DISSERTATIONES
INFORMATICAE
UNIVERSITATIS
TARTUENSIS
40

MOHAMED RAGAB

Bench-Ranking: A Prescriptive Analysis Approach for Large Knowledge Graphs Query Workloads

DISSERTATIONES INFORMATICAE UNIVERSITATIS TARTUENSIS
40

DISSERTATIONES INFORMATICAE UNIVERSITATIS TARTUENSIS
40

MOHAMED RAGAB

Bench-Ranking: A Prescriptive Analysis Approach for Large Knowledge Graphs Query Workloads

UNIVERSITY OF TARTU
Press
1632

Institute of Computer Science, Faculty of Science and Technology, University of Tartu, Estonia.

Dissertation has been accepted for the commencement of the degree of Doctor of Philosophy (PhD) in Computer Science on December 19, 2022 by the Council of the Institute of Computer Science, University of Tartu.

Supervisor(s)

Ass. Prof.	Riccardo Tommasini LIRIS Lab, INSA de Lyon, France
Prof.	Ahmed Awad Institute of Computer Science, Tartu University, Tartu, Estonia

Opponents

Prof. Dr.	Ladjel Bellatreche Laboratory LIAS at ISAE-ENSMA
Ass. Prof.	Ester Zumpano University of Calabria

The public defense will take place on 13 Jan., 2023 at 14:15 in Narva mnt 18-2048.

The publication of this dissertation was financed by the Institute of Computer Science, University of Tartu.

ISSN 2613-5906 (print) ISSN 2806-2345 (PDF)
ISBN 978-9916-27-114-8 (print) ISBN 978-9916-27-115-5 (PDF)

University of Tartu Press
`http://www.tyk.ee/`

To my prophet ("Muhammad" Peace be upon him), my parents "Ragab and Soaad", my beloved kid "Anas", my sincere wife "Shaymaa", and also to my sisters "Hala", "Ghada", and "Shaymaa"...

ABSTRACT

We are living in a world in which data are not just getting bigger, it is also getting more and more interconnected. Graphs are the most intuitive, natural, and flexible data abstraction that handles this interconnectedness in several application domains of our daily lives, e.g., social media, computational biology and protein networks, telecommunications, and many more.

The unprecedented volumes of graph data for these applications demand scalable systems for efficiently processing large amounts of graphs. There has been a significant prevalence of graph processing work in academia and industry. This leads to a surge in the number of systems for storing, managing, and processing graphs. Building a native scalable engine for *querying* graphs is still an open problem. Thus, the current solution for large graph query analytics is to fall back on reusing existing Big Data (BD) systems harnessing the capabilities of their mature distributed relational interfaces.

Despite its flexibility, the relational model requires several additional *design decisions* when used for representing graphs, which cannot be decided automatically, e.g., the choice of the *schema*, the *partitioning technique*, and the *storage formats*. Additionally, changing such design decisions in the future is not an easy task, but it can be even very expensive and time-consuming especially with large volumes of data. For example, changing the logical relational layout (relational schema), or physical data storage (partitioning and storage formats) would require huge data engineering efforts if selected wrongly from the beginning, and discovered performing inefficiently. Moreover, those design decisions may affect each other as they have inherent *trade-offs*. Thus, the BD system performance is exposed to situational performance results with those dimensions. That is, it is hard to identify a clear winner out of the experimental solutions space that guarantees the best performance of those systems. Thus, it is hard to guarantee a fair assessment of relational BD systems performance while querying large graphs.

The state-of-the-art research works centered around performance analyses of querying large graphs are merely *descriptive*, i.e., they answer the question *what happened?* (i.e., comparing performance results of using various experimental configurations) or at most *diagnostic*, i.e., *why did it happen?* (i.e., rationalizing for instance why a BD system X performed better with an experimental configuration $C1$ better than with another configuration $C2$) We argue that such kinds of performance analyses are often overwhelming, and the amount of work left for taking actionable decisions is huge. Conversely, the Prescriptive Performance Analysis (PPA) reduces the need for human intervention even further by making the insights actionable, answering the question *what should we do?*. For instance, guiding the BD practitioners to select the experimental configurations that guarantee an optimal performance of BD systems when querying large graphs.

This thesis investigates the problem of enabling *prescriptive* performance analytics of BD systems that query large (RDF) graphs. The BD PPA aims to guide

the practitioner directly to actionable decisions, navigating complex experimental solution spaces (of multiple experimental dimensions, e.g., schema, partitioning, and storage) without ignoring the underlying dimensions' trade-offs.

In the thesis, we aim to initially make sense of BD systems' performance while querying large (RDF) graphs. We show the limitations of Descriptive and Diagnostic Performance Analysis (DDPA) and showcase how they are inconvenient to attain clear decisions. This is shown even further when ensuring the *replicability* of BD systems that query large (RDF) graphs. For example, the performance of Spark-SQL system with RDF relational schema advancements does not generalize, introducing other new experimental dimensions, e.g., storage and partitioning.

Thus, this calls for designing a prescriptive framework for analyzing the BD performance for querying large (RDF) graphs. The thesis investigates how to enable *prescriptive* analytics via ranking criteria. We designed a PPA framework (called *"Bench-Ranking"*) that employs several *Single-Dimensional* (SD) and *Multi-Dimensional* (MD) ranking criteria for ranking the system's performance with several experimental dimensions. Bench-Ranking provides an accurate yet simple way that supports the practitioners in their evaluation tasks even in the existence of dimensions *trade-offs*. Finally, the thesis provides evaluation metrics for assessing the efficiency of the proposed ranking criteria.

The last but not least contribution of this thesis is wrapping the Bench-Ranking PPA methodology into a reusable and extensible *Python* library (called *PAPyA*). PAPyA aims to hide the complexity of Bench-ranking functionality and metrics and reduces the time and effort in reaching informed decisions in the realm of BD systems scenarios with several design decisions (e.g., querying large graphs).

The thesis concludes the work by giving insights and future directions related to the area of querying and processing large graphs on top of Big Data relational frameworks.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

KGs	Knowledge Graphs
BD	Big Data
RDF	Resource Description Framework
SPARQL	Simple Protocol and RDF Query Language
SQL	Standard Query Language
SW	Semantic Web
ST	Single Statement Table schema
VT	Vertically-Partitioned Table(s) schema
PT	Property Tables schema
WPT	Wide Property Table schema
ExtVP	Extended Vertically-Partitioned Table(s) schema
PBP	Predicate-Based Partitioning
SBP	Subject-Based Partitioning
SD	Single-Dimensional
Rs	Ranking by schema dimension
Rf	Ranking by storage format
Rp	Ranking by Partitioning dimension
MD	Multi-Dimensional
MO	Multi-Objective Optimization
ETL	Extract-Transform-Load
OLAP	Online Analytical Processing
OLTP	Online Transactional Processing
MPPs	Massive Parallel Processing systems
HDFS	Hadoop Distributed File System
BI	Business Intelligence
AI	Artificial Intelligence
ML	Machine Learning
KPIs	Key Performance Indicators

1. INTRODUCTION

Nowadays, we are witnessing an unprecedented growth of interconnected data, which highlights the vital role of graph analytics in our daily lives [Sak+21; Fan+20; Bat+15]. Indeed, graph analytics handle many emerging large data management scenarios in several application domains such as social media, astronomy, computational biology, telecommunications, protein networks, and many more [Bat+15; Sah+20]. Thus, large graph analytics became one of the most effective Big Data (BD) tasks for getting insights from huge volumes of interconnected data [Jun+17]. One recent example on the importance of big graph analytics is the timely *Graphs4Covid-19* [1] initiative that aims at alleviating the global COVID-19 pandemic. This initiative utilized the power of Knowledge Graphs (KGs) which are efficient means that can integrate vast amounts of various related datasets (e.g., *PubMed*, *DrugBank*, *UniProt*, and many others) which include COVID-relevant data about symptoms, protein structures, previous drug treatments, cases, and user demographics. Integrating those relevant datasets into one single source (i.e., the COVID KG), helped greatly to understand the problem better and get valuable insights seeking potential treatments.

The work on graph processing has become increasingly common in both academia and industry [Sah+20; RWE15; Jun+17; Bat+15]. As a result, the number of systems for storing, querying, and analyzing graphs to significantly increase [Sah+20]. For instance, graph database management systems are among the fastest-growing technologies in todays data management industry [RWE15]. These systems are mostly *native*, in the sense that they process a graph stored using dedicated graph data structures that optimize graph query operations such as graph pattern matching and graph traversals. However, native graph database management systems do not show good scalability for large query workloads [RWE15; Jun+17]. Indeed, most graph databases are centralized and require representing the graphs in main memory to maintain nodes and references to their adjacent nodes. On the other side, distributed graph computing engines (e.g., *Apache Giraph*, and *Spark GraphX*, and others) can scale, but they only are dedicated for iterative graph analytics via implementing graph algorithms (e.g., *PageRank*, *Triangle Counting*, *Connected Components*) [Bat+15; Jun+17]. However, these systems are not dedicated to large declarative graph query processing. Furthermore, creating a native, scalable graph query engine is still an open challenge [Sak+21]. As a result, the current approach to handling massive graph query analytics is to rely on reusing already-existing BD systems and utilizing their relational interfaces [Pha+15; Sch+16; Sch+14; MAA18].

Despite its flexibility and scalability, using the relational model to query big graphs necessitates making additional design decisions, such as selecting the *re-*

[1] `https://neo4j.com/graphs4good/covid-19/`

lational schema, the *partitioning technique*, and the storage formats. Unlike the *auto-tuneable* system configurations (known as *knobs* in autonomous database systems) [Van+21; Van+17], it is hard to determine those design decisions automatically. Furthermore, those choices may have an impact on one another due to intrinsic trade-offs [Rag+21].

1.1. Problem Statement

The design of a full pipeline for solving BD problems is already a *time-consuming* task. Indeed, BD frameworks expose hundreds of system parameters for tuning. The problem aggravates even more when the processing tasks become more complex, like in the case of querying large graphs [Sak+21], because of the additional design decisions (e.g., relational schemas, partitioning techniques, and storage formats, etc.) that are hard to be automatically decided [Van+21] just given the graph dataset and workloads. Moreover, changing those design decisions in the future (if selected wrongly from the first design) is extremely costly, as it would require huge data engineering efforts. Additionally, due to the inherent performance *trade-offs* among those design decisions, the performance results are often *situational* [Rag+21]. That is, there is rarely an absolute winner of the several available experimental options. For example, evaluating a query workload with a BD system with a specific underlying relational schema for representing a graph can be efficient with a specific partitioning technique and a specific storage format. Whereas, changing the query workload, the partitioning technique, or even the underlying physical storage formats can dramatically degrade the performance [Rag+21]. As a result, it is challenging to provide *fair* evaluation or benchmarking of relational BD systems' performance when querying massive graphs [RAT21].

Considering existing works that use BD frameworks for querying large Resource Description Framework (RDF) graphs, we can notice the gap in the way they interpret the performance results. Figure 1 shows the *four* levels of analyses that highlight the amount of human intervention required to make a decision and take actions (inspired by *Gartners* analysis taxonomy [Hag17]). The message is intuitive: the simpler the analysis is, the more work is left to do to transform analysis into actionable insights.

In the domain of performance analysis for BD systems, we found that previous works are merely *descriptive*that is, they answer the question "*what happened?*"or, at most diagnostic, i.e., "*why did it happen?*" [Lep+20]. For instance, it can describe that a BD system has better performance (e.g., in terms of query latency) with a specific relational schema (for representing a graph) than with another one. The diagnostic level of analysis can rationalize the reason behind: the query workload has less joins with the first schema than with the second schema. Such analyses of benchmarking and diagnosis also are required to apply to the other experimental dimensions (e.g., partitioning, storage, or others). We argue

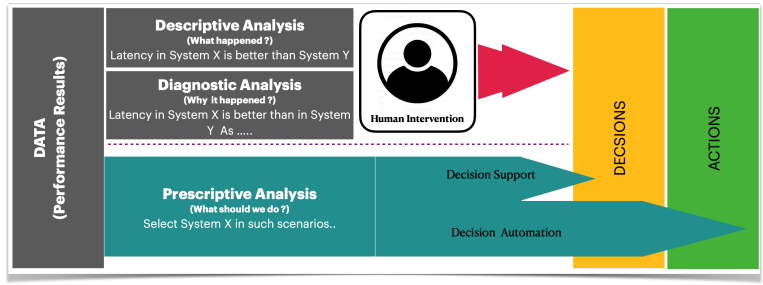


Figure 1: Performance analysis methodology.

that such kinds of analyses are often overwhelming, and the amount of work left for taking actionable decisions (i.e., what experimental options to select) is huge. Moreover, in the presence of the inherent trade-offs among the experimental dimensions, the performance results can disprove when changing experimental design decisions (schema, partitioning, and storage formats) [Rag20]. Figure 2 [2] shows a simplified version of an example of results contradictions (found in [Rag20]) that occur with changing the experimental parameters, for the performance *KPIs*, e.g., query *latency*. The performance of the (BD) system with the *same* schema (i.e., Property Tables schema) with queries (Query1 and Query2) disproves changing the partitioning technique (from *Subject-based* to *Predicated-based*) (details and more examples in Chapter 3). The *prescriptive* performance analysis, on the other hand, decreases the need for human intervention even further by turning the insights into actionable recommendations by relying on *statistical* and *mathematical* models.

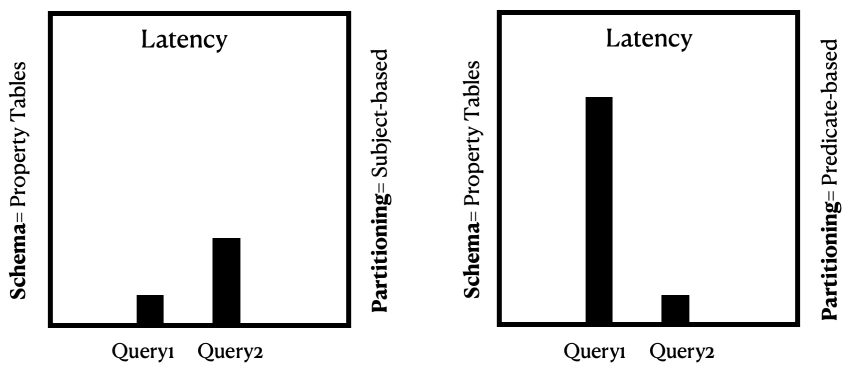


Figure 2: Results disprove (i.e., contradict) while changing the experimental configurations; latency of *Query1* with the *Property Tables* schema is less than *Query2* with the *Subject-based* partitioning, but this turns to dramatically disprove by changing to the *Predicate-based* technique.

Consequently, this thesis explores the issue of providing *prescriptive* [3] ana-

[2] Details about the various RDF relational schemas and partitioning techniques can be found in Chapter 2

[3] Performance predictive analysis is an *orthogonal* level of analysis that aims to forecast potential

Ch.	Summary	References
C1	Lack of systematic performance analysis.	[Abd+17][Sch+16][Sch+14]
C2	Neglecting performance trade-offs.	[ANS18; IP19; al19]
C3	Lack of replicability.	[Rag+21]
C4	Descriptive or at most diagnostics performance analysis.	[Sch+16; Sch+14; Cud+13]

Table 1: Summary of the challenges in state-of-the-art.

lytics in the context of BD systems that query large KGs. This type of analysis implies recommendations while navigating a complex solution space of experimental design selections without neglecting their potential trade-offs in order to take the practitioner directly to decisions that can be put into practice. The following section aims to discuss in more details why abstracting out from the descriptive and diagnostic analyses, and harnessing prescriptive analysis is important for our scenario, reflecting on the literature of large Knowledge graph processing.

1.2. State-of-the-Art

This section presents the state-of-the-art large (RDF) graph processing, highlighting the challenges related to our problem statement (see Table 1 for a summary). A more detailed background can be found in chapter 2.

The literature focuses on optimizing the performance of the BD systems (e.g., Apache Spark, Hive, Impala) when querying knowledge graphs [Abd+17; Sch+16; Sch+14]. However, different experimental dimensions that affect their performance are not systematically studied (*C1*) nor compared (*C2*), as each work focuses only on *one* dimension at a time, e.g., schema [al19], partitioning [ANS18], or storage [IP19].

Focusing *solely* on one experimental design decision might easily neglect the presence of dimensions' trade-offs (*C2*). Thus, the proposed optimizations cannot be replicated when introducing new experimental dimensions (*C3*). For instance, changing the data partitioning techniques or storage encoding disproves the performance of schema advancements [Rag+21].

Finally, prior works stop their analyses at describing or at most diagnosing the performance results (*C4*). For instance, *Shätzle et. al.* show examples of descriptive analyses in their works [Sch+14; Sch+15]. In particular, they presented examples of relational RDF processing systems that come up with proposing novel RDF relational schemas, e.g., the *Wide Property Table* (WPT) schema [Sch+14], and the *Extended Vertically-Partitioned Tables* (ExtVP) schema [Sch+16]. Authors provide descriptive and diagnostic performance analytics of those RDF relational systems' improvements (i.e., mainly due to their novel schema designs).

performance results. (it is out of the scope of this thesis).

Similarly, *Cudré-Mauroux* et.al., and *Abdelaziz* et.al., show other examples of descriptive and diagnostic analyses comparing RDF processing systems in their works [Cud+13; Abd+17]. In particular, [Abd+17] conducted a comprehensive survey of several RDF processing systems, describing their performance (using execution runtime, scalability, etc). This study is followed by diagnosing why one RDF system outperforms another. Similarly, authors in [Cud+13] performed an empirical evaluation of four *NoSQL* processing RDF systems (assessing the metrics of query execution times, loading times). However, we argue that these descriptive or diagnostic research efforts are limited to guide the practitioner to choose one approach (design decisions e.g., schema, partitioning, or storage formats) over another in the presence of performance trade-offs [Rag20].

Differently, *Akhter et. al.* [ANS18] presented an example of prescriptive performance analysis of RDF data systems. The authors adopted a *ranking* measure for comparing RDF partitioning techniques. Nevertheless, their analysis is limited to only one experimental dimension (i.e., partitioning). Thus, this work is not enough for making sense of more complex situations with several experimental dimensions with inherent trade-offs.

It is worth mentioning that, in Appendix 8 we provide detailed discussion (with illustrative tables) of each of the literature approaches/systems and where is the gap(s) that our current proposal aims to cover.

1.3. Research Questions

In this work, we follow the *Macro*, *Mezzo*, *Micro* analysis framework [LG15] for defining the requirements and assumptions to formulate the research questions at these *three* levels of analysis. The *Macro* level is the biggest unit of analysis and is generally related to complex and broad questions that cannot be directly solved but are useful for capturing the vision of analysis. The *Mezzo* level restricts the scope posing some requirements. Finally, at the *Micro* level, we formalize questions that can be tangibly and experimentally evaluated.

1.3.1. Macro Level of Analysis

Herein, we formulate the *Macro* question that frames our research. **Macro:** *"Can we identify the optimal way to process large graphs using relational BD frameworks?"*

The *"optimality"* in this context means the way(s) that achieves the best performance according to certain KPIs (e.g., query latency). This question allows multiple answers and thus, we shall reformulate it into more specific ones. Thus, we list specific *requirements* which help us to narrow down the problem and validate our approach.

1. **(R1) The approach should apply to the problem of querying *large* (RDF) graphs** with *latency* as a reference *KPI*.

2. **(R2) Choose system-agnostic experimental dimensions that directly impact the (BD) systems when querying large graphs.**
3. **(R3) Ensure the *replicability* of BD systems' performance while varying the experimental dimensions.**
4. **(R4) Consider multiple experimental dimensions' *simultaneously* to make sense of their *trade-offs*.**

The requirements state that the approach should apply to querying large (RDF) graphs (**R1**). As we have mentioned, the native graph database systems or triple stores (e.g., *Neo4j*, *RDF-3X*, *Apache Jena*, etc.) can easily handle the query workloads of small to medium volumes of (RDF) graphs [Jun+17]. The problem only arises when those centralized graph DB systems tackle complex graph query workloads against large volumes of graphs [Pha+15]. In this case, we fall back to the distributed relational BD systems that efficiently scale to handle large volumes of data.

However, those BD solutions still incur plenty of additional experimental design decisions that cannot be automatically decided. Seeking for generalizability and portability, this thesis opts for system-agnostic experimental dimensions, e.g., schema, partitioning, and storage (**R2**). However, we can find a number of experimental options on the bucket of each experimental dimension (see Figure 3), creating a complicated space of experimental solutions from which practitioners might choose.

Moreover, those design dimensions have intrinsic performance trade-offs, which can easily impact the replicability of the performance of a relational BD system (e.g., SparkSQL) when querying large KGs (**R3**). Thus, the solution we seek in this thesis opted for considering these multiple experimental dimensions *simultaneously* to make sense of their inherent *trade-offs* (**R4**).

1.3.2. From Macro to Micro Analyses

The requirements above help us narrow down the scope of the problem. In particular, we consider focusing on querying large RDF graphs on top of relational BD systems. Indeed, querying *small-to-medium* graphs can be handled by graph databases or native RDF triple stores. Therefore, the problem of querying KGs emerges only with large scales. Falling back to relational BD frameworks for querying those large RDF graphs entails several design decisions where prescriptive analysis can directly impact.

This thesis investigated the most common and well-studied dimensions in the state-of-the-art that directly impact the performance of a (BD) systems when querying large (RDF) graphs. Specifically, we started the investigation by two core experimental dimensions, i.e., the RDF *relational schemas*, and *storage* formats in a centralized environment [RTS19]. Then, we opted for considering the dimension of *partitioning* techniques seeking scalability in querying large RDF graphs [Rag+20; Rag20; RAT21]. These dimensions (i.e., *relational schemas*,

storage formats, and *partitioning*) directly reflect on the graph query workload operators in terms of query choke-points [Sal+16]. Indeed, the relational schema impacts the query joins, partitioning techniques impact data shuffling in distributed applications whilst storage formats impact physical execution plans, data loading times, as well as the data compression level based on their nature (i.e., *columnar* or *row-oriented*) [RAT21]. Moreover, these experimental dimensions are not *system-specific*, i.e., they can be tested in several BD relational systems.

Considering all these experimental dimensions alongside their underlying range of arbitrary options, the experimental solution space is too complex. Thus, it is hard for practitioners to make an *informed* decision, especially with the presence of experimental *trade-offs*. Thus, we can formulate a *Mezzo* question as follows:

Mezzo: *Can we identify the optimal setup(s) for querying large (RDF) graphs over relational BD systems without ignoring the underlying dimensions' trade-offs and guaranteeing the replicability of the results?*

We centered our case study around *Apache Spark* [Mic15] which is currently the most active and widely used *in-memory* large-scale data processing system in both industry and academia [BQ18]. In particular, *Spark-SQL* offers a prominent relational interface for implementing the relational schemas and answering translated SPARQL queries into SQL. Moreover, SparkSQL supports different partitioning techniques and multiple storage formats. Last but not least, several works in the literature use SparkSQL for processing large RDF graphs [Sch+16; CFL18; MAA18].

Finally, we formalize specific questions at the *Micro* level that can be experimentally evaluated in the following sections alongside an approach to answer them.

1.4. Approach and Contributions

Before digging into the details of our investigation, we shall explain the preliminaries of our experimental solution space.

Definition 1. *A configuration c is a combination of parameters that represent experimental dimensions. The configuration space $\mathscr{C}$ is the Cartesian product of the possible configuration combinations.*

We consider triplet dimensional configurations including relational schemas, partitioning techniques, and storage formats. Figure 3 shows the experimental design space and highlights the example of (*a.ii.3*) configuration, which is akin to Single Triples (*ST*) schema, Subject-based Partitioning (*SBP*) technique, and stored in *ORC* file format.

$$\mathscr{C} = \{Schema\}.\{Partitioning\_Technique\}.\{Storage\_Format\}$$

We run the experiments using two accepted RDF benchmarks (e.g., *WatDiv* [Alu+14] and *SP^{2}Bench* [Sch+08]) that include data generators and sufficiently complex query

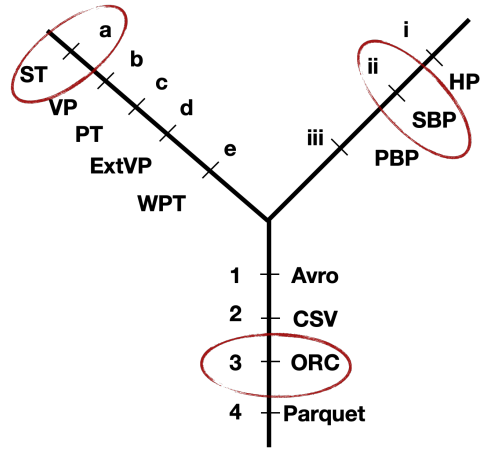


Figure 3: Configuration Space of experimental options composed of three dimensions (schema, partitioning, and storage), e.g., "a.ii.3" represents a configuration combined of the ST schema, SBP partitioning, and the ORC storage format.

workloads. In total, our experiments include 60 configurations (serializing all the combinations of dimension options[4] in Figure 3). Moreover, we consider scalability dimensions, and we conducted our experiments with different dataset sizes (i.e., 100M, 250M, and 500M triples).

1.5. Research Assumptions

Before we dig into how we address the research questions, we list our assumptions. In this thesis, we assume the following:

1. *Assumption 1*: The influence of the query workload in the experimental dimensions (especially the schema) is enough for implementing our prescriptive analysis.
2. *Assumption 2*: The experimental dimensions are discrete. That is, we cannot have hybrid options under the same experimental dimension, e.g., the RDF data stored in a hybrid relational schema of (PT+VP) [CFL18].

1.5.1. Making sense of Descriptive Performance Analysis (Micro 1)

Micro 1: *Can descriptive and diagnostic performance analyses of BD systems directly guide actionable decisions whilst querying large (RDF) graphs?*

Micro 1 investigates whether descriptive performance analysis can directly lead to actionable decisions when applied to BD systems that query large (RDF) graphs. In our scenario, actions could be deciding the optimal configurations for the BD system to execute the query workload in an efficient manner. Thus, this thesis (Chapter 3) will study the limitations of applying descriptive and diagnostic analyses to guide the practitioners in this task. Our results of answering this research question show that, it is hard as well as time-consuming to reach conclusions on the best-performing configurations of a BD system for querying large KGs via merely following descriptive and diagnostic analyses.

[4]C.F. see Chapter 2 for details about schema options (ST, VP,PT), partitioning options (HP, SBP, PBP) as well as storage formats (CSV, Parquet, Avro, ORC).

1.5.2. Assessing the Big Data Systems Replicability (Micro 2)

Micro 2: *can we guarantee replicability of BD systems performance when introducing other new experimental dimensions?*

Micro 2 investigates the effect of changing one experimental dimension on the system's performance replicability [5]. For instance, we check if the relational schema is not the only *impactful* dimension for the performance of relational BD systems for processing large RDF graphs. Changing even one parameter of the experimental setting may invalidate the performance results, making existing benchmarking solutions unfair [Rag+21]. Figure 2 shows a simple example of invalidating the performance of the system with the same schema (i.e., Property Tables), whilst changing the partitioning technique. Thus, we investigate the following hypothesis: *"The replicability of the BD system's performance for querying large (RDF) graphs could be affected by introducing other experimental dimensions".*

The answer of this research question (in Chapter 4) shows an example of how the performance of a BD system (i.e., Spark-SQL) with the RDF relational schema advancements (i.e., ExtVP, and WPT) cannot generalize against the *base-line* ones (i.e., VP, and PT) when introducing different experimental parameters (i.e., partitioning techniques, or storage formats) [Rag+21].

1.5.3. Bench-Ranking: Deciding over Complex Solution Space (Micro 3)

Micro 3: *How can we efficiently select the best-performing configurations out of a complex experimental solution space without ignoring the trade-offs?*

Micro 3 considers the assumption of the query workload's direct influence on the experimental dimensions (assumption 1) and investigates how to select the best experimental configurations. To this extent, we need to raise the level of abstraction going beyond descriptive and diagnostic analyses. That is, instead of comparing a huge number of experiments' performance results (i.e., that sometimes are even *contradicting* due to experimental trade-offs), we provide prescriptive performance analyses that can directly guide the practitioners on this hard task to reach actionable decisions.

Particularly, in this thesis (Chapter 5), we aims to employ *ranking functions* seeking actionable prescriptions. These ranking criteria aim to abstract from the *fine-grained* descriptive performance metrics and enable a *decision-making* model. Hence, this work adopts *Single-Dimensional* (SD) as well as *Multi-Dimensional* (MD) ranking criteria (i.e., *"Bench-Ranking"* [RAT21]) for ranking the performance of dimensions' parameters. Moreover, we discuss metrics for assessing the *goodness* of the proposed ranking criteria [RAT21].

1.5.4. Automating Prescriptive Performance Analysis (Micro 4)

Micro 4: *How can we automate prescriptive performance analysis of BD systems for processing large graphs with a complex experimental solution space?*

The *"Bench-Ranking"* [RAT21]) framework (i.e., the answer for Micro 3) provides the required methodology for attaining actionable prescriptions to BD systems' practitioners

[5] It does worth mentioning that the "replicability" refers to instances in which a researcher aims to arrive at the same scientific findings as a previous study while varying the underlying experimental setups (e.g., data, configurations, etc.) [NM+19].

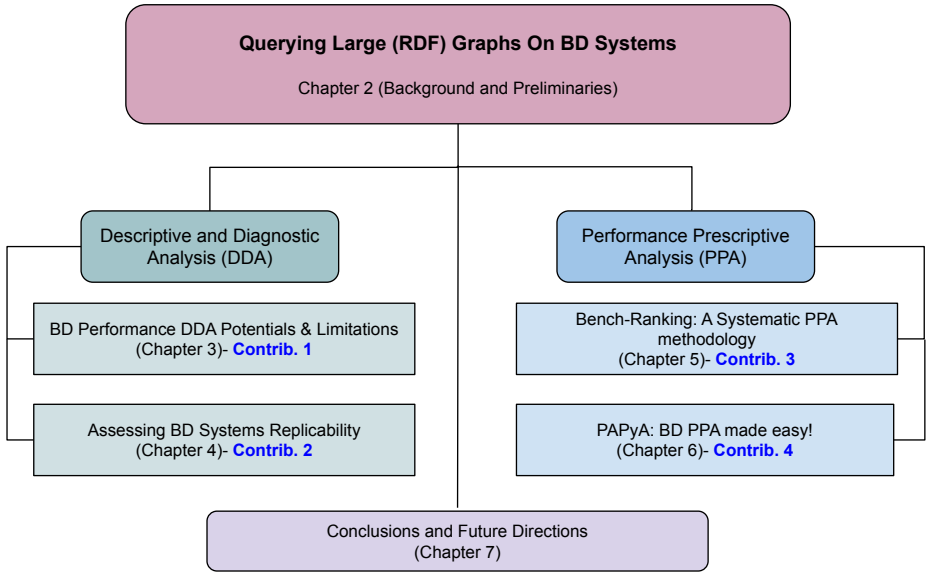


Figure 4: Structure of the thesis.

for querying large (RDF) graphs. However, the amount of experimental work required to implement PPA is still huge. To this extent, *Micro 4* investigates how to *automate* the Bench-Ranking methodology to attain PPA while reducing the amount of effort and time.

The answer of this research question (in Chapter 6) is translated into "*PAPyA* [6]", which is a *Python* library for enabling PPA that allows (1) preparing RDF graphs data for a processing pipeline over relational BD and (2) enables automatic ranking of the performance in a *user-defined* solution space of multiple experimental dimensions; (3) allows user-defined extensions in terms of systems to test, experimental dimensions, as well as ranking methods and criteria metrics.

1.6. Outline of the Thesis

The thesis is structured as follows (see Figure 4):

- Chapter 2 defines the relevant background and preliminary concepts on big data and its challenges, graph processing, graph querying, RDF data model, and SPARQL query language.
- Chapter 3 discusses the experimental solution space that emerges with processing large (RDF) graphs on top relational BD systems and shows why descriptive and diagnostic analyses are limited for this scenario. However, it also aims at making sense of the performance results and provides the best practices for processing large graphs on top of relational systems (e.g., Spark-SQL).
- Chapter 4 discusses the lack of replicability aspect that occur in relational BD systems that process large (RDF) graphs when introducing various experimental configurations.

[6]https://github.com/DataSystemsGroupUT/PAPyA

- Chapter 5 addresses the limitations of descriptive and diagnostic analyses by providing the "*Bench-Ranking*" methodology that helps enabling actionable insights on performance data of BD systems, while taking performance trade-offs into account.
- Chapter 6 presents *PAPyA* tool that wraps the functionality of "*Bench-Ranking*", aiming to reduce the time and effort required to implement prescriptive performance analysis.
- Chapter 7 gives concluding remarks and points out possible future research directions.

2. PRELIMINARIES AND BACKGROUND

In this chapter, we outline the preliminaries necessary to contextualize the information in the remainder of this thesis for readers unfamiliar with the research of (knowledge) graphs, Semantic Web, and relational (big) data processing.

Herein, we define the core technologies and standards of the Semantic Web, i.e., the Resource Description Framework (RDF) graph data model, and the SPARQL graph query language used for querying that model. To understand the terminology used in the following chapters, we also define the most relevant concepts to BD benchmarking efforts for querying large graphs.

2.1. Big Data Analytics and Challenges

The Big Data phenomena is a result of today's data explosion, which is fed by several applications like social networking, Internet of Things (IOT), and many websites that provide crowd intelligence features [Gha+13]. These applications are constantly generating data more than ever seen before. Therefore, BD analytics is a crucial process for analyzing large amounts of data to find information that might assist businesses in making wise decisions about their operations, such as hidden patterns, correlations, market trends, and customer preferences [Han+15; Gha+13], applying various techniques like predictive models, statistical algorithms, and *what-if* analysis.

Big Data can be characterized by the variety, complexity, and speed at which it must be processed or delivered, in addition to the volume of information involved [Gha+13; Sak+21], or as it is called the 3-Vs model: Volume, Variety, and Velocity [Lan+01]. Herein, we explain the Big Data *3V* challenges model as follows.

Volume Volume is the most obvious BD challenge. It represents the amount or size of data. In fact, volumes of data can reach unprecedented rates. It is estimated that more than 2.5 *quintillion* bytes of data are created daily. As a result, there will be more than 180 *zettabytes* of data created by 2025 [1]

Velocity: Velocity refers to the *speed* of generating, updating, or processing data [Han+15]. In some applications, data is generated at such a pace that requires specific processing techniques and fast update and analytics requirements (e.g., e-commerce, social networks, and click streams captured in web server logs). Thus, streaming systems must process data streams in *real-time* to keep up with their arriving speed and update requirements [Gha+13].

Variety: Big Data Variety refers to the range of data types and sources from which data is collected. The huge variety of data sources led to a diversity of data types, which includes *structured* data (e.g. relational tables), *unstructured* data (e.g. text, images, audios, and videos), and *semi-structured* data (e.g. graph and weblogs data).

2.2. Big Data Systems

In practice, the workload type and the BD challenges can characterize a big data system [Han+15]. Traditional data management systems are inadequate for the current chal-

[1] How Much Data Do We Create Every Day? `http://shorturl.at/bCUY4`

lenges of BD (i.e., Volume, Variety, and Velocity). Indeed, the obvious issue with BD is the enormous data scales that cannot fit on the typical data storage systems. For instance, the typical DB management systems struggle to operate efficiently with storing, reading, and writing large volumes of structured data. The volume challenge is not the only problem with these typical systems. As have been mentioned, the data can come in structured and unstructured formats, thus the relational DBs would is not a native fitting solution with (semi) or unstructured data. Similarly, typical DBMs cannot handle processing of large amounts of velocity data with the required minimal latency.

These limitations give the rise to new other efficient data storing and processing BD systems. BD Systems (BDSs) are scalable tools that can efficiently manage and analyze massive amounts of heterogeneous data (i.e., gathered from multiple sources) in a batch, stream, or hybrid fashion, decoupling the data storage from their analytics interfaces. The following are brief introductions of some of the selected big data analysis tools that are relevant to this thesis, along with a brief overview.

2.2.1. Apache Hadoop

Apache Hadoop [Vav+13] is an *open-source Apache* framework that is widely used in the processing of massive data. The *Hadoop Distributed File System* (HDFS) and *Map/Reduce* paradigm serve as the foundations of Apache Hadoop framework. HDFS is a the primary distributed file system in Hadoop applications. The key feature of HDFS is its ability to easily partition the massive data across different machines [Vav+13]. It depends on the design of a master machine (i.e., *NameNode*) that keeps track of how and where data blocks are split up across several data machines (i.e., *DataNode(s)*) on which data blocks are chunked, distributed, and replicated. This architecture helps the HDFS to provide a high-performance access to data. The *Map/Reduce* computing paradigm provides a simple distributed and parallel computing model for processing large data over a commodity cluster of machines. The *map* task divides the incoming data into key value pairs and computes them in parallel. The map task's output is consumed by the *reduce* task(s), which aggregates the output and produces the desired results.

2.2.2. Apache Hive

Apache Hive [Cam+19] is a widely-used data warehousing infrastructure that can be used for several BD *ETL* operations. Typically, Hive is designed as a programming layer on top of the Hadoop MapReduce, loading the data from and into HDFS. In particular, it offers a *declarative* query language (i.e., similar to the *SQL* dialect) called *HiveQL* to organize, aggregate, and execute query analytics against the data. Hive transforms*HiveQL* data processing instructions into distributed *Map-Reduce* or *Apache Tez* [2] jobs to run on the Hadoop cluster. As a result, this declarative querying paradigm saves Hive users from having to write time-consuming, difficult Map-Reduce code in order to manage data stored in HDFS. In a *metastore*, which can be a relational database (like *mysql* or *postgreSQL*) or file, Hive stores the metadata for its databases and tables.

2.2.3. Apache Impala

Apache *Impala* [Bit+15] is an open source *Massive Parallel Processing* (MPP) relational engine for massive volumes of data that is stored in Hadoop cluster. Impala is fully integrated into the Hadoop ecosystem and can perform *SQL* query analytics directly on data

[2] `https://tez.apache.org/`

stored in HDFS without the need for any data movement or transformation. Additionally, it is designed to be compatible with Apache Hive, and Hive users can use Impala with minimal setup time. That is, Impala can shares Apache Hive's metadata, SQL syntax (Hive SQL), *ODBC* driver, and user interface, resulting in a cohesive and comfortable environment for batch or real-time query analytics. However, unlike Apache Hive, Impala is not based on the *Map/Reduce* computing model. In particular, it implements a distributed architecture based on *daemon processes* that are responsible for query execution that runs on the same machines.

2.2.4. Apache Spark & Spark-SQL

Apache Spark [Zah+16; Mic15] is an *in-memory* distributed computing engine for large scale data analytics. It is quickly becoming the *de-facto* and one of the most widely-used large-scale distributed data processing systems [BQ18; SBB20]. Apache Spark relies on core data abstractions called *Resilient Distributed Datasets* (RDDs) and *DataFrames*, both are *immutable* distributed collections of data elements. DataFrames are, in addition, arranged into named and data-typed columns in accordance with a predefined schema, much like a table in a relational database..

Two key features of Apache Spark over its predecessors (e.g., Hadoop) are (1) the availability of *high-level* and declarative programming models such as SQL, as well as (2) the support for a wide variety of storage data formats (details in Section 2.3) [SBB20]. In addition, (3) Spark is a multi-language (i.e., various language bindings expressed in Python, R, java, or Scala) BD framework for applying data engineering, data science, and machine learning, graph analytics on centralized (single-machine) machine or in distributed clusters (of multiple machines). To this end, the Apache Spark engine is preferred by companies and (BD) practitioners for its powerful and high-level libraries, such as SparkSQL [Mic15] for structured and relational processing, Spark Streaming [Zah+13], *MLlib* [Men+16] for machine learning, last but not least GraphX [Gon+14] for implementing distributed large-scale graph analytics. Notably, this thesis focuses on the relational processing paradigm; and thus we particularly focus on the SparkSQL engine.

SparkSQL [Mic15] is a Spark high-level module for processing structured data on top of DataFrames. It offers the DataFrame programming abstraction and functions as a distributed SQL query engine. In particular, SparkSQL enables querying data stored in the DataFrames abstraction using SQL-like languages, optimized using the `Catalyst` query optimizer[3] to make queries fast. As a result, it can runs relational query analytics up to $100x$ faster on existing deployments and data [4]. It also uses the Spark engine, which has complete mid-query fault tolerance and scalable to hundreds of nodes and multi-hour searches, so there is no need to worry about utilizing a different engine for historical data. Additionally, it offers strong interoperability with the remainder of the Spark ecosystem (e.g., integrating SQL query processing with machine learning, or graph query analytics). Last but not least, Spark supports different storage backends for reading and writing data.

2.3. Big Data Distributed Storage

While designing and creating Big Data engineering pipelines, the *storage* is a core design decision. In fact, the *variety* challenge of Big Data incorporates several emergent differ-

[3] `https://databricks.com/glossary/catalyst-optimizer`

[4] `https://www.databricks.com/glossary/what-is-spark-sql`

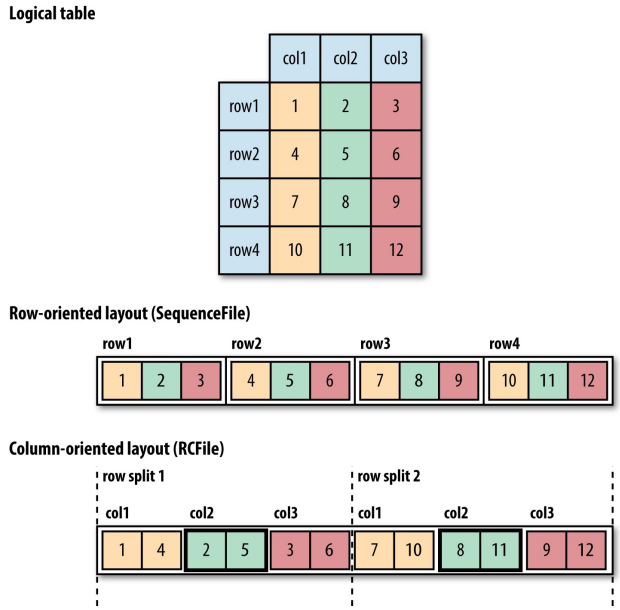


Figure 5: Row-oriented and column-oriented data formats [ada17].

ent file formats and a range of various storage backends. Thus, in Big data applications, choosing the appropriate file format for data analytics is a crucial design decision. Depending on the use cases, each storage choice (i.e., data file format) has different benefits and drawbacks and exists to fulfill one or more objectives. For example, in Business Intelligence (BI), network communication, online applications, and batch or stream processing, various file formats are more practical for specific usages and application situations. Additionally, in reality, deciding on the best-performing storage is mainly dependent on the computational load, i.e., whether analytical (*OLAP*) or transactional (*OLTP*).

Row-oriented File Formats In a *row-oriented* format, the values for each row are stored contiguously in the file. In other words, the first column of a row will be adjacent to the last column of the previous row (see Figure 5). This design makes it easier for adding (writing) data. It is also more efficient in cases where the entire row of data needs to be accessed or processed at once. Thus, row-oriented formats are frequently used with Online Transactional Processing (OLTP) workloads. Indeed, this kind of transactional workload usually processes CRUD queries (Create, Read, Insert, Update and Delete) at a record level. Herein, we provide exemplars of row-oriented file formats.

CSV: The Comma-separated values data format is a widely used and accepted data exchange format that employs commas to separate values. Typically, plain text is used to communicate tabular data across systems. Because CSV is a row-based file format, each line of a CSV file represents a data record with one or more fields, and each record has a consistent list of fields, resulting in a simple information structure.

Avro: is described as a data serialization system similar to Java Serialization. It enables storing complex objects by encoding the schema of their content. In particular, the data is saved in binary format, whereas the Avro schema is kept in *JSON* format. This architecture significantly minimizes file size and maximizes efficiency.

Avro manages fields that have been added, removed, or altered to provide reliable support for schema evolution.

Column-oriented File Formats A column-oriented format, on the other hand, divides a file's rows into row splits, with each split subsequently being stored in a column-oriented manner. Particularly, the values for each row in the first column are stored first, followed by the values for each row in the second column, and so on (see Figure 5). It enables columns that are not visited in a query to be skipped, i.e. *column pruning*. By removing any columns that do not pertain to a given query, columnar data storage prevents the excessive processing delays associated with visiting or extracting *irrelevant* information from a data set. Thus, columnar file formats are most efficient when performing analytical queries that only need a portion of the columns in very large data sets to be analyzed. This way of processing is called OLAP (*Online Analytical Processing*).

Parquet: Apache Parquet is an open-source columnar file format that makes use of nested data structures that were inspired by the*Google Dremel* [Mel+10] framework [IP19]. Parquet supports efficient compression and has encoding schemas support on the columns level. Typically, Parquet works efficiently with Apache Hive and Apache Spark as a way to store columnar data in deep storage that is queried using *SQL(-like)* languages. Additionally, it relies on the *Apache Thrift* [5] framework to define the metadata representations (e.g., file, column, and page header metadata).

ORC: is a *type-aware* columnar file format. ORC is a self-describing format that utilizes metadata such as various statistical information related to the columns, which enables input split elimination based on predicate push downs. ORC format was first created for Hadoop workloads, but is currently used as a general-purpose storage format. Compared to other columnar or row-oriented file formats, ORC offers greater compression ratios. As a result, the ORC format significantly reduces I/O costs to achieve exceptional read performance for a variety of applications, such as large streaming reads, and it also has numerous advantages over its predecessor (i.e., the *RC* file format) [IP19]. Last but not least, ORC offers enhanced support for schema evolution.

2.4. The Semantic Web and Linked Data

The *Semantic Web* refers to the *web of data*, readable and processable by machines [AV04]. The Semantic Web is one of the most essential elements involved in the notion of Knowledge Graphs [GS21]. The main reason for its existence is to allow both humans and machines to interpret and interact with data. It provides an account of inferring the *meaning* in which the logical connection of terms establishes interoperability among systems. To support this end, the *Web Wide Consortium* (W3C [6]) working group defines standards, technologies, and recommendations for organizations to share and publish their data.

Herein, we list the core technologies and standards for representing and querying semantic linked data.

[5] `https://thrift.apache.org/`

[6] `https://www.w3.org/groups/`

2.4.1. Resource Description Framework (RDF)

The Resource Description Framework (RDF [7]) is the standard *W3C* graph data model for establishing semantic interoperability on the Web [GS21]. It represents information about resources on the web by assigning attributes and creating relationships among them via *Unified Resource Identifiers* (URIs). RDF arranges data in the form of *triples* (*Subject(S)*, *Predicate(P)*, *Object(O)*). The subject represents the resource being described, the predicate is the attribute, and the object contains the value associated to the attribute for the given subject. More generally, the predicate represents a relationship/connection between a subject (S) and an object (O) nodes.

RDF data can be viewed as a *directed edge-labeled graph*, with edge labels serving as the predicates and vertices representing the entities. Moreover, an RDF dataset is composed of RDF triples and can be described in a variety of formats and data serializations, i.e., *RDF/XML*, *Turtle*, *N-Triples*, *OWL*, and many more. An RDF resource can take one of the following forms:

IRI Node *IRI Node* provides a global identifier for a resource. It is referred to by others on the Web. An IRI can occur in any position in the triple (s, p, o).

Definition 2 (RDF Triple)**.** *An RDF triple consists of three components:*

- *Subject* $S \in I \cup L \cup V$
- *Predicate* $P \in I \cup V$
- *Object* $O \in I \cup L \cup V$

$$(S,P,O) \in (I \cup B) \times I \times (I \cup B \cup L).$$

where I is the set of IRIs. B is the set of blank nodes. L is the set of literals.

Definition 3 (RDF Graph)**.** *An RDF Graph is a finite set of RDF triples. If G is an RDF Graph, we use I(G), L(G), and B(G) to denote the set of IRIs, literals, and blank nodes. RDF Graphs are organized into datasets.*

Literal node A *Literal node* represents various datatype values such as strings, numbers, and dates. It can be only part of an object (o) in a triple. By default, a literal node is plain (un-typed). However, it is possible to structure a literal by assigning additional information about the interpretation of a literal. This is possible by appending the information of the datatype to the literal value. For example, a plain literal node can be typed as an integer number when we add the suffix "`^^xsd:integer`".

Blank node A *Blank node* is a special unique identifier used to denote the existence of an RDF entity [GS21]. Some RDF data serialization methods allow creating triples with the blank node identifier "`_:`". However, only the subjects and objects are allowed to be defined with blank node identifiers.

Definition 4 (Blank node)**.** *Blank nodes are disjoint from IRIs and literals. RDF makes no reference to any internal structure of blank nodes.*

Definition 5 (RDF Dataset)**.** *An RDF Dataset is a collection of RDF graphs. It contains one RDF graph by default, which is either uniquely named or not. An RDF Dataset holds one or more named graphs:*

$$\{g_0, (u_1, g_1), (u_2, g_2), ...(u_n, g_n)\}$$

[7] `https://www.w3.org/TR/rdf11-concepts/`

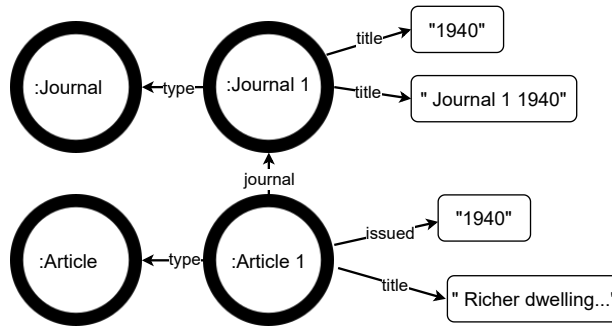


Figure 6: Example of an RDF graph. Prefixes are ignored for simplicity.

