 2008, pp. 1053-1054.

[59] MahmoudiNasab Hooran and Sherif Sakr, "An experimental evaluation of relational RDF storage and querying techniques," in *Database Systems for Advanced Applications*, 2010, pp. 215-226.

[60] MahmoudiNasab Hooran and Sherif Sakr, "Efficient and Adaptable Query Workload-Aware Management for RDF Data," in *WISE'10 of 11th International Conference, on web information system engineering*, 2010, pp. 390-399.

[61] A Kemper and T Neumann, "HyPer: A hybrid OLTP&OLAP main memory database system based on virtual memory snapshots," in *IEEE 27th International Conference on Data Engineering (ICDE 2011)*, 2011, pp. 195-206.

[62] B Motik, Y Nenov, R Piro, I Horrocks, and D Olteanu, "Parallel Materialisation of Datalog Programs in Centralised, Main-Memory RDF Systems," in *AAAI*, 2014, pp. 129-137.

[63] H Garcia-Molina and K Salem, "Main memory database systems: An overview," *IEEE Transactions on knowledge and data engineering*, vol. 4(6), pp. 509-516, 1992.

[64] Herodotou Herodotos, Nedyalko Borisov, and Shivnath Babu, "Query optimization techniques for partitioned tables," in *ACM SIGMOD*, 2011, pp. 49-60.

[65] Jingren Zhou, Nicolas Bruno, and Wei Lin, "Advanced partitioning techniques for massively distributed computation," in *2012 ACM SIGMOD International Conference*

*on Management of Data*, Scottsdale, Arizona, USA, May 20-24, 2012.

[66] C Aggarwal, N Ashish, and A Sheth, "The Internet of Things: A Survey from the Data-Centric Perspective," *Managing and Mining Sensor Data*, pp. 383-428, 2013.

[67] Irene Polikoff. (2014, May) RDF is Critical to a Successful Internet of Things. [Online]. http://www.dataversity.net/rdf-critical-successful-internet-things/

[68] RDF Working Group. (2014, Feb) Resource Description Framework (RDF). [Online]. https://www.w3.org/RDF/

[69] H Hasemann, A Kröller, and M Pagel, "The wiselib tuplestore: a modular RDF database for the internet of things," in *arXiv preprint arXiv:1402.7228*, 2014.

[70] D Pfisterer et al., "SPITFIRE: toward a semantic web of things," *IEEE Communications Magazine*, vol. 49(11), pp. 40-48, 2011.

[71] X Su et al., "Connecting iot sensors to knowledge-based systems by transforming senml to rdf," in *Procedia Computer Science*, vol. 32, Dec 31 2014, pp. 215-222.

[72] S Abbassi and R Faiz, "RDF-4X: a scalable solution for RDF quads store in the cloud," in *International Conference on Management of Digital EcoSystems*, 2016, pp. 231-236.

[73] P Britton, A Kumar, D Bigwood, A DeFusco, and H Greenblatt, "Methods and apparatus for querying a relational data store using schema-less queries," US9529937 B2, Dec 27, 2016.

[74] J Um et al., "Distributed RDF store for efficient searching billions of triples based on Hadoop," *The Journal of Supercomputing*, vol. 72(5), pp. 1825-1840, 2016.

[75] Y Nenov et al., "RDFox: A highly-scalable RDF store," in *International Semantic Web Conference*, 2015, pp. 3-20.

[76] H Bazoobandi et al., "A compact in-memory dictionary for RDF data," in *European Semantic Web Conference*, 2015, pp. 205-220.

[77] M Meimaris and G Papastefanatos, "Double Chain-Star: an RDF indexing scheme for fast processing of SPARQL joins," in *EDBT*, 2016, pp. 668-669.

[78] K Bok, J Lim, K Kim, and J Yoo, "A RDF Indexing Scheme for Large Scale Semantic Web," *International Information Institute (Tokyo) Information*, vol. 19(3), pp. 1011-1019, 2016.

[79] A Cerdeira-Pena, A Farina, J Fernández, and M Martínez-Prieto, "Self-indexing rdf archives," in *Data Compression Conference (DCC 2016)*, 2016, pp. 526-535.

[80] A Katib, V Slavov, and P Rao, "Fast processing of SPARQL queries on RDF quadruples," *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 37, pp. 90-111, 2016.

[81] D Calvanese et al., "Ontop: answering SPARQL queries over relational databases," *Journal of Semaitc Web*, pp. 1-17, 2016.

[82] A Schätzle, M Przyjaciel-Zablocki, S Skilevic, and G Lausen, "S2RDF: RDF querying with SPARQL on spark," in *VLDB Endowment*, vol. 9(10), 2016, pp. 804-815.

[83] F Goasdoué, Z Kaoudi, L Manolescu, J Quiané-Ruiz, and S Zampetakis, "Cliquesquare: Flat plans for massively parallel RDF queries," in *IEEE 31st International Conference*

*on Data engineering (ICDE 2015)*, 2015, pp. 771-782.