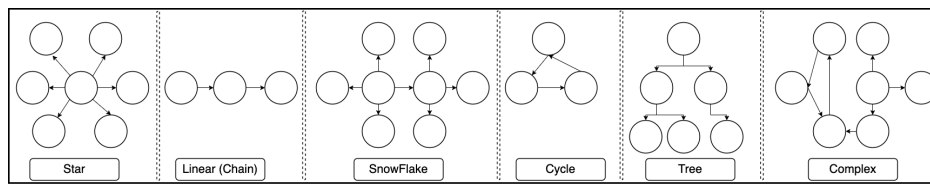


Figure 7: Shapes of SPARQL Queries.

Figure 6 shows an example of RDF graph, while Listing 2.1 shows its representation in the *N-Triples* serialization.

2.4.2. SPARQL Protocol and RDF Query Language (SPARQL)

Multiple languages, like as Cypher, Gremlin, and G-CORE have been proposed for querying property graphs [Cam+19; GS21]. Similarly, SPARQL[8] is the standard query language of RDF datasets that harnesses the simplicity of the RDF model to describe both simple and complex queries. The simplest form of a SPARQL query is called *"Basic Graph Pattern (BGP)"*. In a BGP, the SPARQL query consists of a set of RDF triple patterns that may also contain *variables* (i.e., unbound Subjects, Predicates, or Objects) that may appear in multiple patterns. Terms in BGPs are split into *variables*, which are preceded with *question marks* (like *"?yr"* in Figure 8 and Listing 2.2), and *constants* (like *"Journal 1 (1940)"* in Figure 8 and Listing 2.2). Similar to triples, triple patterns are modeled as directed graphs (Figure 8). Thus, answering a SPARQL BGP query is usually enclosed as a *sub-graph pattern-matching* problem [Zou+11; GS21].

Given an RDF graph G, the evaluation of a BGP B, denoted as $B(G)$, returns a set of *solution mappings*. A solution mapping is a partial mapping from the set V of variables to the set of RDF terms. Figure 8 provides an example of a BGP that is evaluated against the RDF graph in Figure 6. The solution mapping in this example is composed of only one variable (i.e., *yr*) with *one* value mapped to it, i.e., *"1940"*.

SPARQL query shapes: In practice, the SPARQL BGP queries can have different shapes (relying on the position of variables in the triple patterns), which can have severe impacts on query performance [Sch+16]. Figure 7 depicts the most common BGP shapes. (1) *Star* query: consists only of *subject-subject* joins where a join variable is the subject of all

[8] `https://www.w3.org/TR/sparql11-query/`

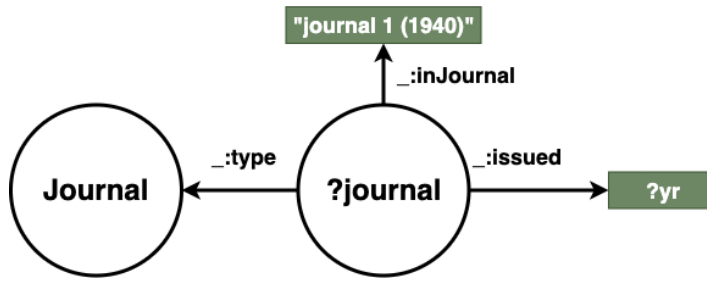


Figure 8: SPARQL query shown as a graph model.

the triple patterns involved in the query. Star patterns frequently exist in the current RDF benchmarks and SPARQL query workloads in general. Thus, many RDF processing systems are optimized for this kind of workload [Sch+16]. (2) *Linear*, *path-shaped* or *Chain* query: is also very common in RDF graph querying. It consists of *subject-object (or subject-object)* joins where the triple patterns are consecutively connected like a line or a chain. (3) *Snowflake-shaped* query: consists of combinations of many star shapes connected by typically short paths. (4) *Cycle* query: contains subject-subject joins, subject-object joins and object-object joins. (5) *Tree* query: consists of subject-subject joins and subject-object joins. (6) *Complex* query: consist of a combination of different shapes.

```
:Journal1 rdf:type :Journal ;
          dc:title "Journal 1 (1940)" ;
          dcterms:issued "1940" .
:Article1 rdf:type :Article ;
          dc:title "richer dwelling scrapped" ;
          dcterms:issued "2019" ;
          :journal :Journal1 .
```

Listing 2.1: RDF example in *N-Triples*. Prefixes are omitted for simplicity.

Listing 2.2: SPARQL Example against RDF graph in Listing 1.1. Prefixes are omitted for simplicity.

```
SELECT ?yr
WHERE { ?journal rdf:type bench:Journal.
        ?journal dc:title "Journal 1 (1940)".
        ?journal dcterms:issued ?yr. }
```

SPARQL 1.1 [Con+13], which originated in 2013, is the latest stable version of the language. It features query language operations such as *aggregates, subqueries, negation, property paths*, and various new functions and operators. For a complete description and formalization of the SPARQL language, readers may refer to the *SPARQL 1.1* specification document and also the work in [Con+13] for an exhaustive formal definition of its semantics.

Typically, the SPARQL query consists of *three* main parts. The pattern matching part that includes a set of BGP triple patterns. the second part includes several matching criteria operators, like *OPTIONAL*, *UNION* of patterns, and *FILTER* operator for restricting

candidate matches. Lastly, the final result modifiers, which states operations of projection (*SELECT*), *DISTINCT*, *LIMIT* and *OFFSET* on the candidate output results. Listing 2.2 illustrates an example of SPARQL against the simple RDF graph in Figure 6. It asks about the year of publication of journal with the title *"Journal 1 (1940)"*.

2.5. Relational Model and Relational Algebra For SPARQL

Relational Model: The relational model, first described by *Edgar F. Codd*, is an approach to managing structured data using a *declarative* language consistent with the *first-order logic* [E09]. The relational model represents data in terms of rows (i.e., tuples) grouped into relations (i.e., tables). A database organized in terms of the relational model is referred to as a relational database.

Relational Algebra [Cyg05] (RA) is used as the core of query language for Relational Databases. RA is an intermediate language commonly used for interpreting the expression of SQL queries. Relational algebra is utilized in a huge portion of work on query planning and optimizations to the SPARQL realm. More specifically, evaluating BGPs returns solution mappings that can be viewed as relations (tables), where variables are attributes/fields (i.e., column names) and tuples(i.e., rows) contain the RDF terms bound by each solution mapping.

The default relational operator for the SPARQL BGPs is the natural inner join ($\bowtie$). While, Complex BGPs support combining and transforming the results of BGPs with language features that include *FILTER*(selection:σ), *SELECT*(projection:Π), *EXISTS*(semi-join:$\ltimes$), MINUS ($\triangleright$), *OPTIONAL*(left-join:$\bowtie$), and *UNION*(union: $\cup$). These language features correspond to the relational algebra defined for SPARQL query clauses.

2.6. RDF Processing Systems

The wide adoption of the RDF data model has triggered the emergence of several efficient and scalable RDF systems. Those systems have been designed to manage, store and query RDF datasets and tackle the challenges that may come with scalability. Generally speaking, these systems can be broadly classified into two main categories: *native* and *non-native* RDF querying/processing systems [Bor+13; Pha+15].

2.6.1. Native RDF Graph Processing Systems

These systems, also called "*triplestores*" (RDF stores) are purpose-built graph databases for managing, storing, and retrieving RDF triples using native semantic queries, i.e., using the SPARQL language. Thus, one can consider a triplestore as the semantic graph DB of RDF data. The next evolution of triplestores is what is called *quad stores* that include information about the *context* or *named graphs*, in addition to the three main components of the triplestores (i.e., Subject, Predicate, and Object). Examples of those native stores are *RDF-3X* [NW08], *4store*[HLS+09], *g-store* [WZZ17], and *Jena TDB/SDB* systems [McB01].

On one hand, RDF triplestores provide the full expressiveness of SPARQL that (technical) users may need in order to capture complex information needs. Moreover, they fully harness the structure and semantics captured by the underlying data. In addition, RDF triple stores are *"schema-last"* approaches for handling the heterogeneous nature of data with the flexibility to handle them with the power of SPARQL (i.e., seamless graph navigation, recursive queries, negations, as well as expressing complex BGP queries).

However, those RDF solutions are centralized, which means they might fall short with large volumes of data or complex workloads [Pha+15]. Moreover, those systems come with limited SPARQL optimizations, as well as no grip on data locality (even with using the *Hexa-storage* that entails indexing all orders of RDF components) [Pha+15; Bor+13].

2.6.2. Non-native (Relational) RDF Processing Systems

Due to the triplestores limitations, the Semantic Web community started to investigate other non-native solutions for handling RDF graphs. Indeed, there is evidence that going native pays in terms of efficiency [Bor+13; Pha+15]. In action, we cannot ignore relational stores that come with 35+ years of research on efficient storage and querying. Moreover, relational solutions provide more important features such as scalability, compression, security, and better query optimization [Bor+13]. Talking only about the scalability, indeed, with the huge growth of RDF data [Aba+07; Sch+16; Sch+14; SA10], the need for large relational systems aggravates. The reason is that these large volumes of RDF data cannot be handled efficiently with the "*centralized*" RDF systems [Pha+15]. While on the other hand, big relational engines are distributed and provide scalability and data partitioning capabilities.

2.7. RDF Relational Schema Representation

It is easy to show that we have different options of representing RDF graphs in various *relational schemas*. The relational schema is a crucial design decision when building data pipelines. This choice hugely impacts the performance of the BD engines [SA10; RTS19]. Herein, we identify the most used RDF relational schema in the state-of-the-art of KGs processing [Abd+17; Aba+07; Cho+05; Sch+16; Sch+14].

For each schema we give an example of data using Listing 2.1, and we provide the respective SQL translation of the SPARQL query in Listing 2.2.

Single Statement Table Schema (*ST*) (also commonly known as the *triples table*) entails storing RDF triples in a single table with three columns that represent the three components of the RDF triple, i.e., *subject*, *predicate*, and *object*. That is, each RDF statement is represented by each table record. The ST schema is widely adopted [Aba+07; Cho+05] in research and in RDF systems,e.g., the major open-source triplestores, i.e., *Apache Jena*, *RDF4J* and *Virtuoso*. The RDF-3X system [NW08] also uses a slight variation of the ST representation that maintains multiple copies of the ST table. Each copy has a clustered index that implements a different sort order of the RDF three components, i.e., (S,P,O),(P,O,S), (S,O,P),..etc. Figure 9 shows the ST schema representation of the sample in Listing 2.1, and the associated SQL translation for the SPARQL query in Listing 2.2.

Property Tables Schema (*PT*) This paradigm suggests the creation of relational-like property tables from RDF data, making it a more feasible scheme for RDF organization in relational databases. It requires to cluster multiple RDF properties as *n-ary* table columns for the same *subject* to group predicates that are pertinent to the same entity. That is, the *subject* column serves as the table key (the unique identifier for the row), while each *property* column may store an *object* value or be *null*. Thus, the PT schema works perfectly with highly structured data, but not with the poorly structured datasets [Sch+08], due to the high number of *null* values that it might incur. Moreover, due to its *sparse* tables representation nature, the PT schema may suffer from high storage overheads when a large number of predicates is present in the RDF data model [Abd+17]. In practice, several systems support property tables for RDF data management. They either derive the

Subject	Predicate	Object
journal1	type	Journal
journal1	title	'Journal(1940)'
journal1	issued	1940
journal1	editor	Sharise_Heagy'
article1	type	Article
article1	title	richer dwelling scrapped'
...	...	...

```
SELECT T3.object AS year
FROM SingleTable T1, SingleTable T2, SingleTable T3
WHERE T2.subject=T2.subject AND
T2.subject=T3.subject AND
T1.object='http://.../sp2bench/Journal" AND
T2.predicate='http://purl.org/dc/elements/1.1/title' AND
T2.object='Journal 1 (1940)' AND
T3.predicate='http://purl.org/dc/terms/issued'
```

Figure 9: ST Schema and an associated SQL query sample. Prefixes are omitted.

type		title		issued	
Subject	Object	Subject	Object	Subject	Object
journal1	Journal	journal1	'Journal(1940)'	journal1	1940
article1	Article	article1	richer dwelling	...	...
...	...	...	...	...	...

```
SELECT T2.object AS year
FROM Title T1, Issued T2, Type T3
WHERE T2.subject=T2.subject AND
T2.subject=T3.subject AND
T3.object='http://.../sp2bench/Journal" AND
T1.object='Journal 1 (1940)'
```

Figure 10: VT Schema and an associated SQL query sample. Prefixes Omitted.

Journal					Article			
ID	title	issue	editor	type	ID	title	jorunal	type
journal1	'Journal(1940)'	1940	Sharise_Heag	Journal	article1	'Journal(1940)'	journal1	Article
...	...	...	...	...	...	...	...	...

```
SELECT J.issued AS year
FROM Journal J
WHERE J.title='Journal 1 (1940)'
```

Figure 11: PT Schema and an associated SQL query sample. Prefixes are omitted.

PT schema structure automatically from the ontology of the dataset (e.g., Sesame) or are manually defined by the application (e.g., Jena). Figure 11 shows the relational flattened property tables of the RDF graph in Listing 2.1 and the associated *SQL* translation for the SPARQL query in Listing 2.2.

Vertically Partitioned Tables Schema (*VT* [9]) is a a specialized version of the Property Tables (PT) schema that requires storing RDF triples into binary tables of two columns (*Subject, Object*) for each *unique* property in the RDF dataset. The VT schema was proposed to speed up the queries over RDF triple stores that used to struggle with the excessive ST table self-joins [Aba+07]. Figure 10 shows the VT representation of the sample RDF graph shown in Listing 2.1, and the associated SQL translation for the SPARQL query in Listing 2.2.

2.7.1. Advancements of RDF Relational Schemas

It is worth mentioning that there are two other advancements of the RDF relational schemata mentioned above. In particular, the Wide Property Tables (WPT) schema [Sch+14] and *Extended Vertical Partitioning* (*ExtVP*) [Sch+16].

Wide Property Table Schema (*WPT*) The entire RDF dataset is represented by this schema into a single *unified* (i.e., *non-normalized*) table. All of the RDF graph dataset's predicates are used as columns in this table. The WPT schema extends the PT schema seeking for optimizing the *star-shaped* SPARQL queries (see SPARQL shapes Section 2.4.2), which are highly common in the SPARQL query workloads. Indeed, the *star-shaped*

[9]The "VT" and "VP" are two different acronyms interchangeably used through the thesis to represent the Vertically Partitioned Tables Schema.

WPT					
ID	title	issued	editor	journal	type
journal1	'journal 1 (1940)'	1940	Sharise_Heag	null	Journal
article1	richer dwelling scrapped	null	null	journal1	Article
...	...	...	...		

```
SELECT T2.object as year
FROM Title T1, Issued T2, Type T3
WHERE T2.subject=T3.subject AND
T1.subject=T2.subject AND
T3.object='Journal' AND
T1.object='Journal 1(1940)'
```

Figure 12: Vertically Partitioned Tables(VP) schema

issued	
subject	object
journal1	1940
...	...

type	
subject	object
journal1	Journal
article1	Article
...	...

issued | title SS	
subject	object
journal1	1940
...	...

title	
subject	object
journal1	'Journa (1940)l'
article1	richer dwelling scrapped
...	...

type | title SS	
subject	object
journal1	Journal
article1	Article
...	...

```
SELECT T1.object
FROM issued|title T1, type|issued T2
WHERE T1.subject=T2.subject
```

Figure 13: Extended Vertically Partitioned Tables(ExtVP) schema

SPARQL queries will require *no* joins to be answered while using the WPT schema [Sch+14]. Unlike the PT schema, this schema does not require any kind of dedicated or ad-hoc clustering algorithms (that are likely to produce *sub-optimal* relational schema for an arbitrary RDF dataset [Sch+14]). Unfortunately, the WPT schema does not completely eliminate the PT model's drawbacks. Poorly formatted RDF data can also have a very sparse WPT representation, which can result in significant storage costs, particularly when there are a lot of multi-valued characteristics included in the RDF dataset. Figure 12 also shows the WPT associated SQL query mapped from the mentioned SPARQL query in Listing 2.2.

Extended Vertical Partitioning (*ExtVP*) is a *query-driven* schema advancement that

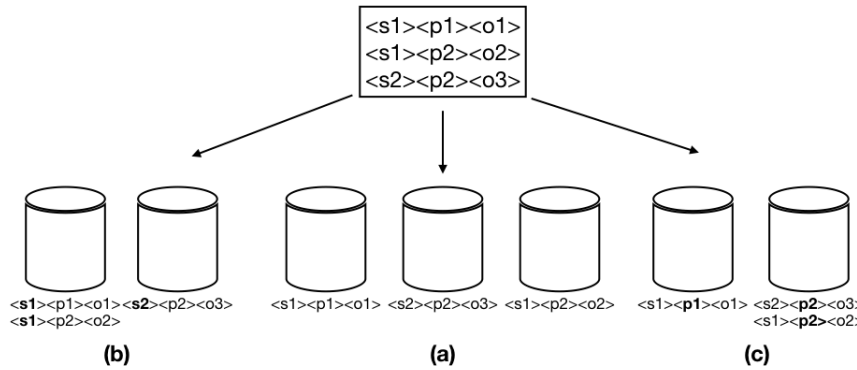


Figure 14: RDF partitioning techniques, (a) HP, (b) SBP (c) PBP.

aims at minimizing the size of the input data during query evaluation. It is motivated by the *Semi-Join* reductions of the possible *VP* tables join correlations that occur among the SPARQL query triple patterns. The ExtVP schema speeds up query answering by *pre-computing* the potential join relations between the VP tables and materializing the results of these *semi-joins* as tables in the storage backend (e.g., HDFS). Particularly, for every two VP relations ExtVP relies on computing the semi-join reductions of *Subject-Subject* (SS), *Subject-Object* (SO), and *Object-Subject* (OS) join patterns. The query input tables are reduced in size; by eliminating *dangling* triples that do not contribute to any join; and will be used in query joins instead of the original VP tables [Sch+16]. However, one of the drawbacks of the ExtVP schema is the excessive additional storage overhead of the materialized ExtVP tables (when compared to the VP schema tables). However, this overhead can be tuned using a *selectivity threshold* to control the materialization of semi-joins tables by keeping only the ones that make significant table reductions [Sch+16]. Figure 13 also shows the ExtVP associated SQL query version of the query in Listing 2.2.

Ii is worth mentioning the we keep more details about the RDF benchmarks relational schema in Appendix 9. Specifically, we provide our designed PT schema of them, because unlike the other RDF relational schemas, i.e., ST, (EXT)VT, WPT, the PT schema requires designing efforts from the (Big) Data engineers.

2.8. (RDF) Graph Partitioning

The ever-growing size of data is often too large to be efficiently managed on a single machine [ANS18; Abb+18; Abd+17]. This challenge has called for the typical design choice of scaling horizontally by splitting the query workload as well as the data over several nodes in distributed systems [Lin18; SÖ18]. Data partitioning is the process of dividing datasets into portions to facilitate better maintenance and access. Data partitioning is the key process used for achieving load balancing, improving system availability, and query processing times in data management systems [ANS18]. Typically, the query execution performance in data systems can be greatly affected by the partitioning technique used in the data store. Thus, partitioning is a key design choice in centralized as well as distributed systems.

As we have mentioned, we witness huge RDF datasets that have been added to the web of data and still growing, e.g., Linked TCGA (around 20 billion triples) and UniProt (over 10 billion triples). Taming the volume of such huge data can be managed using distributed systems and by partitioning this RDF data over several machines. Herein, we

explain commonly used RDF graph partitioning techniques. The RDF graph partitioning problem is defined as follows.

Definition 6 (RDF Graph Partitioning)**.** *Given an RDF graph $G = (V,E)$, such that V and E are the set of all vertices and edges, respectively in the graph G, a partitioning technique* **T** *divides the graph G into n sub-graphs $G1, G2, ...G_n$ such that $G = \bigcup_{i=1}^{n} G_i$. In other words $G1 \cup G2 \cup ..... \cup G_n = G(V,E)$.*

In this thesis, we categorize the RDF partitioning techniques into two categories depending on their nature, i.e., *native* graph partitioning, and *non-native* (*relational*) graph partitioning.

Relational Graph Partitioning: First, we list the following partitioning techniques that can be applied directly to relational tables (relational schemas) generated from an RDF graph.

- Horizontal-Based Partitioning (*HP*) requires partitioning the data evenly on the number of machines in the cluster. In particular, it divides the relational tables, i.e., in *n* equivalent chunks where *n* is the number of machines in the cluster.
- Subject-Based Partitioning (*SBP*) requires distributing triples to the various partitions according to the *hash value* computed for the *subjects*. As a result, all the triples with the same *subject* are assumed to reside on the same partition.
- Predicate-Based Partitioning (*PBP*) requires, similar to the SBP, to distribute triples to the various partitions based on the *hash value* computed for the *predicate*. As a result, all the triples with the same *predicate* are assumed to reside on the same partition.

Native Graph Partitioning: On the other side, distributed native graph processing has been widely adopted in recent years and enables knowledge extraction from large and medium-scale graph-structured datasets using commodity clusters [Abb+18]. Native graph partitioning can be divided into two main approaches, namely *vertex* partitioning and *edge* partitioning. The goal of both approaches is to minimize *cross-partition* dependencies by defining a *minimum-cut* optimization objective [Abb+18].

Underneath those two broad categories, there exist several native graph partitioning algorithms, techniques as well as tools [ANS18]. Particularly speaking about RDF graph partitioning, we discuss the following native graph (RDF-specific) partitioning techniques.

- Recursive-Bisection Partitioning is a *multilevel* graph *bisection* algorithm aiming to solve the *k-way* graph partitioning problem [KK98]. The algorithm works by recursively dividing the graph *G* in two halves until *k* blocks are present [Bul+16] passing by three main phases, namely *Coarsening*, *Partitioning*, and *Uncoarsening*[ANS18].

- TCV-Min Partitioning is similar to the *Recursive-Bisection* technique, as it also aims to solve the *k-way* graph partitioning problem. However, this technique is driven by the objective of minimizing the total *communication volume* of the partitioning [Bul+16].

- Min-Edgecut Partitioning also aims to handle the *k-way* graph partitioning problem. However, unlike *TCV-Min*, the objective is to partition the vertices by minimizing the number of edges connected to them.

- Hierarchical Partitioning is inspired by the assumption that RDF *IRIs* have path hierarchy and IRIs with a common hierarchy *prefix* are often queried together [JST17]. This technique of partitioning is based on extracting path hierarchy from the IRIs and assigning triples having the same hierarchy prefixes into one partition.

3. MAKING SENSE OF BIG DATA SYSTEM PERFORMANCE FOR PROCESSING LARGE KNOWLEDGE GRAPHS

The movement from native processing of RDF datasets (using triple stores) to the non-native relational approaches requires incorporating BD systems for processing. Although Big Data frameworks are not tailored to perform native RDF graphs processing, several approaches exist for processing the RDF data using relational systems [Abd+17; Sch+16; Sch+14; MAA18]. However, the choice of such relational systems for processing and querying large graphs comes with several design decisions to consider while building a BD analytical pipeline. Those additional design decisions are hard to be automatically decided [Van+21] just given the graph dataset and workloads. Moreover, changing those design decisions in the future (if selected wrongly from the first design) has extremely high costs and requires huge data engineering efforts.

In this chapter, we aim at answering the first micro question (Micro 1): "*Can descriptive and diagnostic performance analyses of BD systems directly guide actionable decisions whilst querying large (RDF) graphs*". Particularly, we aim at making sense of relational BD system performance for processing large (RDF) graphs in the presence of complex experimental solution space that constitutes multiple experimental dimensions alongside many alternatives. We consider the most impactful design decisions for processing large RDF graphs in a distributed relational context. By the end of this chapter, we show the limitations of the descriptive and diagnostic system performance analysis, showing the need for prescriptive analysis that directly leads to actionable decisions.

3.1. Experimental Design Decisions

For deciding those dimensions, as well as the system that is subject to evaluation, *Jim Gray's* principles of defining a useful benchmark [Gra93] inspire us to set our requirements as follows:

1. **R.1** *Relevance*: we aim to focus on relevant design decisions that emerge with the problem of processing large RDF graphs on top of relational systems, as well as a relevant system to handle this problem.
2. **R.2** *Scalability*: we consider scalability aspects on the level of data and system, as we target processing large (RDF) graphs.
3. **R.3** *Simplicity*: we advocate for accepted benchmarks that require a well-known language for implementing the analysis (i.e., *SQL*). Moreover, our study is limited to three experimental dimensions (i.e., *Schema*, *Partitioning*, and *Storage formats*).
4. **R.4** *Portability*: we focus on tasks and experimental dimensions that are not *system-specific*.

Derived by the above requirements, we consider relevant design decisions for processing large RDF graphs on top of a BD relational system (**R.1**). In particular, the relational schemas, partitioning techniques, and storage format as our design decisions to set up an analytical pipeline.

Requirement	Details
(R.1) Relevance	- Relevant Dimensions directly impact BD performance. - Relevant BD system (e.g., Spark-SQL).
(R.2) Scalability	- Scalable Benchmarks (Data generation). - Data Partitioning.
(R.3) Simplicity	- Simple language (SQL) for query processing. - Few design decisions.
(R.4) Portability	- Portable (System-agnostic) dimensions. - Standard Query Language (SQL).

Table 2: Requirements that guide our study experimental design space.

Those dimensions directly impact the performance. Indeed, the relational schemas impact the number of joins in the query workload (C.F., Table 3), and the partitioning techniques impact data shuffling in a distributed environment (required for scaling out, distributing data over several machines (**R.2**)), and the storage format impact the query physical plan and data loading time based on their nature (i.e., columnar or row-oriented) and their compression levels. It is also worth mentioning that the aforementioned dimensions do not depend on existing systems, making our investigation *system-agnostic*, i.e., it can be replicated on any distributed BD relational engine for RDF graph processing that supports SQL (**R.4**), e.g., *Apache Hive* [Cam+19], *Apache Impala* [Bit+15], *Apache Drill* [HN13] to name a few.

It is also worth mentioning that those dimensions can be extended with any other *system-agnostic* and *performance-impactful* dimensions. Chapters 5 and 6 will show that our framework solution is flexible to be extended with arbitrary any number of experimental dimensions, and dynamic to adding and removing experimental alternatives across each dimension.

3.1.1. Dimensions' Experimental Space

Intuitively, such experimental dimensions entail different options of choice (we call them dimensions' parameters/options).

Relational schema. Regarding the relational schema, it is easy to show that we have different options and the choice hugely impacts the performance [SA10]. We identify the most used ones in the state-of-the-art of RDF processing: (i) Single Statement Table schema (ST) stores RDF triples in a single table with three columns that represent the three components of the RDF triple, i.e., subject, predicate, and object; (ii) Vertically Partitioned Tables (VT) store RDF triples into tables of two columns (subject, object) for each unique property in the RDF dataset. Finally, (iii) the Property Tables (PT) schema requires combining multiple RDF properties as n-ary columns table for the same subject to group entities that are similar in structure.

Partitioning. Big Data platforms are designed to scale horizontally and, thus, data partitioning is another relevant dimension (**R.2**). However, choosing the right partitioning technique for RDF data is non-trivial, and also greatly impacts the performance. We selected the three techniques that can be applied directly to RDF graphs while being mapped to a relational schema. Namely, (i) Horizontal Partitioning (HP) randomly divides data evenly on the number of machines in the cluster, i.e., n equally sized chunks, where n is the number of machines in the cluster. (ii) Subject-Based Partitioning (SBP) (or (iii)

Predicate-Based Partitioning (PBP)) distributes triples to the various partitions according to the hash value computed for the subjects (predicates). As a result, all the triples with the same subject (predicate) reside on the same partition. Notably, Both SBP and PBP may suffer from fairly severe skew problems and/or lack of parallelism opportunities depending on the nature of the data.

Storage Data Formats. Serializing RDF data also offers many options such as RDF/XML, Turtle, JSON-LD, to name a few. On the other side, BD platforms offer many options for reading/writing to various file formats as well as storage backends. Therefore, we need to consider the variety of storage formats. Those that are relevant for our case study are the various Hadoop Distributed File System (HDFS) file formats. In particular, HDFS supports the following row-oriented formats (e.g., CSV and Avro) and columnar formats (e.g., ORC and Parquet).

Figure 15 shows the experimental solution space of experimental choices (options/-parameters) along side the three mentioned experimental dimensions (i.e., Schema, Partitioning, and Storage).

The BD System Subject to Processing. Outside the broad landscape of relational BD frameworks, we stick with *Apache Spark* which is currently the most active and widely-used large-scale data processing system in both industry and academia [Mic15; BQ18]. In particular, *Spark-SQL* offers a prominent relational interface for implementing the relational schemas and answering translated SPARQL queries (i.e., into SQL). Moreover, Spark-SQL supports different partitioning techniques and multiple storage formats. Last but not least, several works in the literature use Spark-SQL for querying and processing large RDF graphs [Sch+16; CFL18; MAA18; Rag+20].

Table 2 summarizes how we consider the four defined requirements in terms of design decisions, benchmarks (datasets and workloads), and opted BD system subject to experiments.

3.2. Experiments Design

In this section, we explain the experimental design used in this thesis. In particular, we mention the used RDF graph benchmarks, and specifications of their query workloads. We described the process of modelling graphs on top of the relational representations, alongside the partitioning and physical storage of the logical relational layouts.

3.2.1. Benchmark Datasets & Query Workloads

Seeking simplicity (**R.3**), we tested our experiments with two RDF benchmarks, namely *SP^{2}B* [Sch+08], and *Watdiv* [Alu+14]. Both are simple and well-known *synthetic* RDF benchmarks that come with scalable data generators (**R.2**). They are accompanied by sufficiently complex workloads to test the designed solution with different dataset sizes. Moreover, they are centered around simple real-world use cases (**R.3**), academic computer science digital library (*DBLP*), and *online-shop* scenarios for the SP^{2}B and WatDiv, respectively.

On the first hand, the SP^{2}B provides good coverage of the query operators and various RDF access patterns [Sal+19; Sal+16]. Indeed, its queries cover the majority of SPARQL constructs [Sal+16; Bon+18]. Moreover, it also has a reasonable average number of BGPs and triple patterns compared to other SPARQL benchmarks. In addition, SP^{2}B has the lowest mean *selectivity* values of triples, which is required for challenging the relational engine (e.g., Spark-SQL) [Sal+16]. On another hand, with the goal of seeking a variety of

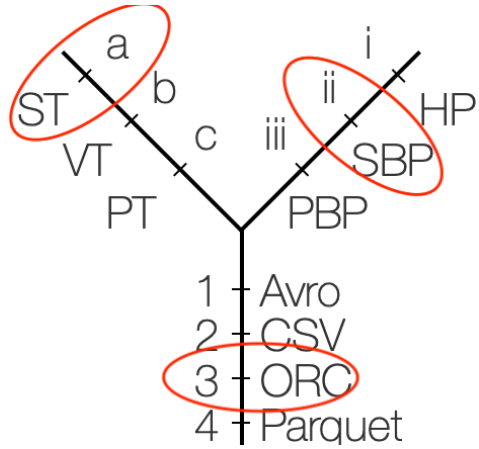


Figure 15: Experimental solution space highlighting an example of a *configuration* (a.ii.3) akin to a combination of the experimental options: (ST schema, SBP partitioning, and ORC file format).

query shapes, we included the WatDiv benchmark in our experiments. WatDiv test queries are more *diverse* within this basic fragment of SPARQL (BGPs) [Alu+14; Sal+19]. Indeed, WatDiv covers more variety of query shapes than in the SP^{2}B benchmark (i.e., *Stars* and *Snow-Flake*) by including other query shapes (i.e., *Linear*, and *Complex*). Table 3 shows the two benchmarks' query complexities in terms of number of joins, number of filters, and number of projections. This table shows how query complexity values can significantly change adapting the same SPARQL query for different relational schemas. Note that, number of projections do not change by SPARQL-to-SQL mappings.

Queries: The SP^{2}B SQL translations of the SPARQL queries are already available[1] on the benchmark page, making the experimentation portable, and reducing the risk of erroneous translation. For the WatDiv, we translated the SPARQL query templates provided by the benchmark (20 queries [2]) into SQL for the ST and PT schema (can be seen in our GitHub repository [3]). For the VT schema, we used the VT-SQL translation provided in [Sch+16]. We checked the correctness of translated SQL queries in WatDiv against the results of the corresponding SPARQL queries over the same dataset, and the results conform. Last but not least, we provide the SQL translations for all the relational schemas for both of the benchmarks on our mentioned repository seeking portability aspects.

To give an indication of the query complexity, we looked at the following query features, i.e., *number of joins*, *number of filters*, and the *number of projected variables*. Table 3 summarizes these complexity measures for SP^{2}B and WatDiv queries in SPARQL, and for the SQL translations of the corresponding RDF relational schemas. For instance, we use the number of *variable projections* in the SQL statements as an indicator for the performance comparison between the data formats of the storage backends in terms of being *row-oriented* (e.g., Avro) or *columnar-oriented* (e.g., Parquet or ORC).

[1] `http://dbis.informatik.uni-freiburg.de/index.php?project=SP2B/translations.html`

[2] `https://dsg.uwaterloo.ca/watdiv/basic-testing.shtml`

[3] `https://github.com/DataSystemsGroupUT/SPARKSQLRDFBenchmarking/tree/master/ProjectSourceCode/src/main/scala/ee/ut/cs/bigdata/watdiv/querying/queries`

WatDiv										
Query	SPARQL				ST-SQL		VT-SQL		PT-SQL	
	Shape	#J	#F	#P	#J	#F	#J	#F	#J	#F
C1	C	7	0	4	7	8	7	0	6	4
C2	C	9	1	4	9	11	9	1	9	5
C3	C	5	0	1	5	6	5	0	2	4
F1	SF	5	2	5	5	8	5	2	3	3
F2	SF	7	1	7	7	9	7	1	2	3
F3	SF	5	1	6	5	7	5	1	4	4
F4	SF	8	2	8	8	11	8	2	4	4
F5	SF	5	1	6	5	7	5	1	3	1
L1	L	2	1	3	2	3	2	1	2	2
L2	L	2	2	2	2	5	2	2	2	2
L3	L	1	1	2	1	3	1	1	1	1
L4	L	1	1	2	1	3	1	1	1	2
L5	L	2	1	3	2	4	2	1	1	2
S1	S	8	1	9	8	10	8	1	3	3
S2	S	3	2	3	3	6	3	2	1	4
S3	S	3	1	4	3	5	3	1	1	3
S4	S	3	2	3	3	6	3	2	1	3
S5	S	3	2	3	3	6	3	2	1	4
S6	S	2	1	3	2	4	2	1	1	2
S7	S	2	1	3	2	4	2	1	1	2
SP2Bench										
Q1	S	3	0	1	2	4	2	2	2	2
Q2	S	8	0	10	9	9	9	0	8	5
Q3	S	1	1	1	1	3	1	2	2	2
Q4	SF	7	1	2	7	9	7	3	8	3
Q5	SF	5	1	2	5	7	5	3	7	3
Q6	SF	8	3	2	8	12	8	3	6	2
Q7	SF	12	2	1	12	14	12	5	4	0
Q8	SF	10	2	1	9	13	9	7	6	4
Q9	S(U)	3	0	1	1	4	1	3	n/a	n/a
Q10	TP(U)	0	0	2	0	1	0	2	5	2
Q11	TP	0	0	1	0	1	0	0	2	0

Table 3: SP^2B & WatDiv queries (SPARQL/SQL) complexity characteristics: number(#) of [J]oins, [P]rojections, and [F]ilters , Shape, i.e., [S]tar, [S]now[F]lake, or a single [T]riple[P]attern; (U)nbounded Predicate Variable, w.r.t schemas (ST, VT, PT).

3.2.2. Experimental Setup and Evaluation Environment

In this section, we describe our experimental environment. In particular, (i) we discuss how we configured our experimental hardware and software components; (ii) we describe how we prepared, partitioned, and stored the datasets, and (iii) we present the details of the design of the experiment.

Hardware and Software Configurations. Our experiments have been executed on a *bare-metal* cluster of *four* machines with a *CentOS-Linux* V7 OS, running on a *32-AMD* cores per node processors, and 128 GB of memory per node, alongside a high speed 2 TB *SSD* drive as the data drive on each node. We used Spark V2.4 to fully support Spark-SQL

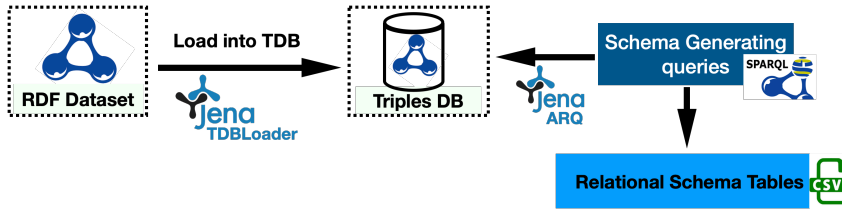


Figure 16: RDF relational Schema generation process.

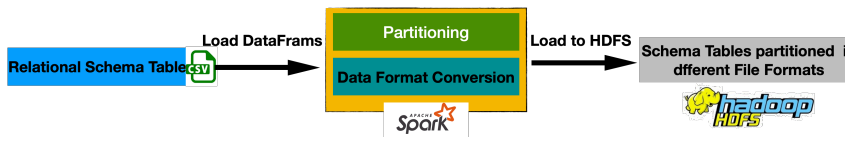


Figure 17: Process of partitioning and storage formats conversion via Spark.

capabilities. We used Hive *V3.2.1*. In particular, our Spark cluster consists of *one* master node and *three* worker machines, while *Yarn* is used as the *resource manager*, which in total uses 330 GB and 84 virtual processing cores.

Data Storage: We generated *synthetic* RDF datasets in the `Notation3` (*n3*) native RDF serialization using the benchmarks' generators. The generated *n3* RDF datasets are converted into CSV relational schemas using *Jena TDB* [4], a disk-based access repository for storing RDF datasets. We further use the *Jena ARQ* [5] for querying these TDB datasets and generating the output schemas tables in the CSV (see Figure 16). Finally, these raw textual CSV documents are loaded to the HDFS. Moreover, we have used the Spark framework to convert the relational schemas data tables from the CSV format into the other HDFS file formats (i.e., Avro, Parquet, and ORC). Spark is also used for partitioning schema tables and loading them into HDFS (See Figure 17).

Data Partitioning. In *Preliminaries* section, we describe the partitioning techniques we selected, i.e., HP, SBP, and PBP. Partitioning impacts data distribution and thus, Spark-SQL performance is greatly affected, especially reading from data backends and data-joining operations. Therefore, when partitioning is required, the goal is to minimize *data shuffling* to the minimum. One should select the technique that best suits the query workload. The mentioned partitioning techniques were originally designed for RDF partitioning. Hence, we defined their equivalent version for tabular RDF representation. In Spark-SQL, join operations require combining two or more *DataFrames* based on certain (*sortable*) keys. That means that the joined data must be transferred to the same node. Thus, we prepared the data in two phases.

First, we use custom Spark partitioners for creating DataFrames that fulfill a certain partitioning technique. Depending on the partitioning techniques of choice (i.e., SBP, PBP), we used as a partitioning key the subject or the predicate of the triples, respectively. For the Horizontal approach, we re-partition the DataFrames into a specific number of partitions according to the number of cores in the machines in our cluster (see Figure 17). Then, we persisted the DataFrames on HDFS (i.e., kept partitioned for later subsequent experiments). We fixed the data partition block size on HDFS as the default block size on Spark (128*MB*). HDFS manages also the replication of these partitioned blocks according

[4] `https://github.com/apache/jena/tree/master/jena-tdb`
[5] `https://github.com/apache/jena/tree/master/jena-arq`

		Format	100M	250M	500M		100M	250M	500M
Sp^{2}B	ST	CSV	13GB	31GB	60GB	watDiv	13.8GB	32.1 G	65.8 G
		Avro	~2GB	~5GB	11GB		1.8GB	4.0 G	8.3 G
		ORC	1.3GB	3.2GB	6.3GB		~1GB	2.6 G	5.4 G
		Parquet	1.7GB	~4GB	~8GB		1.3GB	3.1 G	6.6 G
	VP	CSV	~8.3GB	21GB	41GB		6.1 G	14.3 G	29.4 G
		Avro	~1.7GB	~4.2GB	8.3GB		841.6 MB	1.8 G	3.8 G
		ORC	~1.5GB	~3.8GB	7.3GB		638MB	1.5 G	3.1 G
		Parquet	~1.6GB	4GB	7.8GB		750MB	1.7 G	3.6 G
	PT	CSV	~6.8GB	36GB	33GB		6.2 G	14.4 G	29.6 G
		Avro	1.6GB	6.2GB	7.5GB		1.0 G	2.3 G	4.8 G
		ORC	1.4GB	5.5GB	6.5GB		800.9 M	1.8 G	3.8 G
		Parquet	1.4GB	5.8GB	7GB		908.9 M	2.1 G	4.4 G

Table 4: Dataset sizes w.r.t schemas in different storage file formats.

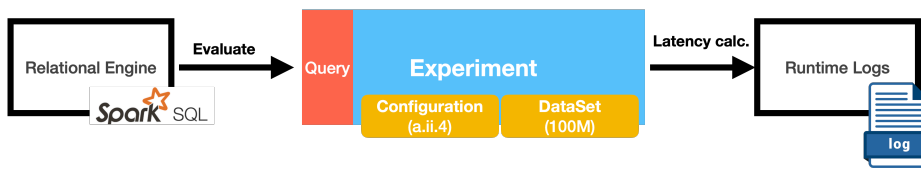


Figure 18: Experiment design and workflow in our scenario.

to a configurable *replication factor(RF)* (i.e. we used the default $RF = 3$).

It is worth mentioning that, partitioning the PT tables by predicates (i.e., using PBP technique) required splitting the tables into two columns tables/DataFrames *(S, P)*. Those DataFrames must be joined in the memory before executing queries against those tables. On the other side, we depend only on the vanilla HDFS partitioning for the PBP technique for the VP tables schema.

Performance Evaluation measure (Latency): We use the *Spark.time* function by passing the *spark.sql()* query execution function as a parameter to measure the query *latency*. An experiment is defined as evaluating a query against a configuration combined of: a relational schema, partitioned, and stored using one of the HDFS data formats (as shown in Figure 18). We run the experiments for all queries *five* times. As a best practice, we exclude the *first cold start* run time, to avoid the *warm-up* bias, and computing an average of the other *four* run times.

3.3. Results and Performance Analysis

Table 5 shows part of the experiments results of running SP2B queries over the 500M dataset and across various configurations [6]. For each dataset size, we have a similar table of results. Each query evaluation has a latency according to the underlying used configuration (Schema, partitioning, and Storage format).

To analyze these huge results in complex scenarios like querying large RDF graphs, we advocate the need for a decision-making framework for making sense of the performance of BD systems. Different levels of analysis are shown in detail in the following sections.

[6]Note: We keep the full results and plots of the benchmarks experiments in the GitHub project repository.

Configuration	*Q1*	*Q2*	*Q3*	*Q4*	...	*Q10*	*Q11*
a.i.1	80.4	270.1	42.5	2688.5	...	28.5	17.9
a.i.2	208.9	793.1	139.2	2891.5	...	61	68.7
...	...	...			...	...	...
b.iii.3	28.9	107.9	7.1	2381.5	...	2.2	1.7
...	...	...			...	...	...
c.iii.4	123	236.7	53.9	2469.1	...	148.6	110.1

Table 5: Spark-SQL query evaluation average run times (*in seconds*) over configurations (SP^2B 500M dataset).

Query	100M		250M		500M	
	BEST	**WORST**	**BEST**	**WORST**	**BEST**	**WORST**
Q1	**c.i.2**	**c.iii.4**	**c.ii.2**	**a.i.2**	**c.ii.2**	**a.i.2**
Q2	**b.ii.3**	**a.i.2**	**b.ii.4**	**a.i.2**	**b.iii.4**	**a.i.2**
Q3	**c.ii.3**	**a.i.2**	**c.ii.4**	**a.i.2**	**c.ii.3**	**a.i.2**
Q4	**a.ii.3**	**b.iii.3**	**c.ii.1**	**a.i.2**	**a.ii.3**	**a.i.2**
Q5	**b.ii.4**	**a.i.2**	**b.ii.4**	**a.i.2**	**b.iii.3**	**a.i.2**
Q6	**c.ii.3**	**a.iii.2**	**c.ii.3**	**a.iii.2**	**c.ii.3**	**a.i.2**
Q7	**b.ii.1**	**c.iii.2**	**b.ii.4**	**c.iii.2**	**b.ii.4**	**c.iii.2**
Q8	**c.iii.4**	**a.iii.4**	**c.iii.4**	**a.iii.2**	**c.iii.4**	**a.i.2**
Q9	**b.ii.4**	**c.i.2**	**b.iii.3**	**c.i.2**	**b.iii.4**	**c.i.2**
Q10	**b.iii.3**	**c.iii.4**	**b.iii.3**	**c.iii.4**	**b.iii.3**	**c.iii.2**
Q11	**b.i.3**	**c.iii.4**	**b.ii.4**	**c.iii.4**	**b.i.3**	**c.iii.2**

Table 6: Best and Worst configurations for the SP2Bench queries across datasets.

First, the *Descriptive* analysis level enables answering factual questions regarding the performance. At this level of analysis, we can extrapolate *fine-grained* insights, e.g., what is happening at the query evaluation level. In particular, we use *descriptive* analysis to identify which queries are long-running, medium-running, or short-running according to their average run times. For each experiment, we can observe which schema, partitioning technique, and storage backend are performing the best or the worst.

The descriptive analysis is usually complemented by *diagnostic* analysis that allows answering the *why* questions regarding the performance phenomena. At this level, we combine factual knowledge from the observed data with domain knowledge to make sense of the performance results. We can enrich the descriptive analyses mentioned above with contextual information about the query complexity and the configurations' attributes.

3.3.1. Descriptive and Diagnostic Analyses' Limitations

Descriptive performance analysis of the results of the experiments cannot provide final actionable answers for the practitioners guiding them to the best-performing configurations [Rag20; RAT21]. Similarly, diagnostic analysis can help understand why a specific dimension is outperforming. For instance, the practitioner can understand why a specific schema *X* outperforms another schema *Y* because the number of join operations entailed

Query	100M		250M		500M	
	BEST	**WORST**	**BEST**	**WORST**	**BEST**	**WORST**
C1	**c.ii.2**	**c.iii.2**	**c.ii.2**	**a.i.2**	**c.ii.2**	**a.i.2**
C2	**b.i.4**	**a.i.2**	**b.i.4**	**a.i.2**	**b.ii.2**	**a.i.2**
C3	**c.ii.4**	**a.i.2**	**c.ii.3**	**a.i.2**	**c.ii.4**	**a.i.2**
F1	**c.ii.2**	**a.i.2**	**c.ii.1**	**a.i.2**	**c.ii.3**	**a.i.2**
F2	**c.i.2**	**a.i.2**	**c.ii.3**	**a.i.2**	**c.ii.1**	**a.i.2**
F3	**b.ii.2**	**a.i.2**	**c.ii.1**	**a.i.2**	**c.ii.4**	**a.i.2**
F4	**c.i.2**	**a.i.2**	**c.ii.3**	**a.i.2**	**c.ii.4**	**a.i.2**
F5	**b.ii.2**	**a.i.2**	**c.ii.2**	**a.i.2**	**c.ii.3**	**a.i.2**
L1	**c.ii.3**	**a.i.2**	**c.ii.1**	**a.i.2**	**c.ii.3**	**a.i.2**
L2	**b.iii.3**	**a.i.2**	**b.iii.2**	**a.i.2**	**b.ii.4**	**a.i.2**
L3	**b.ii.2**	**a.ii.2**	**b.ii.4**	**a.i.2**	**c.ii.4**	**a.i.2**
L4	**b.iii.3**	**c.iii.2**	**b.i.3**	**c.iii.1**	**b.i.4**	**a.i.2**
L5	**c.ii.2**	**a.i.2**	**c.i.4**	**a.i.2**	**c.ii.4**	**a.i.2**
S1	**c.ii.2**	**a.i.2**	**c.i.3**	**a.i.2**	**c.ii.4**	**a.i.2**
S2	**b.iii.2**	**a.i.2**	**c.i.4**	**a.i.2**	**c.ii.3**	**a.i.2**
S3	**c.i.2**	**c.iii.2**	**c.ii.2**	**a.i.2**	**c.i.3**	**a.i.2**
S4	**c.ii.4**	**c.iii.3**	**c.ii.4**	**c.iii.3**	**c.i.4**	**a.i.2**
S5	**c.ii.1**	**c.iii.3**	**c.i.4**	**a.i.2**	**c.ii.4**	**a.i.2**
S6	**c.ii.1**	**c.iii.3**	**c.i.3**	**a.i.2**	**c.ii.3**	**a.i.2**
S7	**c.ii.2**	**c.iii.4**	**c.ii.2**	**c.iii.4**	**c.ii.3**	**a.i.2**

Table 7: Best and Worst configurations for WatDiv queries across datasets.

with querying X is less than in Y or data accessed in X tables is less than in Y. However, due to the inherent *trade-offs* among the experimental dimensions, contradictions still hinder clear decisions at this level of analysis [Rag20; RAT21].

Table 6 and Table 7 show the best and worst-performing configurations for each query in different data scales for SP2Bench, and WatDiv respectively. The experimental results over different settings show *no decisive* configuration over the assessed dimensions, making the practitioner's setup selection a complex task. That is, although we can see an agreement on the worst-performing configurations in both benchmarks (Tables 6 and 7), we cannot decide accurately on the best-performing configurations across queries and across datasets.

Results of both benchmarks show that scaling up to larger datasets does not guarantee decisive best-performing configurations. Indeed, most of the queries show different configurations while moving across datasets. To name a few examples from the SP2B, all queries (except for queries Q6, Q8, and Q10) show different configurations moving from the 100M dataset to the 250M and 500M datasets. To give the intuition, Figure 19 shows an example of disagreements on the best-performing configurations scaling from 100M to 250M in the SP2B benchmark, the distance line (in red) shows the level of disagreement according to the number of non-matching dimensions (i.e., query best configuration contradicts in *one*(e.g., Q1, and Q2, Q3, Q7, Q9, and Q11), *two* (e.g., Q4, and Q11) or all *three* dimensions). The green configurations in the middle of the figure represent the

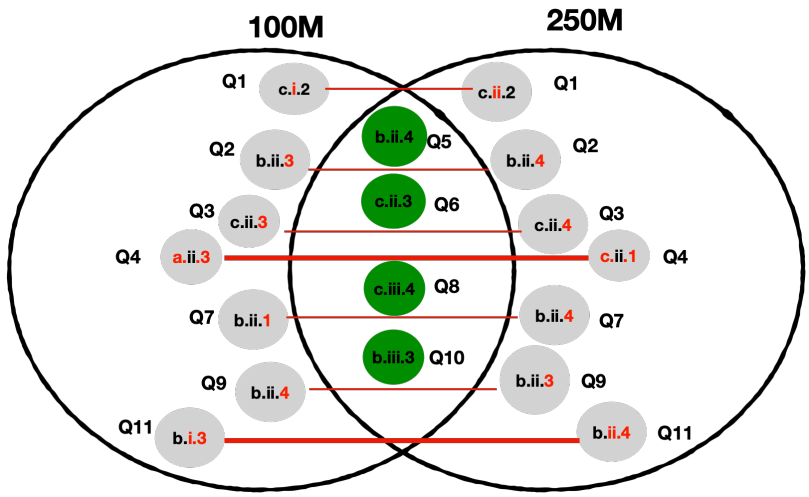


Figure 19: Contradicting best-performing configurations across datasets (SP2Bench 100M to 250M datasets).

agreements when scaling up from the 100M to the 250M-dataset. The same observation is also valid with the *WatDiv* dataset results. All WatDiv queries (except for one query,i.e., $C1$) show different configurations moving from the 100M dataset to the larger datasets (i.e., 100M and 250M), See Table 7.

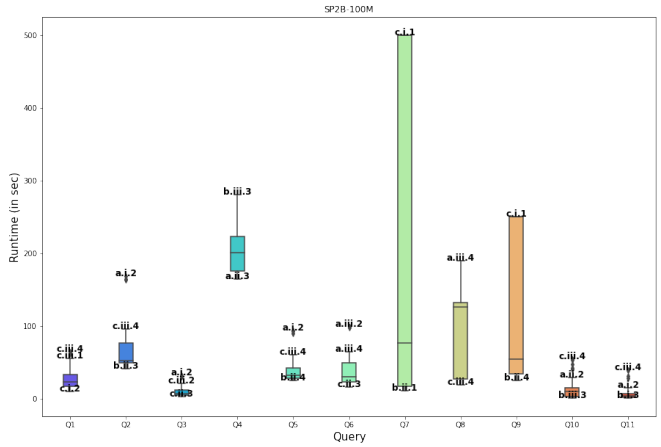
Moreover, moving to the level of queries, we can see inconsistency in the best and worst configurations. A specific dimension option can show the best performance in some queries while showing the worst performance in those queries changing other dimensions [Rag20; Rag+20]. Thus, looking at one dimension at a time might not lead to correct decisions. The efficient performance of a specific schema, partitioning technique, or storage format can be significantly degraded (even to be the worst-performing) when it is used with non-performing other dimensions for query specifications [Rag20; Rag+20; RAT21].

First, we show examples of these observations from the SP2B Benchmark. For instance, changing the partitioning and storage formats can affect the efficient performance of the relational schema. As an example, $Q1$ in the 100M dataset, the best performing schema (PT) in the configuration (best=(c.**i.2**)), becomes the worst performing (c.**iii.4**) changing the partitioning technique and storage format (see Figure 20 (a)). Also, the ST schema shows the best and worst performance in Q4 in the 500M dataset, (best=(a.ii.3), worst=(a.i.2)) (see Figure 20 (c)).

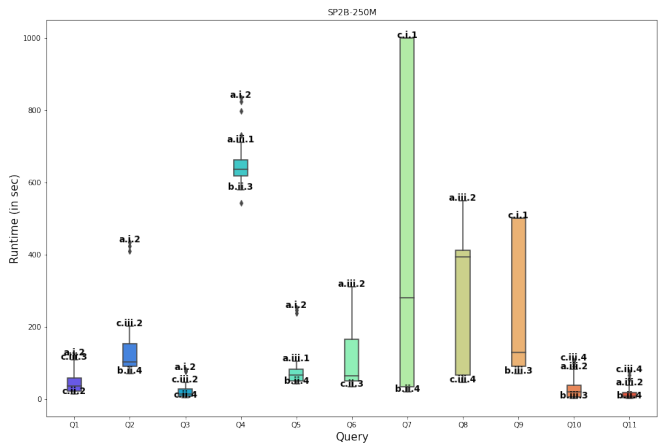
Similarly, changing the schema and storage dimensions degrades the performance of the best-performing partitioning technique in $Q10$ in the 100M and 250M, and 500M experiments (Figure 20 (a), (b), and (c), respectively. For example, Q10 in the 100M and 250M dataset, best= **b**.iii.**3** to worst= **c**.iii.**4**). The same effect occurs in $Q8$ in 250M dataset in Figure 20 (b), (best= **c**.iii.**4** to worst= **a**.iii.**2**).

Last but not least, changing the schema and partitioning dimensions affects the performance of the storage format like in queries Q4 (100M), and Q11 (250M), see Figures 20 (a), (b), respectively.

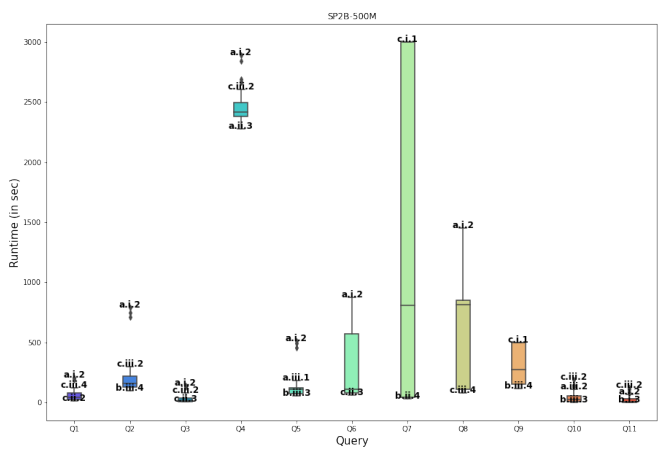
On the second hand, the performance of two well-performing dimensions can be affected by introducing a different third dimension. For example, Figure 20 (a) shows query



(a) (100M-SP2B)



(b) (250M-SP2B)



(c) (500M-SP2B)

Figure 20: SP2B queries with best(worst)-performing configurations.

Q8 (100M dataset) with the partitioning and storage performance (i.e. PBP technique and ORC format) in the best-performing configuration (best=(**c**.iii.4)) is affected to become the worst changing the schema to ST, i.e. (worst=(**a**.iii.4)).

Second, WatDiv results confirm the above observations. Although, there is high agreement on the worst-performing configurations across queries (especially with scaling up to the larger datasets 250M, and 500M) see Table 7. However, we still see contradictions in the best-performing configurations.

The effect of changing the storage formats and partitioning on degrading the efficiency of schema is still valid. For instance, in queries S5 and S6 in the 100M dataset (CF. Table 7), the best performing schema PT (best=(c.**ii.1**)) turns to perform the worst by changing the partitioning technique and the storage format (worst=(c.**iii.3**)). The same effect occurs with queries S4, and S7 in the 100M and 250M datasets, respectively (see Figures 21 (a), (b)).

Similarly, changing the schema and storage degrades the partitioning performance in 100M experiments (queries C2, L4), 250M dataset (queries C2, L5, S2, S5, and S6), and in 500M dataset experiments (queries L4, S3), see in Figures 21 (a-c).

Last but not least, the effect of changing the schema and partitioning on the efficiency of format dimension is shown in queries F1, F4, F5, L5, S1, S2, S3 (in the 100M dataset), C1, F5, L2 (250M experiments), and C1 and C2 (in the 500M dataset experiments), see in Figures 21 (a-c).

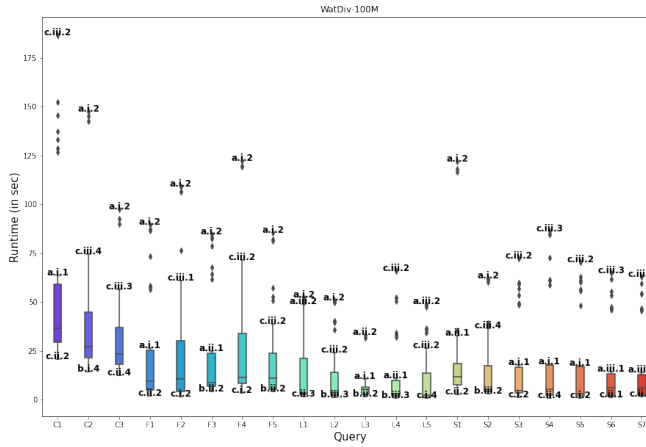
Changing even one dimension can affect the performance of the other two dimensions. For example, in queries C1 of the 100M dataset, the best performing configuration (best= c.*ii*.2) with using the PT alongside the CSV format turns to be the worst by changing the partitioning technique (i.e., worst= c.**iii**.2). Similar effect is observed in query S3 of the 100M dataset, i.e., (best=(c.**i**.2), worst=(c.**iii**.2)), see (Figure 21 (a)). On the same note, changing the schema also can affect the efficiency of the partitioning and storage formats like in queries F2, F4, and L3 in the 100M dataset(CF. (Figure 21 (a))).

Notably, these few examples represent the most extreme case when for the same query, changing one or two dimensions turns the best-performing dimension(s) to provide the absolute worst-performing. That is, there exist some other cases in which changing dimensions in well-performing configurations (not necessarily the best) significantly degrades their performance (not necessarily to the worst).

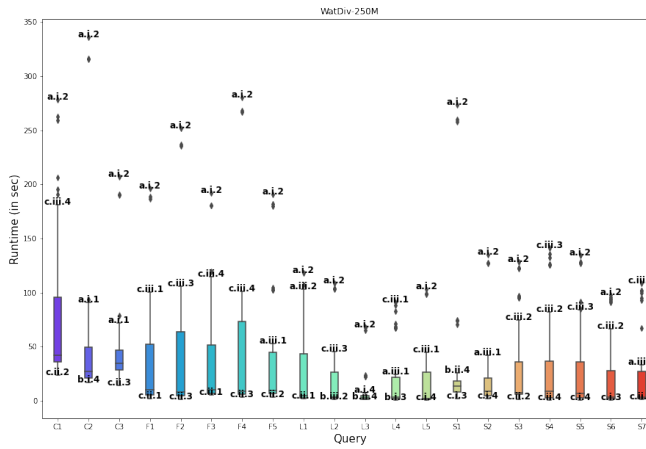
Impact of the Query Workload: What descriptive analysis can also show is the impact of the query workload on the system latency performance with the presence of configurations and datasets. Each query has its own complexity and data selectivity features. Thus, the impact of these queries on the runtimes should differ according to those features and how they change with the underlying configurations.

Figures 22 (*a*) and (*b*) quantitatively show the impact of certain query on the system's performance with the presence of configurations in the SP^{2}B and WatDiv benchmark datasets, respectively. For conciseness, we show the plots of the largest dataset in both of them (i.e., 500*M*). For all of these figures, the reading key is *the lower is the better*. We indicate that a particular configuration outperforms another in a specific query when it takes less execution time.

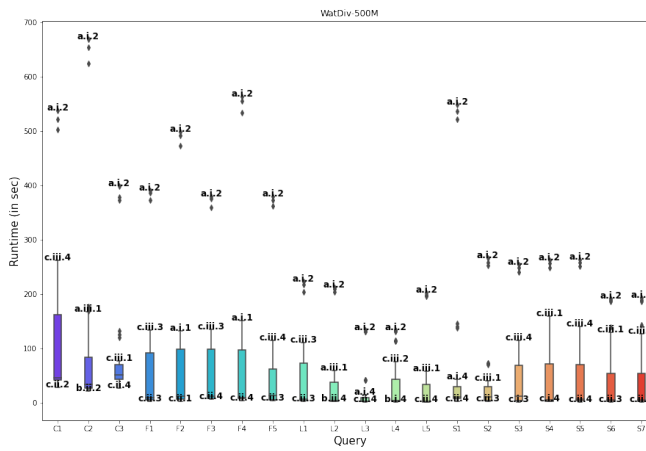
First, we show the results of the queries in SP^{2}B benchmark. We can categorize the queries into *short*, *medium*, and *long*-running queries. For instance, queries ($Q1$, $Q3$, $Q5$, $Q10$, and $Q11$) are the short-running ones. Whereas, queries (Q2, Q5, Q6, Q9) are medium-running queries. Last but not least, queries ($Q4$, $Q7$, $Q8$) are categorized as long-running queries. The SP^{2}B figure also shows that $Q4$ is the query with the highest



(a) (100M-WatDiv)

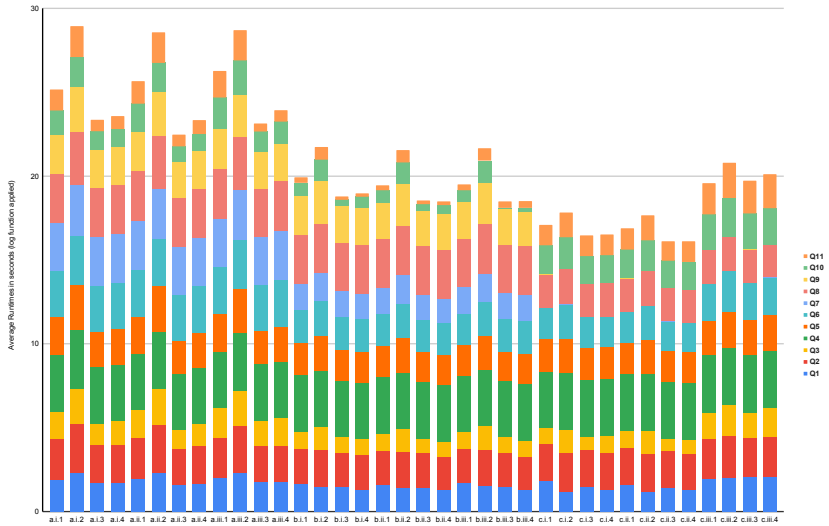


(b) (250M-WatDiv)

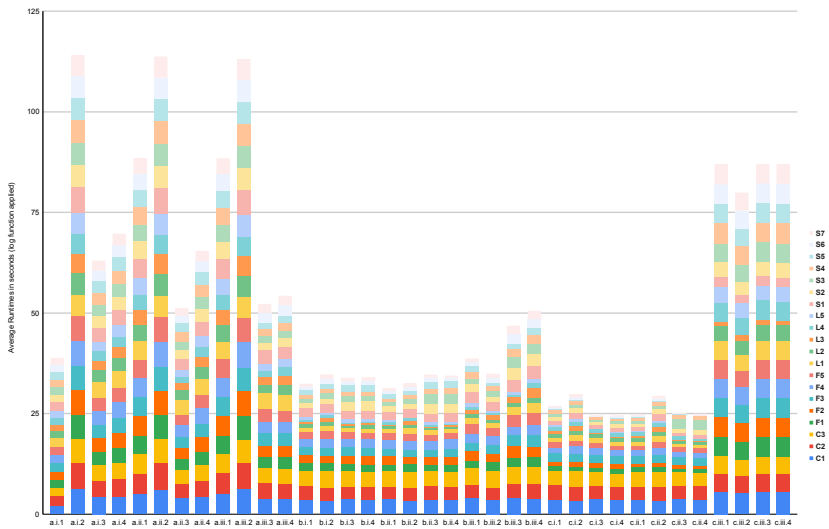


(c) (500M-WatDiv)

Figure 21: WatDiv queries with best(worst)-performing configurations.



(a) (500M-SP2B)



(b) (500M-WatDiv)

Figure 22: Configurations Query Performance for the SP2B and WatDiv queries.

latency (i.e., it takes the highest running times across all the configurations), and $Q8$ which interestingly impacts only the *ST* and *VT* schemas, taking long running times, while having very low running times with the PT schema.

Second, the WatDiv figure shows that the queries of the shape Complex (i.e., $C1$,$C2$, $C3$) are the long-running ones across all of the configurations. The rest of the queries in the other *three* query shapes (i.e., Snow-Flake, Linear, and Star) are ranging from medium

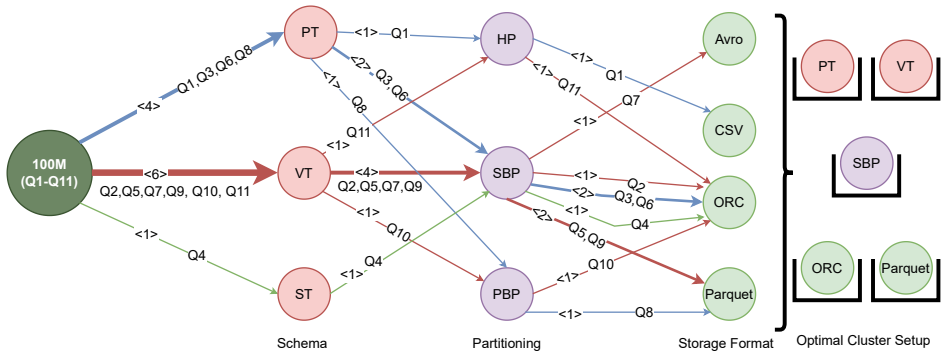


Figure 23: Descriptive analysis may lead to different cluster deployments.

to short-running, across all the configurations.

Interestingly, the performance of the system to evaluate a graph query depends not only on the query complexity features but also according to the experimental configurations. For instance, the complex query $C3$ has the highest impact on the system execution. Indeed, it incurs several triple patterns with a significant low selectivity, thus including high intermediate results that give the required high selectivity results. However, we can notice that its impact clearly aggravates with using the VT schema (with all partitioning techniques and storage formats), and with the PT schema (with the HP and SBP partitioning techniques). However, its impact is significantly reduced with the PT schema using the PBP partitioning technique and with all storage data formats. We have the same observations with the complex query $C2$ (i.e., it shows high impact with the ST and VT schemas, and low impact with the PT schema). On the same note, the complex query $C1$ has a high impact across all the configuration combinations, with a slight low impact (better performance) with the ST schema across its two other dimensions (partitioning and storage formats).

In the line-shaped queries (*Watdiv-L*), we can observe that the PT schema shows a remarkable competitive performance, especially with the HP and SBP partitioning techniques. The VT schema performance directly follows for those queries.

We observed the same note of the PT schema efficient performance (alongside HP and SBP) for the *Watdiv-F* and *Watdiv-S* queries with the shapes "Snow-Flake", and "Star", respectively. In particular, these queries have a significant lower impact with these configurations of PT alongside HP and SBP. However, PT schema with the PBP partitioning has a significant *negative* performance effect on those query shapes. Whereas, this PBP technique with the PT schema interestingly has a significant positive performance impact with the $S1$, and $L3$ queries (i.e., significantly reducing their execution times).

These results affirm the limitations of relying only on descriptive and diagnostic analyses to select efficient configurations from the experimental solution space. Indeed, in such a complex solution space of experimental dimensions and big datasets, getting accurate final options from the descriptive analysis is by nature a hard and time-consuming task.

3.3.2. Performance Complexity Issues

Moreover, Figure 23 shows that the descriptive analyses of the performance results can lead to different cluster deployments with huge redundancy of the data. That is, the configurations that lead to the optimal performance of the system (optimizing the majority of

the queries) may require replicating the data in different schema, partitioned in different techniques, and physically stored in multiple storage file formats on disk. For instance in Figure 23, the best setup that guarantees the best performance of the whole query workload (i.e., generally cover the best-performance for the majority of the queries) requires replicating the data in two RDF relational schema (i.e., PT, and VT), partitioning them using one technique by subject (i.e., SBP), and physically replicating the storage on disk using the best-performing two storage formats (i.e., Parquet, and ORC). Figure shows that how each query is directed to the best-performing configuration options across the figure paths. The thickness of the paths depicts the importance (i.e., the more queries the configuration option covers, i.e., makes it take the minimum latency). For example, VT is in generally very good option according the coverage ratio (6 out of 11 queries).

Intuitively, this replication enhances the performance the best, as the query execution will be directed to the optimal cluster setup of configurations (schema, partitioning, and storage). However, this significantly complicates the *data-warehousing* solutions of processing large graphs on top of relational systems. Besides, wasting huge storage out of the data replication, it requires sophisticated comprehensive data engineering efforts for maintaining data in different logical layouts and physical storage, additionally and most importantly, adapting the query workload for those layouts (i.e., query translation is a schema-dependent task).

3.3.3. Can we make sense of performance results?

Now, we aim at making sense of the performance of Spark-SQL providing some reasoning on the performance phenomena. To this end, we aim at incorporating the domain knowledge about the query workload complexity features, as well as the configuration dimensions characteristics in order to understand the previously shown descriptive *fine-grained* results.

To this end we list a set of hypotheses from the domain knowledge related to the query workload and experimental dimensions (summarized in Table 8):

1. **Schema.** we expect the schema that reduces the number of joins in a query or the one that reduces the input table sizes to be the best-performing one.
2. **Partitioning.** we expect the partitioning technique that achieves the data load balance while not ignoring efficient data locality to be the best-performing.
3. **Storage.** we expect the storage format that optimizes the compression of the data to work better. We also expect that the row-oriented formats are more suitable for the query workload with high projections. Whereas, the columnar formats are more suitable for workloads with few projections.

First, we show the diagnostic analysis over the SP^2B results. In the majority of the results, we can notice that the *ST* is the worst-performing schema. Indeed, it is the schema that requires the maximum number of *self-joins* alongside several *Filters* over the *Predicate* and *Object* column values. Moreover, ST schemas single table is the largest table even after partitioning and thus takes longer execution times. Whereas, the VT schema is mostly the best-performing schema. The reason behind this is that VT tables tend to be relatively smaller in size than other relational schema tables. Thus, Spark query joins have smaller intermediate results entailed in the shuffle operations. The PT schema is yet a competitor to the VT schema since it requires the minimum number of joins and filters (compared to the other two schemas) while translating SPARQL into SQL (C.F., Table 3).

Dimension	When performs well ?
Schema	- reduces the number of query joins. - reduces the the input table sizes.
Partitioning	- achieves the data load balance. - considers efficient data locality.
Storage Format	- optimizes the compression of the data. - row-oriented more suitable to workloads of high projections. - columnar more suitable to workloads of few projections.

Table 8: Domain knowledge related to our experimental dimensions.

However, the PT schema joins are less than in the VT schema queries, the VT schema is generally outperforming the PT schema. The reason behind this is that the size of the binary VT table is less than the n-ary column PT tables. Moreover, some of the VT tables (e.g., *subClassOf*, *References*) are small in size enough to be *broad-casted* while having joins with them. The *broad-cast* joins have a significant positive performance impact in Spark-SQL joins [Mic15].

The reasoning for the query level, $Q4$ shows as the longest-running query across all configurations (C.F. Figure 22 (a)). Indeed, it entails several complex *"joins"* and *filters* in all the schemas, as well as a *negation* filter with another choke-point of duplicate elimination (i.e., DISTINCT). Moreover, it incurs a huge result set, as it touches huge data tables (e.g., Publications, Persons) [eta09]. In this query, the ST is interestingly showing better performance than the other two schemas with this query across all the partitioning techniques and storage formats. The reason is that the number of joins in the ST schema is less than in the PT schema. Nonetheless, the ST schema joins in the VT schema are equal to the ones in the ST schema, ST outperforms the VT schema due to the high number of filters in the ST schema that reduces the intermediary results less than in the VT schema. Similarly, Query $Q8$ is the second longest-running query due to its high number of joins. However, the PT schema significantly enhances the performance of this query due to the lower number of joins in PT schema compared to the other two schemas. Similarly, the complexity of $Q7$ makes it challenging for the system to evaluate in all configurations. It is the query with the highest number of join operations as well as duplicate elimination. The VT schema significantly outperforms the other schemas for evaluating this query due to several broadcast hash joins utilized. The query includes two tables that will be broad-casted to the other machines, i.e., (e.g., *subClassOf*, *References*).

Now, we move to the partitioning dimension impact discussions. The SBP partitioning is generally shown as the best-performing technique. Particularly, it directly outperforms HP, leaving the PBP technique in the worst rank. The most reasonable cause behind this is that most of the query shapes in SP^2B benchmark are stars and snow-flake which are mostly oriented to the RDF subject as the joining key. Therefore, partitioning by the subject key locates the triples or data rows with the same subject on the same machine reducing data shuffling to the minimum, and maximizing the level of parallelism by all workers. This reduces the queries' latency to the minimum. Whereas, this is not satisfied in the Horizontal-based (HP), nor in the Predicate-based PBP partitioning approaches.

Last but not least, regarding storage formats comparison, we can observe that the HDFS `columnar` file formats mostly outperform the row-oriented file formats. In particular, the *ORC* is the best performing storage format followed by *Parquet*. Whereas, *Avro* and *CSV* file formats of HDFS are the worst-performing file formats. The reason behind

these results is that most of the *SP²B* queries are with a few numbers of projections. Indeed, columnar file formats are able to perform better since they are efficient to scan only a subset of columns with filtering out unnecessary columns for the query [IP19]. On the other side, the textual uncompressed *CSV* and the *row-oriented* Avro file formats are shown to have the lowest-performing storage options, respectively.

Second, we show the diagnosis over the WatDiv benchmark results. Similar to the SP²B benchmark, the ST is shown as the worst-performing schema in most of the queries (C.F., Figure 22 (b)). This is also explained by the high number of self-joins in this schema, such in the SP²B benchmark results. Whereas, the PT schema is outperforming the VT schema in the majority of the queries. The reason behind is *two-folds*. First, the high existence of *star patterns* in the *Star* and *Snow-Flake* query shapes, (i.e., WatDiv has 12 out of 20 queries are mainly *star-shaped* (stars and snow-flakes)). Indeed, 88% of the triple patterns in the WatDiv query workload are involved in *star-shaped* joins [Bor+13]. The PT schema is able to answer those star patterns with fewer joins (with no-joins even in some star queries) than with the VT schema. The same reason is valid for the PT schema with *WatDiv-L* queries. These Linear queries consist of *three* triple patterns *two* of which can be found in the same table (i.e., User). In the PT schema, this can be achieved with a *single* join between the *User* and *Product* tables. This results in fewer stages and faster execution than the VT which entails accessing three tables and two joins in those queries. However, the VT schema shows a closer performance to the PT schema when a group of queries contains *fewer* bounded predicates. This explains why for several "L" and "S" queries VP achieves a very closer performance than the PT schema.

Regarding the partitioning dimension impact, the SBP technique (*ii*) shows the best performance in the majority of the queries. The mentioned reason above regarding the star patterns' commonality in the queries also explains this behavior that was similar to the SBP behavior with the SP²B benchmark. However, Subject-based partitioning affects the performance of the VT schema, especially with the Star patterns (in stars and snow-flakes). It gets worse in Linear queries. The reason behind this is that there is no guarantee that the join object variables in the VT tables reside in the same machine (that entails more data shuffling). The SBP technique even affects the performance of the VT schema making it perform worse than the HP partitioning technique. The HP and PBP techniques show; in the majority of the queries; the worst performance, as they do not consider locating the triples with the similar subject key in the same partition. Thus, they fail in optimizing the star-pattern queries. Moreover, the PBP partitioning tends to be the *worst-performing* technique with the PT schema. The reason is PBP technique requires a cumbersome operation of *re-partitioning* the PT tables into its predicates and *re-joining* them again in the memory while querying.

Last but not least, we explain the performance of the storage file formats in the WatDiv benchmark. Similar to the SP²B, the columnar file formats outperform the row-oriented file formats due to the low number of projections. However, the row-oriented file formats (CSV, and Avro) outperform them in several queries that have a high number of projections that reach the extent of accessing the full tables in those queries.

3.4. Best Practices for Querying Large RDF Graphs Using Relational Big Data Systems

In this section, we aim to discuss specific recommendations regarding the case study of querying large RDF graphs over a relational system, e.g., Spark-SQL (see a summary of

Dimension	Practice	Rationale
Schema	VP and PT schemas are preferred over ST schema	• ST table incur plenty of self-joins with complex workloads.
	PT schema is the best with star and snowflake SPARQL queries.	• the number of joins is reduced to the minimum.
	PT performance schema comes with a price	• The burden of designing PT schema (especially for real-world datasets)
	PT schema is not applicable for all triple patterns.	• cannot directly answer unbounded predicates.
	Reducing the number of joins in the execution plan and having a balanced distribution of the data among the cluster nodes.	• Empirically tested in the experiments and shown are more important than the size of the various input tables accessed.
Partitioning	The subject-based partitioning is generally recommended.	• Due to common SPARQL queries that include several star patterns
Storage Format	Columnar formats are more recommended (ORC and Parquet)	• Have high compression values. • Performs better with a few projections.

Figure 24: Experimental dimensions' best practices for the Spark-SQL system.

best practices in Figure 24).

First, the specific recommendations regarding the dimensions of our case study can be listed as follows:

i The Single triples table (ST) is not recommended as a relational schema for representing the RDF datasets in the relational contexts.

ii The VT and PT schema are —in general, recommended relational schema for processing large RDF datasets.

iii The PT schema can be the best option, especially with *Star* and *Snowflake* query workloads (fewer joins are incurred).

iv However, the PT schema comes with the burden of *designing* and *modelling* relational layout, this burden aggravates with *real-world* RDF datasets. Moreover, the PT schema can not directly answer queries with *unbounded* predicates (e.g., $Q9$ in the SP^2B). Thus, this gives an advantage to the deterministic and automatically designed VT schema.

v our experiments show that reducing the number of joins in the execution plan and having a balanced distribution of the data among the cluster nodes are more important than the size of the various tables accessed.

vi The subject-based partitioning is generally recommended with SPARQL queries that include several *star* patterns. Nonetheless, it is not recommended with the VT schema, especially when the query workloads are mostly *linear* or *star-shaped*.

vii On the other side, the Horizontal and Predicate-based partitioning techniques are generally less recommended. However, still better than the default vanilla HDFS partitioning.

viii According to the storage file formats, the columnar file formats are more recommended than the row-oriented file formats when the workloads come with a few numbers of column projections.

ix Specifically, *ORC* shows generally as the best-performing storage option with Spark-SQL in our experiments across the two benchmarks and scalable evaluations, followed by *Parquet*.

3.5. Discussion

To Summarize, processing large RDF graphs in a distributed relational setup is a multi-faceted complex problem, as it includes several factors. First, the query workload has different complexity features, i.e. different shapes, number of triple patterns, and various selectivity values. Relying on relational BD systems like Apache Spark-SQL raises other dimensions to consider while seeking the best performance. The relational schema, the partitioning, and storage formats are important dimensions that directly impact the performance. Each schema has a different representation of the data and thus impacts the query complexity features, e.g., number of joins and number of filters. This can directly affect the intermediate results and their sizes in the query for each schema. With the ever-growing volumes of RDF KGs, the optimal distribution in terms of data replication, as well as load balancing is getting more and more crucial in distributed and cloud computing applications [Cur+15]. Load balancing is particularly an important aspect to consider for achieving efficient data distribution for storage and querying evaluation. Relying only on the relational layouts (schemas) of representing RDF graphs, the optimal distribution of the data may be hard to achieve. In practice, some RDF relational schemas are by nature skewed, e.g., the VP schema (i.e., most of the triples can be related to a few predicates, while very few triples can be related to the majority of predicates). Therefore, in our scenario to mitigate these issues, we consider data partitioning. Partitioning is another crucial design decision in distributed applications for increasing system availability, and reducing query processing times [ANS18]. Indeed, graph partitioning aims to assign portions of the graph to several machines of the distributed cluster in a balanced way (as possible) so that data shuffling (i.e., the amount of data exchanged) across machines is the minimum when executing distributed query analytics. In our work, we aimed to study the behavior of a Big Data system (i.e. Spark-SQL) for querying large RDF graphs with the impact of various RDF partitioning techniques (Horizontal partitioning, Subject(Predicate)-based partitioning) that are different in their data distribution nature, and directly impact the performance of the system. Last but not least, the storage formats affect the compression levels and the data access according to the storage nature (i.e., row and columnar formats).

Descriptive and diagnostic performance analyses in such complex experimental scenarios are a hard and time-consuming task. These levels of analysis cannot provide final actionable answers on the performance analysis in such complex scenarios. Indeed they require a prior deep understanding on the domain knowledge in all aspects, including the workload as well as the reaction of configurations to it. Moreover, inherent trad-offs among the experimental dimensions hinder clear final decisions.

Hence, a prescriptive technique that drafts a clear trade-off across all experimental dimensions will simplify the results presentation and reliably guide selecting advanced settings for their RDF data processing performance. We dedicate the next chapter to discussing the prescriptive analysis that addresses the previous concerns, with an emphasis on the necessity for comprehensive ranking criteria.

4. BIG DATA SYSTEMS PERFORMANCE REPLICABILITY

In this chapter, we discuss the experimental methodology that we used to answer the **Micro 2** question (defined in Section 1.5.2): *can we guarantee replicability of BD systems performance when introducing other new experimental options?*. This question reflects on the replicability of BD systems performance and validates if we can guarantee fair performance assessment of these systems in complex scenarios like big (RDF) graph querying. As have been mentioned in the Introduction chapter, we define the replicability in our context with the ability of a researcher to arrive at the same scientific findings as a previous study while varying the underlying experimental setups (e.g., data, configurations, etc.) [NM+19]. In practice, this Micro question investigates the replicability of system's performance under the effect of changing experimental options (e.g., Schemas, Partitioning techniques, or Storage formats). Thus, we test the hypothesis (mentioned in Section 1.5.2): *HP0: The replicability of the BD system's performance for querying large (RDF) graphs could be affected by introducing other new experimental options.*

In this chapter, we look at the problem from the perspective of the *generalizability* of the state-of-the-art BD systems' performance findings in such complex experimental solution space of multiple dimensions. Chapter 3 shows the limitations of Descriptive and Diagnostic Spark-SQL performance analyses in the complex solution space of multiple experimental dimensions. Indeed, it showed that the *relational schema* is not the only *impactful* dimension for the performance of relational BD systems for processing/querying large RDF graphs. Herein, we further investigate the performance improvement of a relational BD engine (e.g., *Spark-SQL* framework) with two recent schema optimizations (i.e., *Extended Vertically Partitioned Tables* (ExtVP) and *Wide Property Tables* [Sch+14; Sch+16]), w.r.t. their baseline approaches (i.e., *Vertically-Partitioned (VP) Tables* and *Property Tables* (PT)). The state-of-the-art show that those schemata optimizations outperform their baselines. In this chapter, we aim to observe if the performance of the two schema advancements generalizes (i.e., still outperform the baseline ones) over Spark-SQL by introducing *different* RDF partitioning techniques and various storage data formats that are different from the original configurations [Sch+14; Sch+16].

4.1. Methodology and Experiments

Our methodology assesses if we can replicate the state-of-the-art results of the schema optimizations [Sch+14; Sch+16; al19] over the baseline relational schemas performance when introducing different experimental options. Thus, we performed our experiments in an as similar setup as possible to what the original authors have done [Sch+14; Sch+16]. In this regard, we use the baseline (*vanilla*) HDFS partitioning technique. We also use *Parquet* as the baseline storage file format (grey shaded boxes in Figure 25). We then introduce new experimental dimensions to our experiments, such as different partitioning techniques and different storage file formats, testing against various SPARQL query shapes.

Regarding the data partitioning, we introduce the *Horizontal Partitioning* (HP) technique and *Subject-based partitioning* (SBP) for the *WPT* and *PT* schema experiments. On the other hand, *Horizontal*, *Subject* and *Predicate-based* partitioning techniques are

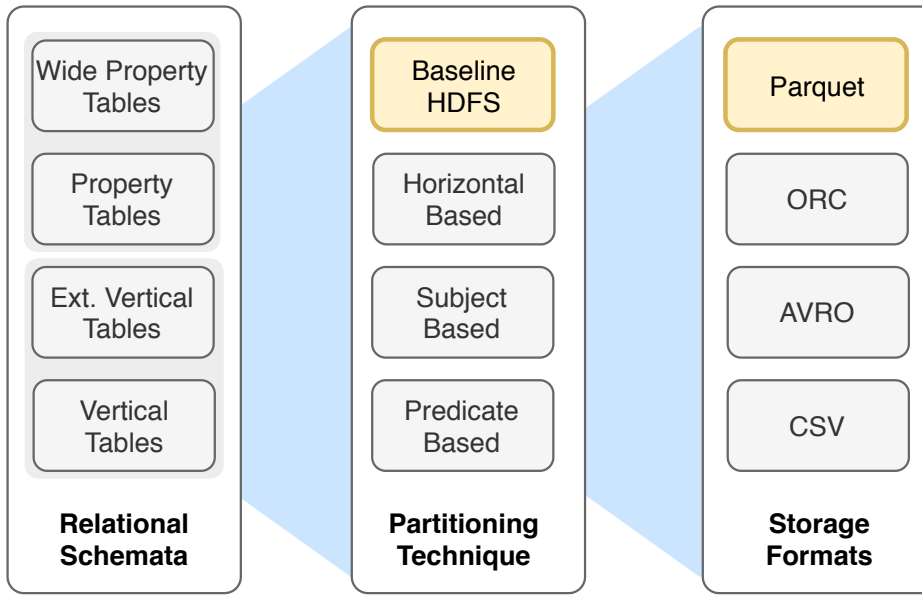


Figure 25: Assessing the performance of Spark-SQL with Schema advancements (WPT, ExtVP) against the baseline schemas (PT, VP), when changing alternatives of other experimental dimensions (e.g., partitioning, and storage).

used for the *VP* and *ExtVP* schema experiments.

We aim to check the impact of these partitioning techniques on the performance of SparkSQL when evaluating SPARQL queries. These partitioning techniques can have significant impacts due to the distribution of the data across the cluster nodes which will force more *shuffling* in the presence of query joins. For example, Horizontal partitioning should have a worse impact than Subject-based partitioning on the PT and WPT schemas, and Predicated-based on (Ext-)VP schemas. These partitioning techniques do not take the query shape into account and possibly place the required data in different nodes.

Regarding the storage of file formats besides the baseline Parquet, we consider an additional *columnar* format, i.e., *ORC*, and two *row-oriented* ones, i.e., *CSV* and *Avro*. We expect columnar formats to perform better for the queries with a subset of column projections since they allow an efficient scan of tables by reading only a portion of columns [IP19].

Finally, we aim to draft our observations and primary findings and propose some more specific best practices related to the schema optimizations (besides the ones in the previous chapter) by discussing and analyzing the experiments' results. Additionally, we aim to highlight the *trade-offs* of combining all these dimensions at the end of this chapter. In particular, we aim to observe the impact of introducing different configurations on these optimizations' and how it would impact the large SPARQL query performance on the SparkSQL engine. Herein, we plan to verify and answer the following questions:

1. How far do RDF partitioning techniques and storage formats impact the replicability of a BD system's query performance in the presence of schema optimizations?
2. How can we systematically compare different RDF relational schemas on top of BD systems in the presence of other performance impactful experimental dimensions (e.g., various Storage formats or Partitioning techniques)?

Table 9: SP^{2}Bench-100M relational schemas size with different file formats

SP^{2}Bench	RDF (n3)	PT	WPT	VP	ExtVP
CSV	**11GB**	∼9.2MB-1.9GB -Total: **6.8GB**	**9.4GB**	8KB-1.9GB -Total: **8.3GB**	- OS (4.9GB) - SS (39GB) - SO (806MB) -Total:∼**45GB**
Avro	**11GB**	980KB-416MB -Total: **1.6GB**	**1.8GB**	8KB-272MB -Total: **1.7GB**	- OS (359MB) - SS (8.8GB) - SO (331MB) -Total:∼**9.5GB**
ORC	**11GB**	620KB-362MB -Total: **1.4GB**	**1.4GB**	8KB-249MB -Total: 1.5GB	- OS (243MB) - SS (7.8GB) - SO (301MB) -Total:∼**8.4GB**
Parquet	**11GB**	620KB-382MB -Total: **1.5GB**	**1.7GB**	8KB-264MB -Total: **1.6GB**	- OS (319MB) - SS (8.4GB) - SO (318MB) -Total:∼**9GB**

3. Last but not least, what are the best practices that guide BD semantic web community towards the best performance with those schema advancements?

4.2. Benchmark & Experimental Setup

This section outlines the experimental setup and the used benchmark alongside a simple analysis of its queries. The experimental setups (presented in Figure 25) summarizes the experiments configurations (*relational schema*, *Partitioning*, *Storage*). We have performed our experiments for *four* different relational schemas, partitioning each schema across *four* various partitioning techniques, i.e., one baseline HDFS, and other three RDF-specific techniques. Last but not least, those schemas tables are stored across *four* different storage formats. In detail:

Benchmark &Dataset: In our evaluation for this chapter's experiments, we used the SP^{2}B SPARQL benchmark [eta09]. Details of the benchmark and its selection criteria can be found in Chapter 3.

Data Storage: We generated a *synthetic* RDF dataset with 100*M* triples size in Notation3 (*n3*) format. This scale size is enough for checking the validity of the literature findings regarding the RDF relational schemas optimizations and maintaining their replicability in a more complex solution space [1].

The generated *n3* RDF dataset is converted into CSV relational schemas using the *Jena TDB* and *Jena ARQ* as described in details in Chapter 3. Finally, these raw CSV tables (relational schemas) are loaded to the HDFS. Then, we use the Spark-SQL framework to transform the various relational schemas data tables from the CSV format into the other HDFS file formats (i.e., Avro, Parquet, and ORC).

Table 9 shows the size of the generated native RDF dataset (i.e., 11*GB*), as well as the storage sizes of each relational schema in the mentioned different file formats on top of HDFS. It is clearly shown how the different relational schemas affect the input data sizes.

[1] Our previous experiments [RTS19] show that the 100M dataset scale is the minimum challenging data-load for SparkSQL on our cluster distributed setup

In action, the PT schema has the smallest table sizes in total, followed by the VP schema, then the WPT table schema. In contrast, the largest storage overheads come with the ExtVP schema. We can also notice how the storage formats affect the sizes of the schemas significantly. In particular, columnar-oriented formats have the minimum table sizes across all the schemas. Indeed, ORC is shown to have the minimum table sizes, followed by Parquet. Whereas, the Avro row-oriented formats have quite larger schema sizes, and CSV has the largest table sizes.

Queries: As have been mentioned, SP^{2}Bench queries have different complexities and a high *diversity* of features [Sal+19]. These queries implement meaningful requests on top of RDF data. In our experiments, we reused the SQL version of the queries associated with the SP^{2}B benchmark [2] for the mentioned RDF relational schemas. However, for the new relational schema advancements (e.g., ExtVP, WPT) that are missing on the benchmark website, we have manually translated these queries into SQL, and we provide all these translated queries in our project repository [3]. We have evaluated all of these 11 queries of type *SELECT*, except $Q9$ and $Q11$, which are not applicable (*'NA'*) for the PT and the WPT relational schemas. Also, $Q7$ is not applicable in the VP and ExtVP schemas. Notably, for generating the ExtVP tables, we configured the default selectivity threshold of 1 [Sch+16] to generate the full versions of ExtVP table joins. Table 3 (in Chapter 3) shows the SP^{2}B benchmark query shapes besides several complexity features, in terms of the number of joins, filters, and projections.

Environment Setup: Our experiments were executed on the *bare-metal* cluster of 4 machines with the same specifications mentioned in Chapter 3.

RDF Data Partitioning: We used Spark for partitioning the relational schemas. This is required to persist those DataFrames on top of the HDFS default file blocks partitioning level. We use the resulting DataFrames as the input for the query engine. In our experiments, we have the baseline HDFS partitioning (grey partitioning box (see Figure 25). While other RDF partitioning techniques also have been tested, namely *HP*, *SBP*, and *PBP* approaches. These techniques depend on partitioning the data horizontally across machines (i.e., HP) or based on a partitioning key of the RDF subject or *predicate* (i.e., SBP and PBP, respectively).

Performance Evaluation measure (Latency): We used the *Spark.time* function by passing the *spark.sql()* query execution function as a parameter to measure the query *latency*. We run the experiments for all queries *five* times (excluding the *first cold start* run time, to avoid the *warm-up* bias, and computing an average of the other *four* run times).

4.3. Replicability Results

In this section, we discuss the experiments' results. Particularly, we compare the optimized relational schemas (i.e., WPT and ExtVP) against their baseline schemas (i.e., PT, and VP, respectively) according to our methodology.

#Joins	*Q1*	*Q2*	*Q3*	*Q4*	*Q5*	*Q6*	*Q8*	*Q10*	*Q11*
PT	2	9	2	8	7	6	9	5	2
WPT	0	0	0	3	3	3	10	3	0

Table 10: SP2Bench queries: Number of Joins of PT vs WPT.

WPT vs. PT	**Avro**	**CSV**	**ORC**	**Parquet**
Vanilla HDFS Partitioning	2/9	2/9	8/9	9/9
Horizontal Partitioning	2/9	3/9	6/9	6/9
Subject-based Partitioning	2/9	2/9	6/9	6/9

Table 11: Number of queries for which WPT outperforms PT for data formats and partitioning techniques.

4.3.1. WPT versus PT Schema Results

Table 10 shows the SP[2]Bench queries' number of joins when translated into SQL concerning the PT and WPT schemas. Except for *Q*8 (which requires many *self-joins* of the WPT table), the number of joins always decreases, adopting the WPT schema. Moreover, we expect that the WPT schema query performance (i.e., in terms of *latency*) will outperform other relational schemas [al19]. In this regard, the Parquet data format efficiently handles the sparsity caused by the WPT table schema —as *Null* values are efficiently ignored in this file format [Sch+14].

Table 11 shows the overall benchmark results of the WPT performance over the PT schema across all file formats (*horizontally* in the table), and across the different partitioning techniques (*vertically*). Values in this table specify the number of queries in which the WPT schema outperforms [4] the baseline PT schema. The *green* color indicates that WPT performing the best, while the *yellow* color indicates that its performance is above 50% over the PT schema, and the *red* means that performance is less than 50%.

Our results confirm that the WPT schema performs better than the baseline PT schema in all the queries (i.e., 9 queries out of 9 in the benchmark) with Parquet file format and using the baseline HDFS partitioning technique. Indeed, these results confirm the findings in [Sch+14; al19] assessing the replicability regarding the WPT schema optimization.

To investigate how the performance difference between the WPT and PT schemas changes, we introduce two new dimensions, i.e., various file formats and different partitioning techniques. In this regard, Table 12 shows the effect of data partitioning (*left* of the table) and storage formats (*right* of the table) considering the other new factors across all the experiments. To this extent, we have calculated the percentages as follows, for the partitioning factor's impact, we pivoted on each partitioning technique and counted the percentage of how much the WPT schema performance in SparkSQL is better than the PT schema one across all the queries while considering all the changes of the storage file formats (moving across them). We calculated the partitioning effect similarly by pivoting

[2] `http://dbis.informatik.uni-freiburg.de/index.php?project=SP2B/queries.php`

[3] `https://datasystemsgrouput.github.io/SPARKSQLRDFBenchmarking/`

[4] Notably, in this chapter when mentioning a schema outperforms another schema, we specifically mean that the system (e.g., Spark-SQL) query latency with using the first schema is less (better) than with the other schema.

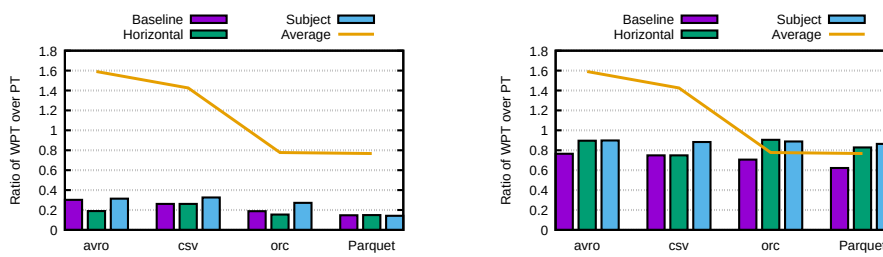
WPT/PT	Partitioning effect			Storage effect	
	Baseline_HDFS_Part	58.33%		***Parquet***	77.78%
	Horizontal Partitioning	47.22%		***ORC***	74.07%
	Subject-based	44.44%		***CSV***	25.93%
	Predicate-based	NA		***AVRO***	22.22%

Table 12: The effect of other partitioning techniques, and other storage formats on the reproducibility of the WPT S.O.T.A findings.

on the storage file format and moving across the partitioning techniques in all queries.

Table 12 also demonstrates that in such a complex space of different relational schema, data partitioning, and storage file formats, schema-based query optimization is not straightforward. As we can see, WPT outperforms PT schema only for 58% in the queries using only the baseline default HDFS partitioning technique regarding the storage formats and only about 78% for the Parquet file format. The determination of this result shows the *trade-off* of considering alternative storage file formats and partitioning techniques alongside the experiments' query evaluation.

Regarding the storage, we can see that ORC, another columnar file format, performs closer to the baseline columnar Parquet file format with 74%. However, the baseline Parquet is yet better, as unlike ORC, Parquet can efficiently handle the WPT table's sparsity [Sch+14]. On the other side, we can see that the row-oriented formats have a significant negative effect on the performance of the WPT schema. The WPT schema performance is better than the PT schema, with only 22% and 25% in all Avro and CSV queries, respectively. In action, SP^{2}Bench queries only have one query (i.e., $Q2$) with more than 2 column projections. This justifies why *column-oriented* formats (in-general) give better results for the WPT than the row-based ones. In general, we can state that file formats affected the replicability of the state-of-the-art results for the WPT schema. In more specific words, the WPT schema advancement performance only replicates (i.e., being better than the PT baseline schema) with the baseline storage format (i.e., Parquet), but not when changing other columnar or row-oriented storage formats.