[84] (2017, July) Materialized view. [Online]. https://en.wikipedia.org/wiki/Materialized_view

[85] Tim Finin. (2005, Jan) UMBC ebiquity FOAF Dataset. [Online]. http://ebiquity.umbc.edu/blogger/2005/01/25/foaf-dataset-available/

[86] Sidirourgos Lefteris, Romulo Goncalves, Martin Kersten, Niels Nes, and Stefan Manegold, "Column-store support for RDF data management: not all swans are white," in *VLDB*, 2008, pp. 1553-1563.

[87] Apache Jena. (2011) Jena RDF/XML How-To. [Online]. https://jena.apache.org/documentation/io/rdfxml_howto.html

[88] Michael Stonebraker, "Implementation of integrity constraints and views by query modification," in *SIGMOD*, 1975, pp. 65-78.

[89] R Agrawal and R Srikant, "Fast Algorithm for Mining Association Rules," in *VLDB*, vol. 1215, 1994, pp. 487-499.

[90] (2009, Oct) SwetoDblp. [Online]. https://datahub.io/dataset/sweto-dblp

[91] (2011) The DBLP Computer Science Bibliography. [Online]. https://old.datahub.io/dataset/dblp

[92] DCMI. Dublin Core Metadata Initiative. [Online]. http://dublincore.org

[93] (2014, Feb) RDF 1.1 Primer W3C Working Group Note. [Online]. https://www.w3.org/TR/2014/NOTE-rdf11-primer-20140225/

[94] H Patni, C Henson, and A Sheth, "Linked sensor data," in *2010 International Symposium on InCollaborative Technologies and Systems (CTS 2010)*, 2010, pp. 362-370.

[95] P Barnaghi and M Presser, "Publishing linked sensor data," in *3rd International Conference on Semantic Sensor Networks*, vol. 668, November 2010, pp. 1-16.

[96] LinkedSensorData. [Online]. http://wiki.knoesis.org/index.php/LinkedSensorData#Linked_Observation_Data

[97] (2016, Nov) Postgres-XL 9.5 R1.4. [Online]. http://www.postgres-xl.org

## Publications

a. Trupti Padiya and Minal Bhise, DWAHP: Distributed Workload-Aware Hybrid Partitioning for RDF Data, in *ACM International Database Engineering & Applications Symposium IDEAS '17* , 2017, pp. 235-241.
b. Anubha Jain, Trupti Padiya, and Minal Bhise, Log-based Method for Faster IoT Queries, in *IEEE Region 10 Symposium TENSYMP'17* , 2017, pp. 1-4.
c. Trupti Padiya, Jai Jai Kanwar, and Minal Bhise, Workload-Aware Hybrid Partitioning, in *ACM Compute 2016*, 2016, pp. 51-58.
d. Trupti Padiya and Minal Bhise, Hot and Cold Data Classification for Main Memory Databases, in *Ph.D. Forum collocated with IEEE International Parallel and Distributed Processing Symposium IPDPS 2015*, 2015.
e. Trupti Padiya, Minal Bhise, and Prashant Rajkotiya, Data Management for Internet of Things, in *IEEE Region 10 Symposium TENSYMP IEEE*, 2015, pp. 62-65
f. Bhavik Shah, Trupti Padiya, and Bhise Minal, Query Execution for RDF Data Using Structured Indexed Vertical Partitioning, in *IEEE International Parallel and Distributed Processing Symposium Workshop IPDPSW* , 2015, pp. 575-584.
g. Trupti Padiya, Minal Bhise, Sandeep Vasani, and Mohit Pandey, Query Execution for RDF Data on Row and Column Store, in *International Conference on Distributed Computing and Internet Technology*, 2015, pp. 403-408.
h. Trupti Padiya and Minal Bhise, Query Execution for Partitioned RDF Data, in *ACM International Conference for Women in Engineering*, 2013, pp. 1-8.
i. Sandeep Vasani, Mohit Pandey, Minal Bhise, and trupti Padiya, Faster Query Execution for Partitioned RDF Data, in *International Conference on Distributed Computing and Internet Technology*, 2013, pp. 547-560.
j. Trupti Padiya, Minal Bhise, and Sanjay Chaudhary, Semantic Web Data Partitioning, in *Advancing Information Management through Semantic Web Concepts and Ontologies*.: ISBN 978-1-4666-2494-8, 2013, ch. 8, pp. 154-165.
k. Trupti Padiya, Mohit Ahir, Minal Bhise, and Sanjay Chaudhary, Data Partitioning for Semantic Web, *International Journal of Computer and Communication Technology*, ISSN (Online): 2231 – 0371, ISSN (Print): 0975 – 7449, Vol 3, Issue 2, 2012, pp. 32-35