(a) Q2 (b) Q4

Figure 26: The performance of the WPT schema over PT schema in $Q2$ and $Q4$ (values below '1' means WPT is better than PT)

At last, we enroll in three specific queries, namely, $Q2$, $Q4$, and $Q8$. We selected these queries as good representatives to exemplify our findings. However, the reader can find the rest of the query figures in our mentioned GitHub repository.

Figure 26 (a), (b) and Figure 27 depict the performance of SparkSQL for these queries under a various combination of file formats and partitioning techniques. In particular,

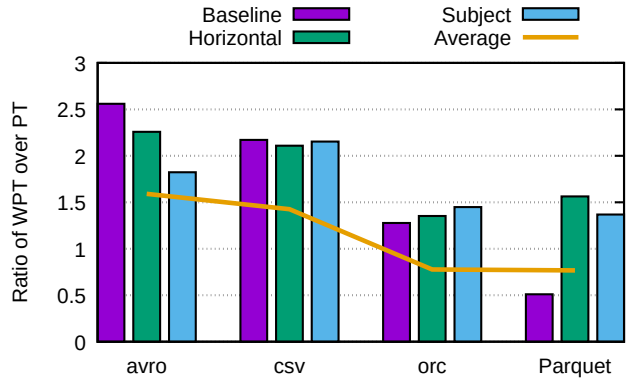


Figure 27: The performance of WPT over PT schema in Q8. values (below '1' means WPT is better than PT)

these figures combine the ratios of the WPT schema performance (in terms of the system's query latency) being better than the PT schema in those mentioned queries. Particularly, the Y-axis shows the ratios of the system's query latency (runtimes) with the WPT over the query latency with using the PT schema, across various storage formats and partitioning techniques. The ratios less than 1 indicate better performance of WPT over PT in that query and across the different configuration settings. The 'yellow' lines in the figures show the average of the various partitioning techniques ratios (mentioned above, i.e., WPT/PT query latency) across each storage format.

There is a tremendous performance enhancement in WPT over PT in $Q2$ and $Q4$. The reason behind this refers that the number of SparkSQL joins of WPT is significantly less than the joins in PT schema (cf. Table 10). Particularly, in $Q2$ number of joins in PT (SQL-version) is 9 compared to *no-joins* in WPT schema. While in $Q4$ with PT schema, we have 8 SQL joins compared to 3 *self-joins* of the WPT table. Interestingly, we have more joins in WPT than the baseline PT schema in $Q8$, i.e., 10 self-joins and 8 joins, respectively.

Not surprisingly, we can notice that $Q8$ is the only query that witnesses worse performance for the WPT compared to the PT schema. Figure 27 shows that most of the ratios of 'WPT over PT' is greater than 1 in the baseline-partitioned data experiments (i.e., partitioned only with HDFS) and other file formats instead of Parquet. Notably, all the results (i.e., total query runtimes) and query histograms can be found on our mentioned GitHub repository.

4.3.2. ExtVP versus VP Schema Results

According to [Sch+16], ExtVP outperforms or at least performs similarly to the VP schema. The reason is that queries are similar, and the number of SQL joins in the VP, and ExtVP schemas are the same. This clarification is reflected in Table 13. The performance improvement depends mainly on the percentage of reductions in the input table sizes that the ExtVP optimization might introduce out of the join correlations for each query [Sch+16]. Table 13 also presents the percentage of ExtVP reductions of the processed tables' rows for each query over the original input tables processed rows with the baseline VP tables. The *semi-join* reductions provided by the ExtVP optimization help speeding-up the performance of SparkSQL by reducing the size of the shuffled data.

Query	VP	ExtVP	Input tables size reductions
Q1	2	2	58%
Q2	9	9	77%
Q3	1	1	59%
Q4	7	7	96%
Q5	5	5	60%
Q6	9	9	31%
Q8	9 & 1 Union	9 & 1 Union	5%
Q9	2 & 1 Union	2 & 1 Union	0%
Q10	1 Union	1 Union	0%
Q11	0	0	0%

Table 13: Number of joins and percentage of input tables sizes reductions.

ExtVP VS. VP	**Avro**	**CSV**	**ORC**	**Parquet**
Baseline HDFS Partitioning	6/10	6/10	5/10	7/10
Horizontal Partitioning	3/10	3/10	3/10	3/10
Predicate-based Partitioning	2/10	3/10	6/10	6/10
Subject-based Partitioning	2/10	3/10	3/10	3/10

Table 14: Comparison of ExtVP schema with the VP schema in different storage formats, and in different partitioning techniques.

In more details, ExtVP optimizes specific queries according to the join correlations between triple patterns in those queries [Sch+16], namely, in *Subject-to-Subject(SS)*, *Object-to-Subject(OS)*, and *Subject-to-Object(SO)* join relations [Sch+16]. Thus, we expect some queries to give similar results to the VP schema queries (i.e., No reductions occurred in the VP tables by the ExtVP schema optimization). In our experiments, $Q9$, $Q10$, and $Q11$ do not present any input data reductions. Thus, we state that it is expected that their performance to be very close to the baseline VP schema performance.

We adopt the same approach of the (WPT to PT) schemas performance comparisons for evaluating the performance of ExtVP against the VP experiments.

First, we check if our experiments' results confirm the state-of-the-art results regarding the ExtVP schema optimization over the baseline VP schema performance.

Table 14 shows the total number of queries in which the ExtVP performance is better than the VP schema performance across all the benchmark queries. For our baseline HDFS partitioning technique and with the Parquet file format, we can see that some queries do not benefit from the optimizations of the ExtVP. Indeed, 3 queries out of 10 fail to utilize the optimized ExtVP technique. The reason behind such behavior is that those queries have *unbounded* predicates that can not be optimized by the ExtVP schema [Sch+16] (see $Q9$ and $Q10$ in Table 3), or they have no effective join reductions (see $Q9$, $Q10$, $Q11$ in Table 13). The performance of these queries is a subject of in-detail discussion in the next sections.

Second, similarly to what we have done for the WPT schema optimization, we now investigate how generalizable the state-of-the-art results are when we introduce different file formats or partitioning techniques over the data for both the ExtVP and VP schemas.

Similarly, Table 15 shows how far the data partitioning (*left* of the table) and data formats (*right* of the table) impact the results of ExtVP in comparison to the VP schema

ExtVP/VP	Partitioning effect			Storage effect	
	Baseline HDFS _Part	67.5%		***Parquet***	55%
	Horizontal Partitioning	35%		***ORC***	45%
	Predicate-bsed	55%		***AVRO***	42.5%
	Subject-based	30%		***CSV***	42.5%

Table 15: The effect of other partitioning techniques, and other storage formats on the reproducibility of the ExtVP S.O.T.A findings

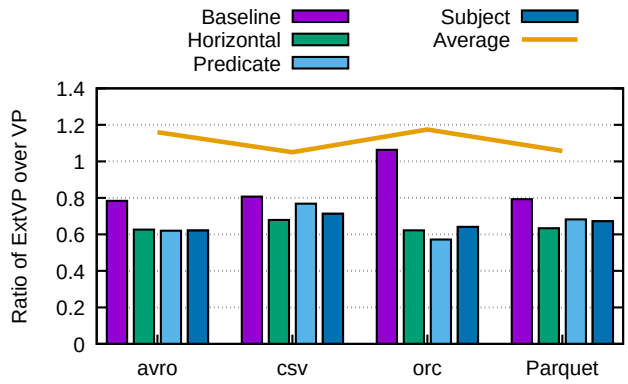
performance. Notably, this table's percentage values are also calculated similarly to how we have calculated the WPT against the PT. We pivoted on the analysis dimension of choice, i.e., file format X or partitioning technique Y, and we calculated how many times SparkSQL performs better using ExtVP than using the baseline VP approach.

Regarding the partitioning techniques' effect on ExtVP, our expectations are confirmed. In particular, we can observe that the partitioning techniques degraded the performance of ExtVP significantly. Only 35% and 30% of the experiments adopting Horizontal and Subject-based partitioning, respectively, show a performance improvement in using ExtVP over VP schema. Adopting Predicate-based partitioning slightly reduces this negative effect (i.e., 55% of the queries show that performance improvement).

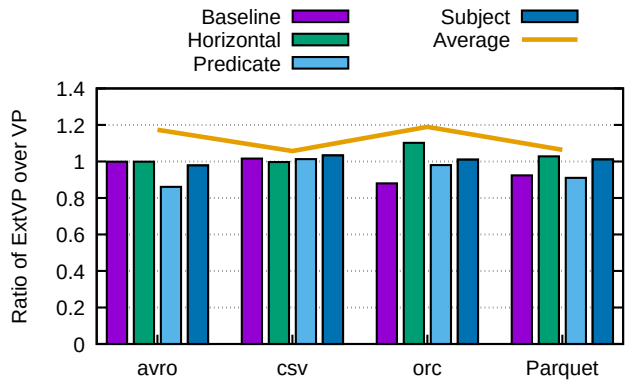
From Table 15, we can also see that the ExtVP schema is only outperforming the VP schema, with 67% of the queries using the baseline HDFS partitioning scenario. Thus, we can see the *trade-off* of considering various storage file formats. We can see also that the baseline Parquet file format is the one that has less impact on the overall performance for ExtVP. Indeed, in 55% of the cases where Parquet is used, ExtVP outperforms the VP performance. Additionally, the ORC columnar file format provides high performance of ExtVP over VP schema with an overall 45% of the cases. However, there is a clear difference from the Parquet file format with 10% better performance.

On the other hand, the row-oriented formats degrade the performance of ExtVP. For only 42.5% of the experiments that adopt either Avro or CSV, ExtVP performance beats the performance of the VP schema. Such behavior is related to the number of column projections in the SP^{2}Bench queries, which are the minimum in this benchmark scenario. Thus, columnar file formats can fit such query workloads better than row-oriented ones.

Last but not least, herein, the most notable query examples are introduced, confirming our previous findings but with more details. First, $Q4$ is revealed to be the query with the most benefit with the ExtVP optimization. The reason behind this is that $Q4$ includes a high number of joins (i.e., 7 joins) and has the maximum number of input tables' rows reductions while using the ExtVP schema optimization with 96% of reduced processed rows (see Table 13). This query is directly followed by $Q2$ with 77%. Although $Q2$ has a higher number of table joins than $Q4$, the reductions in input table sizes in $Q4$ are more significant. On the other side, $Q9$, $Q10$, and $Q11$ do not benefit from the ExtVP optimization, i.e., ExtVP does not provide any input table size reductions. In particular, $Q9$ and $Q10$ have unbounded predicate variables in the original SPARQL queries. ExtVP cannot *directly* handle this type of queries [Sch+16]. While $Q11$ has only a single triple pattern, and thus it has no joins in optimizing the ExtVP optimization approach. Figures 28 (a) and (b) show the performance of SparkSQL for $Q4$ and $Q9$, respectively, under various combinations of formats and partitioning techniques in the ExtVP experiments. Figure 28 (a) shows that $Q4$ is always below the line of all the other queries' average runtimes. Whereas, ExtVP does not show a remarkable difference over the VP schema in $Q9$, i.e.,



(a) Q4



(b) Q9

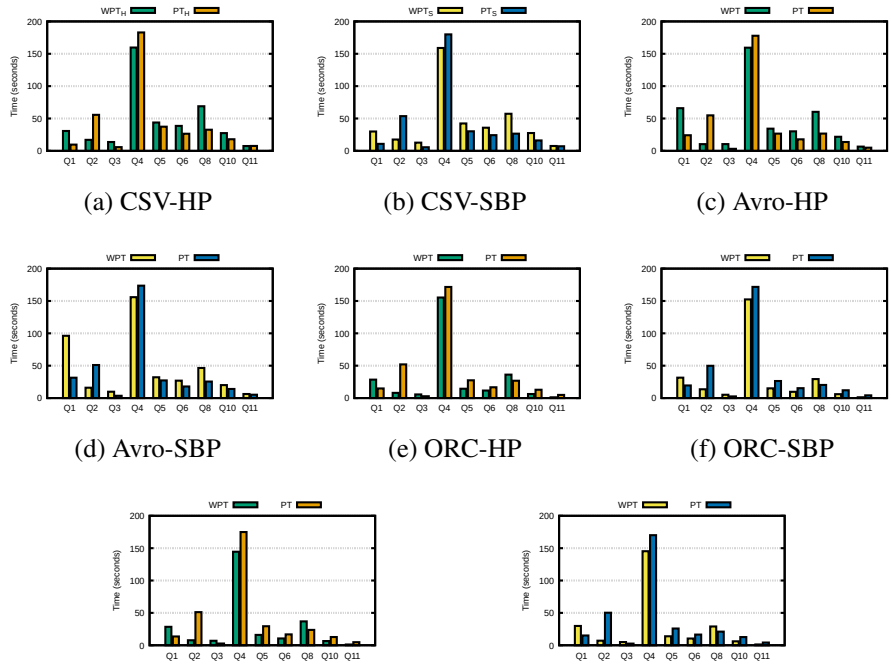
Figure 28: The performance of ExtVP over VP schema in Q9. (values below '1' indicates that ExtVP is better than VP)

they show pretty close performance to each other.

In the next section, we discuss in further detail the experiment findings against the current S.O.T.A regarding the superiority of ExtVP and PT.

4.4. Discussion

This chapter helps to assess the replicability of BD systems performance while introducing various new experimental dimensions. Specifically, it characterizes and classifies the RDF relational schemas and their optimizations within the SparkSQL realm introducing new experimental dimensions (e.g., partitioning and storage). Thus, that helps data architects and practitioners interested in large RDF processing to understand the RDF relational schema potentials better using different partitioning techniques and storage formats. This understanding will lead to a better selection of the most suitable and performance-optimized solution that adequately suits their needs. Achieving that will also accommodate better design and development of new SPARQL systems, leading to reliable RDF services with high Spark performance. Taking our experiments' findings



(a) CSV-HP (b) CSV-SBP (c) Avro-HP

(d) Avro-SBP (e) ORC-HP (f) ORC-SBP

(g) Parquet-HP (h) Parquet-SBP

Figure 29: WPT Vs. PT schemata performance using different partitioning techniques and storage formats

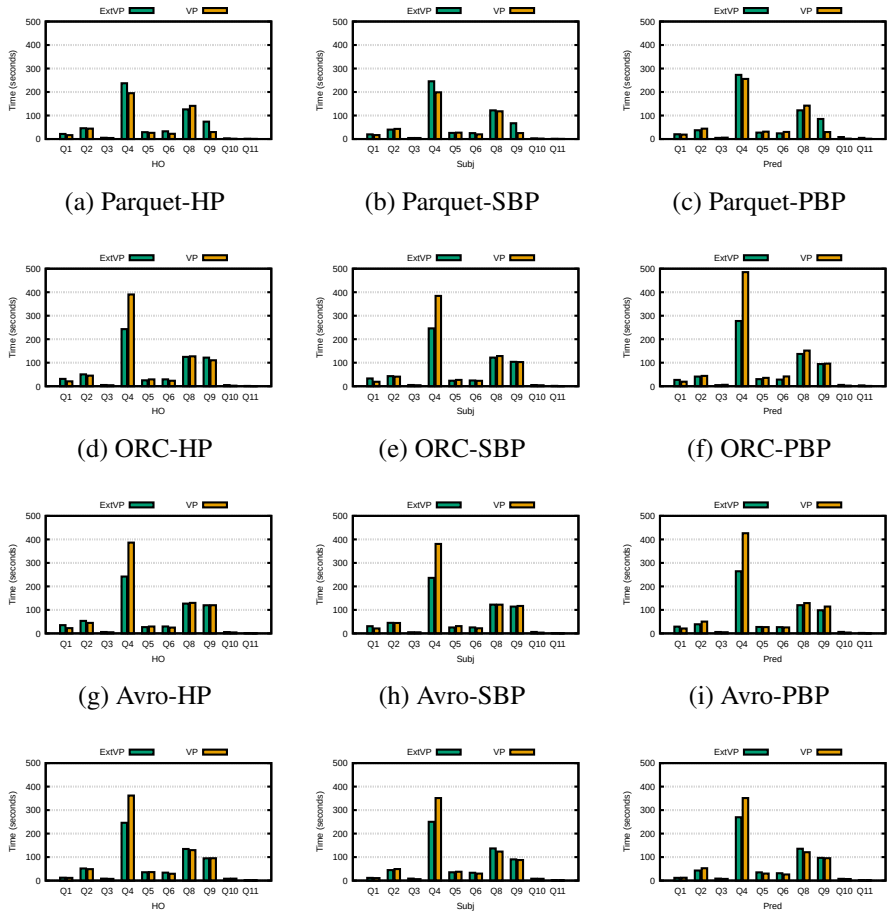
into consideration, we discuss our results and give some insights on processing RDF best practices on a large scale. Next, we place the literature assumptions on the relational schema optimizations' superiority against our experimental findings. We follow this by recommendations to the large RDF processing practitioners.

4.4.1. Hypothesis 1: The WPT schema always outperforms PT schema

According to [Sch+14; al19], we expect that the performance of the WPT schema outperforms the PT schema, especially with the "*star-shaped*" queries. Star-shaped queries can be answered when the WPT table is queried with *no-joins* included because all the properties relevant to the same subject are present in the same WPT table.

The state-of-the-art findings of the WPT schema are fully reproduced with the default HDFS partitioning and with using the baseline Parquet file format. That is, the performance of Spark using the WPT schema for representing the RDF dataset always outperforms the baseline PT schema.

Nevertheless, our results show when we deviate from the original setup [Sch+14; al19] introducing new experimental factors, the solution space increases in complexity. Consequently, the trade-offs between relational schema, partitioning techniques, and storage formats make the WPT optimization replicability not straightforward. Using other partitioning techniques alongside the baseline Parquet format affected the replicability of the WPT schema optimizations. Only 78% of the queries' results conform with the fact that WPT is better than the PT schema (Table 12).



(a) Parquet-HP (b) Parquet-SBP (c) Parquet-PBP

(d) ORC-HP (e) ORC-SBP (f) ORC-PBP

(g) Avro-HP (h) Avro-SBP (i) Avro-PBP

(j) CSV-HP (k) CSV-SBP (l) CSV-PBP

Figure 30: ExtVP Vs. VP schemata performance using different partitioning techniques and file formats

Figure 29 shows the schemas' performance when the solution adopts different partitioning techniques and file formats. Figures 29 (a-h) show clearly the effect of partitioning techniques on the replicability of the WPT optimizations across all the different file formats. For instance, notably the horizontal partitioning (Figures 29 (a,c,e,g)) affected the performance of WPT, making its performance in SparkSQL worse than the baseline PT schema in most of the queries (i.e., $Q1$, $Q3$, $Q5$, $Q6$, $Q8$, $Q11$). Similarly, we can observe the negative effect of the subject-based technique on WPT schema (Figures 29 (b,d,f,h)) in the same queries.

The impact of file formats aside the baseline Parquet format is even worse. Even using the baseline (HDFS) partitioning technique affects the replicability of the WPT schema optimizations. Overall, only 58% of the query results conform with the fact that WPT outperforms PT schema (Table 12). The experiments show that columnar file formats, e.g., ORC, and Parquet, are the best for representing such wide tables (WPT and PT). Columnar file formats are the best for sparse queries (i.e., queries with few column projections or columns to access) out of the wide tables. They perform better than the

row-oriented file formats, e.g., CSV and Avro, which would be only better with queries that require full-row reading.

Figure 29 shows the performance degradation considering different file formats. For instance, moving from Parquet and ORC in Figures 29 (e-h) to other row-oriented file formats such as Avro and CSV in Figures 29 (a-d), we can notice the performance degradation of the queries with the WPT schema optimizations.

4.4.2. Hypothesis 2: The ExtVP always outperforms VP schema

According to [Sch+16], we expect that ExtVP provides better or at least similar performance gains than the VP schema, as the queries are similar, and the number of SQL joins in the VP schema is equal to the ExtVP joins. Nevertheless, one should keep in mind that ExtVP improvements mainly depend on the possible reductions in the table input data size and excluding the dangling triples (rows that do not contribute to any joins) [Sch+16]. Typically, the queries for the ExtVP schema are similar to the VP ones; the only difference realizes in the queried input tables(i.e., their sizes are reduced by the ExtVP schema, or their sizes are still equal to the original VP tables). Thus, the relational engine's performance, e.g., Spark with the ExtVP, should be equivalent to or better than its performance with the VP schema.

Based on our experiments, the findings of the ExtVP schema are not fully reproduced, even considering the default HDFS partitioning and the baseline Parquet file format. Some queries do not benefit from the ExtVP optimizations ($Q9$, $Q10$, $Q11$); no input size reductions occurred in those queries) as shown in Table 13. Only on the original configurations (except for those mentioned queries), we can confirm the state-of-the-art results regarding the ExtVP optimization. However, our results show that schema-based query optimization is not straightforward when changing the default configurations.

Regarding the partitioning techniques, using an alternative to the baselines technique (HDFS) affects the replicability of the ExtVP optimizations even if the storage format is Parquet. Only 55% of the queries' results show that ExtVP is superior to the VP schema (cf. Table 15). Moreover, Figure 30 shows the effect of other RDF partitioning techniques on the replicability findings of the ExtVP optimization. For instance, deviating from the baseline partitioning technique to other RDF-based techniques with the same baseline Parquet, i.e., Figures 30(a-c) degrades the results of ExtVP and makes it perform worse than the baseline VP schema in several queries ($Q1$, $Q4$, $Q5$, $Q6$, $Q8$) with the Horizontal and Subject-based partitioning. The predicate-based partitioning in Figure 30(c) has a better performance with this schema, which has a close performance to the VP schema in the previously-mentioned queries.

Similarly, using storage formats different from Parquet affects the ExtVP optimizations' replicability, even with the baseline (HDFS) partitioning technique. Indeed, we have only 67.5% of the query results of ExtVP outperforming VP (cf. Table 15). Similarly, Figure 30 shows the effect of other file formats other than the baseline Parquet, i.e Figures 30 (d-l) for ORC, Avro, and CSV respectively. We can notice the queries' performance degradation with the ExtVP schema optimizations moving vertically to these other formats.

Finally, from our experiments, we observe that columnar file formats are better than Row-oriented ones. However, the performance difference is not significant with such similar schemas. The table structure is the same table of two columns Predicate (Subject-Object) in both vertical schemas. Moreover, both schemas do not have wide tables compared to the WPT and PT schemas. That is, these schemas will not benefit a lot from

the columnar file formats. The performance gain of columnar over the row-oriented file formats is because SP2Bench queries have a few numbers of column projections. Thus, it would work better with columnar rather than row-based file formats.

4.5. Concluding Remarks and Best Practices

In this chapter, we evaluated the replicability of the performance of one of the BD systems (i.e., Spark-SQL) by introducing various experimental options. In particular, we presented a comprehensive empirical evaluation of the state-of-the-art RDF relational schemas alongside introducing three RDF partitioning techniques and four storage formats. Our analysis demonstrates decisively variant *trade-offs* using different data partitioning and storage file formats against those schema optimizations. The experiments show that the performance replicability of the SparkSQL system with the relational schema optimizations can be easily affected by new experimental factors such as data partitioning or using various storage data formats.

	Avro	**CSV**	**ORC**	**Parquet**
Baseline-HDFS	✗	✗	✓*	✓**
Horizontal	✗	✗	✓	✓
Subject-based	✗	✗	✓	✓

Where ✓is good practice, ✗is bad practice,and ˜ has the same performance compared to PT.
* WPT had very competitive performance
** WPT had the best performance

Table 16: Mapping the partitioning technique to the storage format best practices in the WPT schema.

4.5.1. Best Practices with Schema Optimizations

Herein, we discuss the best practices for keeping the best performance of the state-of-the-art RDF relational schemas. Tables 16 and 17 provides an abstracted map of good and bad storage format and partitioning techniques along with the schema dimension.

The results in Figure 29 and Table 16, show that partitioning the WPT table has, in the majority, a negative effect on the WPT optimization, making it perform even worse than its baseline approach, i.e., the PT schema. The effect of the storage formats is more significant in the WPT optimization (cf. Tables 12, 16). Therefore, this WPT schema's storage format selection decision should be dealt with as a *first-class citizen* in such experiments.

The horizontal and subject-based partitioning techniques are not recommended with ExtVP optimization. However, Predicate-based still gives better results than those two other RDF partitioning techniques (cf. Tables 15 and 17). Also, columnar file formats are still recommended with the ExtVP schema optimization. However, it was noticed that the effect of the partitioning is more significant to this optimization (cf. Figure 30, Tables 15, and 17). Thus, the partitioning selection decision of this ExtVP schema should be highly considered in these experiments.

Also, our analysis yields the following recommendations

1. With WPT, it is recommended to use the columnar storage formats rather than row-oriented ones (cf. Table 16).

	Avro	CSV	ORC	Parquet
Baseline-HDFS	✓	✓	~	✓*
Horizontal	✗	✗	✗	✗
Subject-based	✗	✗	✗	✗
Predicate-based	✗	✗	✓	✓

Where ✓is good practice, ✗is bad practice, and ~ has the same performance compared to VP.
* ExtVP had a very competitive performance

Table 17: Mapping the partitioning technique to the storage format best practices in the ExtVP schema.

2. With the WPT schema, Parquet is yet the best columnar file format to select because it efficiently handles its sparsity.
3. With WPT, it is recommended to use the native HDFS partitioning rather than selecting an RDF-oriented partitioning technique.
4. With ExtVP, the baseline HDFS partitioning is more recommended than specific RDF ones. However, larger datasets would require partitioning anyway.
5. With ExtVP, the columnar file formats are a recommended performance optimization option.

5. BENCH-RANKING: A FRAMEWORK BD PRESCRIPTIVE PERFORMANCE ANALYSIS

In this chapter, we aim to provide answers to the second micro question which is formulated as follows:

Micro 2: *How can we efficiently select the best-performing configurations out of a complex experimental solution space without ignoring their performance trade-offs?*

The limitations of *descriptive* and *diagnostic* analyses (shown in Chapter 3) require comparing a huge number of experiments' results, and sometimes the results are even *contradicting* due to experimental trade-offs. We suggest raising the level of abstraction that reduces the comparative performance analysis reaching actual decision making. Therefore, we advocate for applying various ranking functions seeking this goal (we call it *"Bench-Ranking"* [RAT21]). Ranking functions show effective roles in decision support in several application-domains [GM20]. We utilize ranking functions [1] aiming to abstract from the *fine-grained* descriptive performance results and enable a *decision-making* model.

In this chapter, we answer *Micro 2* the following three phases. First, we employ ranking functions that aggregate performance indicators of each dimension, i.e., *Single-Dimensional* (SD) ranking. The SD ranking functions help assess the system's performance while using a specific dimension cross set of tasks (e.g., queries). We show the results of the SD ranking criteria and their limitations (i.e., optimizing towards a single dimension).

Second, we discuss how we mitigate these SD ranking limits via using *Multi Dimensional* (MD) ranking criteria. MD criteria select the best-performing configurations with any arbitrary number of dimensions while optimizing their performance simultaneously.

Last but not least, we discuss the Bench-Ranking metrics to evaluate the proposed ranking criteria (i.e., SD and MD criteria), and we show SD and MD ranking criteria evaluation results according to two defined metrics (i.e., *conformance*, and *coherence*).

5.1. *Bench-Ranking* Preliminaries

In this section, we summarize the concept of Bench-Ranking as a means for Prescriptive Performance Analysis. Bench-Ranking is based on *four* fundamental notions, i.e., *Configuration*, *Ranking Score*, *Ranking Function*, and *Ranking Set*, defined below.

Definition 7. *A configuration c is a combination of parameters (options) that represent experimental dimensions. The configuration space $\mathscr{C}$ is the Cartesian product of the possible configuration combinations.*

In [RAT21], we consider *triplet* dimensional configurations, including relational schemas, partitioning techniques, and storage formats. Figure 31 shows the experimental design space and highlights the example of (*a.ii.3*) configuration, which is akin to Single Triples (*ST*) schema, Subject-based Partitioning (*SBP*) technique, and stored in *ORC* file format.

Definition 8. *A ranking score R is a numerical value that decides the ranking of an element in a set according to a specific criteria (e.g., the query latency for a configuration).*

[1] Ranking criteria and ranking functions are interchangeably used in the text.

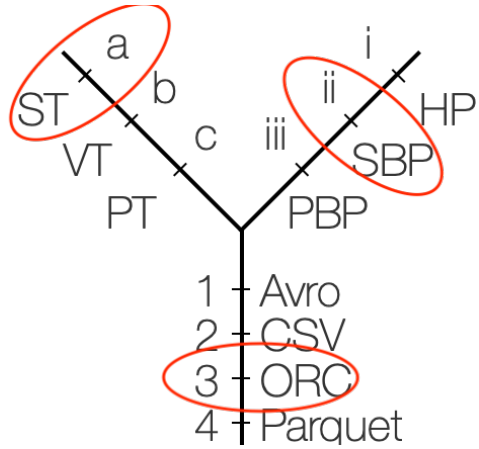


Figure 31: The configuration space $\mathscr{C}$.

Given two elements i and j , and R_i,R_j their ranking score, we say that i has a higher rank than j (i.e., i outperforms j) iff $R_i > R_j$.

Herein, we will use a simple running example, we consider a set of three competitive configurations [*a.i.1*, *b.ii.2*, *c.iii.3*]. Each configuration in the list represents an element of a set. An example of a ranking score (Definition 8) can be the time required (i.e., *latency*) for executing a query (e.g., Q_1 of a benchmark) by each of the selected configurations (in seconds). Let's assume the latency values are equal to 30, 50, and 40, for the three configurations, respectively. The association with each configuration with its ranking score happens according to the query evaluation (runtimes). We can generalize this by introducing the notion of *ranking function* (defined below).

Definition 9. *Let E is the input list of elements to be ranked, the ranking function f_R is a function $E \rightarrow \mathscr{R}$ that associates a ranking score to every element in E.*

Considering the ranking scores above, the configuration set E can be sorted to produce the sorted set $\mathscr{R}$=[a.i.1, c.iii.3, b.ii.2]. The lower the execution time (latency), the better. More formally, we denote $\mathscr{R}$ as a *rank set*.

Definition 10. *A rank set $\mathscr{R}$ is an ordered set of elements ordered by a ranking score. The rank index r_i is the index of a ranked element i within a ranking set $\mathscr{R}$, i.e., $\mathscr{R}[r_i]$=i. We denote with $\mathscr{R}^k$ the leftmost (top-ranked) subset of $\mathscr{R}$ of length k, and we denote with $\mathscr{R}_x$ the rank set calculated according to the Rank score R_x.*

Table 19 shows an example of actual query ranks of the configurations according to the query runtimes shown in Table 18, e.g., $c.i.2$ is at the first 1^{st} rank for running $Q1$, while $c.iii.4$ comes at the last (36^{st} rank.

5.2. Single-Dimensional (SD) Ranking Criteria

Ranking criterion like the one proposed in [ANS18] for RDF partitioning techniques helps to provide a high-level view of the system performance across a set of tasks. Applying the ranking criteria independently for each dimension supports explanations of the results [ANS18]. In our work, we generalize *Akhter's* proposal for calculating ranking scores about any experimental dimensions, as shown by the following ranking function:

Conf. \ Query	Q1	Q2	...	Q10	Q11
a.i.1	34.93	79.98	...	10.19	5.35
a.i.2	49.61	168.67	...	28.63	14.29
...	...	...	...	...	...
b.iii.4	18.63	44.10	...	1.45	1.13
...	...	...	...	...	...
c.i.2	9.67	55.67	...	17.93	7.61
c.iii.4	64.76	95.28	...	54.08	39.27

Table 18: Configurations' query execution runtimes (in seconds).

Conf. \ Query	Q1	Q2	...	Q10	Q11
a.i.1	29^{th}	28^{th}	...	19^{th}	25^{th}
a.i.2	33^{th}	36^{th}	...	30^{th}	32^{th}
...	...	...	...	...	...
b.iii.4	11^{th}	$\mathbf{35}^{th}$	...	5^{th}	8^{th}
...	...	...	...	...	...
c.i.2	$\mathbf{1}^{st}$	23^{th}	...	29^{th}	29^{th}
c.iii.4	36^{th}	33^{th}	...	36^{th}	36^{th}

Table 19: Configurations' rankings according to the query evaluation runtimes.

$$R = \sum_{r=1}^{d} \frac{O_{dim}(r) * (d-r)}{|Q| * (d-1)}, 0 < R \leq 1 \tag{5.1}$$

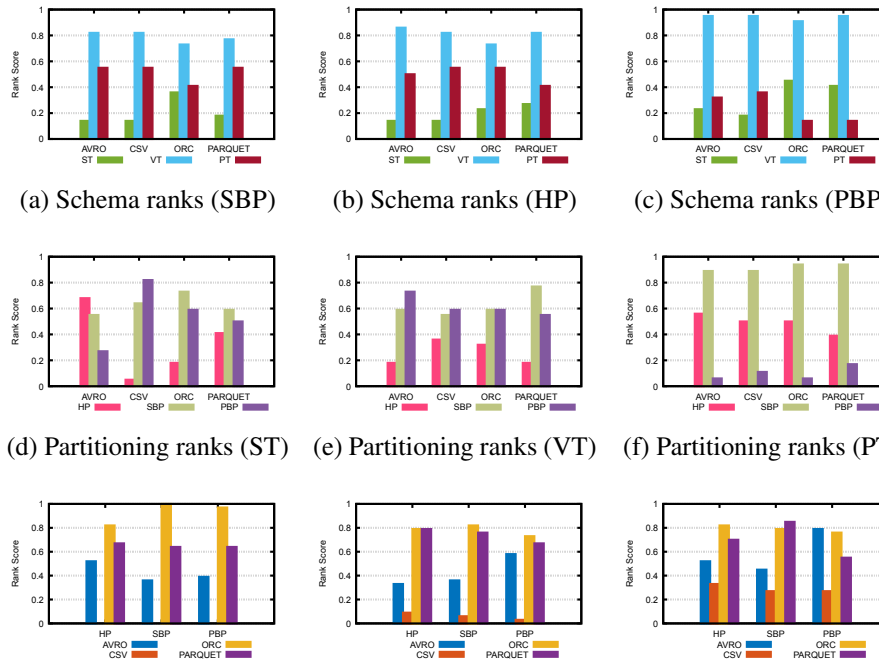
In particular, R is the *Rank Score* of the ranked dimension (i.e relational schema, partitioning technique, or storage backend). Such that, d represents the total number of variants (options) under the ranked dimension, $O_{dim}(r)$ denotes the number of occurrences of the dimension being placed at the rank r (1^{st}, 2^{nd},..) (see Table 20). In the formula, $|Q|$ represents the total number of queries, as we have 11 queries in our SP^{2}B benchmark or 20 queries in WatDiv (i.e., $|Q| = 11$ or $|Q| = 20$). The *Rank Score* value lies between [0,1], where "0" stands for the lowest possible score and ("1") points to the highest possible rank score.

Equation 5.1 evaluates the performance of a set of parameters across several different challenges. In our case, the parameters correspond to configurations and challenges to benchmark queries. For example, we can use Equation 5.1 for ranking the relational schemata, i.e., we pose $d = 3$, as we have *three* different schemata (i.e., ST, VT, and PT) in this paper[2]. Table 20 shows a simple example of applying the above formula for comput-

Schema	1^{st}	2^{nd}	3^{rd}	R
ST	1	3	7	0.23
VT	6	4	1	0.73
PT	4	4	3	0.55

Table 20: Example of *Rank Scores* for *schema* dimension.

[2]It will be the same (i.e., $d = 3$) while applying the equation for ranking partitioning techniques, matching the *three* given techniques (i.e HP, SBP, PBP). Whereas applying the equation for ranking



(a) Schema ranks (SBP) (b) Schema ranks (HP) (c) Schema ranks (PBP)

(d) Partitioning ranks (ST) (e) Partitioning ranks (VT) (f) Partitioning ranks (PT)

(g) Storage ranks (ST) (h) Storage ranks (VT) (i) Storage ranks (PT)

Figure 32: Single-dimensional ranking scores for the *SP²B 500M* triples datasets (The higher the better).

ing the rank scores of the relational schema dimension. For instance, ST was placed in the *"first"* rank only *once*, *"second" three* times, and *"third" seven* times. Thus, its overall ranking is 0.23. In contrast, the VT performed better with 0.73, and PT with 0.55. Intuitively, this means that VT schema in this example is the best (i.e., has the highest rank score).

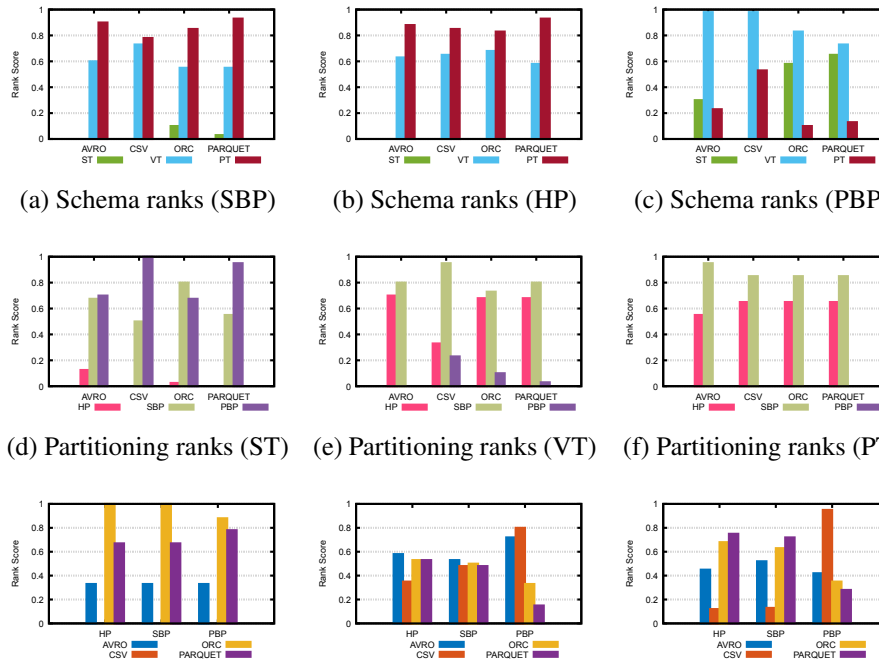
In the following, we will consider the ranking for each dimension independently (i.e $R \xrightarrow{as} \mathscr{R}_s$ for ranking relational schema, $R \xrightarrow{as} \mathscr{R}_p$ for ranking partitioning techniques, $R \xrightarrow{as} \mathscr{R}_f$ for storage formats).

5.2.1. Single-Dimensional Ranking Analysis Results

In the following figures, we show the performance of each dimension's (i.e., schema, partitioning, and storage) parameters in terms of *rank scores*. For simplicity, we only show the largest dataset scale (i.e., 500M). However, we keep all the figures of the other two datasets (100M, and 250M) on the mentioned *GitHub* repository page.

Figure 32 and Figure 33 show how many times a particular dimension achieves the highest or the lowest ranking scores, respectively, considering the results of all the experiments in SP²B and WatDiv benchmarks, respectively. Particularly, Figures (*a*), (*b*), and (*c*) show the ranking scores of the relational schema performance alongside various storage formats pivoting different partitioning techniques SBP, HP, and PBP, respectively.

the storage formats, we pose $d = 4$, as we have *four* different storage file formats (i.e, CSV, AVRO, Parquet, ORC).



(a) Schema ranks (SBP) (b) Schema ranks (HP) (c) Schema ranks (PBP)

(d) Partitioning ranks (ST) (e) Partitioning ranks (VT) (f) Partitioning ranks (PT)

(g) Storage ranks (ST) (h) Storage ranks (VT) (i) Storage ranks (PT)

Figure 33: Single-dimensional ranking scores for the *WatDiv 500M* triples datasets (The higher the better).

Figures (*d*), (*e*), and (*f*) show the ranking scores of the partitioning techniques' performance alongside various storage formats pivoting different relational schemas ST, VT, and PT, respectively. Last but not least, Figures (*g*), (*h*), and (*i*) show the ranking scores of the storage formats' performance alongside various partitioning techniques pivoting different relational schemas ST, VT, and PT, respectively. Notably, when a rank score of a dimension is *zero* (i.e., it always comes at the last rank [i.e., 3^{rd} rank in the case of schema or partitioning, 4^{th} rank in the case of storage dimension], see Table 20), we omit it from the figure.

SP^{2}B SD Ranking: First, we start with the SP^{2}B single-dimensional rankings. Specifically, schema ranking scores for the 500*M* datasets (Figures 32 (*a*), (*b*), and (*c*)), we observe that the ST schema is always the worst performing with 100% (i.e., in all cases) when HP and SBP are chosen as partitioning techniques. However, the ST schema has the second-highest scores after the VT schema for the PBP partitioning with 50% of all cases. On the other side, the VT schema has the highest rank scores by 100% in the ranking scores of the relational schemata. The PT schema performance ranking falls between the VT schema and the ST schema by more than 83%. The observations are still confirmed for the 100*M* and 250*M* triples dataset (check the repository page).

Figures 32 (*d*), (*e*), and (*f*) show the different partitioning techniques rankings. In the 500*M* triples dataset, the SBP technique outperforms the other techniques with more than 58% of the ranking times. The HP is performing the worst in most ranking times, with more than 83% of the ranking times. It comes as the top-ranked technique using *Avro* file format. However, in the PT schema, the PBP technique always beats the HP (i.e., taking

its second rank position with 100%). The same pattern is still valid for the smaller datasets (i.e., 100*M*, and 250*M*), showing that the SBP is the top-ranked technique. However, the PBP technique interestingly performs as the worst technique. On the other hand, the performance of the HP technique is somewhere between the other two techniques. Even in the 100*M*, HP surprisingly outperforms the SBP having a higher rank using the PT schema and Avro file format.

Last but not least, Figure 32 (*g*), (*h*), and (*i*) show the storage ranking scores. In the 500*M* triples datasets using the ST relational schema and HP and SB partitioning techniques, we observe that HDFS *ORC* is the best performing backend with 100%. The *Parquet* format performance immediately follows. On the other hand, *CSV* and *Avro* file formats are respectively the worst performing on HDFS with 100% of the mentioned cases. However, for the lower dataset (100*M*) using the ST schema and the PBP partitioning technique, *Avro* is the second best-performing storage after *ORC*. Using the VT schema and HP and SBP partitioning techniques, we can still observe that *ORC* and *Parquet* share the best-performing rank. HDFS *CSV* and *Avro* file formats respectively keep performing the worst, having the lowest rank scores with 100% of these mentioned cases. However, with the PBP partitioning in the smaller datasets (100*M*, and 250*M*) with the VT schema, *Avro* is the best performing backend, followed by Parquet.

WatDiv SD Ranking: Second, we move to the WatDiv single-dimensional rankings. Herein, we discuss the rankings of the three datasets for the three dimensions. However, we show only the ranking figures of the largest dataset (i.e., 500*M*). We put the ranking plots of the other datasets on the repository page.

Regarding the schema dimensions rankings, ST shows as the worst-performing schema (i.e., has the lowest ranking score) with 100% using the HP partitioning technique. Interestingly, ST performs better than VT schema in the 100*M* dataset (with the SBP partitioning and columnar file formats ORC and Parquet). Moving to the SBP, and PBP, the performance of the ST schema is still the worst across the different dataset scales. However, the ST schema shows a very competitive performance compared to the other two schemas with the PBP technique and the columnar file formats, especially in the 250*M* dataset. However, scaling up to the 500*M* dataset (Figures 32 (*a*), (*b*), and (*c*)), it shows as the second best-performing schema after the VT schema with columnar file formats and the row-oriented Avro file format. On the other side, the PT schema is the best-performing schema with 100% of the ranking cases when using the HP or SBP techniques and with using any storage file format across the different scales of the datasets. However, the PT schema performance significantly degrades when using the PBP partitioning technique, especially with the columnar file formats (ORC and Parquet). Nonetheless, in the row-oriented file formats, it is still better than ST schema only with the CSV file format. However, the ST schema performs better than it with the Avro file format across scales of the datasets.

Regarding the partitioning techniques rankings, it is seen that the HP technique shows the worst ranking scores with the ST schema with more than 92% of the ranking times across the dataset scales. Moving to the results of HP with the VT schema, it shows a better performance, i.e., it ranks as the second-performing technique by 50% of the ranking cases. Interestingly, it is shown as the best-performing technique in the 250*M* dataset with Parquet file format and the VT schema. On the other side, The PBP partitioning technique showed the worst-performing technique with the PT schema with 100% of the ranking cases and across all the dataset sizes. However, it shows a very competitive performance with the ST schema. In particular, it showed the second best-performing technique with

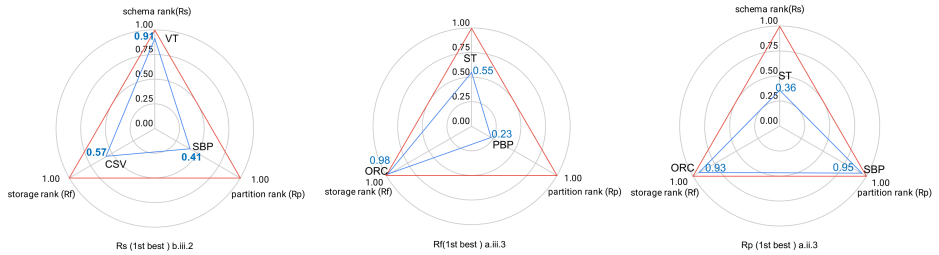
Dataset \ $\mathscr{R}^3_x$	SP^{2}B				WatDiv		
	$\mathscr{R}^3_f$	$\mathscr{R}^3_p$	$\mathscr{R}^3_s$		$\mathscr{R}^3_f$	$\mathscr{R}^3_p$	$\mathscr{R}^3_s$
100*M*	a.iii.**3** a.ii.**3** a.i.**3**	a.**ii**.3 a.**ii**.4 c.**ii**.2	**b**.iii.2 **b**.iii.1 **b**.iii.4		a.ii.**3** a.i.**3** b.ii.2	c.**ii**.2 b.iii.3 c.**ii**.1	**b**.iii.2 **b**.iii.1 c.i.4
250*M*	a.iii.**3** a.ii.**3** b.ii.4	b.**ii**.3 c.**ii**.3 c.**ii**.1	**b**.iii.1 **b**.iii.2 **b**.iii.3		a.ii.**3** a.i.**3** b.iii.2	a.**ii**.1 a.**ii**.2 c.i.4	**b**.iii.2 **b**.iii.1 c.i.4
500*M*	a.ii.**3** a.iii.**3** c.ii.4	c.**ii**.4 c.**ii**.3 c.**ii**.1	**b**.iii.1 **b**.iii.2 **b**.iii.4		a.ii.**3** a.i.**3** c.iii.2	a.iii.2 a.iii.4 c.**ii**.1	**b**.iii.2 **b**.iii.1 c.ii.4

Table 21: Top-3 configurations of the *rank sets* ($\mathscr{R}^3_f$, $\mathscr{R}^3_p$, $\mathscr{R}^3_s$) across datasets. For example the Top-3 confs ranking by format ($\mathscr{R}^3_f$) in 100M are from top to down (aiii.3, aii.3, ai.3).

the ST schema by more than 58%. Moreover, it showed the best-performing technique with those schemata (ST, and VT) by more than 33%, and 25% for the ST and VT, respectively. However, the PBP technique still shows a significant low performance with the VT schema across all the dataset scales. In particular, it showed the worst-performing technique by more than 66.5% of all cases.

Regarding the storage formats rankings, for all scales of the datasets and for the ST schema, we observe that columnar file formats (ORC and Parquet) significantly outperform the row-oriented file formats (CSV, and Avro) with 100% of the ranking cases. Moreover, we observe that CSV is the worst performing file format with 100% (its bar is always absent from the ST figures for ranking the storage formats). However, moving to the VT and PT schema, we observe remarkable performance enhancement for the row-oriented file formats over the columnar ones. The row-oriented formats showed as the best performing formats with almost 89% of the ranking cases. In particular, CSV shows as the best-performing one with the VT schema by more than 78% of the rankings across the VT figures (data scales), directly followed by Avro (i.e., as the second best-performing format) by more than 44% of the cases. Interestingly, Avro shows as the best performing format with the HP and SBP partitioning techniques when scaling to the 500*M* dataset. On the other side, ORC and Parquet are yet competing Avro on the second best-performing rank (i.e., coming after CSV) in this schema (i.e., VT). They beat Avro on that goal only by 30% of the cases. The same trend is still valid for the row-oriented file formats with the PT schema, but with less significance. In particular, CSV is the best performing format only with 55.5% of the ranking cases. Columnar file formats show a better performance with the PT schema than with the VT schema. In particular, the Parquet file format shows a very competitive performance compared to the other file formats (i.e., it outperforms the other file formats by more than 44% of the ranking cases of the datasets). On the other side, Avro and ORC compete to get the second position across the data scales. However, Avro significantly outperforms ORC by more than 66% of the ranking cases across datasets.

Table 21 shows the top-3 configuration combinations according to the single-dimensional ranking criteria. Moreover, the table highlights the best-performing dimension (marked



(a) Ranking by Schema (b) Ranking by Storage (c) Ranking by Partitioning

Figure 34: Example of dimensions' trade-offs effect on single-dimensional ranking criteria ($\mathscr{R}_s$,$\mathscr{R}_p$, and $\mathscr{R}_f$).

by the green color across the same dimension, i.e., vertically). For example, in the SP^{2}B benchmark, ranking by schema we mark *VT (b)* as the best; ranking by partitioning, we mark the *SBP (ii)*. Finally, we can roughly mark *ORC (3)* followed by *Parquet (4)* are the best ones when ranking by format. Similarly, in WatDiv benchmark, ranking by the schema marks roughly the *VT (b)* schema as the best followed by *PT (b)* ranking by partitioning also marks roughly the *SBP (ii)*, and the following is *PBP (ii)* as the best-performing techniques. Last but not least, we can roughly mark *ORC (3)* followed by *CSV (2)* as the best ones when ranking by storage format. However, ranking over one dimension (using those single-dimensional criteria) and ignoring the other dimensions ends up with selecting different configurations.

Figure 34 shows the single-dimensional ranking criteria w.r.t a simple geometrical representation that depicts the triangle subsumed by each ranking criterion ($\mathscr{R}_s$, $\mathscr{R}_p$, and $\mathscr{R}_f$). The triangle sides present the trade-offs ranking dimensions. The *red* triangles represent the full ranking optimization, i.e, full rank scores, $R_x = 1$. The blue triangles in the plots represent the *actual* ranking scores for the selected configurations. Single-dimensional ranking criteria maximize the score for only one dimension while ignoring the other two dimensions. For instance, ranking by schema dimension in Figure 34 (a) shows how the schema dimension is perfectly optimized, while ignoring optimizations of the other two sides (dimensions). The same effect of trade-offs is shown in Figure 34 (a) , and (b) for ranking by storage and partitioning, respectively.

5.3. Multi-Dimensional (MD) Ranking Criteria

While single-dimensional ranking supports explanations of the results [ANS18], we observed that ranking prescriptions are incoherent across dimensions. The most reasonable explanation is that single-dimensional ranking criteria cannot capture a general view. Indeed, single-dimensional ranking criteria neglect the presence of trade-offs as they rank alongside a single experimental dimension (i.e., showing which parameters under a specific dimension are the best-performing). For instance, ranking by the schema (i.e., by $\mathscr{R}_s$) aims at finding the best ranking scores that optimize the performance of the schema dimension while ignoring the other dimensions' performance. This intuition leads to extend the *Bench-ranking* into a *multi-objective optimization* problem in order to optimize all the dimensions at the same time.

In our experiments, we adopt the standard *Pareto front* optimization techniques to consider all the experimental dimension altogether [NZE05; RK15]. The Pareto front framework conducts the MO optimization over several dimensions in order to find the

		SP2B									
		1st	**2nd**	**3rd**	**4th**	**5th**	**6th**	**7th**	**8th**	**9th**	**10th**
500M	Par_Agg	c.ii.4	b.ii.4	b.iii.1	b.iii.3	b.iii.4	b.ii.3	a.ii.3	a.iii.3	b.i.4	-
	Par_Q	b.ii.3	b.ii.4	b.iii.3	b.iii.4	b.iii.1	b.i.4	b.ii.1	c.ii.4	a.ii.3	c.ii.3
		WatDiv									
		1st	**2nd**	**3rd**	**4th**	**5th**	**6th**	**7th**	**8th**	**9th**	**10th**
500M	Par_Agg	c.ii.4	a.iii.4	c.ii.1	c.i.4	a.iii.3	b.iii.2	a.ii.3	c.iii.2	a.iii.2	-
	Par_Q	c.ii.4	c.i.4	c.ii.3	c.ii.1	c.i.3	c.i.1	b.ii.1	b.ii.2	c.ii.2	b.i.1

Table 22: Top-10 Pareto solutions for the 500*M* dataset for SP^{2}B and WatDiv.

best possible parameter configurations for the learning outcome of the highest performance [Xu19; Bar+18; RK15]. In practice, Pareto front aims at finding a set of *non-dominated*/optimal solutions if no objective can be improved without sacrificing at least one other objective. On the other hand, a solution *X* is pointed as *dominated* by another solution *Y* if, and only if, *Y* is equally good or better than *X* with respect to all the objectives. In particular, we utilized the *Non-dominated Sorting Genetic Algorithm (NSGA-II)* [Deb+02a] as one of the most popular Pareto front algorithms in order to find the best configuration combinations in our complex experimental solution space. The *NSGA-II* is a modified version of the *NSGA* algorithm that is widely used in many real-world applications. Its main procedure is sorting the search space using the non-dominated procedure [Deb+02a]. Its technique is powerful–yet simple, not only in capturing a global search space and finding well-distributed Pareto fronts but also in reducing the time to solve complex MO optimization problems as with further details mentioned by *Deb et.al.* [Deb+02b]. Furthermore, this algorithm can handle any arbitrary number of dimensions and objective functions.

5.4. Multi-Dimensional Criteria Results

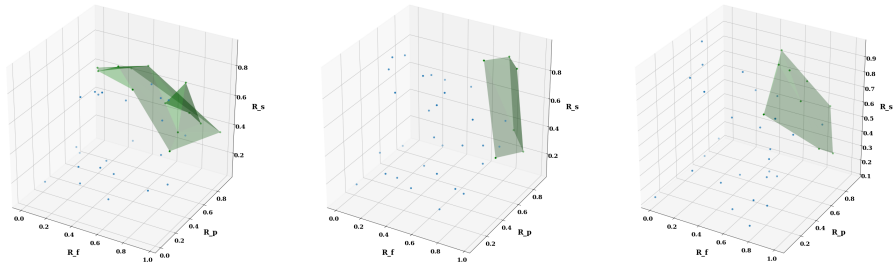
In contrast to SD ranking criteria that optimize one dimension at a time, MO optimization techniques (e.g., Pareto *NSGA-II*) aim to find the optimal non-dominated solutions (i.e., configuration combinations in our case) by optimizing all dimensions at the same time.

In our experiments, we calculate the Pareto fronts for our Bench-Ranking problem in two ways. The first way, which we call ($Pareto_Q$), applied *NSGA-II* algorithm considering the rank sets obtained by sorting w.r.t each query result individually (see Table 19). The algorithm aims at *minimizing* the query runtimes globally.

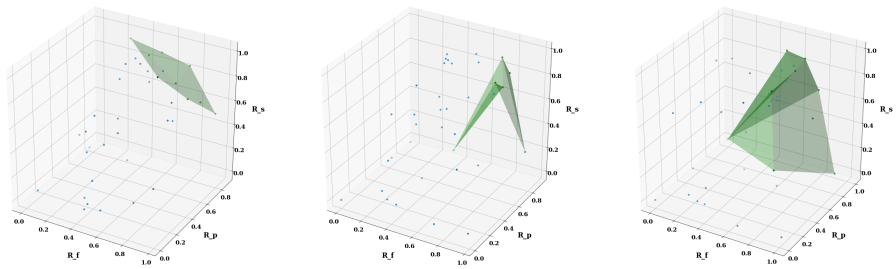
The second way, called ($Pareto_{Agg.}$), operates on the single-dimensional ranking criteria, i.e., $\mathscr{R}_s$, $\mathscr{R}_p$, and $\mathscr{R}_f$. In this case, the algorithm aims at maximizing the performance of the three ranks altogether. Notably, for both implementations of the algorithm, the search space is given in the form of configurations' query runtimes (configurations query positions, see Table 19), or in configurations' rank scores ($\mathscr{R}_s$, $\mathscr{R}_f$, $\mathscr{R}_p$) in the aggregated form of Pareto, instead of generating a search space. Table 22 shows the *top-10* Pareto fronts for both approaches, i.e., non-dominated solutions, which correspond to the overall optimal configurations for the 500*M* dataset in both of the benchmarks. For conciseness, we keep the results of Pareto for the other two datasets (100*M*, 250*M*) in the project repository.

Figures 35 (a)-(f) show the Pareto fronts (depicted by the green shaded areas) of the three dimensions of the Bench-ranking for $Pareto_{Agg.}$[3] for both of the benchmarks, i.e.,

[3] $Pareto_Q$ cannot be visualized as it uses 11 dimensions one for each of the SP^{2}B benchmark



(a) SP^{2}B (100M) (b) SP^{2}B (250M) (c) SP^{2}B (500M)



(d) WatDiv (100M) (e) WatDiv (250M) (f) WatDiv (500M)

Figure 35: Aggregated Pareto-Fronts for the SP^{2}B and WatDiv Benchmark datasets.

SP2B and WatDiv. Each point of those figures represents a solution of rank scores (i.e, configuration in our case).

5.5. Evaluating Ranking Criteria

This section focuses on answering the question of what makes a ranking criterion adequate for our purposes. This problem is well-known in *Information Retrieval* (IR) applications, where several metrics, e.g., *precision* and *recall*, are used to validate the ranking. Nevertheless, the existing *IR* approaches often require a ground truth, which may lack in the context of performance analysis. The most reasonable solution is to employ multiple ranking criteria and compare the prescriptions with the actual experimental results. However, this approach falls back to the problem related to *ranking consensus*[4] [LC15].

In *Bench-ranking*, we can consider a ranking criterion *"good"* if it does not suggest a *low-performing* configuration. In other words, we are not interested to be the best at any particular query as long as we are never the worst. To this extent, we identified two evaluation metrics, namely ***conformance*** and ***coherence***.

Definition 11. *Ranking conformance is a measure of the adherence of the top-ranked configurations according to the actual query positioning of those configurations. To measure the conformance, we propose Equation 5.2.*

queries.

[4]Ranking consensus is related to choosing between two preference sets.

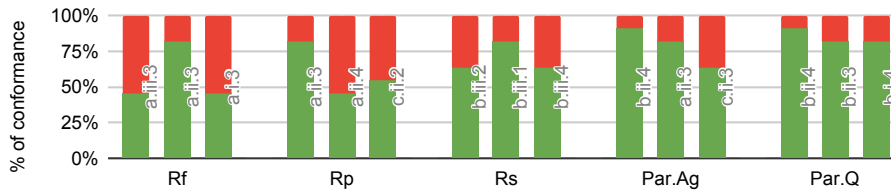


Figure 36: Ranking criteria *conformance* for the top-3 ranked configurations.

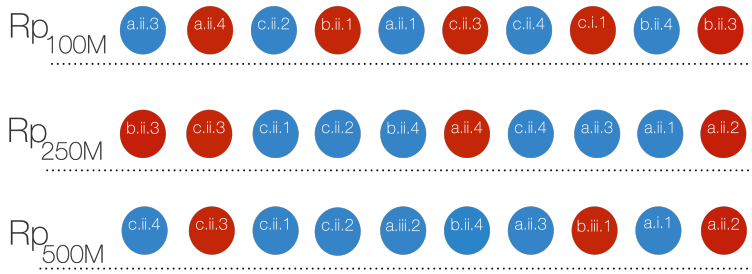


Figure 37: Ranking by Part. coherence example across dataset scales.

$$A(\mathscr{R}^k) = 1 - \sum_{i=0}^{|Q|} \sum_{j=0}^{k} \frac{\bar{A}(i,j)}{|Q| * k}, \quad \bar{A}(i,j) = \begin{cases} 1 & \mathscr{R}^k[j] \in \mathscr{Q}i_h^i \\ 0 & otherwise \end{cases} \tag{5.2}$$

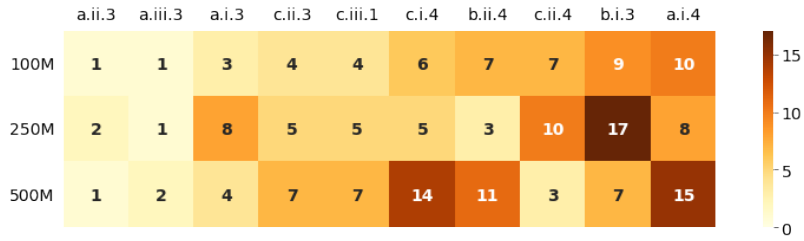
Given the *top-**k*** subset of the ranking set $\mathscr{R}$, we count how many times its elements occur in the *bottom-**h*** subset of the ranking set $\mathscr{Q}_h^i$, which corresponds to the ranking set obtained by using the execution time of query Q_i as ranking criterion for each query (see Table 19). For instance, let's consider the R_s ranking and the $100M$ dataset evaluation. The *top-3* ranked configurations are $\mathscr{R}_s^3$={b.iii.2,b.iii.1,**b.iii.4**} , that overlaps only with the *bottom-3* ranked configurations in query $Q2$. This can be checked in Table 19 that shows sample of actual rankings of the configuration combinations based on the query runtimes. $\mathscr{Q}_2^{h=3}$={b.iii.3,**b.iii.4**,a.iii.2}, i.e b.iii.4 is in the 35^{th} position out of 36 ranks/-positions (i.e., the rank before last). Thus, $A(\mathscr{R}_s^3) = 1 - 1/(11 * 3)$.

To give the intuition of the conformance metric, Figure 36 shows the level of *conformance* of the top-ranked *three* configurations (see Definition 11), such that, the *green* color represents the level of conformance.

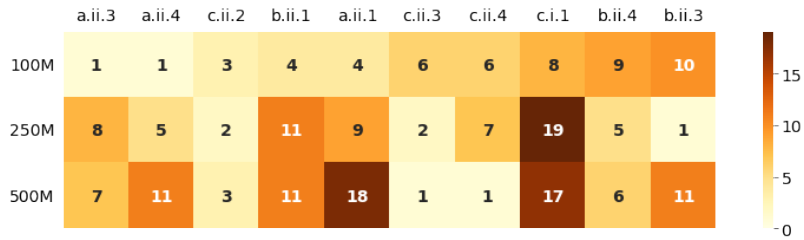
Definition 12. *Ranking coherence is a measure of the level of agreement between two ranking sets that use the same ranking criterion across different experiments (e.g., a dimension of scalability).*

To measure the *coherence* of a ranking criterion, we opt for *Kendall* index, which is a common measure to compare the outcomes of ranking functions [GM20]. Particularly, it counts the number of pairwise disagreements between two rank sets: the larger the distance, the more dissimilar the rank sets are. On a side note, we assume that rank sets have the same number of elements. Kendalls distance between two rank sets $\mathscr{R}1$ and $\mathscr{R}2$, where P represents the set of unique pairs of distinct elements in the two sets, can be calculated using the following equation:

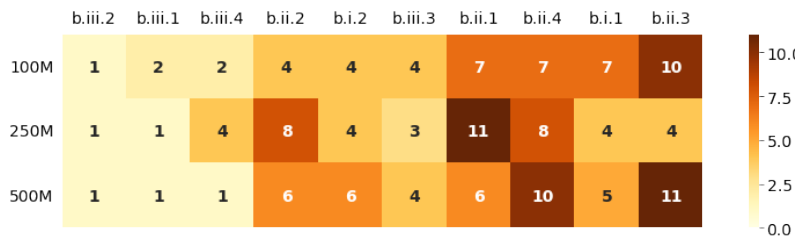
$$K(\mathscr{R}1, \mathscr{R}2) = \sum_{\{i,j\} \in P} \frac{\bar{K}_{i,j}(\mathscr{R}1, \mathscr{R}2)}{|P|} \tag{5.3}$$



(a) Storage Format ranking ($\mathscr{R}_f$) coherence across datasets



(b) Partitioning ranking ($\mathscr{R}_p$) coherence across datasets



(c) Schema ranking ($\mathscr{R}_s$) coherence across datasets

Figure 38: Coherence Heat-Maps for the SP2B experiments.

$$\bar{K}_{i,j}(\mathscr{R}1,\mathscr{R}2) = \begin{cases} 0 & \mathscr{R}1[r_i^1] = \mathscr{R}2[r_i^2] = i \wedge \mathscr{R}1[r_j^1] \\ & = \mathscr{R}2[r_j^2] = j \wedge \\ & r_i^1 - r_j^1 = r_i^2 - r_j^2 \\ 1 & \text{otherwise} \end{cases}$$

For instance, the K index between $\mathscr{R}_f^3$ (top-3 ranked configurations) for $100M$ and $250M$ is 0.33, i.e., one disagreement out of three pair comparisons (Table 21). Figure 37 gives the intuition of the coherence metric by showing the *top-10* ranked configuration for the same ranking criterion($\mathscr{R}_p$) (i.e., $\mathscr{R}_p^{10}$) and across three different data scales. We can see examples of pairwise disagreements that occur by scaling from $100M$ to $250M$, and also from $100M$ to the $500M$ dataset. For instance, in ($100M$-to-$250M$), *b.ii.3* was at the 10^{th} rank in $100M$, while being swapped to be at the 1^{st} position in the $250M$, and the 1^{st} ranked configuration (*a.ii.3*) in the $100M$ swapped to be at the 8^{th} rank in the $250M$. Similar kind of disagreements are shown in the ($100M$-to-$500M$) scale-up transition. Moreover, Figure 38 shows heat maps of the (dis)agreements on the top configurations of SD criteria (a) ranking by format (b) ranking by partitioning, and ranking by schema while scaling up to larger datasets.

Criteria	SP^{2}B				WatDiv		
	100M	**250M**	**500M**		**100M**	**250M**	**500M**
$\mathscr{R}_f$	58%	82%	70%		67%	50%	7%
$\mathscr{R}_p$	61%	70%	58%		85%	33%	37%
$\mathscr{R}_s$	70%	79%	79%		82%	78%	75%
$Pareto_{Agg}$	**79%**	**100%**	82%		**88%**	75%	68%
$Pareto_Q$	**85%**	**100%**	**97%**		**95%**	**98%**	**97%**

Table 23: Ranking criteria *conformance* across datasets, k=3, h=17.

5.5.1. Ranking Criteria Evaluation Results

In this section, we discuss the results of the ranking criteria evaluation metrics, i.e. we calculate the *conformance* and the *coherence* for these selected ranking criteria using Equation 5.2 and Equation 5.3, respectively.

Table 23 shows the conformance of each ranking criterion *top-3* configurations not being worse than the worst 17 ranked configurations (i.e., better than the 17 ones, half of the distribution) according to the queries' ranked sets (C.F., Table 19). In other words, the parameters of Equation 5.2 are arbitrary chosen to be $k = 3$ and $h = 17$, which can be of any other choice. For instance, if the practitioner aim the top-ranked *five* configurations not being worse than the lowest performing *five* configurations, then parameters will be $k = 5$ and $h = 31$ (total number of configurations is 36).

All the selected ranking criteria perform very well for all the datasets. However, the single-dimensional criteria $\mathscr{R}_f$, $\mathscr{R}_p$, and $\mathscr{R}_s$ have lower conformance than multi-dimensional ranking criteria (i.e., the ones based on Pareto). For instance in the SP^{2}B , in the 100M, 250M, and 500M datasets, $Pareto_{Agg.}$ has a conformance of 79%, 100%, and 82%, respectively. The same pattern repeats with the $Pareto_Q$ version (with 85%, 100%, and 97%, respectively). In contrast, single-dimensional ranking criteria have relatively lower conformance of 58%, 82%, and 70% for $\mathscr{R}_f$, 61%, 70%, and 58% for $\mathscr{R}_p$, and 70%, 79%, and 79% for $\mathscr{R}_s$, accordingly. Turning to the WatDiv benchmark, the same trend still valid for the $Pareto_Q$ criterion (i.e., having a better conformance than the single-dimensional ones). Although this is also valid with the $Pareto_{Agg}$ in the 100M dataset, scaling up to the 250M, and 500M slightly degrades the conformance of $Pareto_{Agg}$ in comparison to the $\mathscr{R}_s$ criterion. The main reason behind these results is that single-dimensional criteria do not consider trade-offs across experimental dimensions, ultimately selecting the configuration that may *under-perform* in some queries. Meanwhile, *Pareto-based* MD ranking considers those *trade-offs* while optimizing all the dimensions simultaneously.

We measure the *coherence* of the ranking criteria across different experiments (dataset sizes). Table 24 shows the results where the reading key is *the lower is the better* (i.e., high Kendall's index means high disagreement across two *rank sets*). All the ranking criteria show high coherence across different scales of the datasets. In particular, we notice small distances (i.e. indicating low swaps/disagreements in the ranking ordinals, C.F., Figure 37) for both single and multi-dimensional ranking criteria. Indeed, scaling the datasets does not excessively impact the rank sets' order in all the ranking criteria in both of the SP^{2}B and WatDiv benchmarks.

Coherence (Kendall)	SP^2B				
Di-Dj	$\mathscr{R}_f$	$\mathscr{R}_p$	$\mathscr{R}_s$	**Pareto**$_{Agg.}$	**Pareto**$_Q$
100M-250M	0.13	0.18	0.06	0.19	0.24
100M-500M	0.16	0.29	0.06	0.23	0.16
250M-500M	0.13	0.19	0.07	0.13	0.18
Di-Dj	*WatDiv*				
100M-250M	0.21	0.25	0.08	0.2	0.13
100M-500M	0.35	0.35	0.11	0.28	0.18
250M-500M	0.19	0.23	0.07	0.2	0.09

Table 24: Coherence results across datasets for SP^{2}B & WatDiv benchmarks.

5.6. Discussion

There is still a lack of prescriptive and quantitative analytics in benchmarking the performance of BD systems. Bench-ranking takes the first steps in filling this research gap. In particular, it shows the value of ranking criteria evaluating BD frameworks. This framework reveals its importance in the use cases that involve several experimental dimensions with inherent performance trade-offs. For instance, processing large RDF datasets with a relational BD system (e.g., Spark-SQL) that includes several dimensions, i.e., relational schemata, partitioning techniques, and data formats.

Bench-ranking criteria provide an accurate yet simple way that supports the practitioners in their evaluation task even in the existence of dimensions' trade-offs. Moreover, we propose two metrics to evaluate the ranking criteria *goodness*, i.e., the ranking *conformance* and *coherence*. Looking at the problem as an MD optimization problem allows for capturing more accurate and general reasonable configuration combinations. Indeed, the Pareto front-ranking criteria showed more conformance for its suggested ranked configurations than the single-dimensional ranking criteria. The proposed Pareto criteria handle dimensions' trade-offs efficiently. Moreover, the multi-dimensional ranking using such techniques is scalable to any arbitrary number of dimensions for optimization.

5.6.1. Bench-Ranking Opportunities and Further Improvements

There are several ways to build on top of the Bench-ranking framework for further improvements. We discuss these opportunities in this section with three possible scenarios.

The first is regarding the used case study that is categorical and very coarse. In other words, we did not investigate configurations that combine hybrid schemata or storage formats; due to their complexity. Nevertheless, this can be taken into consideration as a new parameter under the schema dimension. For example, we can have a mix of RDF relational schemata, i.e., VP and PT [CFL18], or a mixture of storage formats for reading the data tables from HDFS.

The second point relates to the ranking functions, where the proposed approach is not workload-aware. Several alternative ways exist to model the workload within the ranking. For instance, further investigation could consider some query complexity dimensions. However, consideration related to the query algebra would require examining the actual query plan, which is inherently *system-specific*, which contradicts the *portability* requirement (Chapter 3 **R.4**). In the future, we plan to design a *workload-aware*

ranking based on benchmark *choke-points* [5] [Sal+16], which are sufficiently high-level to remain *system-agnostic*. Moreover, we aim to employ other ranking functions as well as MD optimization techniques and algorithms. For instance, we aim to enrich the Bench-Ranking framework with implementing other techniques based on the *Bayesian* strategy or the *Nelder and Mead* algorithms [SN09].

The last possible improvement relates to the ranking criteria evaluation metrics. Interestingly, such metrics do not leverage any prior knowledge about the workload. For instance, we know that configuration using columnar file formats, e.g., *ORC* should outperform those using the row-oriented formats for queries with few projections. Hence, we plan to design a hypothesis-based evaluation metric that leverages such domain knowledge for evaluating ranking functions. For example, distance metrics can be enriched with penalization/rewarding criteria when a hypothesis is rejected/confirmed.

[5] `https://projects.ics.forth.gr/isl/RDF-Benchmarks-Tutorial/DiscussingRDFBenchmarks.pdf`

6. PAPYA: A TOOL FOR AUTOMATING PRESCRIPTIVE PERFORMANCE ANALYSIS OF LARGE RDF GRAPHS PROCESSING

In this chapter, we answer to the *fourth* micro question that is formulated as follows: **Micro 4**: *How can we automate prescriptive performance analysis of BD systems for processing large graphs with a complex experimental solution space?*

The previous chapter introduced the concepts and techniques of *Bench-Ranking* as a means for enabling *Prescriptive Performance Analysis* for BD systems. PPA is an alternative to descriptive/diagnostic discussions that aims to answer the question *What should we do?* [Lep+20]. In practice, *Bench-Ranking* has shown to be more useful than descriptive and diagnostic analyses, enabling informed decision-making without neglecting the complexity of the performance analysis [RAT21]. Nevertheless, we remark that still determining the best way to execute a SPARQL workload over a large RDF graph using a BD framework is a *time-consuming* task. Our direct experience with big RDF graphs processing shows that the most *time-consuming* phases were the *data preparation* and *performance analytics*. Although the Bench-Ranking methodology (Chapter 5) simplifies the performance analyses, its current implementation is still limited as it does not follow any specific software engineering best practices. Thus, practitioners who implement the Bench-Ranking methodology may face the following challenges:

1. **Experiment Preparation (C1)** requires huge data engineering efforts to build the full pipeline for processing large graphs on top of BD systems, to put the data in the *logical* and *physical* representations that adapt with *relational distributed* environments. Moreover, the current experimental preparation in Bench-Ranking requires incorporating several systems, e.g., *Apache Jena* [1] (i.e., for logical schema definitions), and *Apache SparkSQL* [Mic15] (i.e., for physical partitioning and storage).
2. **Portability and Usability (C2)**: deciding new requirements in the Bench-Ranking framework's current implementation (e.g., changes over the number of tasks (i.e., queries) or changes over the experimental dimensions/configurations.) would lead to repeating vast parts of the work.
3. **Flexibility and Extensibility (C3)** : the current implementation of the Bench-Ranking framework does not fully reflect the flexibility and extensibility of the framework in terms of experimental dimensions and ranking criteria extensibility.
4. **Complexity and Compoundnesss (C3)**: practitioners may find Bench-Ranking criteria and evaluation metrics quite complex to implement. Moreover, the current implementation does not provide an interactive interface that eases interconnections of various modules of the framework (e.g., data and performance visualization).

To mitigate these problems, we design and implement an open-source library called *PAPyA* (*Prescriptive Analysis in Python Actions*), which contributes with (1) reducing the engineering work required for graph processing preparations and data loading; (2) reproducing existing experiments (according to user needs and convenience) for relational processing of SPARQL queries using SparkSQL. This will lead to reducing massive efforts

[1] `https://jena.apache.org/`

Challenges	Requirements	PAPyA Solutions
C1: Experiments Preparation	**R4, R5**	- Data Preparator that generates and loads graph data for distributed relational setups. - User-defined *YAML* configurations files.
C2: Portability and Usability	**R2, R3, R4, R5**	- Data Preperator prepares graphs data ready for processing with any arbitrary relational BD system. - Internals & abstractions enable plugin-in new modules and programmable artifacts. - Checking performance replicability whilst configuration changes.
C3: Flexibility and Extensibility	**R1, R3, R5**	- Allow adding/excluding experimental dimensions. - Allow adding new ranking algorithms. - Flexibility in shortening and enlarging the configuration space.
C4: Compoundness and Complexity	**R1, R4**	- Interactive Jupyter Notebooks. - Variety of data and ranking visualizations

Table 25: Summary of challenges and requirements along with PAPyA solutions.

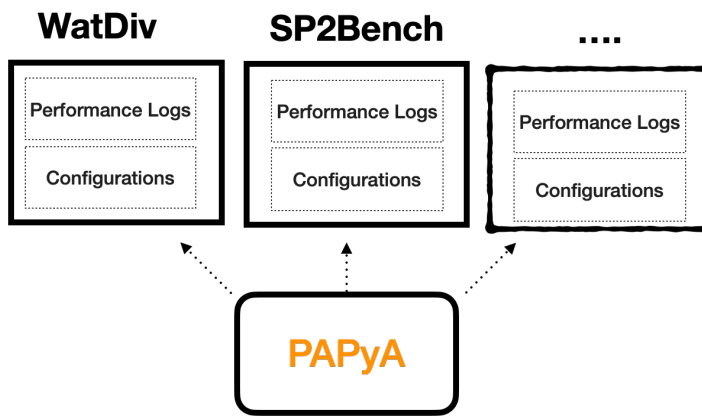


Figure 39: PAPyA dynamicity to provide PPA for RDF benchmarks, and datasets.

for building analytical pipelines from scratch for relational BD systems that are subject to the experiments. Moreover, PAPyA also aims at (3) automating the Bench-Ranking methods for prescriptive performance analysis (described in chapter 5), In practice, PAPyA facilitates navigating the complex solution space via packaging the functionality of different ranking functions as well as *Multi-Optimization* (MO) techniques into *interactive* programmatic library interfaces. Moreover, (4) PAPyA demonstrates the applicability and re-usability of the Bench-ranking methods in actual scenarios. Last but not least, (5) checking the replicability of the relational BD systems' performance for querying large (RDF) graphs within a complex experimental solution space.

The focus of this chapter is to show the internals and functionality of PAPyA as a means for providing PPA for BD relational systems that query large (RDF) graphs.

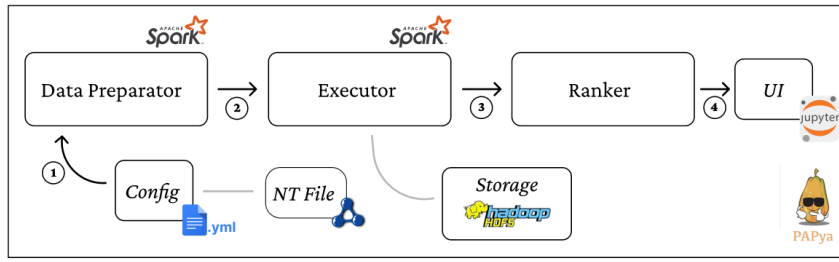


Figure 40: Papaya architecture and workflow.

Thanks to PAPyA, we can easily perform the PPA for any RDF benchmark, pointing out to the performance results, and specifying the experimental configurations (see Figure 39).

6.1. PAPyA Requirements

In this section, we first present the requirement analysis for the design of our PAPyA library, and then we present its architecture. We elicit our requirements for the library based on the above mentioned challenges, and relying on existing research efforts on benchmarking BD systems for processing and querying large RDF graphs [Sch+16; Sch+14; RAT21; Rag+20; CFL18]. Thus, the requirements analysis comprises:

1. **Support for Prescriptive Performance Analysis (R1)**: PAPyA shall support the necessary abstractions required to support PPA. Moreover, by default, it shall support existing Bench-Ranking [RAT21] techniques in chapter 5.
2. **Independence from the Key Performance Indicators (KPIs) (R2)**: PAPyA must enable PPA independently from the chosen KPI. In [RAT21] we opted for query latency, yet one may need to analyze the performance in terms of other metrics (e.g., memory consumption).
3. **Independence from Experimental Dimensions (R3)**: PAPyA must allow the definition of an arbitrary number of dimensions, i.e., allow the definition of *n-dimensional* configuration space. In Bench-Ranking, we opted for relational schemas, partitioning, and data formats, which lead to a *3-dimensional* solution space (C.F., Figure 42).
4. **Usability (R4)**: PAPyA supports decision making, simplifying the analysis of performance data. To this extent, data visualization techniques as well as a simplified API are both of paramount importance.
5. **Extensibility (R5)**: PAPyA should be extensible both in terms of architecture and programming abstractions. It should decouple data and processing abstractions to ease the integration of new components, tools and techniques.

6.1.1. Architecture, Abstractions, and Internals

This section presents the PAPyA's main components and shows how they fulfill the requirements. Table 25 summarizes the requirements to challenges mappings alongside the PAPyA solutions. PAPyA allows its user to build an entire pipeline for querying big RDF datasets and analyzing the performance results. In particular, it facilitates building the experimental setting considering the configuration space (described in Definition 1

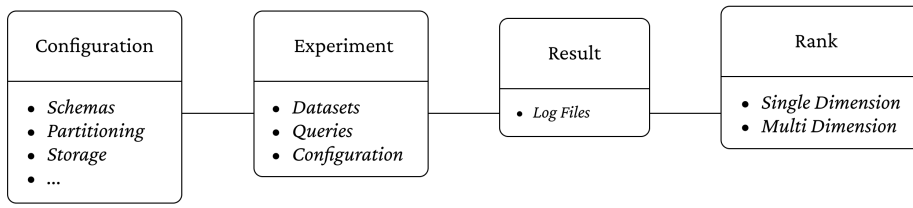


Figure 41: PAPyA internal abstractions.

in Chapter 1) specified by users. This entails preparing and loading the graph data in a *user-defined relational* configuration space, then performing experiments (executing a query workload on top of a relational BD framework), and finally analyzing and providing prescriptions of the performance.

To achieve that, PAPyA includes *three* core modules depicted in Figure 40, i.e., the *Data Preparator*, the *Executor*, and the *Ranker*. Moreover, PAPyA relies on few core abstractions depicted in Figure 41, i.e., *Configuration*, *Experiment*, *Result*, and *Rank*. While detailing each module's functionalities, we introduce PAPyA workflow, which also appears in Figure 40, starting with the input is a configuration file that points to the input *N-Triple* file with the RDF graph (Figure 40 step (A)).

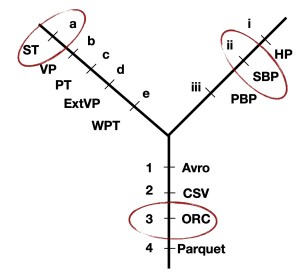


Figure 42: Configuration space $\mathscr{C}$.

The first actor in the pipeline is the *Data Preparator* (DP), which takes as input the configuration files, looks up the experimental dimensions of interest, and loads the RDF dataset. The configuration file is represented by the *configuration* abstraction (see Figure 41), which enables extensibility (R.R5). Indeed, DP allows defining an arbitrary number of dimensions with as many options as necessary (R.R3). In particular, it considers the dimensions specified in Bench-Ranking framework in chapter 5 (R.R1), i.e., relational schemas, storage format, and partitioning technique. Therefore, the DP automatically generates the relational schemas for the input RDF dataset according to the specified configurations.

More specifically, DP currently includes three relational schemas commonly used for RDF processing, i.e., (i) Single Statement (ST) which prescribes to store triples using a ternary relation (subject, predicate, object), often requires many self-joins. ST is used by several native triple-stores like *Apache Jena*, *RDF4J*, and *Virtuoso* [Cho+05], (ii) Vertical-Partitioned Table (VP) proposed by Abadi et.al. [Aba+07] to mitigate issues of self-joins in ST schema proposing to use *binary* relations (subject, object) for each unique predicate in the dataset, (iii) the Wide Property Table (WPT) schema that attempts to encode the entire dataset into a single denormalized table. WPT is initially proposed for *Sempala* system by *schatzle et.al.* [Sch+14], who also proposed (iv) an extended version of the VP schema (ExtVP) [Sch+16] that precomputes *semi-join* VP tables to reduce data shuffling. For partitioning, DP currently supports three partitioning techniques, i.e., (i) horizontal partitioning that divides data evenly over n equivalent chunks where n is the number of machines in the cluster, (ii) subject-based, (iii) predicate-based, that distribute data across various partitions according to the *hash value* computed for the subject or predicate keys, respectively. Finally, DP allows to store data using various HDFS file formats (i) *CSV*

```
from papya import DataPreperator
class SparkDataPreparator(DataPreparator):
  def __init__(self, RDFGraph g):
  def generateST(self,g): # Step (1) in Figure 43
    ST=map(lambda x: x.split("\t"), g)
  def getUniquePredicates(self,ST):# unique property list
    runSQL("SELECT DISTINCT p from ST").toList()
  def generateVP(self,ST): # Step (2) in Figure 43
    uniquePredicates=getUniquePredicates(ST)
    for p in uniquePredicates:
     VPTable=runSQL(f"SELECT s, o From ST WHERE p={p}")
  def generateWPT(self,ST):# Step (3) in Figure 43
    uniquePredicates=getUniquePredicates(ST)
      for p1  in uniquePredicates:
        for p2 in uniquePredicates:
          if(p1 != p2):
            q = f"SELECT s, Pr1.o, Pr2.o
             FROM (SELECT T1.s, T1.o
                   FROM ST T1 WHERE T1.o={p1}) Pr1
             LEFT JOIN (SELECT T2.s, T2.o
                   FROM ST T2 WHERE T2.o={p2}) Pr2"
    wptable=runSQL(q)
  def getRelatedPredicates(self,pred1, relType):
    RelPredsCmd = ("SELECT DISTINCT p FROM ST t1
                    LEFT SEMI JOIN "+
    if (relType == "SS"){
      RelPredsCmd += "(t1.s=t2.s)"}
    else if (relType == "OS"){
      RelPredsCmd += "(t1.s=t2.o)"}
    else if (relType == "SO"){
      RelPredsCmd += "(t1.o=t2.s)"}
    runSQL(RelPredsCmd).toList()
  # Step (4) in Figure 43
  def generateExtVP(self,VP[],relType):
    relpreds = getRelatedPredicates(p1, relType)
    for p2 in relpreds:
      extVPcommand = ("SELECT t1.s, t1.o
                       FROM {p1}  t1
                       LEFT SEMI JOIN  {p2} t2 ON")
        #VP tables joined subject-to-subject
        if (relType == "SS"){
          extVPcommand += "(t1.s=t2.s)"
        } else if (relType == "OS"){#object-to-subject
          extVPcommand += "(t1.o=t2.s)"
        } else if (relType == "SO"){#subject-to-object
          extVPcommand += "(t1.s=t2.o)"}
    extVpTables = runSQL(extVPcommand)
```

Listing 6.1: RDF relational schema generation in PAPyA Data Perperator.

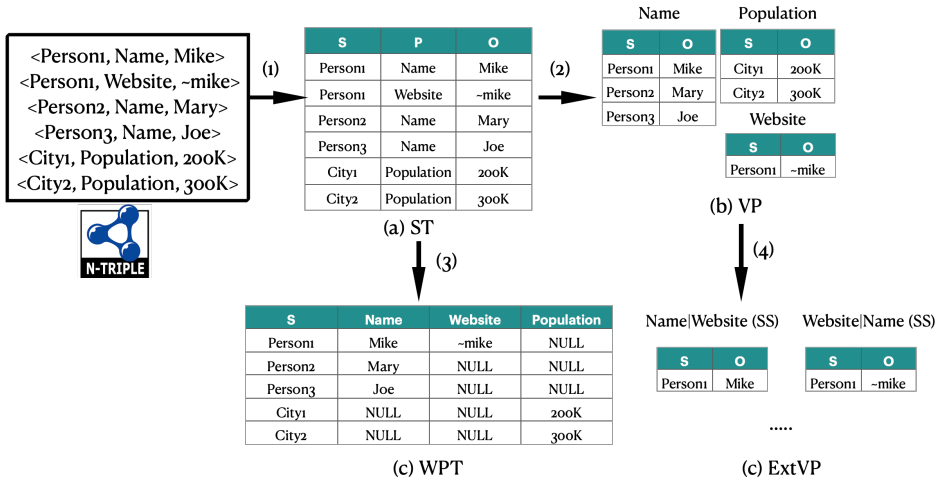


Figure 43: RDF relational schema transformations in PAPyA DP.

or (ii) *Avro*, which are row-oriented, and (iii) *ORC* or (iv) *Parquet*, which are column-oriented.

The DP interface is generic, and the generated data are *agnostic* to the underlying relational system. Seeking scalability, the current DP implementation relies on *SparkSQL*, which allows implementing RDF relational schema generation using the SQL transformations shown in Listing 6.1. Notably, *Apache Hive* or *Apache Impala* could be potential candidates for an alternative implementation executors. However, SparkSQL also supports different partitioning techniques and multiple storage formats, making it ideal for our experiments.

Figure 43 shows sample of schema generation in PAPyA DP component. First, the DP transforms the input RDF graph (N-Triples file(s)) into an ST table schema (i.e., Figure 43 Step(1)), and then other schemas are generated using parameterized SQL queries. For instance, VP and WPT schemas are generated using SQL queries against the ST table (i.e., Figure 43 Step(2), and (3), respectively). While, ExtVP schema generation relies on VP tables to exist first (i.e., Step(4) in Figure 43).

```
1 from abc import ABC, abstractmethod
2 class Executor(ABC):
3     @abstractmethod
4     def run(self, experiment, runs, dataPath, logsPath):
5         pass
```

Listing 6.2: Abstract Executor definition in PAPyA.

The **Executor** is the following module in PAPyA workflow, which is the system that is subject to experimentation (see Figure 40 step B). For instance, in [Rag+21; RAT21; Sch+16; CFL18] the considered system is Apache SparkSQL. The executor offers an abstract API to be extended (R5). In practice, it (i) starts the execution pipeline in the external system, (ii) it collects the performance logs (R2), and currently, it persists them on a file system, e.g., HDFS. The Executor expects a set of experiments to run defined in terms of (i) a set of queries, (ii) an RDF dataset (size), and (iii) a configuration (defined in Definition 1 in Chapter 1). The *Experiment* abstraction is defined in Figure 41. We decide to wrap the running experiments in a *SparkSQL-based* executor (*SparkExecutor*). The experiment specifications (alongside the configurations) are passed to this wrapper

```
from papya import Rank
class SDRank(Rank):
  def __init__(self,dimension, q):#q is number of queries
  def eval(self):
   d = len(self.dimension.index)
   rank_score = []
   for _, row in self.dimension.iterrows():
    for r in range(d):
     s = s + (row[r+1]*(d-(r+1)) / (self.q*(d-1)))
    rank_score.append(s)
   return rank_score
  def plot(self, rank_score):
   display(rank_score)
```

Listing 6.3: SD Ranking Function w.r.t Equation (5.1) in Chapter 5.

as parameters. It is worth mentioning that the Executor assumes the query workload is available in the form of the *SQL* queries [2].

The results logs are then loaded by the **Ranker** component into *python Dataframes* to make them available for analysis (see Figure 40 step (C)). The Ranker reduces the time required to calculate the rankings, obtain useful data visualizations, and determine the *best-performing* configurations while checking the performance replicability. To fulfill R2, i.e., the Ranker component operates over a log-based structure whose schema shall be specified by the user in the input configurations. Moreover, to simplify the usage and the extensibility (R5), we decoupled the performance analytics (e.g., ranking calculation) from its definitions and visualization. In particular, the *Rank* class abstraction reflects on the *ranking function* (Definition 9 in Chapter 5) that takes as input data elements and returns a *ranking set*. To fulfill R4, a default data visualization for the rank shall be specified. However, this is left for the user to specify due to the specificity of the visualization.

```
from papya import Rank
class MDRankPareto(Rank):
  def eval(self, ds, dims): # ds:dataset; dims:dimensions
   # renkSet, function to get ranks of dataframe
   allDim = rankSet(ds,dims).reset_index().values
   inputPnt = np.array(rankSet(ds,dims)[:]).tolist()
   # pass nsga algorithm with prtQ function
   paretoPnts, dominatedPnts = nsga2(inputPnt, prtQ)
   # getConfs sort pareto solutions from best to worst
   paretoQ = sort(paretoPnts, allDim)
   # pass nsga algorithm with prtAgg function
   paretoPnts, dominatedPnts = nsga2(inputPnt, prtAgg)
   paretoAgg = sort(paretoPnts, allDim)
   return paretoQ, paretoAgg
  def plot(self, paretoType):
   diagramPareto(paretoType)
```

Listing 6.4: MD Ranking criterion (Pareto Frontier).

The Rank call allows defining additional ranking criteria (R.R5). In addition, to fulfill R.R1, PAPyA already implements SD as well as *Multi-Dimensional* (MD). Listing 6.3 and Listing 6.4 respectively present the implementation of the generalized ranking for-

[2] In the current stage, PAPyA does not support SPARQL query translation nor SQL query mappings.

```
from papya import Ranker
class BenchRanker(Ranker):
  def conformance(rankSet,q,k, h):
      criteria_table = rankSet.loc[bestOfSD(k)]
      criteria_table = criteria_table[criteria_table>h]
      count = criteria_table.count(axis=1)
      sum = count.sum(axis=0)
      conformance = 1-(sum/(k*q))
      return conformance
  def coherence(rankSet1, rankSet2):
      # count number of (dis)agreements in two rank sets.
         n = len(rankSet1)
         assert len(rankSet2) == n, # equal size of lists.
         i, j = np.meshgrid(np.arange(n), np.arange(n))
         a = np.argsort(rankSet1)
         b = np.argsort(rankSet2)
         kendall_idx = np.logical_or(np.logical_and(a[i]
         < a[j], b[i] > b[j])
         , np.logical_and(a[i] > a[j], b[i] < b[j])).sum()
         return kendall_idx/(n*(n-1))
```

Listing 6.5: Ranking criteria evaluation as per the conformance and coherence metrics (Chapter 5).

mula presented in Equation (5.1) and the implementation of the Pareto front MD ranking criterion. The Equation and in implementation details discussed in Chapter 5.

PAPyA allows its users to interact with the experimental environment (R5) using *Jupyter Notebook*. To facilitate the analysis, it integrates data visualization (R4) that can aid decision-making. Thus, the *Rank* class includes a specific method to implement, where to specify the default visualization.

Finally, to evaluate the raking criteria, we introduced the notions of coherence and conformance (R1) in Chapter 5. Ranking criteria evaluation metrics are employed to select which ranking criterion is *effective* (i.e., if it is not suggesting low-performing configurations). In our experiments, we use such metrics by looking at all ranking criteria and comparing them with the results across different scales, e.g., dataset sizes (100M, 250M, and 500M). Notably, to minimize the dependencies, we implemented the ranking algorithms and evaluation metrics from scratch.

6.2. PAPyA in Practice

In this section, we explain how to use PAPyA in practice, showcasing its functionalities with a focus on performance data analysis and visualization. In particular, we design our experiments in terms of (i) a set of queries (originally in SPARQL and manually translated into SQL accordingly with the different relational schemas), (ii) RDF datasets of different sizes automatically prepared using our *Spark-based DataPreparator*, and (iii) a configuration based on three dimensions as in chapter 5 (i.e., Bench-Ranking framework), i.e., schema, partitioning techniques, and storage formats. We present the results of experiments that use the *WatDiv* benchmark. In our experiments, we evaluate the performance of SparkSQL as a relational engine for evaluating the query workload. In particular, our *KPI* of choice is query *latency*, but it could be easily extended to other metrics, e.g., memory consumption or throughput (triple per second). Our analyses are based on the average

```
1  from papya import Configuration, SparkExecutor
2  from papya import data_preparator as dp
3
4  #Configurations
5  confs = Configuration({
6  "schemas":["ST","VP","WPT","ExtVP"],
7  "partition":["HP,SBP","PBP"],
8  "storage":["CSV","Avro","Parquet", "ORC"]})
9
10 #Executor
11 QW_ST_Schema=[q_ST_1,q_ST_2,..,q_ST_20]
12 exp = dp.experiment(dataset="100M", QW_ST_Schema, confs)
13 spe = SparkExecutor(master="local[*]")
14 res=spe.run(exp,runs=5,dataPath="hdfs:.."
15       ,logsPath="hdfs:..")
16
17 #Bench-Ranking
18 #(1) SD-Ranking Criteria
19 schemaSDRanks = SDRank(res,dim=list(conf.keys())[0],
20 q=len(Q), d=len(list(conf.values())[0]) #schema SDranking
21 partitioningSDRanks = ... #partitioning SDRanking
22 storageSDRanks = ... #storage SDRanking
23
24 #(2) MD-Ranking (Pareto)
25 paretoFronts_Q=MDRankPareto(ds="100M",dims=Q)
26 paretoFronts_Agg=MDRankPareto(ds="100M"
27 ,dims=[schemaSDRanks,partitioningSDRanks,storageSDRanks])
28
29 #Visualization
30 SDRank.plot(schemaSDRanks) #plot SDRanking for schema
31 MDRrankPareto.plot(paretoFronts) #plot MD-Ranking Pareto
32 #Ranking Criteria evaluation
33 conf=Ranker.conformance(schemaSDRanks,q=20,k=3, h=45)
34 coh=Ranker.coherence(schemaSDRanks_100M,schemaSDRanks_250M)
```

Listing 6.6: Experiment design example in PAPyA.

results of five different experiment runs [3].

Listing 6.6 shows a full example of PAPyA pipeline, starting by deciding the configurations (in terms of three dimensions and their options, e.g., list of schemas, partitioning techniques, storage formats to prepare, load, and benchmark) (Listing 6.6 lines 5-10). Then, an experiment is set up for running, defining the dataset size (e.g., "100M" triples), a list of queries to execute or exclude from the workload, and the configurations (Listing 6.6 line 12). An executor is defined for running the experiment along with the number of times experiments will be run (Listing 6.6 line 13). The results (runtime logs) are kept in log files in a specified path (e.g., HDFS or a local disk). The *Bench-Ranking* phase starts when we have the results in logs (Listing 6.6 line 17). It is worth noting that the performance analyses, e.g., Bench-Ranking, could start directly if the performance data (logs) are already present. For instance, we call the *SDRank* (Listing 6.6 line 19) for calculating rank scores for the "schema" dimension, alongside specifying the number of queries ("q"), and number of options under this dimension ("d" in Equation (5.1) Chapter 5). The MD-Ranking(i.e., Pareto fronts) is applier in two ways. The first one is called Pareto$_Q$ (Listing 6.6 line 25), which applies the *NSGA-II* algorithm by considering the ranking sets obtained while sorting each query results individually. Using the first

[3] experiments and queries: https://datasystemsgrouput.github.io/SPARKSQLRDFBenchmarking

Di	WatDivmini					
	Rf	Rp	Rs	ParetoQ	ParetoAgg	Rta
100M	a.ii.3	a.ii.3	c.i.4	c.ii.2	a.ii.3	a.ii.3
	b.ii.2	a.ii.4	c.ii.2	c.i.2	c.ii.2	c.ii.2
	a.i.3	a.ii.2	c.i.3	b.ii.2	b.ii.2	b.ii.2
250M	a.ii.3	a.ii.3	c.i.4	c.i.4	c.i.4	c.i.4
	a.i.3	a.ii.4	c.i.3	c.ii.2	b.ii.2	b.ii.2
	c.i.4	a.ii.2	c.i.2	c.i.2	a.ii.3	a.ii.3
500M	a.ii.3	a.ii.3	c.ii.4	c.ii.3	b.ii.2	b.ii.2
	a.i.3	a.ii.4	c.i.4	c.ii.4	c.ii.3	c.ii.3
	c.i.3	a.ii.2	c.i.3	c.i.3	c.ii.4	c.ii.4

Table 26: WatDiv-mini best-performing (Top-3) configurations according to the SD and MD ranking criteria.

Di	WatDivfull					
	Rf	Rp	Rs	ParetoQ	ParetoAgg	Rta
100M	a.ii.3	c.ii.2	d.iii.4	d.iii.2	c.ii.2	c.ii.2
	a.i.3	b.iii.3	d.iii.1	d.iii.3	d.iii.3	d.iii.3
	b.ii.2	c.ii.1	d.iii.3	d.iii.4	b.ii.2	b.ii.2
250M	a.ii.3	a.ii.1	d.iii.4	d.iii.2	d.iii.3	d.iii.3
	a.i.3	d.iii.3	d.iii.3	d.iii.3	c.i.4	c.i.4
	b.iii.2	a.ii.2	d.iii.2	c.i.4	a.iii.4	a.iii.4
500M	a.ii.3	c.ii.2	d.iii.2	d.ii.2	d.ii.2	c.ii.3
	a.i.3	a.iii.2	d.iii.4	c.ii.3	c.ii.3	d.ii.2
	c.iii.2	a.iii.4	d.iii.1	a.iii.4	a.iii.4	a.iii.4

Table 27: WatDiv-Full best-performing (Top-3) configurations according to the SD and MD ranking criteria.

method, the algorithm aims at *minimizing* the query runtimes across all dimensions. The second one is called the Pareto$_{Agg}$ (Listing 6.6 line 26), which operates on the SD ranking criteria. By using the second method, the algorithm aims to *maximize* the rank scores of the three SD-ranking criteria altogether, i.e., R_s, R_p, and R_f. The user can plot the SD rank scores and the MD Pareto ranking criterion (Listing 6.6 line 29). In addition, the user can evaluate the effectiveness of the ranking criterion using conformance and coherence metrics (Listing 6.6 line 32).

Tables 26 and 27 show the *top-3* ranked configuration according to the various ranking criteria, i.e., Single-Dimension and Multi-Dimensional (Pareto) for the WatDiv datasets (i.e., 100M, 250M, 500M triples). In addition, Tables 29 and 28 provide the ranking evaluation metrics.

6.2.1. Rich Visualizations

To fulfill R4, PAPyA decouples data analytics from visualizations. Meaning that the user can specify his/her visualizations of interest with the performance data. Nevertheless, PAPyA yet provides several interactive and extensible default visualizations that help

	WatDivMini					
	Conformance			Coherence		
	D1	D2	D3	D1-D2	D1-D3	D2-D3
Rs	88.33%	91.67%	93.33%	0.09	0.12	0.06
Rp	38.33%	13.33%	5.00%	0.14	0.14	0.14
Rf	63.33%	46.67%	35.00%	0.16	0.39	0.3
paretoQ	95.00%	98.33%	95.00%	0.14	0.25	0.14
paretoAgg	88.33%	73.33%	93.33%	0.16	0.25	0.2
Rta	88.33%	73.33%	93.33%	0.2	0.24	0.17

Table 28: Ranking *Coherence* (Kendall distance, the lower the better) & *Conformance* across $WatDiv_{mini}$ datasets (D1=100M, D2=250M, D3=500M).

	WatDifull					
	Conformance			Coherence		
	D1	D2	D3	D1-D2	D1-D3	D2-D3
Rs	96.00%	94.00%	93.00%	0.1	0.13	0.07
Rp	73.00%	39.00%	36.00%	0.09	0.28	0.17
Rf	64.00%	32.00%	43.00%	0.14	0.25	0.15
paretoQ	92.00%	98.00%	98.00%	0.1	0.15	0.07
paretoAgg	87.00%	71.00%	76.00%	0.17	0.24	0.15
Rta	89.00%	84.00%	76.00%	0.15	0.22	0.13

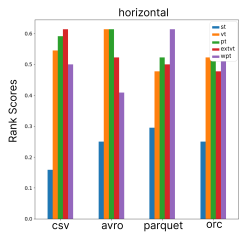
Table 29: Ranking *Coherence* (Kendall distance, the lower the better) & *Conformance* across $WatDiv_{full}$ datasets (D1=100M, D2=250M, D3=500M).

practitioners rationalize the performance results and final prescriptions. In addition, visualizations are simple and intuitive for understanding several Bench-Ranking definitions, equations, and evaluation metrics. For instance, Figure 44 (a-c) shows three samples of SD-ranking criteria plots. In particular, they show how many times a specific dimension's (e.g., schema in Figure 44 (a)) alternatives/options (ST, VP, PT,..etc) achieve the highest or the lowest ranking scores. Figure 34 shows the SD ranking criteria w.r.t a simple geometrical representation (detailed in the following sections) that depicts the triangle subsumed by each dimension's ranking criterion (i.e., Rs, Rp, and Rf). The triangle sides present the trade-offs ranking dimensions and show that the SD-ranking criteria may only optimize towards a single dimension at a time. The MD-ranking criteria, i.e., Pareto_{Agg} [4] results are depicted using a *3D* plot in Figure 46 (a). Pareto fronts are depicted by the green shaded area of the three experimental dimensions of the Bench-Ranking (for WatDiv 500M triples dataset [5]). Each point of this figure represents a *solution* of rank scores (i.e., a configuration in our case).

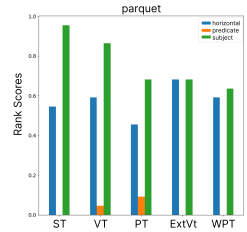
PAPyA visualizations allow explaining the conformance and coherence results using simple plots. For instance, Figure 47 (a) shows the coherence of the top-3 ranked configurations of the Rs criterion in the 100M dataset while scaling to the larger datasets, i.e., 250M and 500M. PAPyA explains the conformance of the Bench-Ranking criteria by visualizing the conformance of the top-3 ranked configurations (or any arbitrary number of configurations) with the actual query rankings (Table 19 in Chapter 5). The green color represents the level of conformance, and the red depicts a configuration is performing worse than the h worst rankings. Thus, this may explain why $\mathscr{R}_p$ and $\mathscr{R}_f$ criteria have low conformance results in Tables 26 and 27 , while the other criteria have relatively

[4] Pareto_Q cannot be visualized, i.e., as it uses more than *three* dimensions, one for each query of the workload [RAT21].

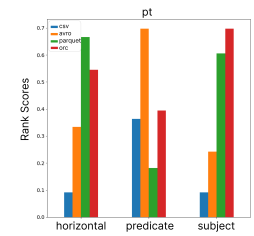
[5] see other Pareto figures in Chapter 5



(a) SD Schema Ranks

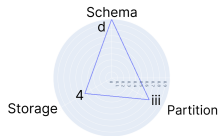


(b) SD Partition Ranks

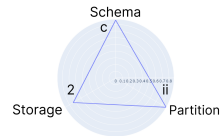


(c) SD Storage Ranks

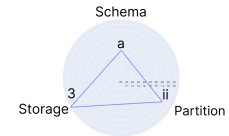
Figure 44: Examples on SD Rank Scores over different dimensions (100M), the higher the better.



(a) Ranking by Schema

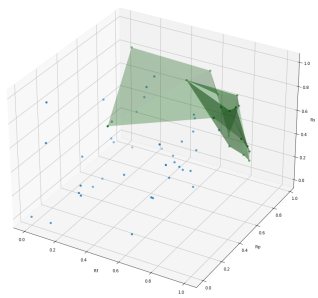


(b) Ranking by Partition

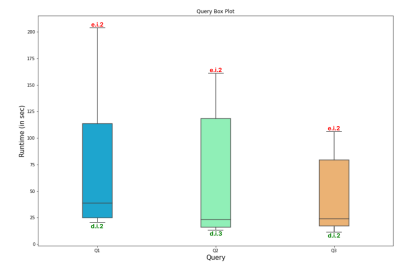


(c) Ranking by Storage

Figure 45: Dimensions trade-offs using single-dimensional ranking (R_s,R_f, and R_p).

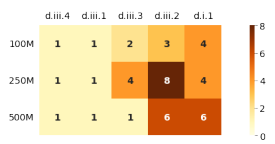


(a) Pareto Fronts for WatDiv$_{full}$

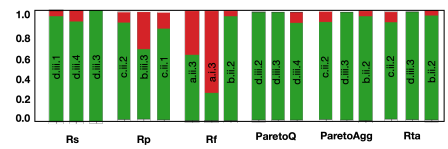


(b) Best and worst-performing configuration for each query

Figure 46: Pareto Fronts, and queries best-worst configuration examples.



(a) Coherence



(b) Conformance

Figure 47: Heatmap shows the coherence of the R_s criterion (Top-5 configurations) scaling from 100M to larger dataset scales. The stacked plot shows the Conformance of the top-3 ranked configurations.

Schema	Full conf. Space	PBP	!PBP	CSV	!CSV
Extvp	0.82	0.96	0.75	0.78	0.83
PT	0.60	0.32	0.74	0.69	0.57
VP	0.55	0.71	0.46	0.74	0.48
ST	0.32	0.52	0.23	0.20	0.37
WPT	0.21	0.00	0.31	0.09	0.25

Table 30: Schemas global ranking across various configurations.

higher conformance values.

Practitioners can also use PAPyA visualizations for *fine-grained* ranking details. For instance, showing the best and worst configurations for each query (as shown in Figure 46 (b) for example of *three* queries of the WatDiv workload). Such detailed visualizations could help the user rationalize the final prescriptions of PAPyA.

6.2.2. PAPyA Flexibility & Extensibility

```
#(1)Full-WatDiv Configurations
dimensions:
  schemas: ["st", "vp", "pt", "extvp", "wpt"]
  partition: ["horizontal","subject","predicate"]
  storage: ["Avro", "CSV", "ORC", "Parquet"]
query: 20
#(2) Mini-WatDiv Configurations
dimensions:
  schemas: ["st", "vp", "pt"]
  partition: ["horizontal", "subject"]
  storage: ["csv", "orc", "parquet"]
query: 10
#(3)WatDiv without Partitioning (i.e.,Centralized)
dimensions:
  schemas: ["st", "vp", "pt", "extvp", "wpt"]
  partition: null
  storage: ["Avro", "CSV", "ORC", "Parquet"]
query: 20
```

Listing 6.7: Configurations yaml file for different experiments in PAPyA.

Adding/Removing Configurations/Queries. To show an example of the extensibility of PAPyA, we implement the Bench-Ranking criteria over a subset of the configurations and subset of the WatDiv benchmark tasks (i.e., queries); we call it $WatDiv_{mini}$. In particular, we run PAPyA Bench-Ranking with WatDiv excluding two schemas (i.e., schema advancements: *ExtVP*, and *WPT*), one partitioning technique (i.e., Predicate-based), and one storage format (i.e., Avro). Then, we include all the configurations back to test the extensibility with the WatDiv experiments (see Tables 27 and Tables 26). The configurations' exclusion and inclusion is specified easily from the *YAML* configuration file (as shown in Listing 6.7 lines 7-12), i.e., PAPyA considers only the specified configurations and rank according to them.

Tables 27 and Tables 26 shows the top-ranked *three* configurations according to the specified configurations in WatDiv-full and WatDiv-mini, respectively. Intuitively, results differ according to the available ranked configuration space. For instance, with the inclusion of the ExtVP schema (i.e., '*d*'), it dominates instead of the PT schema (i.e., '*c*')

in $WatDiv_{mini}$ for ranking by schema (Rs) criterion. In $WatDiv_{mini}$, excluding the Predicate partitioning ('*iii*'), the *subject-based* partitioning ('*ii*') completely dominates (one exception) in the $\mathscr{R}_p$ criterion across the different dataset sizes. Including it back, the predicate-based partitioning ('*iii*') significantly competes with the *subject-based* partitioning technique in most of the ranking criteria, i.e., $\mathscr{R}_p$, $\mathscr{R}_s$, $\text{Pareto}_{Agg/Q}$, and $\mathscr{R}_{ta}$.

With such flexibility, PAPyA also provides several dynamic views on the ranking criteria. For example, Table 30 shows the SD ranking of the schema dimension with changing the configuration space. Particularly, it shows how the *global* ranking of each relational schema (or any other specified dimension) could change by including/excluding configurations of the other dimensions. The table shows that the order of the global schema ranks changes by including all configurations ("Full Conf. Space") than including/excluding the predicate partitioning or CSV format, i.e., "PBP/!PBP", "CSV/!CSV", respectively. For instance, the PT schema global ranking is interestingly oscillating with those changes in the available configurations.

D_i	$\mathscr{R}_s$	$\mathscr{R}_f$	Pareto_Q	Pareto_{Agg}
100M	d.i.1	a.i.3	c.i.2	c.i.2
	c.i.2	b.i.2	d.i.2	b.i.2
	d.i.2	e.i.4	d.i.4	d.i.1
250M	c.i.1	a.i.3	d.i.2	d.i.2
	d.i.2	e.i.4	c.i.4	c.i.1
	c.i.3	d.i.2	c.i.2	e.i.4
500M	d.i.2	a.i.3	d.i.2	d.i.2
	c.i.3	d.i.2	c.i.3	a.i.3
	c.i.1	e.i.4	c.i.4	c.i.3

Table 31: Best-performing configurations, *excluding the partitioning* dimension.

Metric	**D**	$\mathscr{R}_s$	$\mathscr{R}_f$	**Pareto**$_Q$	**Pareto**$_{Agg}$
Conform.	100M	97.50%	67.50%	95.00%	92.50%
	250M	97.50%	30.00%	100.00%	97.50%
	500M	100.00%	52.50%	100.00%	52.50%
Cohere.	100M-250M	0.11	0.17	0.11	0.19
	100M-500M	0.28	0.16	0.30	0.18
	250M-500M	0.17	0.08	0.15	0.09

Table 32: Criteria evaluation (conform.ance, and coher.ence), *excluding partitioning*.

Adding/Removing Full Experimental Dimension. PAPyA'flexibility extends to the experimental dimensions, i.e., it is possible to add/remove dimensions easily. For instance, we can fully exclude the partitioning dimension in case experiments are executed on a single machine (see Listing 6.7 lines 13-18). In [RTS19], we run experiments on SparkSQL with different relational schemas and storage backends yet without data partitioning. Table 31 shows the best-performing (top-3) configurations in WatDiv experiments when excluding the partitioning dimension. Table 32 shows the conformance and coherence metrics' results for the various ranking criteria [6].

[6]Notably, the Rp and Rta criteria cannot be calculated when excluding the partitioning dimen-

```
from papya import Rank
#SD Ranker (Implementation of Equation (1))
class SD_Ranking(Rank):
    ...
#MD Ranker (Pareto-NSGA2)
class MD_Ranking(Rank):
    ...
#Add Triangle_Area as a new ranking criterion.
class RtaCriterion(Rank):
  def calculate_rta(self):
      r_scores = SDRank(...).calculateRank()
      rta_scores = []
      for i in range (len(r_scores)):
        rs = r_scores[0][i]
        rp = r_scores[1][i]
        rf = r_scores[2][i]
        # RTA Formula (Equation 4)
        y = (math.sin(math.radians(120)))/2
        outer_triangle_area = y * (1+1+1)
        rta = y * (rf*rp + rs*rp + rf*rs)
        rta_scores.append(rta)
    return rta_scores
  def plot(self):# plots (Figure 7)
    ...
```

Listing 6.8: Plugin the Triangle-Area as new Ranking criterion.

Adding Ranking Criterion. PAPyA abstractions enable users to plug-in new ranking criterion besides the already existing ones (i.e., the abstract Rank class, Section 6.1). Let's assume we seek usage of a simple ranking criterion that leverages a *geometric* interpretation of the SD rankings of the three experimental dimensions based on the *triangle area* subsumed by each ranking criterion (R_s, R_p, and R_f).

In Figure 48, the triangle sides represent the SD-ranking dimensions' rank scores. Thus, this criterion aims to maximize this triangle's area (i.e., the *blue* triangle). The closer to the ideal (outer red triangle), the better it scores. In other words, the bigger the area of this triangle covers, the better the performance of the three ranking dimensions altogether. The *red* triangle represents the case with the maximum/ideal rank score, i.e, $R = 1$ for the three dimensions (as, $0 < R <= 1$). Equation (5.1, Chapter 5) defines the blue triangle area; named as ranking by triangle area (R_{ta}).

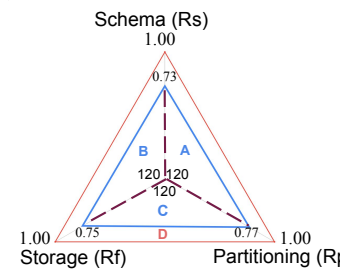
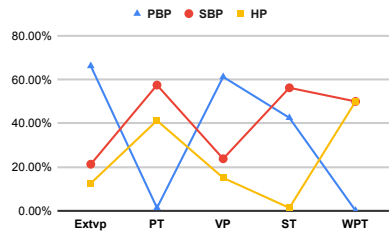


Figure 48: Triangle Area criterion.

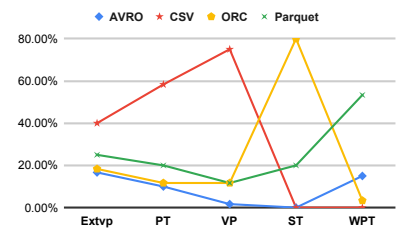
$$TriangleArea(R_{ta}) = \frac{1}{2} sin(120) * (R_f * R_p + R_s * R_p + R_f * R_s) \quad (6.1)$$

The formula (Cf. Equation 6.1) computes the actual triangle area. Simply, it sums up the triangle area of the three triangles A, B, and C by two of its sides which are the rank scores of each dimension, i.e., R_s, R_p, or R_f (dashed triangle sides), and the angle between both of them (i.e 120 in this case). For example, the actual area of the *blue*

sion.



(a) Impact of partitioning on the schema performance



(b) Impact of storage on the schema performance

Figure 49: Schema Replicability across changing partitioning or storage formats.

triangle is $R_{ta} = \frac{1}{2}\sin(120)(0.75*0.771+0.73*0.77+0.75*0.73) = 0.73$. In addition to the SD and MD ranking criteria classes, Listing 6.8 shows how to extend PAPyA with a new Ranker class, i.e., *RtaCriterion*.

It is worth mentioning that the idea behind R_{ta} is intuitively similar to Pareto$_{Agg}$ because both aim to maximize the rank scores of the three dimensions altogether. However, unlike Pareto$_{Agg}$ that is multi-dimensional, R_{ta} cannot extend to dimensions above three. For simplicity, we used R_{ta} as an exemplar to showcase that PAPyA abstractions enable extending new ranking algorithms/criteria. Tables 26 and 27 show the best-performing (top-3) configurations ranked by R_{ta}. Results show that Pareto$_{Agg}$ results perfectly conform with R_{ta} top-ranked configurations (especially in the top-3 ranked configurations). Table 28 and 29 also show that R_{ta} criterion scores high conformance ratios across WatDiv benchmark datasets. It also scores high coherence (few disagreements) through the scalability of WatDiv datasets. In both WatDiv experiments (i.e., with mini and full dimensions inclusion), the R_{ta} conformance and coherence values are very close to the Pareto$_{Agg}$ criterion.

6.2.3. Checking Performance Replicability

PAPyA also activates the functionality of checking the BD system's performance replicability when introducing different experimental dimensions. In particular, it enables checking the system's performance with one specific dimension while changing the parameters of the other dimensions. For example, Figures 49 (a) and (b) respectively show the impact of the partitioning and storage on the performance of the schema dimension. The Figures show how the performance of the system with a configuration can significantly change with changing other dimensions.

PAPyA can also check the performance replicability by comparing two configurations [Rag+21], as discussed in Chapter 4. For instance, PAPyA can compare the *schema* optimizations (i.e., *WPT*, and *ExtVP*) w.r.t their *baseline* ones (i.e., *PT*, and *VP*) while introducing different partitioning techniques and various HDFS storage formats that are different from the baseline configurations [Sch+16; Sch+14].

Table 33 shows the effect of introducing partitioning techniques (right of the table) and different file formats (left of the table) different from the baseline configurations (i.e., Vanilla HDFS partitioning, and Parquet as storage format). The *trade-offs* effect is clear on the replicability results. Indeed, WPT outperforms PT schema only with 54.16% in the queries using the baseline Vanilla HDFS partitioning technique across all storage formats and only about 39% for the baseline Parquet format across all partitioning techniques. On

Partitioning	ExtVP VS.VP	WPT VS. PT	Storage	ExtVP VS. VP	WPT VS. PT
V. HDFS	97.5%	54.16%	Parquet	75.0%	38.8%
Horizontal	67.5%	8.3%	ORC	73.33%	18.5%
Predicate	61.4%	NA	Avro	63.3%	16.6%
Subject	66.25%	6.9%	CSV	93.3 %	16.6 %

Table 33: The *replicability* of schema advancements (i.e., WPT, ExtVP) VS. baselines (i.e., PT, VP), WatDiv 500M dataset.

the other side, we observe significant degradation of WPT schema optimization moving to other configurations with both partitioning and storage dimensions. For instance, WPT outperforms PT only with about 8% and 7% using other different partitioning techniques, i.e., Horizontal and Subject, respectively. The same occurs with changing the storage formats different from baseline Parquet. Similarly, ExtVP versus VP schema performance results confirm our observations. PAPyA enables showing those *trade-offs* of considering alternative storage file formats and partitioning techniques alongside the experiments query evaluation.

6.3. Conclusion and Road-map

In this chapter, we present PAPyA, an extensible library that reduces the efforts needed to analyze the performance of BD systems used for processing large (RDF) graphs. PAPyA implements the performance analytics methods adopted in [Sch+14; Sch+16; RTS19] including an novel approach for prescriptive performance analytics we presented in [RAT21].

Inspired by Gartner's analysis methodology [Hag17], Figure 50 reflects the amount of human intervention required to make a decision with the descriptive and diagnostic analyses of the performance results. Descriptive and diagnostic analytics are limited, and cannot guide practitioners directly to the best-performing configurations in a complex solution space. This is shown in this paper with the lack of performance replicability (shown Section 6.2.3). Indeed, the performance of the BD system is affected by changing the configurations, e.g., oscillating schema performance with changing partitioning, and storage options (Figure 49). On the other side, PAPyA reduces the amount of work required to interpret performance data. It adopts the Bench-ranking methodology with which practitioners can easily decide the *best-performing* configurations given an experimental solution space with an arbitrary number of dimensions. Although, descriptive discussions are limited, PAPyA still provides several descriptive analytics and visualizations on the performance to explain the final decisions given by PAPyA. PAPyA also aims to reduce the engineering work required for building an analytical pipeline for processing large RDF graphs. In particular, PAPyA prepares, generates, and loads data ready for big relational RDF graph analytics.

PAPyA is developed considering the ease of use and the flexibility aspects allowing extending the library with an additional arbitrary number of experimental dimensions to the solution space. Moreover, PAPyA provides abstractions on the level of ranking criteria, meaning that the user can use his/her ranking functions for ranking the solution space. Seeking availability, we provide PAPyA as an open-source library under MIT license and published at a persistent URI. PAPyA's GitHub repository includes tutorials and documentation on how to use the library.

As a maintenance plan, PAPyA's roadmap includes:

1. covers the phase of query evaluation into PAPyA pipeline. In particular, we plan to

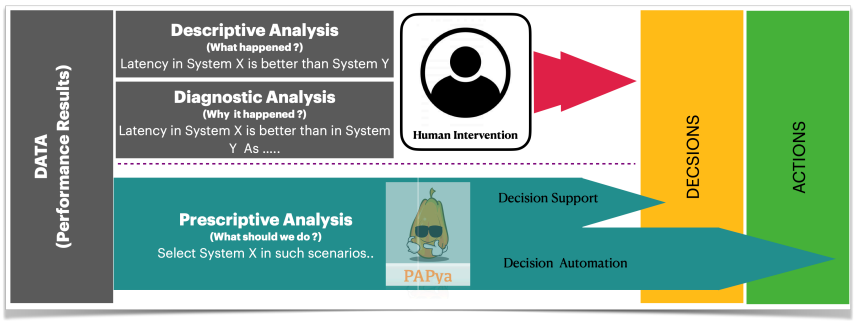


Figure 50: Performance analysis methodology, and how PAPyA reduces human intervention in BD performance analyses.

provide native support of SPARQL via incorporating native triple stores for query evaluation.

2. incorporate SPARQL into SQL translation using advancements of *R2RML* mapping tools (e.g., *OnTop*) [Bel+21].
3. wraps other *SQL-on-Hadoop* executors to PAPyA; thus, the performance of the engines could also be compared.
4. use orchestration tools (such as *Apache Airflow*) to monitor the PAPyA pipelines.
5. enable predictive analysis in PAPyA, to predict the performance of BD systems for processing (RDF) graphs, based on previous experiments, and query workloads.
6. integrate PAPyA with other tools like *gmark* [Bag+16] that generates graphs and workloads, making them ready for distributed relational setups.

7. CONCLUSION AND FUTURE DIRECTIONS

Leveraging relational Big Data (BD) processing frameworks to process large knowledge graphs yields a great interest in optimizing query performance. Modern BD systems are yet complicated data systems, where the configurations notably affect the performance. Benchmarking different BD frameworks and configurations provides the community with best practices for better performance. However, most of these benchmarking efforts are classified as descriptive and diagnostic analytics. In this thesis, we discuss how we abstract from the complexity of descriptive and diagnostic performance analyses, guiding the practitioner directly to actionable informed decisions. Particularly, we investigate how to enable prescriptive analytics for BD systems when querying large KGs, via ranking functions, as well as multi-dimensional optimization techniques (called *Bench-Ranking* framework). This Bench-Ranking framework builds on the state-of-the-art benchmarking efforts in the area of querying large (RDF) graphs.

7.1. Bench-Ranking Requirements & Research questions

This work follows the *Macro*, *Mezzo*, *Micro* framework for framing the research problem [LG15], which requires to formulate the research questions at *three* levels of analysis.

"Can we guarantee fair benchmarking assessment of relational BD systems performance while querying big graphs?" is the *Macro* question that frames our research. The question is open to numerous interpretations and answers; thus, we shall reformulate it into more specific ones. Thus, we list few specific *requirements* which enable us to narrow down the problem and validate our approach.

1. **(R1) The approach should apply to the problem of querying *large* (RDF) graphs** with *latency* as a reference *KPI*.
2. **(R2) Choose system-agnostic experimental dimensions that directly impact the (BD) systems when querying large graphs.**
3. **(R3) Ensure the *replicability* of BD systems' performance while varying the experimental dimensions.**
4. **(R4) Consider multiple experimental dimensions' *simultaneously* to make sense of their *trade-offs*.**

Thus, we formulated the *Mezzo* question based on the aforementioned requirements as follows: *Can we aid decision making for benchmarking big (RDF) graph processing over relational systems?* The objective is to provide guidance to practitioners in choosing the *best-performing* configurations (Definition 1 in Chapter 1) of a BD system for querying large (RDF) KGs from a complex solution space of experimental dimensions with arbitrary multiple options and inherent trade-offs.

In the following sections, the Micro questions are mentioned alongside the summary and conclusions of the contribution(s) regarding those questions.

7.2. Summary of Contributions

This section offers an overview of the thesis contributions regarding the problems solved and how they offer a valid solution for the research questions (see Table 34 for a summary).

RQs	Chllenges	Publication	Chapters
Micro 1	C1,C2	[RTS19],[Rag+20]	Chapter3
Micro 2	C2,C3	[Rag+21]	Chapter 4
Micro 3	C4	[RAT21]	Chapter 5
Micro 4	C4	Under Review [RAT22]	Chapter 6

Table 34: Summary of research questions, the challenges that they address (see Table 1, in Chapter 1); the related publications, and the chapters that discuss them.

7.2.1. Making sense of Big Data Descriptive Performance Analysis

The state of the art research works that adopt large (RDF) graph processing is limited in terms of the systematic coverage experimental dimensions that impact the performance (see Table 1, in Chapter 1). Moreover, those research efforts are limited in the maturity level of performance analysis. They provide mere descriptive and diagnostic performance analytics.

Thus, Chapter 3 provides answers to the *Micro 1* question (formulated in the Introduction chapter, Section 1.5.1). Chapter 3 discusses the comprehensive experimental solution space of the design decisions that emerge with querying large (RDF) knowledge graphs on top relational BD systems, i.e., schema, partitioning, and storage formats. It also discusses other experimental options, such as the BD system dedicated to the experiments (i.e., Apache Spark-SQL) and the RDF benchmarking datasets and workloads (i.e., SP2B and WatDiv). We approached answering the research question(s) in this chapter via:

1. performing extensive benchmarking experiments on the performance of the combination of the configurations.
2. making sense of the performance results following the descriptive and diagnostic analyses.
3. giving best practices for processing large KGs alongside mentioned dimensions (see Chapter 3 Figure 24)).
4. showcasing the limitations of the descriptive and diagnostic analyses.

Thus, this chapter shows that the descriptive performance and diagnostic analyses are *insufficient* to provide actionable indicators. That is, comparing descriptive results in such complex scenarios to reach a decision can be an overwhelming task. Although, it is possible to utilize diagnostic analysis to rationalize the performance utilizing the knowledge attained around the workload and the experimental dimensions, experiments showed that those dimensions *trade-offs* still hinder clear directions towards selecting the best-performing configurations. Moreover, following the descriptive and diagnostic analyses may result in multiple cluster configuration setups/deployments with huge data redundancy to ensure the most efficient execution of the query workload (see Figure 23 in Section 3.3.1 in Chapter 3). This thesis provides a framework (i.e., Bench-Raking)

that aims to guide the practitioners directly to the best-performing configurations with minimum costs in terms of cluster management.

7.2.2. Assessing Big Data Replicability

Chapter 4 continues to show the current big graph performance analytics limitations. It provides answers to the *Micro 2* question (formulated in the introduction Chapter 1, Section 1.5.2). The Micro 2 question investigates the effect of changing one or more experimental dimension(s) and their options/alternatives on the replicability of the performance of a BD system for querying large (RDF) graphs.

To answer the *Micro 2* question in this thesis, we followed the following approach. We check the validity of the hypothesis mentioned in (Chapter 1, Section 1.5.2). Particularly, we investigate the performance improvement of one of BD systems (i.e., *Spark-SQL*) with two recent schema optimizations (i.e., *Extended Vertically-Partitioned Tables* (ExtVP)[Sch+16] and *Wide Property Tables* [Sch+14]), w.r.t. their baseline approaches (i.e., *Vertically-Partitioned (VP) Tables* and *Property Tables* (PT)). We observe if the performance of the two schemas advancements generalizes (i.e., still outperform the baseline ones) over Spark-SQL with introducing different RDF partitioning techniques and various *HDFS* storage data formats that are different from the studied *baseline* configurations (i.e., *Vanilla HDFS* partitioning, and *Parquet* as storage format) [Sch+16; Sch+14; Rag+21].

Answering **Micro 2** unveils the replicability issue in Big Data systems performance when querying large graphs. The results of our replicability experiments show that we cannot even guarantee the optimal performance of BD system with one of the dimensions (e.g., schema optimizations)if we changed or introduced other experimental options (e.g., partitioning techniques, or storage formats). In practice, we show that the RDF relational schema optimizations outperform the baseline ones (table percentages) [Sch+16; Sch+14] *only* with the *vanilla* configurations. However, this performance behavior *does not generalize* while introducing other new partitioning techniques (e.g., *Horizontal*, *Subject(Predicate)-based* partitioning), or other storage file formats (e.g., *Avro*, *CSV*, or *ORC*) [Rag+21]. Thus, the outcomes of this contribution can be summarized as follows:

1. The schema dimension is not the *only* design decision to consider while processing large KGs on top relational systems.
2. The Partitioning and storage formats are key factors that directly impact the BD system's performance.
3. BD systems replicability is indeed affected by introducing different experimental dimensions.
4. Descriptive and diagnostic performance analyses are limited in the presence of replicability lack.

The results of this chapter showed the effect of experimental configurations trade-offs. Therefore, we aim for a framework to mitigate these issues and guide optimal solutions while considering the performance trade-offs.

7.2.3. Bench-Ranking:The Big Data Prescriptive Performance Analysis (PPA)

Motivated by the limitations of descriptive analysis (shown in Chapter 3 and chapter 4), Chapter 5 proposes a *"Bench-Ranking"* framework. The Bench-Ranking framework frame-

work aims to enable prescriptive performance analysis that can help decide the best-performing configurations over the complex solution space alongside considering the inherent trade-offs. In particular, the contribution answers the *Micro 3* question (formulated in the introduction Chapter 1, Section 1.5.3). The Micro 3 question investigates the level of abstraction required for selecting the *best-performing* experimental configurations instead of comparing a huge number of experiments' performance results (i.e., that sometimes are even *contradicting* due to experimental trade-offs). To guide the practitioners on this hard task, the contributions of the Bench-ranking framework (Chapter 5) are summarized as follows:

1. Employing SD ranking Criteria seeking actionable prescriptions, aiming to abstract from the *fine-grained* descriptive performance metrics and enable a *decision-making* model.
2. designing the Bench-ranking as *Multi-Dimensional* (MD) optimization problem for optimizing the performance of dimensions' parameters altogether.
3. Proposing metrics for assessing the *efficiency* of the proposed ranking criteria [RAT21].

SD Ranking Criteria generalize the ranking function proposed in [ANS18] that ranks several RDF partitioning techniques across different datasets. Our generalized ranking function enables calculating the *ranking scores* for the experimental dimensions' options of the mentioned dimensions. The *rank scores* of the SD ranking criteria help to provide a *high-level* view of the system performance across a set of tasks (e.g., workload queries) [RAT21]. However, our experiments show that the SD ranking prescriptions are not *coherent* across experimental dimensions [RAT21]. Indeed, the SD ranking criteria cannot generalize as they neglect the presence of *trade-offs* as they rank (i.e., optimize) alongside a single experimental dimension [RAT21] neglecting the other ones.

MD Ranking Criteria. The limitations of SD criteria led to extending the *Bench-Ranking* into a *Multi-Dimensional* (MD) optimization problem to optimize all the dimensions at the same time. We adopt the standard *Pareto frontier* technique to consider the experimental dimensions altogether [Deb+02]. In particular, we utilized the *Non-dominated Sorting Genetic Algorithm* (*NSGA-II*) [Deb+02]. The algorithm operates on the ranking scores of the SD ranking criteria (i.e., we call them R_s, R_p, and R_f).

Evaluating Ranking Criteria. In Bench-Ranking framework, we consider a ranking criterion *"good"* if it does not suggest *low-performing* configurations. Herein, we discuss how we measure the ranking criteria *goodness* using *two* metrics. The first metric is the *Conformance* that measures the *adherence* of the *top-ranked* configurations w.r.t actual query rankings (i.e., positioning of those configurations [1]) [RAT21]. The second metric is the *Coherence* which measures the level of agreement between two ranking sets that use the same ranking criterion across different experiments (e.g., different dataset scales). To measure how coherent the ranking criteria are while scaling up to larger datasets, we employ *Kendall's index* [2]

Our experimental results[3] [RAT21] show that all the ranking criteria achieve high *Coherence* across different scales of the datasets. This shows that scaling the datasets does not excessively impact the rank sets' order in all the ranking criteria, whereas MD ranking

[1] Each configuration C has a rank according to its running time of the queries.

[2] Kendall's index is a common measure to compare the ordering of ranking functions.

[3] *Conformance* and *Coherence* results [RAT21] are omitted due to space limits.

criteria (i.e., Pareto fronts) show better *Conformance* results than the SD ranking criteria. Indeed, the Pareto ranking technique considers optimizing the performance of all dimensions simultaneously, whereas the SD considers only one dimension at a time while ignoring the other dimensions.

7.2.4. PAPyA: Big Data Bench-Ranking Made Easy

Prescriptive Performance Analysis (PPA), in particular utilizing ranking functions, has shown to be more useful than traditional *descriptive* and *diagnostic* analyses for making sense of Big Data frameworks' performance. Bench-Ranking criteria (i.e., SD and MD) draw the guidelines toward actionable insights. However, the amount of experimental work required to implement BD PPA is still huge.

To this end, Chapter 6 answers the Micro 4 question (formulated in the introduction Chapter 1, Section 1.5.4) by introducing PAPyA [4], an open-source python library for enabling and providing PPA for Big Data systems when querying large KGs. The requirements for PAPyA are based on the existing research efforts on benchmarking BD systems for processing and querying large RDF graphs [Sch+16; Sch+14; RAT21; Rag+20; CFL18].

PAPyA wraps the functionality proposed in Bench-Ranking [RAT21] (Chapter 5) in one single tool that reduces the efforts needed to select the best-performing configurations that guarantee the optimal performance of BD systems for large RDF graphs processing. In Bench-Ranking Chapter 5, we used the query *latency* as the Bench-Ranking KPI, yet PAPyA supports analyzing the performance in terms of other metrics (e.g., memory consumption, CPU usage). In Bench-Ranking, we also opted for the relational schemas, partitioning, and data formats, which lead to a *3-dimensional* solution space. Hiding the complexity of Bench-Ranking, PAPyA provides an interactive environment that eases interconnections of various modules of the framework (e.g., data and performance visualization). Moreover, PAPyA also aims to reduce the time required to build an analytical BD pipeline to process large RDF graphs. In particular, PAPyA prepares, generates, and loads data ready for big relational RDF graph analytics.

7.3. Open Challenges and Future Directions

We believe the work presented so far opens up several interesting research directions. Here, we highlight key directions for future research.

7.3.1. Workload-driven Automatic Configuration Mining

Our current proposed SD and MD ranking criteria (i.e., *"Bench-Ranking"*) do not explicitly consider the query workload. This means that *"Bench-Ranking"* framework cannot provide prescriptions across *different* benchmarks' query workloads [RAT21].

On another side, the flexibility of RDF graphs (i.e., *schema-last* data model) brings challenges to the storage and management of RDF graphs in a relational environment [Pha+15]. Indeed, the relational model requires several *design decisions* when used for representing and processing graphs, which cannot be decided automatically, e.g., the choice of the *schema*, the *partitioning technique*, and the physical *storage formats*. There is no *One-Solution-Fits-All* on the level of these experimental dimensions. That is, the configurations made of several experimental dimensions cannot fit all the query shapes.

[4] https://github.com/DataSystemsGroupUT/PAPyA

Indeed, our experiments have demonstrated that no configuration is dominant for the different families of RDF SPARQL queries [RAT21].

For instance, when considering the schema dimension, we find that each RDF relational schema is excellent for a particular class of queries [AÖD14]. One RDF relational schema may outperform others while evaluating a particular family of queries, but it increases the sparsity of the data and results in space waste. The most popular relational schemas for RDF graphs are (i) Single Statement (ST) Table, (ii) Vertical Table (VT), and (ii) Wide Property Table (WPT) [Hog+21] (detailed specifications in Chapter 2). Such schemas are designed according to different requirements that are often in contrast. For example, the ST schema is *triple-based*, i.e., it is agnostic from the underlying graph structure. This approach eases data integration (i.e., merging two datasets does not require any schema change) but increases the number of *self-joins* required to answer complex queries. The WPT is *dataset-based* schema, i.e., the schema attempts to encode the entire dataset into a *single denormalized* relation. The schema strictly depends on the graph structure and aims at reducing the cost of *star-shaped* queries [Rag+21]. It allows answering this family of queries with *no* joins [Sch+14]. However, it significantly degrades the performance of the relational system for other query shapes that require several joins over the large sparse WPT table [Rag+21; Sch+14]. Moreover, the WPT schema suffers from huge *redundancy* and *sparsity* (i.e., containing many *NULL values*), and thus it incurs large storage overhead. Last but not least, the VT schema is *graph-based*, i.e., it requires one binary relation (Subject,Object) for each unique predicate (*labeled edge*) in the graph structure. The VT schema aims at reducing redundancy by normalizing the relational graph structure. However, it may incur huge data *skewness* over the predicate tables and low performance with complex SPARQL queries (i.e., too many VT table joins).

On the other side, combining multiple schemas (partitioning techniques or storage formats) to attain *hybrid* configurations that obtain the best of the underlying alternatives represents a valuable research direction. We also argue that building such *hybrid* configurations requires *tailored* solutions that consider the underlying dataset characteristics and the query workload adjustments, entailing huge data engineering efforts. To this end, we aim to propose algorithms that automate designing *hybrid* configurations that adapt to the query workload covering a wide range of graph query shapes without ignoring the loading times and the storage overheads.

Therefore, we propose investigating such research direction in the future. We start with the hybrid schema generation, as it is the most studied dimension in the literature and the one directly reflects on the query workload complexity. The investigation process will take the following steps.

Designing algorithms for Automated Hybrid Schema migration A major problem in hybrid schema definition for graph data is the lack of an automated procedure for schema definition and migrations. Considering the schemas ST, VT, and WPT as starting schema points, we will develop an algorithm for schema migration that considers the dataset characteristics and the impact on the workload. Therefore, defining a dataset profile that reflects the graph-specific workloads is a crucial step.

The algorithm should take as an input a dataset profile and the workload, which consists of a set of SPARQL queries and their SQL translations. The algorithm will be *iterative* and attempts to modify the input schema for the given query workload to reach an optimal hybrid schema. As an extension of such an algorithm, we plan to employ the *reinforcement learning* methods [SB18] to automatically tune the hybrid relational schema migrations to adapt to the prospective changes in the query workload. Investigating hy-

brid relational schema generation for graph streaming applications is one of our future directions to check its feasibility and challenges.

Empirically Validate the Performance To validate the effectiveness of the designed algorithm, we plan to re-use the state-of-the-art RDF benchmarks like *LUBM*, *SP2Bench*, *WatDiv*, and *LDBC*. These benchmarks include a data generator and sufficiently complex workloads to test the designed solution with different dataset sizes. Moreover, we also plan to test the approach on real-world datasets like *Wikidata* and *Yago* [Hog+21] that present additional challenges in data preparation and consistency. Last but not least, harnessing scalability in our experiments, we plan to use *Apache Spark* the *de-facto* standard for Big Data analytics.

7.3.2. Bench-Ranking alongside Multi-Query Optimization techniques

Typically, the query optimization is a challenging task in database systems. Although typical relational DBs have the luxury of using indexes that eases the mission of query optimizers, most of the current Big Data management systems lack this feature. Moreover, the *schema-free* nature of RDF graphs with complex SPARQL workloads (i.e., several joins) aggravates the problem of query optimization for processing the large knowledge (RDF) graphs [SA10; Pha+15]. Several techniques have been proposed to solve this problem such as query rewriting [Cor+10; CMC16], vertical partitioning [Aba+07], as well as scalable join processing [NW09]. On another side, several works tackled such problem via the classical Multi-Query Optimization (MQO) technique in the context of RDF and SPARQL [RW16; GGZ19; Zer+20; Pen+19]. The main idea of MQO is to identify and exploit common *sub-expressions* across a set of running queries in order to reduce evaluation cost. In particular, MQO generates a global optimized plan (via query rewriting, join reorders, or sharing the intermediate results within the same group of queries) aiming for reducing the evaluation costs of multiple queries.

To reflect on our work in this thesis, we aim to study the reflection of MQO techniques on guiding the design decisions of querying large KGs on top of relational BD systems. In particular, we aim to investigate how the graph layouts and partitioning across a distributed storage can be re-designed with applying MQO techniques and detecting the SPARQL query templates [Has+16] on the given query workload. We aim also to further investigate how we can periodically and automatically re-partition the RDF graph to optimize the query evaluation to adapt with the query workload changes.

Automatic Query Workload Translation. The state-of-the-art proposes several approaches for the problem of SPARQL into SQL mappings/translations [CLF09; PCS14; RR15]. However, most of these works pre-assume a predefined relational schema for query translation. Indeed, Translating *SPARQL* query workload into *SQL* is a *schema-dependent* task. For instance, in this thesis, the process of translating the native SPARQL query workload into SQL for each RDF relational schema is done manually or reused existing benchmarks (WatDiv, and SP2Bench) query translations for the specified schemas [Sch+16; Sch+14]. However, to the best of our knowledge, there is no automatic approach for translating the SPARQL query workload based on the underlying hybrid relational schema (i.e., a mixture of ST, VT, PT,.., etc). Therefore, we plan to design mapping algorithms that automatically translate the query workload according to the designed relational layout. This will be more beneficial for the hybrid relational schema generation because the query translation process will automatically adapt to the generated underlying hybrid schema. As an initial step, the automatic translation will follow the generic relational

algebra transformations [Cyg05], as an intermediary abstract dialect for the workload operators and clauses. Further, the underlying schema specifications are employed with the existing advancements of SPARQL-to-SQL translations such as *R2RML* mapping (e.g., *OnTop*) [Bel+21].

7.3.3. Learning to Rank (LtR): Predicting the Optimal configurations

The Bench-Ranking methodology is a post-hoc prescriptive performance analytics framework. It analyzes the performance results and aims to guide the practitioner directly to the optimal configurations that guarantee the best performance of a BD system for querying large (RDF) graphs. This means that the practitioner should first execute experiments and get the performance results of the experimental dimensions, and then Bench-Ranking (with the help of our PAPyA tool [RAT22]) points to the best-performing configurations. To reduce the experiments needed for Bench-Ranking, we aim for enabling *pre-hoc* prescriptive performance analysis by predicting the best-performing configurations for a new query workload. To achieve this, we plan to employ Machine Learning (ML) [MRT18] techniques that utilize the previous performance results (as a labeled training dataset) and learn about the best configurations for executing new different instances of the query workload.

To this end, we aim to employ the *Learning-to-Rank* (LtR) ML techniques for this problem [Liu+09]. LtR is the state-of-the-art *Information Retrieval* (IR) technique that develops ranking models from the labeled training data. Thus, we aim to utilize *query-independent* (dataset profile characteristics) and *query-dependent* LtR techniques aiming to learn the ranking function of the best configurations for querying graphs in distributed relational environments [Dal+12]. Therefore, the query profiling [Has+16] is a crucial phase in this scenario, as it enables extracting query features of the existing training workload [MSO12]. Most of Big Data systems (e.g., Apache Spark-SQL, Impala, Apache Drill, and Hive) provide a query "*explain*" service that enables understanding the execution approach taken by the engine for evaluating the query. Thus, we aim to elicit a query profile (i.e., query execution features) from the query execution plans using such services. Additionally, query similarity techniques [Arz+19; Has+16] and clustering techniques such as *K-Nearest-Neighbour* (KNN) can be employed to detect which query (in the training dataset) is the nearest to the features of the test query.

As a result, the LtR models, alongside the domain knowledge (i.e., query complexity features, dataset features/statistics, cluster specifications, etc.), can tune that ranking for the new test queries and directly recommend optimal solutions (configurations) for the given query workload.

7.3.4. Bench-Ranking with Costs and Energy Consumption Estimation

Benchmarking is a crucial task that checks the performance capabilities of systems under specific tasks evaluating their potentials and bottlenecks [Bon+18; Szá19]. Benchmarking efforts can push technological progress turning research into a competition with clearly-defined goals. This stimulates a research community to produce more and better results where the end users can understand and decide on the basis of those defined objectives.

However, benchmarking Big Data solutions to select the best setup(s) in cloud services can come with huge costs, as it would require huge data engineering efforts [Wan+14]. For example, in our scenario selecting the best-performing configurations (relational schema,

partitioning technique, and storage format) for a BD for querying and processing vast amounts of is not a trivial task. Changing those design decisions in the future (if selected wrongly from the first design) would come with several burdens and huge costs at the production level. For instance, deciding to change the relational schema of such large datasets require huge engineering efforts [DL20]. These efforts can be exerted for adapting the data to the new data layouts, as well as for adapting the query workload to work with the new schema structures.

Our Bench-Ranking framework can jump to the scene, providing prescriptive analysis for practitioners that can guide them in such tasks. Nonetheless, it is still important to develop analytical models for estimating how much costs Bench-Ranking would save by deciding the best-performing configurations, especially after enabling the *pre-hoc* analyses in Bench-Raking framework with applying *LtR* techniques [Liu+09] mentioned above. On another side, with such intensive BD analytical computations, the attention to the energy consumption, energy-aware query optimizations, and green computing should be more and more considered [GBB21]. Indeed, in recent years, computer systems' energy consumption has sharply increased [Tu+14; GBB21]. One key factor in energy greediness is thought to be lowering of query processors energy consumption using several techniques such as selecting *Bitmap Join Indexes* (BJIs) with query analytics [GBB21]. Therefore, it is also interesting to investigate the energy consumption estimation (as a Non-Functional Requirement (NFR))in cloud services and cluster distributed setups in general when processing and querying large KGs, and implementing the Bench-Ranking framework for relational BD systems.

7.4. Concluding Remarks

In this thesis, we discussed *how* and *why* it is important to enable and provide *prescriptive* performance analysis for Big Data systems, focusing on ranking the complex solution space of the experimental dimensions that emerge with querying large (RDF) graphs over relational BD systems.

The Bench-Ranking criteria that we provide in this work is an accurate yet simple way that supports the practitioners in their evaluation tasks even in the existence of experimental dimensions trade-offs. It aims at abstracting away from the complexity of descriptive and diagnostic BD performance analyses that are shown to be limited and might require huge human interventions to provide accurate actionable insights on the problem of querying large KGs (Chapters 3 and 4). On the other side, the prescriptive performance analysis in Bench-Ranking framework reduces the need for human intervention by making the insights actionable, utilizing Single-Dimensional and Multi-Dimensional ranking techniques for making sense of the BD systems performance analyses. Thus, it can directly guides the practitioner to the *best-performing* configuration combinations of complex solutions space of various multiple experimental dimensions.

The SD ranking criteria seek actionable indicators via employing ranking functions of the experimental dimensions (i.e., schema, partitioning, and storage) that abstract out from *fine-grained* performance observations and benchmarking and lead to actual decision making. The results show that the SD ranking are efficient for guiding the practitioner to the best-performance across one dimension. For instance, if the practitioner aims solely for the optimizing the schema dimension ignoring the other dimensions, SD ranking criteria would be an option. However, the SD criteria cannot generalize over multiple dimensions as they neglect the presence of trade-offs as they rank (i.e., opti-

mize) alongside a single experimental dimension neglecting the other ones. Therefore, this thesis recommends extending the BenchRank into a Multi-Dimensional optimization problem in order to optimize all the dimensions at the same time.

Evaluating the ranking criteria themselves is a core pillar in the Bench-Ranking framework. In particular, we discuss that it is important to measure the ranking criteria goodness to opt for one. Our Bench-Ranking framework provides two metrics, the first metric is the *Conformance* that measures the adherence of the top-ranked configurations w.r.t actual query rankings (i.e., ranking positioning of those configurations). The second metric is the *Coherence* which measures the level of agreement between two ranking sets that use the *same* ranking criterion across different experiments (e.g., different dataset scales).

Last but not least, we aim to provide the hook rather than providing the fish. That is, we do not focus much on providing prescriptive analysis for a specific KG, nor a specific RDF dataset (though, we provide general best-practices from the previous work as well as our experiments 3). However, we rather provide the tool (i.e., PAPyA) that the practitioner can utilize for his/her own use-case, scenario and RDF datasets, and still can directly guide him/her with prescriptive analysis, seeking optimal performance results.

In particular, PAPyA tool wraps up the Bench-Ranking ranking techniques as well as the evaluation metrics in one ope-source python library (Chapter 6). Big Data practitioners can utilize this tool when processing and querying large graphs to (1) prepare the solution space of various experimental dimensions (e.g., schema, partitioning, and storage), and load data for experimentation, (2) run experiments with a BD system (e.g., Spark-SQL), (3) automatically collect performance results from logs, and provide prescriptive analysis on the best-performing configurations. PAPyA is extensible and flexible tool in terms of removing or adding (new) experimental dimensions, or alternatives/options alongside each experimental dimension. Moreover, it can extend the ranking techniques, as well as the ranking criteria evaluation metrics.

BIBLIOGRAPHY

[Aba+07] Daniel J Abadi et al. “Scalable semantic web data management using vertical partitioning”. In: *VLDB*. 2007.

[Aba+09] Daniel J Abadi et al. “SW-Store: a vertically partitioned DBMS for Semantic Web data management”. In: *The VLDB Journal* 18.2 (2009), pp. 385–406.

[Abb+18] Zainab Abbas et al. “Streaming Graph Partitioning: An Experimental Study”. In: *Proc. VLDB Endow.* 11.11 (2018), pp. 1590–1603. ISSN: 2150-8097. DOI: 10.14778/3236187.3236208. URL: https://doi.org/10.14778/3236187.3236208.

[Abd+15] Ibrahim Abdelaziz et al. “SPARTex: A Vertex-Centric Framework for RDF Data Analytics”. In: *PVLDB* 8.12 (2015), pp. 1880–1883. DOI: 10.14778/2824032.2824091. URL: http://www.vldb.org/pvldb/vol8/p1880-abdelaziz.pdf.

[Abd+17] Ibrahim Abdelaziz et al. “A survey and experimental comparison of distributed SPARQL engines for very large RDF data”. In: *Proceedings of the VLDB Endowment* (2017).

[ada17] adarsh-timepasstechies. *row-oriented and column-oriented file formats in hadoop*. https://timepasstechies.com/row-oriented-column-oriented-file-formats-hadoop/. [Online; accessed 9-Nov-2022]. 2017.

[AES22] Hanan E Alhazmi, Fathy E Eassa, and Suhelah M Sandokji. “Towards big data security framework by leveraging fragmentation and blockchain technology”. In: *IEEE Access* 10 (2022), pp. 10768–10782.

[Aga+18] Giannis Agathangelos et al. “RDF Query Answering Using Apache Spark: Review and Assessment”. In: *34th IEEE International Conference on Data Engineering Workshops, ICDE Workshops 2018, Paris, France, April 16-20, 2018*. 2018, pp. 54–59. DOI: 10.1109/ICDEW.2018.00016. URL: https://doi.org/10.1109/ICDEW.2018.00016.

[Ahm+18] Yousuf Ahmad et al. “LA3: A scalable link-and locality-aware linear algebra-based graph analytics system”. In: *Proceedings of the VLDB Endowment* 11.8 (2018), pp. 920–933.

[al19] Arrascue Ayala et al. “Relational schemata for distributed SPARQL query processing”. In: *Proceedings of the International Workshop on Semantic Big Data*. 2019, pp. 1–6.

[Ala19] Khadija Alaoui. “A categorization of RDF triplestores”. In: *Proceedings of the 4th International Conference on Smart City Applications*. 2019, pp. 1–7.

[Ale+01] Sofia Alexaki et al. “On Storing Voluminous RDF Descriptions: The Case of Web Portal Catalogs”. In: *Proceedings of the Fourth International Workshop on the Web and Databases, WebDB 2001, Santa Barbara, California, USA, May 24-25, 2001, in conjunction with ACM PODS/SIGMOD 2001. Informal proceedings*. 2001, pp. 43–48.

[Alu+14] Güne Aluç et al. “Diversified stress testing of RDF data management systems”. In: *International Semantic Web Conference*. Springer. 2014, pp. 197–212.

[Ang+17] Renzo Angles et al. “Foundations of modern query languages for graph databases”. In: *ACM Computing Surveys (CSUR)* 50.5 (2017), pp. 1–40.

[Ang+18] Renzo Angles et al. “G-CORE: A core for future graph query languages”. In: *Proceedings of the 2018 International Conference on Management of Data*. 2018, pp. 1421–1432.

[Ang+20] Renzo Angles et al. “The LDBC social network benchmark”. In: *arXiv preprint arXiv:2001.02299* (2020).

[Ang+21] Renzo Angles et al. “Pg-keys: Keys for property graphs”. In: *Proceedings of the 2021 International Conference on Management of Data*. 2021, pp. 2423–2436.

[ANS18a] Adnan Akhter, Axel-Cyrille Ngonga Ngomo, and Muhammad Saleem. “An Empirical Evaluation of RDF Graph Partitioning Techniques”. In: *EKAW*. 2018. DOI: 10.1007/978-3-030-03667-6_1. URL: https://doi.org/10.1007/978-3-030-03667-6%5C_1.

[ANS18b] Adnan Akhter, Axel-Cyrille Ngomo Ngonga, and Muhammad Saleem. “An empirical evaluation of RDF graph partitioning techniques”. In: *European Knowledge Acquisition Workshop*. 2018.

[AÖD14] Güne Aluç, M Tamer Özsu, and Khuzaima Daudjee. “Workload matters: Why RDF databases need a new design”. In: *Proceedings of the VLDB Endowment* 7.10 (2014), pp. 837–840.

[Ard+18] Danilo Ardagna et al. “Performance prediction of cloud-based big data applications”. In: *Proceedings of the 2018 ACM/SPEC International Conference on Performance Engineering*. 2018, pp. 192–199.

[Arz+19] Natalia Arzamasova et al. “On the usefulness of SQL-query-similarity measures to find user interests”. In: *IEEE Transactions on Knowledge and Data Engineering* 32.10 (2019), pp. 1982–1999.

[ATT19] Renzo Angles, Harsh Thakkar, and Dominik Tomaszuk. “RDF and Property Graphs Interoperability: Status and Issues.” In: *AMW* 2369 (2019).

[AV04] Grigoris Antoniou and Frank Van Harmelen. *A semantic web primer*. MIT press, 2004.

[AVS17] Maribel Acosta, Maria-Esther Vidal, and York Sure-Vetter. "Diefficiency metrics: measuring the continuous efficiency of query processing approaches". In: *International Semantic Web Conference*. 2017, pp. 3–19.

[Awa+20] Feras M Awaysheh et al. "Next-generation big data federation access control: A reference model". In: *Future Generation Computer Systems* 108 (2020), pp. 726–741.

[Awa+21] Feras M Awaysheh et al. "Big Data Resource Management & Networks: Taxonomy, Survey, and Future Directions". In: *IEEE Communications Surveys & Tutorials* (2021).

[Bag+16] Guillaume Bagan et al. "gMark: Schema-driven generation of graphs and queries". In: *IEEE Transactions on Knowledge and Data Engineering* 29.4 (2016), pp. 856–869.

[Bar+18] Cristóbal Barba-González et al. "jMetalSP: a framework for dynamic multi-objective big data optimization". In: *Applied Soft Computing* 69 (2018), pp. 737–748.

[Bat+15] Omar Batarfi et al. "Large scale graph processing systems: survey and an experimental evaluation". In: *Cluster Computing* 18.3 (2015), pp. 1189–1213.

[BDA18] Angela Bonifati, Stefania Dumbrava, and Emilio Jesús Gallego Arias. "Certified graph view maintenance with regular datalog". In: *Theory and Practice of Logic Programming* 18.3-4 (2018), pp. 372–389.

[Beb+18] Bradley R Bebee et al. "Amazon Neptune: Graph Data Management in the Cloud." In: *ISWC (P&D/Industry/BlueSky)*. 2018.

[Bee+16] Wouter Beek et al. "LOD Laundromat: Why the Semantic Web Needs Centralization (Even If We Don't Like It)". In: *IEEE Internet Comput.* 20.2 (2016), pp. 78–81.

[Bel+21] Matteo Belcao et al. "Chimera: A Bridge Between Big Data Analytics and Semantic Technologies". In: *International Semantic Web Conference*. Springer. 2021, pp. 463–479.

[BH18] Maciej Besta and Torsten Hoefler. "Survey and taxonomy of lossless graph compression and space-efficient graph representations". In: *arXiv preprint arXiv:1806.01799* (2018).

[Bit+15] MKABV Bittorf et al. "Impala: A modern, open-source sql engine for hadoop". In: *Proceedings of the 7th biennial conference on innovative data systems research*. 2015.

[BMT20] Angela Bonifati, Wim Martens, and Thomas Timm. "An analytical study of large SPARQL query logs". In: *The VLDB Journal* 29.2 (2020), pp. 655–679.

[Bon+18] Angela Bonifati et al. "A survey of benchmarks for graph-processing systems". In: *Graph Data Management*. Springer, 2018, pp. 163–186.

[Bor+13] Mihaela A. Bornea et al. “Building an Efficient RDF Store over a Relational Database”. In: *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*. SIGMOD ’13. New York, New York, USA: Association for Computing Machinery, 2013, pp. 121–132. ISBN: 9781450320375. DOI: 10.1145/2463676.2463718. URL: https://doi.org/10.1145/2463676.2463718.

[BQ18] Luciano Baresi and Giovanni Quattrocchi. “Towards vertically scalable spark applications”. In: *European Conference on Parallel Processing*. Springer. 2018, pp. 106–118.

[Bul+16] Aydn Buluç et al. “Recent advances in graph partitioning”. In: *Algorithm engineering* (2016), pp. 117–158.

[Cam+19] Jesús Camacho-Rodrguez et al. “Apache Hive: From MapReduce to Enterprise-grade Big Data Warehousing”. In: *Proceedings of the 2019 International Conference on Management of Data, SIGMOD Conference 2019, Amsterdam, The Netherlands, June 30 - July 5, 2019*. 2019, pp. 1773–1786. DOI: 10.1145/3299869.3314045. URL: https://doi.org/10.1145/3299869.3314045.

[Car+15] Paris Carbone et al. “Apache flink: Stream and batch processing in a single engine”. In: *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering* 36.4 (2015).

[CFL18] Matteo Cossu, Michael Färber, and Georg Lausen. “Prost: Distributed execution of SpaRQL queries using mixed partitioning strategies”. In: *arXiv preprint arXiv:1802.05898* (2018).

[Cho+05] Eugene Inseok Chong et al. “An efficient SQL-based RDF querying scheme”. In: *Proceedings of the 31st international conference on Very large data bases*. 2005, pp. 1216–1227.

[CLF09] Artem Chebotko, Shiyong Lu, and Farshad Fotouhi. “Semantics preserving SPARQL-to-SQL translation”. In: *Data & Knowledge Engineering* 68.10 (2009), pp. 973–1000.

[CMC16] Jean-Paul Calbimonte, Jose Mora, and Oscar Corcho. “Query rewriting in RDF stream processing”. In: *European semantic web conference*. Springer. 2016, pp. 486–502.

[Cod89] Edgar F Codd. “Relational database: A practical foundation for productivity”. In: *Readings in artificial Intelligence and Databases*. Elsevier, 1989, pp. 60–68.

[Con+13] World Wide Web Consortium et al. “SPARQL 1.1 overview”. In: (2013).

[Cor+10] Gianluca Correndo et al. “SPARQL query rewriting for implementing data integration over linked data”. In: *Proceedings of the 2010 EDBT/ICDT Workshops*. 2010, pp. 1–11.

[Cud+13] Philippe Cudré-Mauroux et al. “NoSQL databases for RDF: an empirical evaluation”. In: *International Semantic Web Conference*. Springer. 2013.

[Cur+15] Olivier Curé et al. “On the evaluation of RDF distribution algorithms implemented over apache spark”. In: *arXiv preprint arXiv:1507.02321* (2015).

[Cyg05] Richard Cyganiak. “A relational algebra for SPARQL”. In: *Digital Media Systems Laboratory HP Laboratories Bristol. HPL-2005-170* 35.9 (2005).

[Dai+20] Yuanfei Dai et al. “A survey on knowledge graph embedding: Approaches, applications and benchmarks”. In: *Electronics* 9.5 (2020), p. 750.

[Dal+12] Lorand Dali et al. “Query-independent learning to rank for rdf entity search”. In: *Extended Semantic Web Conference*. Springer. 2012, pp. 484–498.

[Dav+16] Ankur Dave et al. “Graphframes: an integrated api for mixing graph and relational queries”. In: *Proceedings of the fourth international workshop on graph data management experiences and systems*. 2016, pp. 1–8.

[Dav19] Timothy A Davis. “Algorithm 1000: SuiteSparse: GraphBLAS: Graph algorithms in the language of sparse linear algebra”. In: *ACM Transactions on Mathematical Software (TOMS)* 45.4 (2019), pp. 1–25.

[Deb+02a] K. Deb et al. “A fast and elitist multiobjective genetic algorithm: NSGA-II”. In: *IEEE Transactions on Evolutionary Computation* 6.2 (2002), pp. 182–197. DOI: 10.1109/4235.996017.

[Deb+02b] Kalyanmoy Deb et al. “A fast and elitist multiobjective genetic algorithm: NSGA-II”. In: *IEEE Trans. Evol. Comput.* 6.2 (2002), pp. 182–197.

[Del+14] Daniele Dell’Aglio et al. “RSP-QL semantics: A unifying query model to explain heterogeneity of RDF stream processing systems”. In: *International Journal on Semantic Web and Information Systems (IJSWIS)* 10.4 (2014), pp. 17–44.

[Deu+19] Alin Deutsch et al. “Tigergraph: A native MPP graph database”. In: *arXiv preprint arXiv:1901.08248* (2019).

[Dev14] Ramalingam Devakunchari. “Analysis on big data over the years”. In: *International Journal of Scientific and Research Publications* 4.1 (2014), pp. 1–7.

[DL20] Ali Davoudian and Mengchi Liu. “Big data systems: A software engineering perspective”. In: *ACM Computing Surveys (CSUR)* 53.5 (2020), pp. 1–39.

[Dua+11] Songyun Duan et al. “Apples and oranges: a comparison of RDF benchmarks and real RDF datasets”. In: *Proceedings of the 2011*

ACM SIGMOD International Conference on Management of data. 2011, pp. 145–156.

[Dwi+20] Vijay Prakash Dwivedi et al. "Benchmarking graph neural networks". In: *arXiv preprint arXiv:2003.00982* (2020).

[DX15] Lian Duan and Ye Xiong. "Big data analytics and business analytics". In: *Journal of Management Analytics* 2.1 (2015), pp. 1–21.

[E09] EF E F. Codd. "Derivability, redundancy and consistency of relations stored in large data banks". In: *ACM SIGMOD Record* 38.1 (2009), pp. 17–36.

[Ein05] Albert Einstein. "Zur Elektrodynamik bewegter Körper. (German) [On the electrodynamics of moving bodies]". In: *Annalen der Physik* 322.10 (1905), pp. 891–921. DOI: http://dx.doi.org/10.1002/andp.19053221004.

[eta09] Michael Schmidt et.al. "SP
2Bench: A SPARQL Performance Benchmark". In: *ICDE 2009*. Ed. by Yannis E. Ioannidis, Dik Lun Lee, and Raymond T. Ng. 2009, pp. 222–233.

[Fan+20] Yixiang Fang et al. "A survey of community search over big graphs". In: *The VLDB Journal* 29.1 (2020), pp. 353–392.

[Fan+21] Wenfei Fan et al. "GraphScope: a unified engine for big graph processing". In: *Proceedings of the VLDB Endowment* 14.12 (2021), pp. 2879–2892.

[Fer+17] Javier D. Fernández et al. "LOD-a-lot - A Queryable Dump of the LOD Cloud". In: *The Semantic Web - ISWC 2017 - 16th International Semantic Web Conference, Vienna, Austria, October 21-25, 2017, Proceedings, Part II*. Ed. by Claudia d'Amato et al. Vol. 10588. Lecture Notes in Computer Science. Springer, 2017, pp. 75–83.

[Fra+18] Nadime Francis et al. "Cypher: An evolving query language for property graphs". In: *Proceedings of the 2018 International Conference on Management of Data*. 2018, pp. 1433–1445.

[FRP15] Jing Fan, Adalbert Gerald Soosai Raj, and Jignesh M Patel. "The Case Against Specialized Graph Analytics Engines." In: *CIDR*. 2015.

[Gao+18] Libo Gao et al. "Stream WatDiv: A streaming RDF benchmark". In: *Proceedings of the International Workshop on Semantic Big Data*. 2018, pp. 1–6.

[GBB21] Issam Ghabri, Ladjel Bellatreche, and Sadok Ben Yahia. "Energy Efficiency vs. Performance of Analytical Queries: The case of Bitmap Join Indexes". In: *2021 IEEE International Conference on Big Data (Big Data)*. 2021, pp. 3066–3074. DOI: 10.1109/BigData52589.2021.9671307.

[GGZ19] Xintong Guo, Hong Gao, and Zhaonian Zou. "Leon: A distributed rdf engine for multi-query processing". In: *International Conference*

on Database Systems for Advanced Applications. Springer. 2019, pp. 742–759.

[Gha+13] Ahmad Ghazal et al. “Bigbench: Towards an industry standard benchmark for big data analytics”. In: *Proceedings of the 2013 ACM SIGMOD international conference on Management of data*. 2013, pp. 1197–1208.

[GM20] Abraham Gale and Amélie Marian. “Explaining monotonic ranking functions”. In: *Proceedings of the VLDB Endowment* 14.4 (2020), pp. 640–652.

[GMS93] Michel Goossens, Frank Mittelbach, and Alexander Samarin. *The LaTeX Companion*. Reading, Massachusetts: Addison-Wesley, 1993.

[Gon+14] Joseph E Gonzalez et al. “{GraphX}: Graph Processing in a Distributed Dataflow Framework”. In: *11th USENIX symposium on operating systems design and implementation (OSDI 14)*. 2014, pp. 599–613.

[Goy+19] Palash Goyal et al. “Benchmarks for graph embedding evaluation”. In: *arXiv preprint arXiv:1908.06543* (2019).

[Gra+16] Damien Graux et al. “Sparqlgx: Efficient distributed evaluation of sparql with apache spark”. In: *International Semantic Web Conference*. Springer. 2016, pp. 80–87.

[Gra93] Jim Gray. “Database and Transaction Processing Performance Handbook”. In: *The Benchmark Handbook for Database and Transaction Systems (2nd Edition)*. 1993.

[GS21] Claudio Gutierrez and Juan F Sequeda. “Knowledge graphs”. In: *Communications of the ACM* 64.3 (2021), pp. 96–104.

[Hag17] John Hagerty. *Planning Guide for Data and Analytics*. `https://www.cartagena99.com/recursos/alumnos/apuntes/2017_planning_guide_for_data_analytics.pdf`. [Online; accessed 4-Sep-2021]. 2017.

[Hal+16] Mara Hallo et al. “Current state of Linked Data in digital libraries”. In: *Journal of Information Science* 42.2 (2016).

[Han+15] Rui Han et al. “Benchmarking big data systems: State-of-the-art and future directions”. In: *arXiv preprint arXiv:1506.01494* (2015).

[Has+16] Ali Hasnain et al. “Sportal: profiling the content of public sparql endpoints”. In: *International Journal on Semantic Web and Information Systems (IJSWIS)* 12.3 (2016), pp. 134–163.

[Has+17] Mohamed S. Hassan et al. “Empowering In-Memory Relational Database Engines with Native Graph Processing”. In: *CoRR* abs/1709.06715 (2017). arXiv: `1709.06715`. URL: `http://arxiv.org/abs/1709.06715`.

[Has+18] Mohamed S Hassan et al. “Grfusion: Graphs as first-class citizens in main-memory relational database systems”. In: *Proceedings of*

the 2018 International Conference on Management of Data. 2018, pp. 1789–1792.

[HB19] Mahmudul Hassan and Srividya K. Bansal. "Data Partitioning Scheme for Efficient Distributed RDF Querying Using Apache Spark". In: *2019 IEEE 13th International Conference on Semantic Computing (ICSC)*. 2019, pp. 24–31. DOI: 10.1109/ICOSC.2019.8665614.

[HL14] H Howie Huang and Hang Liu. "Big data machine learning and graph analytics: Current state and future challenges". In: *2014 IEEE international conference on big data (Big Data)*. IEEE. 2014, pp. 16–17.

[HLS+09] Steve Harris, Nick Lamb, Nigel Shadbolt, et al. "4store: The design and implementation of a clustered RDF store". In: *5th International Workshop on Scalable Semantic Web Knowledge Base Systems (SSWS2009)*. Vol. 94. 2009.

[HMF15] Antonio Hernández-Illera, Miguel A Martnez-Prieto, and Javier D Fernández. "Serializing RDF in compressed space". In: *2015 Data Compression Conference*. IEEE. 2015, pp. 363–372.

[HN13] Michael Hausenblas and Jacques Nadeau. "Apache drill: interactive ad-hoc analysis at scale". In: *Big data* 1.2 (2013), pp. 100–104.

[Hog+21] Aidan Hogan et al. "Knowledge graphs". In: *Synthesis Lectures on Data, Semantics, and Knowledge* 12.2 (2021), pp. 1–257.

[Hu+20] Weihua Hu et al. "Open graph benchmark: Datasets for machine learning on graphs". In: *Advances in neural information processing systems* 33 (2020), pp. 22118–22133.

[Hua+14] Yin Huai et al. "Major technical advancements in apache hive". In: *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*. 2014, pp. 1235–1246.

[Ios+20] Alexandru Iosup et al. "The LDBC Graphalytics Benchmark". In: *arXiv preprint arXiv:2011.15028* (2020).

[IP19] Todor Ivanov and Matteo Pergolesi. "The impact of columnar file formats on SQL-on-hadoop engine performance: A study on ORC and Parquet". In: *Concurrency and Computation: Practice and Experience* (2019), e5523.

[JS22] Guodong Jin and Semih Salihoglu. "Making RDBMSs Efficient on Graph Workloads Through Predefined Joins". In: *Proc. VLDB Endow.* 15.5 (2022), pp. 1011–1023. URL: https://www.vldb.org/pvldb/vol15/p1011-jin.pdf.

[JST17] Daniel Janke, Steffen Staab, and Matthias Thimm. "Koral: A Glass Box Profiling System for Individual Components of Distributed RDF Stores." In: *BLINK/NLIWoD3@ ISWC*. 2017.

[Jun+15] Martin Junghanns et al. “Gradoop: Scalable graph data management and analytics with hadoop”. In: *arXiv preprint arXiv:1506.00548* (2015).

[Jun+17] Martin Junghanns et al. “Management and analysis of big graph data: current systems and open challenges”. In: *Handbook of Big Data Technologies*. Springer, 2017, pp. 457–505.

[Kan+17] Chathura Kankanamge et al. “Graphflow: An active graph database”. In: *Proceedings of the 2017 ACM International Conference on Management of Data*. 2017, pp. 1695–1698.

[Kha+16] Mahmoud Abo Khamis et al. “Joins via geometric resolutions: Worst case and beyond”. In: *ACM Transactions on Database Systems (TODS)* 41.4 (2016), pp. 1–45.

[Kim+15] Jinha Kim et al. “Taming subgraph isomorphism for RDF query processing”. In: *arXiv preprint arXiv:1506.01973* (2015).

[KK98] George Karypis and Vipin Kumar. “A fast and high quality multilevel scheme for partitioning irregular graphs”. In: *SIAM Journal on scientific Computing* 20.1 (1998), pp. 359–392.

[KNT15] Foteini Katsarou, Nikos Ntarmos, and Peter Triantafillou. “Performance and scalability of indexed subgraph query processing methods”. In: *Proceedings of the VLDB Endowment* 8.12 (2015), pp. 1566–1577.

[Knu] Donald Knuth. *Knuth: Computers and Typesetting*. URL: `http://www-cs-faculty.stanford.edu/%5C~%7B%7Duno/abcde.html`.

[LA15] Juan C Leyva Lopez and Pavel A Alvarez Carrillo. “Accentuating the rank positions in an agreement index with reference to a consensus order”. In: *International Transactions in Operational Research* 22.6 (2015), pp. 969–995.

[Lan+01] Doug Laney et al. “3D data management: Controlling data volume, velocity and variety”. In: *META group research note* 6.70 (2001), p. 1.

[Lan01] Douglas Laney. *3D Data Management: Controlling Data Volume, Velocity, and Variety*. Tech. rep. 2001.

[LBV18] Matteo Lissandrini, Martin Brugnara, and Yannis Velegrakis. “Beyond macrobenchmarks: microbenchmark-based graph database evaluation”. In: *Proceedings of the VLDB Endowment* 12.4 (2018), pp. 390–403.

[LC15] Juan Carlos Leyva López and Pavel Anselmo Álvarez Carrillo. “Accentuating the rank positions in an agreement index with reference to a consensus order”. In: *Int. Trans. Oper. Res.* 22.6 (2015), pp. 969–995.

[Le+12] Wangchao Le et al. “Scalable multi-query optimization for SPARQL”. In: *2012 IEEE 28th International Conference on Data Engineering*. IEEE. 2012, pp. 666–677.

[Lep+20] Katerina Lepenioti et al. “Prescriptive analytics: Literature review and research challenges”. In: *International Journal of Information Management* 50 (2020), pp. 57–70.

[LG15] Jeffrey R Lacasse and Eileen Gambrill. “Making assessment decisions: Macro, mezzo, and micro perspectives”. In: *Critical Thinking in Clinical Assessment and Diagnosis*. 2015.

[Lin18] Jimmy Lin. “Scale up or scale out for graph processing?” In: *IEEE Internet Computing* 22.3 (2018), pp. 72–78.

[Liu+09] Tie-Yan Liu et al. “Learning to rank for information retrieval”. In: *Foundations and Trendső in Information Retrieval* 3.3 (2009), pp. 225–331.

[LM09] Justin J Levandoski and Mohamed F Mokbel. “RDF data-centric storage”. In: *2009 IEEE International Conference on Web Services*. IEEE. 2009, pp. 911–918.

[Low+14] Yucheng Low et al. “Graphlab: A new framework for parallel machine learning”. In: *arXiv preprint arXiv:1408.2041* (2014).

[Lu+19] Jiaheng Lu et al. “Speedup your analytics: Automatic parameter tuning for databases and big data systems”. In: *Proceedings of the VLDB Endowment* (2019).

[MAA18] Amgad Madkour, Ahmed M Aly, and Walid G Aref. “Worq: Workload-driven rdf query processing”. In: *International Semantic Web Conference*. Springer. 2018, pp. 583–599.

[Mal+10] Grzegorz Malewicz et al. “Pregel: a system for large-scale graph processing”. In: *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*. 2010, pp. 135–146.

[Mar+15] Claudio Martella et al. *Practical graph analytics with apache giraph*. Vol. 1. Springer, 2015.

[Mar14] Volker Markl. “Breaking the chains: On declarative data analysis and data independence in the big data era”. In: *Proceedings of the VLDB Endowment* 7.13 (2014), pp. 1730–1733.

[McB01] Brian McBride. “Jena: Implementing the RDF Model and Syntax Specification.” In: *SemWeb*. Vol. 1. 2001, pp. 23–28.

[Mel+10] Sergey Melnik et al. “Dremel: interactive analysis of web-scale datasets”. In: *Proceedings of the VLDB Endowment* 3.1-2 (2010), pp. 330–339.

[Men+16] Xiangrui Meng et al. “Mllib: Machine learning in apache spark”. In: *The Journal of Machine Learning Research* 17.1 (2016), pp. 1235–1241.

[Mic15] Reynold S. Xin et.al. Michael Armbrust. “Spark SQL: Relational Data Processing in Spark”. In: *Proceedings of the 2015 ACM SIG-*

MOD International Conference on Management of Data, Australia. 2015, pp. 1383–1394.

[Mil13] Justin J Miller. "Graph database applications and concepts with Neo4j". In: *Proceedings of the southern association for information systems conference, Atlanta, GA, USA*. Vol. 2324. 36. 2013.

[MRT18] Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar. *Foundations of machine learning*. MIT press, 2018.

[MS19] Amine Mhedhbi and Semih Salihoglu. "Optimizing subgraph queries by combining binary and worst-case optimal joins". In: *arXiv preprint arXiv:1903.02076* (2019).

[MSO12] Craig Macdonald, Rodrygo LT Santos, and Iadh Ounis. "On the usefulness of query features for learning to rank". In: *Proceedings of the 21st ACM international conference on Information and knowledge management*. 2012, pp. 2559–2562.

[MW95] Alberto O Mendelzon and Peter T Wood. "Finding regular simple paths in graph databases". In: *SIAM Journal on Computing* 24.6 (1995), pp. 1235–1258.

[Ngo+14] Hung Q. Ngo et al. "Beyond worst-case analysis for joins with minesweeper". In: *Proceedings of the 33rd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS'14, Snowbird, UT, USA, June 22-27, 2014*. Ed. by Richard Hull and Martin Grohe. ACM, 2014, pp. 234–245. DOI: 10.1145/2594538.2594547. URL: https://doi.org/10.1145/2594538.2594547.

[Ngo+18] Hung Q Ngo et al. "Worst-case optimal join algorithms". In: *Journal of the ACM (JACM)* 65.3 (2018), pp. 1–40.

[NM+19] Engineering National Academies of Sciences, Medicine, et al. "Reproducibility and replicability in science". In: (2019).

[NW08] Thomas Neumann and Gerhard Weikum. "RDF-3X: a RISC-style engine for RDF". In: *Proceedings of the VLDB Endowment* 1.1 (2008), pp. 647–659.

[NW09] Thomas Neumann and Gerhard Weikum. "Scalable join processing on very large RDF graphs". In: *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*. 2009, pp. 627–640.

[NZE05] Patrick Ngatchou, Anahita Zarei, and A El-Sharkawi. "Pareto multi objective optimization". In: *Proceedings of the 13th International Conference on, Intelligent Systems Application to Power Systems*. IEEE. 2005, pp. 84–91.

[OG+08] Alisdair Owens, Nick Gibbins, et al. "Effective benchmarking for RDF stores using synthetic data". In: (2008).

[OS16] Dan Olteanu and Maximilian Schleich. "Factorized databases". In: *ACM SIGMOD Record* 45.2 (2016), pp. 5–16.

[PAG09] Jorge Pérez, Marcelo Arenas, and Claudio Gutierrez. “Semantics and complexity of SPARQL”. In: *ACM Transactions on Database Systems (TODS)* 34.3 (2009), pp. 1–45.

[PAG10] Jorge Pérez, Marcelo Arenas, and Claudio Gutierrez. “nSPARQL: A navigational language for RDF”. In: *Journal of Web Semantics* 8.4 (2010), pp. 255–270.

[Par15] Marcus et. al. Paradies. “GRAPHITE: an extensible graph traversal framework for relational database management systems”. In: *Proceedings of the 27th International Conference on Scientific and Statistical Database Management*. 2015, pp. 1–12.

[PCS14] Freddy Priyatna, Oscar Corcho, and Juan Sequeda. “Formalisation and experiences of R2RML-based SPARQL to SQL query translation using morph”. In: *Proceedings of the 23rd international conference on World wide web*. 2014, pp. 479–490.

[Pen+19] Peng Peng et al. “Optimizing Multi-Query Evaluation in Federated RDF Systems”. In: *IEEE Transactions on Knowledge and Data Engineering* (2019).

[Per+15] Yonathan Perez et al. “Ringo: Interactive graph analytics on big-memory machines”. In: *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*. 2015, pp. 1105–1110.

[Pha+15] Minh-Duc Pham et al. “Deriving an emergent relational schema from RDF data”. In: *Proceedings of the 24th International Conference on World Wide Web*. 2015, pp. 864–874.

[PMV22] George Papastefanatos, Marios Meimaris, and Panos Vassiliadis. “Relational schema optimization for RDF-based knowledge graphs”. In: *Information Systems* 104 (2022), p. 101754.

[Rag+20] Mohamed Ragab et al. “Towards Making Sense of Spark-SQL Performance for Processing Vast Distributed RDF Datasets”. In: *Proceedings of The International Workshop on Semantic Big Data@ Sigmod’20*. New York, NY, USA, 2020. ISBN: 9781450379748.

[Rag+21] Mohamed Ragab et al. “An In-depth Investigation of Large-scale RDF Relational Schema Optimizations Using Spark-SQL”. In: *Processing of Big Data (DOLAP) co-located with the 24th (EDBT/ICDT 2021), Nicosia, Cyprus,2021*. 2021.

[Rag20] Mohamed Ragab. “Large Scale Querying and Processing for Property Graphs”. In: *Proceedings of the 22nd International Workshop on Design, Optimization, Languages and Analytical Processing of Big Data (DOLAP) co-located with EDBT/ICDT 2020 Joint Conference , Copenhagen, Denmark, March 30, 2020*. Ed. by Il-Yeol Song, Katja Hose, and Oscar Romero. Vol. 2572. CEUR Workshop Pro-

ceedings. CEUR-WS.org, 2020, pp. 79–83. URL: http://ceur-ws.org/Vol-2572/short3.pdf.

[RAT21] M. Ragab, F. M. Awaysheh, and R. Tommasini. "Bench-Ranking: A First Step Towards Prescriptive Performance Analyses For Big Data Frameworks". In: *2021 IEEE International Conference on Big Data (Big Data)*. Los Alamitos, CA, USA: IEEE Computer Society, 2021, pp. 241–251. DOI: 10.1109/BigData52589.2021.9671277. URL: https://doi.ieeecomputersociety.org/10.1109/BigData52589.2021.9671277.

[RAT22] Mohamed Ragab, Adam Satria Adidarma, and Riccardo Tommasini. *PAPyA: Performance Analysis of Large RDF Graphs Processing Made Easy*. 2022. DOI: 10.48550/ARXIV.2209.06877. URL: https://arxiv.org/abs/2209.06877.

[RK15] M Janga Reddy and D Nagesh Kumar. "Elitist-Mutated multi-objective particle swarm optimization for engineering design". In: *Encyclopedia of Information Science and Technology, Third Edition*. IGI Global, 2015, pp. 3534–3545.

[RN11] Marko A Rodriguez and Peter Neubauer. "A path algebra for multi-relational graphs". In: *2011 IEEE 27th International Conference on Data Engineering Workshops*. IEEE. 2011, pp. 128–131.

[Rod15] Marko A Rodriguez. "The gremlin graph traversal machine and language (invited talk)". In: *Proceedings of the 15th Symposium on Database Programming Languages*. 2015, pp. 1–10.

[RR15] Mariano Rodriguez-Muro and Martin Rezk. "Efficient SPARQL-to-SQL with R2RML mappings". In: *Journal of Web Semantics* 33 (2015), pp. 141–169.

[RTS19] Mohamed Ragab, Riccardo Tommasini, and Sherif Sakr. "Benchmarking Spark-SQL under Alliterative RDF Relational Storage Backends." In: *QuWeDa@ ISWC*. 2019, pp. 67–82.

[Rud+13] Michael Rudolf et al. "The graph story of the SAP HANA database". In: *Datenbanksysteme für Business, Technologie und Web (BTW) 2037* (2013).

[RW16] Xuguang Ren and Junhu Wang. "Multi-query optimization for subgraph isomorphism search". In: *Proceedings of the VLDB Endowment* 10.3 (2016), pp. 121–132.

[RWE15] Ian Robinson, Jim Webber, and Emil Eifrem. *Graph databases: new opportunities for connected data*. " O'Reilly Media, Inc.", 2015.

[SA10] Sherif Sakr and Ghazi Al-Naymat. "Relational processing of RDF queries: a survey". In: *ACM SIGMOD Record* 38.4 (2010), pp. 23–28.

[Sag+22] Tomer Sagi et al. "A design space for RDF data representations". In: *The VLDB Journal* 31.2 (2022), pp. 347–373.

[Sah+20] Siddhartha Sahu et al. “The ubiquity of large graphs and surprising challenges of graph processing: extended survey”. In: *The VLDB Journal* 29.2 (2020), pp. 595–618.

[Sak+21a] Sherif Sakr et al. “The future is big graphs: a community view on graph processing systems”. In: *Communications of the ACM* 64.9 (2021), pp. 62–71.

[Sak+21b] Sherif Sakr et al. “The future is big graphs: a community view on graph processing systems”. In: *Communications of the ACM* 64.9 (2021), pp. 62–71.

[Sak09] Sherif Sakr. “GraphREL: A Decomposition-Based and Selectivity-Aware Relational Framework for Processing Sub-graph Queries”. In: *DASFAA*. 2009.

[Sal+16] Muhammad Saleem et al. “SPARQL Querying Benchmarks”. In: *Tutorial at ISWC* (2016).

[Sal+19] Muhammad Saleem et al. “How Representative Is a SPARQL Benchmark? An Analysis of RDF Triplestore Benchmarks?” In: *The World Wide Web Conference*. ACM. 2019, pp. 1623–1633.

[SB18] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[SBB20] Filippo Schiavio, Daniele Bonetta, and Walter Binder. “Dynamic speculative optimizations for SQL compilation in Apache Spark”. In: *Proceedings of the VLDB Endowment* 13.5 (2020), pp. 754–767.

[Sch+08a] Michael Schmidt et al. “An Experimental Comparison of RDF Data Management Approaches in a SPARQL Benchmark Scenario”. In: *The Semantic Web - ISWC 2008, 7th International Semantic Web Conference, ISWC 2008, Karlsruhe, Germany, October 26-30, 2008. Proceedings*. 2008, pp. 82–97.

[Sch+08b] Michael Schmidt et al. “An Experimental Comparison of RDF Data Management Approaches in a SPARQL Benchmark Scenario”. In: *The Semantic Web - ISWC 2008, 7th International Semantic Web Conference, ISWC 2008, Karlsruhe, Germany, October 26-30, 2008. Proceedings*. Ed. by Amit P. Sheth et al. Vol. 5318. Lecture Notes in Computer Science. Springer, 2008, pp. 82–97. DOI: 10.1007/978-3-540-88564-1_6.

[Sch+14] Alexander Schätzle et al. “Sempala: Interactive SPARQL query processing on hadoop”. In: *ISWC*. 2014.

[Sch+15] Alexander Schätzle et al. “S2X: graph-parallel querying of RDF with GraphX”. In: *Biomedical Data Management and Graph Online Querying*. Springer, 2015, pp. 155–168.

[Sch+16] Alexander Schätzle et al. “S2RDF: RDF querying with SPARQL on spark”. In: *Proceedings of the VLDB Endowment* 9.10 (2016), pp. 804–815.

[Sch+19] Guilherme Schievelbein et al. “Exploiting Wide Property Tables Empowered by Inverse Properties for Efficient Distributed SPARQL Query Evaluation.” In: *ISWC (Satellites)*. 2019, pp. 81–84.

[Sid+08] Lefteris Sidirourgos et al. “Column-store support for RDF data management: not all swans are white”. In: *PVLDB* 1.2 (2008), pp. 1553–1563. DOI: 10.14778/1454159.1454227. URL: http://www.vldb.org/pvldb/1/1454227.pdf.

[Sim+14] David Simmen et al. “Large-scale graph analytics in aster 6: bringing context to big data discovery”. In: *Proceedings of the VLDB Endowment* 7.13 (2014), pp. 1405–1416.

[SN09] Saa Singer and John Nelder. “Nelder-mead algorithm”. In: *Scholarpedia* 4.7 (2009), p. 2928.

[SÖ18] Semih Salihoglu and M Tamer Özsu. “Response to Scale Up or Scale Out for Graph Processing”. In: *IEEE Internet Computing* 22.5 (2018), pp. 18–24.

[SPL11] Alexander Schätzle, Martin Przyjaciel-Zablocki, and Georg Lausen. “PigSPARQL: Mapping SPARQL to pig latin”. In: *Proceedings of the International Workshop on Semantic Web Information Management*. 2011, pp. 1–8.

[SS15] Reza Shokri and Vitaly Shmatikov. “Privacy-preserving deep learning”. In: *Proceedings of the 22nd ACM SIGSAC conference on computer and communications security*. 2015, pp. 1310–1321.

[Ste18] US Stefan Plantikow. “Summary Chart of Cypher, PGQL, and G-Core”. In: (2018).

[Sun+15] Wen Sun et al. “Sqlgraph: An efficient relational-based property graph store”. In: *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*. 2015, pp. 1887–1901.

[Szá19] Gábor Szárnyas. “Query, analysis, and benchmarking techniques for evolving property graphs of software systems”. In: (2019).

[Tah+19] Ruby Y Tahboub et al. “Towards compiling graph queries in relational engines”. In: *Proceedings of the 17th ACM SIGPLAN International Symposium on Database Programming Languages*. 2019, pp. 30–41.

[Tan+15] Mingjie Tang et al. “Similarity group-by operators for multi-dimensional relational data”. In: *IEEE Transactions on Knowledge and Data Engineering* 28.2 (2015), pp. 510–523.

[Tia+20] Yuanyuan Tian et al. “IBM db2 graph: Supporting synergistic and retrofittable graph queries inside IBM db2”. In: *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 2020, pp. 345–359.

[Tu+14] Yi-Cheng Tu et al. “A system for energy-efficient data management”. In: *ACM SIGMOD Record* 43.1 (2014), pp. 21–26.

[Van+17] Dana Van Aken et al. “Automatic database management system tuning through large-scale machine learning”. In: *Proceedings of the 2017 ACM International Conference on Management of Data*. 2017, pp. 1009–1024.

[Van+21] Dana Van Aken et al. “An inquiry into machine learning-based automatic configuration tuning services on real-world database management systems”. In: *Proceedings of the VLDB Endowment* 14.7 (2021), pp. 1241–1253.

[Vav+13] Vinod Kumar Vavilapalli et al. “Apache hadoop yarn: Yet another resource negotiator”. In: *Proceedings of the 4th annual Symposium on Cloud Computing*. 2013, pp. 1–16.

[Wan+14] Lei Wang et al. “Bigdatabench: A big data benchmark suite from internet services”. In: *2014 IEEE 20th international symposium on high performance computer architecture (HPCA)*. IEEE. 2014, pp. 488–499.

[Wil+03] Kevin Wilkinson et al. “Efficient RDF Storage and Retrieval in Jena2.” In: *SWDB*. Vol. 3. Citeseer. 2003, pp. 131–150.

[WKB08] Cathrin Weiss, Panagiotis Karras, and Abraham Bernstein. “Hexastore: sextuple indexing for semantic web data management”. In: *PVLDB* 1.1 (2008).

[WS19] Marcin Wylot and Sherif Sakr. “Framework-Based Scale-Out RDF Systems”. In: *Encyclopedia of Big Data Technologies*. 2019. DOI: 10.1007/978-3-319-63962-8_225-1. URL: https://doi.org/10.1007/978-3-319-63962-8%5C_225-1.

[Wyl+18] Marcin Wylot et al. “RDF data storage and query processing schemes: A survey”. In: *ACM Computing Surveys (CSUR)* 51.4 (2018), p. 84.

[WZZ17] Dong Wang, Lei Zou, and Dongyan Zhao. “g-store: Querying Large Spatiotemporal RDF Graphs”. In: *Data and Information Management* 1.2 (2017), pp. 84–103.

[XD17] Konstantinos Xirogiannopoulos and Amol Deshpande. “Extracting and analyzing hidden graphs from relational databases”. In: *Proceedings of the 2017 ACM International Conference on Management of Data*. 2017, pp. 897–912.

[Xio+13] Wen Xiong et al. “A characterization of big data benchmarks”. In: *2013 IEEE international conference on big data*. IEEE. 2013, pp. 118–125.

[XKD15] Konstantinos Xirogiannopoulos, Udayan Khurana, and Amol Deshpande. “Graphgen: Exploring interesting graphs in relational data”. In: *Proceedings of the VLDB Endowment* 8.12 (2015), pp. 2032–2035.

[Xu19] Chonghuan Xu. “A big-data oriented recommendation method based on multi-objective optimization”. In: *Knowledge-Based Systems* 177 (2019), pp. 11–21.

[Zah+13] Matei Zaharia et al. “Discretized streams: Fault-tolerant streaming computation at scale”. In: *Proceedings of the twenty-fourth ACM symposium on operating systems principles*. 2013, pp. 423–438.

[Zah+16] Matei Zaharia et al. “Apache Spark: a unified engine for big data processing”. In: *Commun. ACM* 59.11 (2016), pp. 56–65. DOI: 10.1145/2934664. URL: http://doi.acm.org/10.1145/2934664.

[Zer+20] Eleftherios Zervakis et al. “Efficient Continuous Multi-Query Processing over Graph Streams”. In: *23rd International Conference on Extending Database Technology, EDBT 2020*. OpenProceedings. org. 2020, pp. 13–24.

[Zha+19] Kangfei Zhao et al. “Sql-g: Efficient graph analytics by sql”. In: *IEEE Transactions on Knowledge and Data Engineering* 33.5 (2019), pp. 2237–2251.

[Zha17] Yan Zhang. “Efficient Structure-aware OLAP Query Processing over Large Property Graphs”. MA thesis. University of Waterloo, 2017.

[Zou+11a] Lei Zou et al. “GStore: Answering SPARQL Queries via Subgraph Matching”. In: *Proc. VLDB Endow.* 4.8 (2011), pp. 482–493. ISSN: 2150-8097. DOI: 10.14778/2002974.2002976. URL: https://doi.org/10.14778/2002974.2002976.

[Zou+11b] Lei Zou et al. “gStore: answering SPARQL queries via subgraph matching”. In: *Proceedings of the VLDB Endowment* 4.8 (2011), pp. 482–493.

[ZY17] Kangfei Zhao and Jeffrey Xu Yu. “All-in-one: graph processing in RDBMSs revisited”. In: *Proceedings of the 2017 ACM International Conference on Management of Data*. 2017, pp. 1165–1180.

8. APPENDIX A: REFLECTIONS ON THE STATE-OF-THE-ART OF PROCESSING AND QUERYING LARGE KNOWLEDGE GRAPHS

In this appendix, we present more details about the state-of-the-art related work in the area of big (graph) data processing and benchmarking. We opted to put these details here as reference point for the readers who may keen to know more about the underlying details of this area of research.

8.0.1. Graph Management Systems

There has been a significant prevalence of the work on graph processing in both academia and in industry[Sak+21b]. This led to a surge in the number of different commercial and research systems for storing, querying, processing, and managing graphs [Sah+20]. It is important to understand that the graph management systems can be divided into *two* main categories, based on their design purpose and type of workloads [Sah+20]. The first category includes graph processing engines, which analyze the graph *iteratively* to gain global insights. The second category includes graph databases to query graphs.

Table 35 summarizes a simple comparison between those two categories of graph computation systems. Nonetheless, we discuss in detail the two categories with example systems from both academia and industry in the following sections.

1. **Graph Query Systems (Graph Databases)**: Recently, graph databases are the most popular choice for performing querying and local computations over graphs [Sah+20; Jun+17]. In such systems, data is typically stored natively in the form of graphs instead of relational tables. Thus, data manipulation is expressed by *graph-oriented* operations and type constructors. Graph Databases mainly use *declarative* query languages for interacting with graph query workloads [Ang+17]. The nature of graph query language (e.g., *Cypher*, *SPARQL*, *Gremlin*, *PGQL*, and *GraphQL*) depends on the underlying nature of graph model that the graph DB supports [Ang+17]. Currently, the most two popular graph data models are *Edge-Labeled Graphs* (i.e., *RDF* data model), where nodes are connected by directed, labeled edges, and *Property Graphs*, where both nodes and edges can further embed attributes [Sak+21b]. Examples of popular graph databases designed for managing property graphs are *Neo4j* [1], *Amazon Neptune* [2], *Microsoft CosmosDB* [3], *Titan* [4], and *Apache TinkerPop* [5]. Other popular RDF graph databases also exist, e.g., *Allegrograph* [6], *Stardog* [7], *Virtuoso* [8], and *Blazegraph* [9].

[1] `https://neo4j.com/`
[2] `https://aws.amazon.com/neptune/`
[3] `https://azure.microsoft.com/en-us/products/cosmos-db/`
[4] `https://titan.thinkaurelius.com/`
[5] `https://tinkerpop.apache.org/`
[6] `https://allegrograph.com/`
[7] `https://www.stardog.com/platform/features/high-performance-graph-database/`
[8] `https://virtuoso.openlinksw.com/`
[9] `https://blazegraph.com/`

	Graph Database Systems	Graph Analytics Systems
Purpose	Querying	Analytical (algorithms)
Workloads Excel In	OLTP Workloads ((local traversals)	OLAP & iterative, recursive workloads
Interact with	Typically, limited portion of the graph	Entire graph (can be more than once)
Response	Real Time	Long Running (large graphs)
Computation Nature	Centralized	Distributed
Declarative Graph Querying	Yes	No
Large Query Workloads	Limited	No
System Examples	Neo4j, Neptun, StarDog, CosmosDB	Apache Giraph, Gelly, Spark GraphX

Table 35: Graph Database Systems and Distributed Graph Processing Systems.

2. **Graph Analytics Systems**: Although the fact that, graph databases excel at querying graphs, they usually cannot efficiently process large graphs in an iterative manner [Jun+17]. Thus, graph processing engines are used mainly for implementing graph algorithms such as *PageRank*, *Triangle Counting* or *Connected Components*, and many more. These algorithms require iterative processing over the whole graph, while other algorithms such as *Single Source Shortest Path* might require touching a large portion of the graph (i.e., OLAP graph processing).

   Graph analytical engines are mainly *distributed*, and they can be divided into *two* main categories based on their programming model. The first category is the "*Vertex-Centric*" ("*Think-Like-A-Vertex*") processing programming paradigm. This defines graph computations as an iterative process between communicating vertices in the graph. This model defined the *Pregel* [Mal+10] model firstly developed by *Google* and afterwards was adopted by several graph processing frameworks such as *Apache Spark GraphX* [Gon+14], *Apache Flink* (i.e., *Gelly* [Car+15] and *Gradoop* [Jun+15] systems), and the *Apache Giraph* [Mar+15]. The second category is the *linear algebra*-based systems, which defines the graph computation with matrix operations. Graph processing systems that reside in this category are such as *GraphBLAS* [Dav19] and *LA3* [Ahm+18].

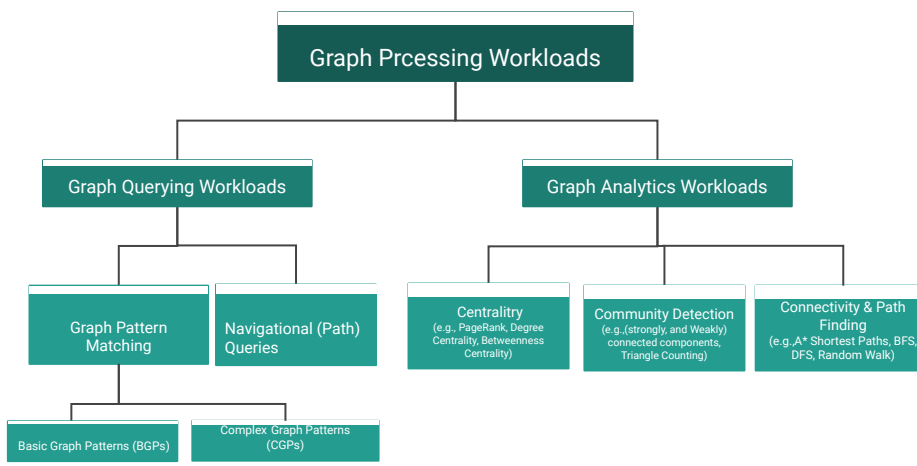


Figure 51: Graph processing workloads main categories, with some examples.

8.1. Graph Analytics Systems meet Graph Querying Systems

With the enormous growth in volumes of graph data, the data community aims to address all aspects of graph query and processing workloads (see Figure 51). One of the common challenges of large-scale graph processing is the efficient evaluation of graph queries [Sak+21; Rag20]. The distributed nature of large graph processing systems fueled the need for large graph analytics. However, the area of querying large graphs is still under development [Sak+21]. Indeed, a distributed native graph solution for querying large graphs is still missing [Sak+21].

There are two attempts towards filling this gap. First, *GraphFrames* [Dav+16] is a package[10] designed on top of Sparks DataFrames. GraphFrames benefits from the scalability and high performance of Spark DataFrames. This package provides an interface for implementing graph algorithms on top of DataFrames, as well as incorporating *simple* graph pattern matching with *fixed-length* patterns (i.e., called *motifs*). Thus, GraphFrames provides an interface for querying large graphs but in a limited (*non-declarative*) manner. Another project was launched by *Neo4j* in a system called *"Morpheus"*. Morpheus [11] aimed at enabling large graph querying over large volumes of graphs. Particularly, it was designed to enable the evaluation of a declarative graph query language (i.e., *Cypher* [Fra+18]) over large property graphs using the capabilities of DataFrames on top of Apache Spark. Nevertheless, *Neo4j* declared that this project is no longer actively maintained.

Thus, the area of enabling graph querying over large graphs using declarative languages is not fully mature [Sak+21; Rag20]. Currently, practitioners seeking large graph query workloads can only use the already existing limited large graph querying systems such as GraphFrames, or Morpheus (with no support).

8.2. Relational Graph Processing: State-of-the-Art

The increasing demand for querying large graphs has triggered the BD community to direct attention to falling back to the mature relational (BD) systems for performing large graph querying using declarative capabilities of SQL [Sch+16; Sch+14; CFL18; Aba+07]. These relational solutions come with *35+* years of research on efficient storage and querying. More importantly, relational-backed stores offer important features that are mostly lacking from native graph querying systems, namely, scalability, industrial-strength transaction support, compression, and security, to name a few.

8.2.1. Querying Large (RDF) Knowledge Graphs

The same challenges of finding a suitable native graph system for querying large graphs exist in the realm of (RDF) Knowledge Graphs (KGs). The KGs empower several applications via representing large collections of structured knowledge. Vast RDF KGs (e.g. *DBPedia*, *WikiData*, *Yago*, and *Bio2RDF*), and are now publicly available, and have billions of triples.

Centralized RDF engines, e.g., Apache Jena [McB01], *RDF- 3X* [NW08], and *gStore* [WZZ17], provide native graph processing and querying of RDF datasets with the full expressive capabilities of the *SPARQL* for capturing complex information needs, as well as

[10] `https://github.com/graphframes/graphframes`

[11] `https://github.com/opencypher/morpheus.`

harnessing the underlying data semantics. Moreover, those SPARQL triplestores are flexible and have the power of graph navigation with recursive queries and applying negations and other complex Basic Graph patterns (BGPs).

Nonetheless, those native solutions are centralized and thus cannot handle large-scale RDF datasets effectively [Bor+13; Pha+15]. They are also limited in terms of optimizers. SPARQL optimization is limited after exceeding a specific number of triple pattern joins (e.g., 12 in Virtuoso) [Pha+15]. RDF architect has no grip on data locality, even *Hexastorage* (i.e., indexing all orders of RDF components permutations) does not save from lack of locality. Thus, the need for processing large RDF datasets calls for innovative solutions to store, analyze, and query these massive RDF datasets. This call leads the Semantic Web community to leverage Big Data (BD) processing frameworks (e.g., Apache Spark, Hive, and Impala) to process large RDF datasets.

8.2.2. Examples of relational BD RDF querying solutions

Spark-based RDF querying solutions: Many systems exploited the relational interface of the Spark framework (i.e., Spark-SQL) for building scalable RDF querying engines. For example, *S2RDF* (SPARQL on Spark for RDF) [Sch+16] introduced a relational schema for encoding RDF data called ExtVP (the Extended Vertical Partitioning) that extends the Vertical Partitioning (VP) schema introduced by *Abadi et.al.* [Aba+07]. The ExtVP uses a *semi-join*-based reduction technique in order to minimize the query input size regardless of its pattern shape (i.e., *Star-shaped*, *Linear-Shaped*, *Snowflake*, or any other *Complex*(i.e. Hybrid) shapes). In S2RDF, based on this schema the SPARQL queries are easily mapped into SQL queries using *Jena ARQ* framework. On the same side of S2RDF, *WORQ* [MAA18] is a query workload-driven framework that optimizes the performance of Spark-SQL for querying large RDF graph datasets represented in the VP relational schema. In particular, WORQ utilized *Bloom* filters for computing online join reductions (among VP tables) of frequent triple patterns in the workload. In addition, this framework showed that caching those intermediate join reductions excel caching final results. Last but not least, *Prost* [CFL18] is another spark-based system that used the relational (SQL) interface of Spark for querying large RDF graphs. In particular, *Prost* combined a *mixture* of two existing RDF relational schemas, i.e., Vertical Partitioning tables (VP) with the Property Table (PT) schema.

Other relational RDF querying solutions: relied on the distributed nature of other big relational systems, such as Hadoop, Impala, Hive. For example, *Sempala* [Sch+14] is a distributed RDF query system that translates SPARQL into SQL which runs on top of *Apache Impala* [12]. The *Sempala* system stores the whole RDF graph in a unified relational layout (table) called "Wide Property Table" (WPT). In order to mitigate the effect of WPT sparsity, Sempala stored this unified sparse table in the columnar Parquet file format over HDFS. Parquet is very efficient in storing wide schemes with hundreds of columns while accessing only a few of them in a request. Moreover, NULL values are not stored explicitly in Parquet as they can be determined by the definition levels [Sch+14]. *PigSPARQL* [SPL11] follows a similar approach as Sempala but uses *Pig* as the underlying system. It stores RDF data in a vertically partitioned schema (VP) similar to S2RDF [Sch+16].

[12] `https://impala.apache.org/`

8.3. Challenges of Relational BD Systems for Querying of Large Graphs

Although it may seem logical to implement graph processing within relational systems, the difference among the relational and graph representations, and *functional mismatch* among graph traversal algebra and relational algebra bring several challenges while processing graphs on top of relational systems [Has+18; Tan+15]. Indeed, the design space for processing graphs in the relational realm is different, and several performance factors need to be considered (i.e., *the core of this thesis*). Moreover, graph workloads, by their nature, are iterative and depend extensively on efficient traversal of adjacency structures that are not typically implemented in relational interfaces including the SQL query language. Relational systems are not optimized for fast graph traversals [Par15]. Implementing such adjacency structures in relational systems is also challenging due to the complexity of the *under-hood* relational system's implementation internals.

Moreover, this thesis discussed that processing and querying large RDF graphs over the non-graph (relational) BD systems require several additional design decisions [RTS19; RAT21]. Relational RDF systems that target large RDF querying mainly focus on the dimensions that directly affect the performance, (1) **Relational schema**: indeed the relational layout directly affect the performance of systems as it affects the number of joins, filters of the query workloads.

The most common schemas for representing RDF graphs in the relational world are the ST, VP, WPT, PT, and ExtVP. For example, systems like RDFMATCH [Cho+05], Apache Jena 1 [McB01], and RDF-3X [NW08] utilize the Single Statement (ST) schema. Whereas, the binary Vertically-Partitioned (VP) tables schema is used to represent RDF graphs in several other relational systems like PigSPARQL and SW-Store [Aba+09]. Moreover, the extension of this schema (ExtVP) is used in systems like S2RDF [Sch+16], and WORQ [MAA18] providing some join reductions using the semi-join reductions, and Bloom filters, respectively. Sempala used the unified WPT schema for representing the RDF graphs over HDFS. Whereas, the Property Tables (PT) schema is used in RDF relational systems such Db2RDF [Bor+13] and Apache Jena2 [Wil+03]. Last but not least a hybrid schema of PT and VP is used by the Prost system over HDFS ans using Spark-SQL for querying them. Table 36 shows the relational RDF schemas used to represent RDF graphs in relational RDF query systems and databases.

The Schema dimension is followed by data (2) **Partitioning** as the second considered dimension of relational systems for querying large RDF graphs. The effect of partitioning over relational systems for querying large RDF datasets was not neglected in the state-of-the-art. for instance, *Akhter et.al.* [ANS18] evaluated the performance of seven RDF partitioning techniques (i.e., Horizontal, Subject-Based, Predicate-Based, Hierarchical, Recursive-Bisection, TCV-Min, and Min-Edgecut Partitioning)over federated and non-federated systems. Anthony et.al. [al19] showed the effect of the *subject-based* partitioning technique for partitioning different RDF relational schemata over the Spark-SQL framework. Last but not least, *Mohamed Ragab et.al.* [Rag+20; Rag20; RAT21], showed the effect of three partitioning techniques (i.e., Horizontal, Subject-based, Predicate-based) on the performance of relational schema advancements for querying large RDF datasets over Spark-SQL engine.

Last but not least, we cannot ignore the effect of the (3) **Storage** dimension. For instance, *Ivanov et.al.* [IP19] evaluated the impact of two different columnar file formats (i.e., ORC and Parquet) over *SQL-on-Hadoop* engines (e.g., Spark-SQL, Hive). On the

RDF Relational Querying System	ST	VP	PT	ExtVP	WPT
RDF-3X [NW08]	✓				
S2RDF [Sch+16]				✓	
Sempala [Aba+09]					✓
DB2RDF [Bor+13]			✓		
WORQ [MAA18]		✓			
PigSPARQL [SPL11]		✓			
SPARQL-GX [Gra+16]		✓			
SW-Store [Aba+09]		✓			
Prost [CFL18]		✓	✓		
Hassan et.al. [HB19]		✓	✓		
Schievelbein et.al. [Sch+19]					✓
RaxonDB [PMV22]			✓		
Apache Jena 1 [McB01]	✓				
Apache Jena 2 [Wil+03]			✓		
Pham et.al. [Pha+15]			✓		

Table 36: RDF relational schemas used/proposed in the state-of-the-art of RDF querying relational systems.

same note, *Ragab et.al.* [RTS19] evaluated the performance of Spark-SQL for querying RDF datasets stored over HDFS in both row-oriented file formats (i.e., Avro, CSV), and Columnar file formats (i.e., ORC, Parquet). Moreover, *Mohamed Ragab et.al.* [Rag20] showed the impact of those different row-oriented and column-oriented storage formats on the performance of two RDF relational schema advancements (i.e., WPT and ExtVP) for querying large RDF datasets on top of the Spark-SQL engine. The experiments showed that storage is indeed an impactful dimension to consider for querying RDF graphs over relational systems, and it impacts the replicability of those schema advancements.

8.4. RDF Graph Processing Benchmarking Efforts

Benchmarking is a crucial task that checks the performance capabilities of systems under specific tasks evaluating their potentials and bottlenecks [Bon+18; Szá19]. In fact, benchmarking efforts can push technological progress turning research into a competition with clearly-defined goals. This stimulates the research community to produce more and better results where the end users can understand and decide on basis of those defined objectives. For example, the KGs community is synergizing efforts pushing technological progress through benchmarking initiatives like the *Linked Data Benchmark Council* (LDBC[13]). Research-wise, the community proposed datasets like *LOD-a-lot*[Fer+17], test-beds like *LOD-Laundromat* [Bee+16], and *full-fledge* benchmarks like *SP*^{2}*Bench* [eta09], *LUBM* [14], *WatDiv*[15] and many more. Recently, the community started studying benchmarking methodologies [Sal+19] and metrics [AVS17], and it is questioning whether SPARQL benchmarks are or not representative of the actual workload [Dua+11].

[13] ldbcouncil.org

[14] `http://swat.cse.lehigh.edu/projects/lubm/`

[15] `https://dsg.uwaterloo.ca/watdiv/`

Reference	Dimension		
	Partitioning	Schema	Storage
Abdelaziz et.al [Abd+17]	✗	✗	✗
Sakr et.al. [SA10]	✗	✗	✗
Mauroux et.al. [Cud+13]	✗	✗	✗
Sempala [Sch+14]	✗	**! (WPT)**	✗
S2RDF [Sch+16]	!(HDFS)	✓	! (Parquet)
Ayla et.al. [al19]	!(Subj.)	✓	! (Parquet)
Ragab et.al. [RTS19]	✗	✓	✓
Ivanov et.al. [IP19]	✗	✗	! (Parquet and ORC)
Akther et.al. [ANS18]	✓	✗	✗
Bench-Ranking [RAT21]	✓	✓	✓

Table 37: Neglecting performance trade-offs in the state-of-the-art when covering the experimental dimensions, i.e., Relational Schema, Partitioning, and Storage Formats. ✓ Full presence, ✗ Full Absence, **!** exists with some limitations.

Reference	Analysis class			
	Desc.	Diag.	Pres.	Multi-Dim.
Abdelaziz et.al [Abd+17]	✓	✓	✗	✗
Sakr et.al. [SA10]	✓	✗	✗	✗
Mauroux et.al. [Cud+13]	✓	✓	✗	✗
Sempala [Sch+14]	✓	✓	✗	✗
S2RDF [Sch+16]	✓	✓	✗	✗
Ayla et.al. [al19]	✓	✓	✗	✗
Ragab et.al. [RTS19]	✓	✓	✗	✗
Ivanov et.al. [IP19]	✓	✓	✗	✗
Akther et.al. [ANS18]	✓	✓	**!(Partition)**	✗
Bench-Ranking [RAT21]	✓	✓	✓	✓

Table 38: State-of-the-art Lack of Prescriptive Performance Analysis for Processing Large KGs on top of BD Relational Systems. ✓Analysis Provided) , ✗Analysis Missing, **!** Analysis exists with some limitations.

8.4.1. Big RDF Benchmarking Challenges

In the introduction (Chapter 1), we briefly mentioned the challenges of benchmarking efforts of BD relational systems for querying large knowledge graphs (see Chapter 1 Table 1). In this section, we aim at expanding the details of the research gap that this thesis covers, reflecting on those challenges.

Neglecting Performance Trade-offs & Lack of Replicability: First, the state-of-the-art research focused on optimizing the RDF systems' performance rather than systematically benchmarking the underlying experimental dimensions that affect their performance. For instance, *Abdelaziz et.al.* [Abd+17], *Sakr et.al.* [SA10], and *Mauroux et.al.* [Cud+13] conducted comprehensive surveys of benchmarking several distributed RDF processing systems, comparing their performance without investigating the underlying impactful di-

mensions (e.g., schema, partitioning, or storage). Moreover, even the research efforts that investigate those dimensions focused only on *one dimension* at a time (even with limited coverage of alternatives), e.g., relational schema [RTS19; al19], partitioning [ANS18], or storage [IP19]. For instance, Sempala system [Sch+14] proposed the WPT schema advancement for enhancing the performance of *Apache Impala* relational system when querying massive RDF datasets, while neglecting other dimensions such as partitioning, storage options. Similarly, *DB2RDF* [Bor+13], a system proposed by *IBM* research, also proposed an efficient clustering algorithm for designing a relational Property Tables (PT) schema for representing KGs. However, this work also neglects the other dimensions (e.g., storage and partitioning). The S2RDF system [Sch+16] for querying large RDF datasets over Spark-SQL also suggested a novel schema advancement (i.e., a extension of VP schema) called *"ExTVP"* for optimizing the performance of Spark without considering the other dimensions that directly impact the performance. Table 37 shows the dimensions covered in the literature. It shows the gap of covering a benchmarking study of the three dimensions altogether.

As we have shown through this thesis, focusing solely on one experimental dimension neglects the presence of other dimensions trade-offs. Thus, introducing one new experimental dimension, or even changing one experimental option can affect the BD systems' performance to generalize. Chapter 4 discussed extensive experiments that shows the significant impact of changing experimental options on the BD systems replicability for querying large (RDF) graphs.

Lack of Systematic Prescriptive Performance Analysis: Second, there is a gap in the performance analysis maturity in existing works that use BD frameworks for querying large RDF graphs. The studies show that the *descriptive* and *diagnostic* analyses are the most common in the literature [Abd+17; Cud+13; Sch+16; Sch+14]. Prior works stop their analyses at describing the performance of engines around a specific experimental dimension. For instance, *Shätzle et.al.* [Sch+14; Sch+15] described the performance of big relational systems (i.e., comparing *loading times* and *query execution time*). In particular, they show *why* systems e.g., *S2RDF* [Sch+16] and *Sempala* [Sch+14] outperform other RDF systems in those metrics due to their proposed schema optimizations, i.e., the Wide Property Table (WPT) schema [Sch+14], and the Extended Vertically-Partitioned Tables (ExtVP) schema [Sch+16]. Moreover, *Abdelaziz et.al.* [Abd+17] conducted a comprehensive descriptive and diagnostic survey of several RDF processing systems, describing their performance in terms of *scalability*, *query efficiency*, and *workload adaptability* metrics. The authors followed their analysis by diagnosing why a distributed RDF system *X* outperforms another system *Y* without giving prescriptions over the underlying experimental dimensions. Similarly, authors in [Cud+13] performed an empirical evaluation of four *NoSQL* processing RDF systems describing and comparing their *query execution times*, and *loading times*. However, none of these works provide prescriptions over the level of the mentioned experimental dimensions altogether without neglecting their trade-offs.

In this thesis, we argue that the *Prescriptive* analysis reduces the need for human intervention by making the descriptive insight actionable. The prescriptive analysis relies on *statistical* and *mathematical* models that aid in answering the question of *'what should be done?'*. Regarding benchmarking, the prescriptive analysis provides the criteria for selecting the best possible given approaches. A promising example of prescriptive performance analysis of RDF data systems was introduced by Akhter et.al. [ANS18]. The authors adopted a ranking measure to decide which RDF partitioning techniques have the

best performance over various RDF datasets. Nonetheless, prescriptive analysis that have been provided in that work is limited to one experimental dimension (i.e., partitioning). Table 38 shows the shows the gap of providing prescriptive performance analysis for large KGs processing in top of BD relational systems.

How Bench-Ranking Framework handles those limitations: To the best of our knowledge, the work done in this thesis [RAT21; Rag+20] is the only work that covers benchmarking the combination of three impactful dimensions (see last row in Table 37), i.e., relational schema, storage, and partitioning dimensions into unified experiments while assessing the performance of relational systems (e.g., Apache Spark-SQL) for querying large RDF datasets. Moreover, the Bench-Ranking framework; with means of ranking techniques; provides prescriptive performance analysis for large KGs practitioners to guide them to select the best performing experimental options over the complex solution space of multiple dimensions. The framework also models the *trade-offs* of the experimental dimensions as a Multi-Dimension (MD) optimization problem aiming to find those optimal experimental options across multiple dimensions (e.g., schema, partitioning, and storage), see last row in Table 38.

9. APPENDIX B: BENCHMARKS RELATIONAL SCHEMAS & QUERY WORKLOAD

9.1. Benchmarks Relational Schemas

In this section, we describe the relational schema of the the two used RDF benchmarks of this thesis (i.e., SP2B and WatDiv).

More Specifically, we show here the Property Tables (PT) schema. The reason of showing only the PT schema is that the other RDF graphs relational schemas in this thesis are deterministic in their design (e.g., *ST*, *WPT*, *VT* and *ExtVT*). In other words, those schemas are agnostic from the underlying graph structure. For instance, The ST is a ternary single relation (table) of the three components of RDF statements (i.e., Subject, Predicate, and Object). The WPT encodes the entire dataset into a single denormalized relation of all predicates in the graph. the *VT* and *ExtVT* are graph-based schemas, i.e., these schemas require one binary relation (Subject, Object) for each unique predicate (labeled edge) in the graph structure.

In contrast, the PT schema requires designing efforts from the (Big) Data engineer. Therefore, as a reference we provide our designed PT schema for the two RDF benchmarks as depicted in the simplified *ERD* diagram of Figures 52, and 53 for WatDiv and SP2B RDF benchmarks, respectively.

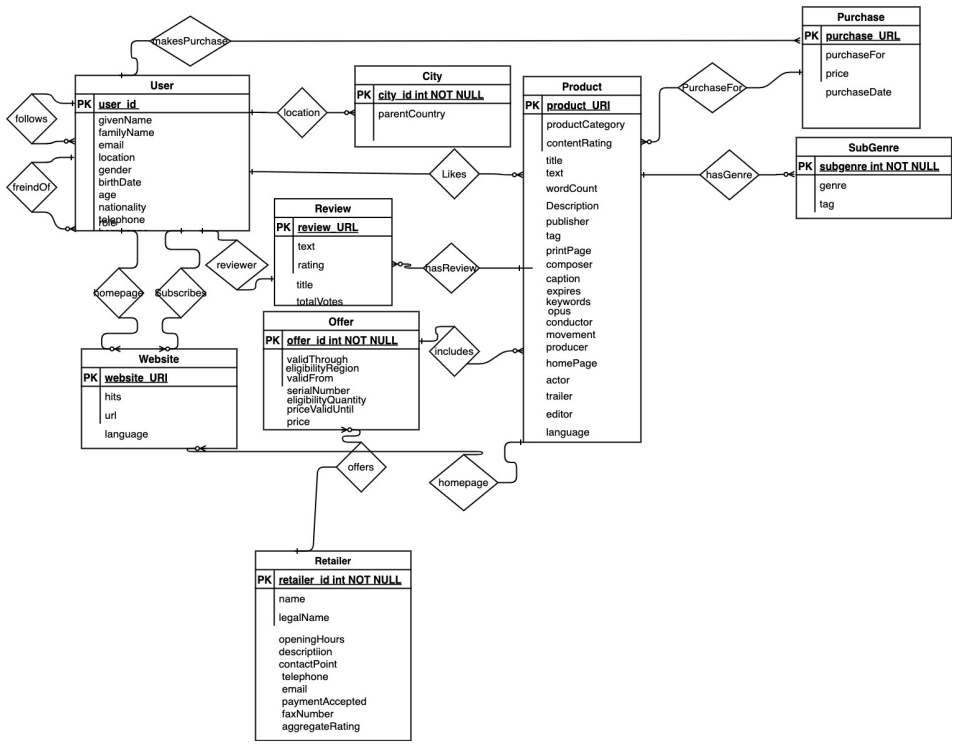


Figure 52: WatDiv Property Tables (PT) relational Schema.

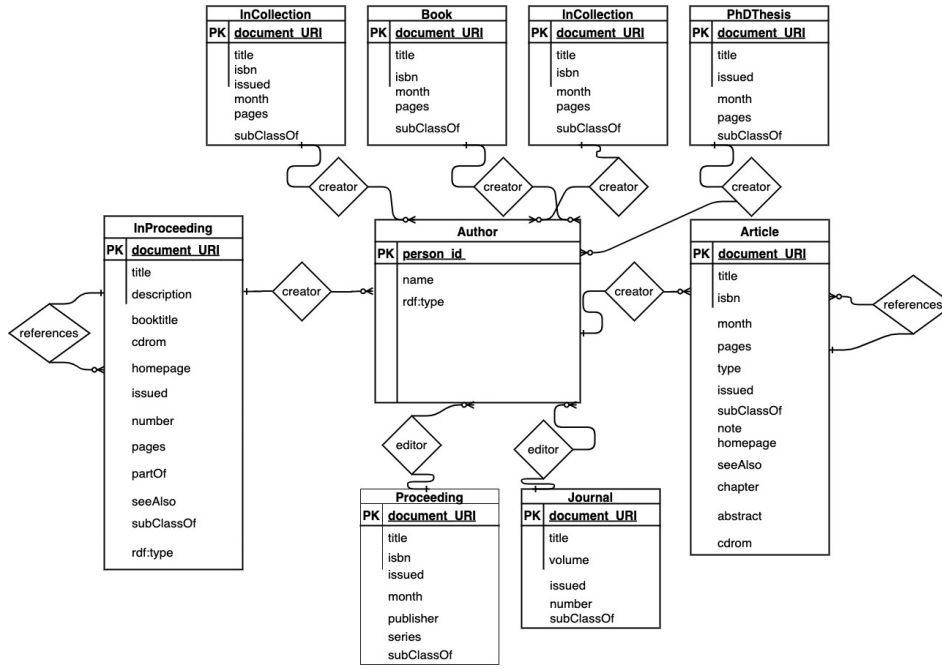


Figure 53: SP2B Property Tables (PT) relational schema.

9.2. Query Workload with Relational Schema Translations

In this section, we provide the set queries in the two RDF Benchmarks (i.e., SP2B and WatDiv) in native SPARQL, as well as when translated into the different relational schemas (i.e., ST, (Ext)VT, PT, and WPT).

1. WatDiv Query workload alongside SQL translations: The WatDiv SPARQL query workload [1] consists of 20 templates that can generate arbitrary any number of queries replacing the placeholders with parameter values. We provide SQL translations for each of the above mentioned schemas in our GitHub repository [2].

2. SPARQL Query Workload alongside SQL Translations: The SP2B SPARQL query workload [Sch+08] [3] consists of 11 SPARQL queries. We provide the SQL translations of those queries for each of the above mentioned schemas in our GitHub repository [4].

[1] WatDiv SPARQL query templates: `https://dsg.uwaterloo.ca/watdiv/basic-testing.shtml`

[2] WatDiv SQL translations: `https://github.com/DataSystemsGroupUT/SPARKSQLRDFBenchmarking/tree/master/ProjectSourceCode/src/main/scala/ee/ut/cs/bigdata/watdiv/querying/queries`

[3] SP2B SPARQL queries: `https://dbis.informatik.uni-freiburg.de/index.php?project=SP2B/queries.php`

[4] SP2B SQL translations: `https://github.com/DataSystemsGroupUT/SPARKSQLRDFBenchmarking/tree/master/ProjectSourceCode/src/main/scala/ee/ut/cs/bigdata/sp2b/querying/queries`

ACKNOWLEDGEMENTS

First and foremost, I thank Almighty God for giving me the opportunity and ability to pursue this research. In addition, may peace and salutation be given to our beloved and our ultimate role model; *prophet Muhammad* (PBUH) who has taken all human beings from the darkness to the lightness.

I dedicate this thesis to the soul of my previous supervisor Prof. *Sherif Sakr* who unfortunately passed away during the pandemics (2020). Sherif was a respected professor and *first-class* researcher at Tartu Universitys Institute of Computer Science in the field of Big Data technologies. I would say that Sherif greatly helped me to develop and grow on several levels, academic and personal.

I also express my thanks to my current supervisor *Prof. Ahmed Awad* for his guidance, support and feedback. I also thank him for his time and discussions related to my PhD topic, and even beyond it.

Special thanks and gratitude go to my supervisor *Ass. Prof. Riccardo Tommasini* who always had time for me, devoting much time to ensuring that I am aiming high and pushing me to maximize my potential during my Ph.D. research journey.

I am grateful to the reviewers of this work (*Prof. Bellatreche*, and *Ass. Prof. Zumpano*) for their valuable comments and rich feedback that have significantly improved my thesis.

I would not forget to thank my friends and colleagues at the university of Tartu, for their support. I want to thank my colleague *Kristo Raun* for helping me with the Estonian translations in this thesis.

Words cannot express my deepest gratitude and appreciation to my family for their unconditional love and emotional support. I say special thanks to my parents (my father "Ragab" and my mother "Sooad"), and my beloved sisters (*"Hala"*, *"Ghada"*, and *"Shaimaa"*). I thank my wife *"Shaymaa Heidr"* for her constant support and care, and my fabulous kid *"Anas"*, for his hugs and laughs that brighten my days. They were always around and accompanied me in moments of hardships and frustration.

Thank you all.

SISUKOKKUVÕTE

Bench-Ranking: ettekirjutav analüüsimeetod suurte teadmiste graafide päringutele

Me elame maailmas, kus ei suurene mitte ainult andmemahud, vaid ka andmete vahelised seosed. Graafid on kõige intuitiivsemad, loomulikumad ja paindlikumad andmeabstraktsioonid mis käsitsevad antud seoseid mitmetes rakendustes meie igapäevaelus.

Antud rakenduste graafiandmete enneolematud mahud nõuavad suure hulga graafide tõhusaks töötlemiseks skaleeritavaid süsteeme. Graafi töötlemine on laialt levinud nii akadeemilistes ringkondades kui ka tööstuses. See on tinginud graafide töötlemise, hoiustamise ja haldamise süsteemide arvu tõusu. Loomulikult skaleeruva graafide pärimise süsteemi ehitamine on veel lahtine probleem. Seega tuleb suurte graafipäringute analüüsi jaoks endiselt taaskasutada olemasolevaid suurandmete (BD) süsteeme ja nende relatsioonilisi liideseid.

Vaatamata oma paindlikkusele nõuab relatsioonimudel graafide esitamisel mitmeid täiendavaid disainiotsuseid, mida ei saa automaatselt otsustada: skeemi valik, jaotustehnika ja salvestusvormingud. Lisaks võivad need disainiotsused üksteist mõjutada, kuna nõuavad omavahelisi kompromisse. Seega on BD süsteemi jõudlus sõltuvuses antud otsustest, mis tähendab et selget võitjat on raske kindlaks teha. Sellest tulenevalt on suurte graafide pärimisel raske tagada relatsiooniliste BD süsteemide jõudluse õiglast hindamist.

Tipptasemel uurimistööd, mille keskmes on suurte graafide pärimise tulemuslikkuse analüüs, on vaid kirjeldavad nad vastavad küsimusele "mis juhtus?"või paremal juhul diagnostilised miks see juhtus?. Meie väidame, et sedalaadi tulemusanalüüsid on sageli üle jõu käivad ja otsuste tegemiseks jääb suur hulk tööd. Tõepoolest, mainitud eksperimentaalsete konfiguratsioonide sisemised kompromissid põhjustavad vastuolusid ja seega ei saa sellise ebaküpse analüüsi taseme järel otsustada lõplike parimate konfiguratsioonide üle. Vastupidiselt, ettekirjutav jõudlusanalüüs (PPA) vähendab inimsekkumise vajadust veelgi, muutes teadmise teostatavaks, vastates küsimusele "mida me peaksime tegema?".

See lõputöö uurib ettekirjutava analüütika võimaldamise probleemi BD süsteemide kontekstis, mis pärivad suuri (RDF) graafe. PPA eesmärk on suunata praktik otse rakendatavate otsuste juurde, navigeerides keerulistes eksperimentaalsetes lahendusruumides, jätmata tähelepanuta aluseks olevate mõõtmete vajalikke kompromisse.

Käesolevas töös püüame esialgu mõista BD süsteemide jõudlust suurte (RDF) graafipäringute tegemise ajal. Näitame kirjeldava ja diagnostilise jõudlusanalüüsi (DDPA) piiranguid ja demonstreerime kuidas need on selgete otsuste tegemisel ebamugavad. Seda veelgi enam kui on tarvis tagada (RDF) graafe pärivate BD süsteemide taasesitus. Näiteks Spark-SQL-i süsteemi jõudlust RDF-i relatsiooniskeemi täiustustega ei ole võimalik üldistada, kuna see lisab uusi eksperimentaalseid mõõtmeid, näiteks andmete salvestuse ja jaotuse osas.

Seega on vaja koostada ettekirjutav raamistik analüüsimaks BD süsteemide jõudlust suurte (RDF) graafipäringute teostamiseks. Lõputöös uuritakse, kuidas võimaldada ettekirjutavat analüütikat järjestuskriteeriumide abil. Töötasime välja PPA raamistiku Bench-Ranking, mis kasutab mitut ühemõõtmelist (SD) ja mitmemõõtmelist (MD) järjestamise kriteeriumi süsteemi jõudluse järjestamiseks mitme eksperimentaalse mõõtmega. Varasemate SD-järjestusmeetodite eesmärk on järjestada eksperimentaalsed mõõtmed, arvutades eksperimentaalsete mõõtmete valikute jaoks paremusjärjestuse skoorid. Hilisem MD-tehnika laiendab Bench-Rankingi mitme eesmärgi optimeerimise probleemiks, et opti-

meerida kõiki mõõtmeid korraga. Eksperimentaalsete mõõtmete täielikuks arvestamiseks kasutame standardset Pareto piiri tehnikat. Bench-Ranking pakub täpset kuid lihtsat viisi, mis toetab praktikuid nende hindamisülesannete täitmisel isegi mõõtmete kompromissi-de olemasolul. Viimaks pakub lõputöö hindamismõõdikud välja pakutud järjestuskriteeriumide tõhususe hindamiseks. Lisaks pakume välja pakutud järjestuskriteeriumide tõhususe hindamise mõõdikud. Esimene mõõdik on Vastavus, mis mõõdab kõrgeima asetusega konfiguratsioonide vastavust tegelikele päringujärjestustele (st nende konfiguratsioonide paigutusele). Teine mõõdik on "sidusus", mis mõõdab kahe järjestuskomplekti kokkuleppe taset, mis kasutavad erinevates katsetes sama järjestuskriteeriumi (nt erinevad andmestiku skaalad).

Selle lõputöö viimane, kuid mitte vähem oluline panus on Bench-Ranking PPA metoodika pakkimine korduvkasutatavasse ja laiendatavasse Pythoni teeki (PAPyA).

PAPyA eesmärk on varjata Bench-Ranking funktsionaalsuse ja mõõdikute keerukust ning vähendab aega ja vaeva teadlike otsuste tegemiseks BD-süsteemide stsenaariumide valdkondades kus on vaja teostada mitu disainiotsust, nt suurte graafide pärimine. Lisaks võimaldab see ette valmistada andmeid töötlemiskonveieri jaoks relatsioonilise suurandmete raamistiku kaudu; mis võimaldab konfiguratsioonide automaatset järjestamist.

Lõputöö võtab tehtu kokku, arutades teadmust ja tulevikusuundi mis on seotud suurte graafide pärimise ja töötlemisega BD relatsiooniliste raamistike peal.

CURRICULUM VITAE

Personal data

Name:	Mohamed Ragab Moawad Mohamed
Date of Birth:	25.08.1992
Nationality:	Egyptian
Language Skills:	Arabic(Native), English
Email:	mohamed.ragab@ut.ee

Education

2018–2022	University of Tartu, Estonia, Ph.D. in Computer Science
2013–2018	Cairo University, Egypt, Masters (M.Sc) in Information Systems
2009–2013	Fayoum University, Egypt, Bachelors (B.SC) in Information Systems.

Employment

2018–2022	Junior Research Fellow, University of Tartu, Estonia.
2014–2018	Teaching Assistant, Nahda University, BeniSuef, Egypt.
2013–2014	Web Developer, ITIS Company for Development.

Scientific work

Main fields of interest

- Big Data Management
- Semantic Web
- Large Graph Data Processing

ELULOOKIRJELDUS

Isiklikud andmed

Nimi:	Mohamed Ragab Moawad Mohamed
Sünniaeg:	25.08.1992
Kodakondsus:	Egiptuse
Keeleoskus:	Araabia (emakeel), inglise keel
E-post:	mohamed.ragab@ut.ee

Haridus

2018-2022	Tartu Ülikool, Eesti, Ph.D. arvutiteaduses
2013-2018	Kairo Ülikool, Egiptus, magistrikraad (M.Sc) teabe alal Süsteemid
2009-2013	Fayoumi ülikool, Egiptus, informaatika bakalaureuseõpe (B.SC) süsteemid.

Teenistuskäik

2018-2022	nooremteadur, Tartu Ülikool, Eesti.
2014-2018	õppeassistent, Nahda ülikool, BeniSuef, Egiptus.
2013-2014	veebiarendaja, ITIS arendusettevõte.

Teadustegevus

Peamised uurimisvaldkonnad:

- Suurandmete haldus
- Semantiline veeb
- Suure graafiku andmetöötlus

LIST OF ORIGINAL PUBLICATIONS

Publications included in the thesis

I **Mohamed Ragab**. "Towards Prescriptive Analyses of Querying Large Knowledge Graphs". *The 26th European Conference on Advances in Databases and Information Systems (ADBIS). Springer, Cham, September 2022. P.639–647* (**Best Doctoral Symposium Award**). *Lead author. The author performed the implementation and the analysis of the experiments and contributed substantially to the ideas and the writing.*

II **Mohamed Ragab**, Adam Satria, Riccardo Tommasini. "`PAPyA`: a Library for Performance Analysis of SQL-based RDF Processing Systems". *Semantic Web Journal (SWJ), 2022, DOI: 10.48550/ARXIV.2209.06877.URL: https://arxiv.org/abs/2209.06877.* (**Under Review**). *Lead author. The author performed the implementation and the analysis of the experiments and contributed substantially to the ideas and the writing.*

III **Mohamed Ragab**, Feras M. Awaysheh, and Riccardo Tommasini. "Bench-Ranking: A First Step Towards Prescriptive Performance Analyses For Big Data Frameworks". *2021 IEEE International Conference on Big Data (Big Data). IEEE Computer Society, December 2021, P.241–251. Lead author. The author performed the implementation and the analysis of the experiments and contributed substantially to the ideas and the writing.*

IV Sherif Sakr, Angela Bonifati, Hannes Voigt, Alexandru Iosup, Khaled Ammar, Renzo Angles, Walid Aref, Marcelo Arenas, Maciej Besta, Peter A. Boncz, Khuzaima Daudjee, Emanuele Della Valle, Stefania Dumbrava, Olaf Hartig, Bernhard Haslhofer, Tim Hegeman, Jan Hidders, Katja Hose, Adriana Iamnitchi, Vasiliki Kalavri, Hugo Kapp, Wim Martens, M. Tamer Özsu, Eric Peukert, Stefan Plantikow, **Mohamed Ragab**, Matei R. Ripeanu, Semih Salihoglu, Christian Schulz, Petra Selmer, Juan F. Sequeda, Joshua Shinavier, Gábor Szárnyas, Riccardo Tommasini, Antonino Tumeo, Alexandru Uta, Ana Lucia Varbanescu, Hsiang-Yun Wu, Nikolay Yakovets, Da Yan, Eiko Yoneki. "The future is big graphs: a community view on graph processing systems". *Communications of the ACM (CACM), September 2021, P.62–71. Co-Author: Contributed in writing the initial draft of the paper in two sections, i.e., System Requirements (Scale-up vs. scale-out) for large graph processing. Relational meets Graphs.*

V **Mohamed Ragab**, Riccardo Tommasini, Feras M. Awaysheh, Ramos JC. "An In-depth Investigation of Large-scale RDF Relational Schema Optimizations Using Spark-SQL". *Proceedings of the 23rd International Workshop on Design, Optimization, Languages and Analytical Processing of Big Data (DOLAP) co-located with the 24th International Conference on Extending Database Technology and the 24th International Conference on Database Theory (EDBT/ICDT), CEUR-WS.org, March 2021, P.71–80. Lead author. The author performed the implementation and the analysis of the experiments and contributed substantially to the ideas and the writing.*

VI **Mohamed Ragab**, Riccardo Tommasini, Sherif Sakr. "Comparing Schema Advancements for Distributed RDF Querying Using SparkSQL". *Proceedings of the ISWC 2020 Demos and Industry Tracks, CEUR-WS.org, November 2020, P.30–34. Lead author. The author performed the implementation and the analysis of the experiments and contributed substantially to the ideas and the writing.*

VII **Mohamed Ragab**, Riccardo Tommasini, Sadiq Eyvazov, Sherif Sakr. "Towards making sense of Spark-SQL performance for processing vast distributed RDF datasets". *Semantic Big Data (SBD) co-located with SIGMOD PODS, ACM, June 2020, P.1–6. Lead author. The author performed the implementation and the analysis of the experiments and contributed substantially to the ideas and the writing.*

VIII **Mohamed Ragab**. "Large Scale Querying and Processing for Property Graphs". *Proceedings of the 22nd International Workshop on Design, Optimization, Languages and Analytical Processing of Big Data co-located with EDBT/ICDT joint conference, CEUR-WS.org, March 2020, P.79–83. Lead author. The author performed the implementation and the analysis of the experiments and contributed substantially to the ideas and the writing.*

IX **Mohamed Ragab**, Riccardo Tommasini, Sherif Sakr "Benchmarking Spark-SQL under Alliterative RDF Relational Storage Backends". *Proceedings of the QuWeDa 2019 co-located with 18th International Semantic Web Conference (ISWC), CEUR-WS.org, October 2019, P.67–82. Lead author. The author performed the implementation and the analysis of the experiments and contributed substantially to the ideas and the writing.*

Publications not included in the thesis

I Riccardo Tommasini, **Mohamed Ragab**, Alessandro Falcetta, Emanuele Della Valle, Sherif Sakr. "A First Step Towards a Streaming Linked Data Life-Cycle". *The 19th International Semantic Web Conference(ISWC), Springer, November 2020, P.634–650). Co-Author: Contributed with the implementations and the analysis of the experiments, with reviewing the final manuscript before submission.*

II **Mohamed Ragab**, Mohamed Maher, Ahmed Awad, Sherif Sakr "MINARET: A Recommendation Framework for Scientific Reviewers". *The 22nd International Conference on Extending Database Technology, EDBT, OpenProceedings.org, March 2019, P.538–541. Lead author. The author performed the implementation and the analysis of the experiments and contributed substantially to the ideas and the writing.*

III Riccardo Tommasini, **Mohamed Ragab**, Alessandro Falcetta, Emanuele Della Valle, Sherif Sakr. "Bootstrapping the Publication of Linked Data Streams". *Proceedings of the ISWC 2019 Satellite Tracks (Posters & Demonstrations, Industry, and Outrageous Ideas) co-located with 18th International Semantic Web Conference (ISWC 2019), CEUR-WS.org, October 2020, P.29–32. Co-Author: Contributed with the implementations and the analysis of the experiments, with reviewing the final manuscript.*

DISSERTATIONES INFORMATICAE PREVIOUSLY PUBLISHED IN DISSERTATIONES MATHEMATICAE UNIVERSITATIS TARTUENSIS

19. **Helger Lipmaa.** Secure and efficient time-stamping systems. Tartu, 1999, 56 p.
22. **Kaili Müürisep.** Eesti keele arvutigrammatika: süntaks. Tartu, 2000, 107 lk.
23. **Varmo Vene.** Categorical programming with inductive and coinductive types. Tartu, 2000, 116 p.
24. **Olga Sokratova.** Ω-rings, their flat and projective acts with some applications. Tartu, 2000, 120 p.
27. **Tiina Puolakainen.** Eesti keele arvutigrammatika: morfoloogiline ühestamine. Tartu, 2001, 138 lk.
29. **Jan Villemson.** Size-efficient interval time stamps. Tartu, 2002, 82 p.
45. **Kristo Heero.** Path planning and learning strategies for mobile robots in dynamic partially unknown environments. Tartu 2006, 123 p.
49. **Härmel Nestra.** Iteratively defined transfinite trace semantics and program slicing with respect to them. Tartu 2006, 116 p.
53. **Marina Issakova.** Solving of linear equations, linear inequalities and systems of linear equations in interactive learning environment. Tartu 2007, 170 p.
55. **Kaarel Kaljurand.** Attempto controlled English as a Semantic Web language. Tartu 2007, 162 p.
56. **Mart Anton.** Mechanical modeling of IPMC actuators at large deformations. Tartu 2008, 123 p.
59. **Reimo Palm.** Numerical Comparison of Regularization Algorithms for Solving Ill-Posed Problems. Tartu 2010, 105 p.
61. **Jüri Reimand.** Functional analysis of gene lists, networks and regulatory systems. Tartu 2010, 153 p.
62. **Ahti Peder.** Superpositional Graphs and Finding the Description of Structure by Counting Method. Tartu 2010, 87 p.
64. **Vesal Vojdani.** Static Data Race Analysis of Heap-Manipulating C Programs. Tartu 2010, 137 p.
66. **Mark Fišel.** Optimizing Statistical Machine Translation via Input Modification. Tartu 2011, 104 p.
67. **Margus Niitsoo**. Black-box Oracle Separation Techniques with Applications in Time-stamping. Tartu 2011, 174 p.
71. **Siim Karus.** Maintainability of XML Transformations. Tartu 2011, 142 p.
72. **Margus Treumuth.** A Framework for Asynchronous Dialogue Systems: Concepts, Issues and Design Aspects. Tartu 2011, 95 p.
73. **Dmitri Lepp.** Solving simplification problems in the domain of exponents, monomials and polynomials in interactive learning environment T-algebra. Tartu 2011, 202 p.

74. **Meelis Kull.** Statistical enrichment analysis in algorithms for studying gene regulation. Tartu 2011, 151 p.
77. **Bingsheng Zhang.** Efficient cryptographic protocols for secure and private remote databases. Tartu 2011, 206 p.
78. **Reina Uba.** Merging business process models. Tartu 2011, 166 p.
79. **Uuno Puus.** Structural performance as a success factor in software development projects – Estonian experience. Tartu 2012, 106 p.
81. **Georg Singer.** Web search engines and complex information needs. Tartu 2012, 218 p.
83. **Dan Bogdanov.** Sharemind: programmable secure computations with practical applications. Tartu 2013, 191 p.
84. **Jevgeni Kabanov.** Towards a more productive Java EE ecosystem. Tartu 2013, 151 p.
87. **Margus Freudenthal.** Simpl: A toolkit for Domain-Specific Language development in enterprise information systems. Tartu, 2013, 151 p.
90. **Raivo Kolde.** Methods for re-using public gene expression data. Tartu, 2014, 121 p.
91. **Vladimir Šor.** Statistical Approach for Memory Leak Detection in Java Applications. Tartu, 2014, 155 p.
92. **Naved Ahmed.** Deriving Security Requirements from Business Process Models. Tartu, 2014, 171 p.
94. **Liina Kamm.** Privacy-preserving statistical analysis using secure multi-party computation. Tartu, 2015, 201 p.
100. **Abel Armas Cervantes.** Diagnosing Behavioral Differences between Business Process Models. Tartu, 2015, 193 p.
101. **Fredrik Milani.** On Sub-Processes, Process Variation and their Interplay: An Integrated Divide-and-Conquer Method for Modeling Business Processes with Variation. Tartu, 2015, 164 p.
102. **Huber Raul Flores Macario.** Service-Oriented and Evidence-aware Mobile Cloud Computing. Tartu, 2015, 163 p.
103. **Tauno Metsalu.** Statistical analysis of multivariate data in bioinformatics. Tartu, 2016, 197 p.
104. **Riivo Talviste.** Applying Secure Multi-party Computation in Practice. Tartu, 2016, 144 p.
108. **Siim Orasmaa.** Explorations of the Problem of Broad-coverage and General Domain Event Analysis: The Estonian Experience. Tartu, 2016, 186 p.
109. **Prastudy Mungkas Fauzi.** Efficient Non-interactive Zero-knowledge Protocols in the CRS Model. Tartu, 2017, 193 p.
110. **Pelle Jakovits.** Adapting Scientific Computing Algorithms to Distributed Computing Frameworks. Tartu, 2017, 168 p.
111. **Anna Leontjeva.** Using Generative Models to Combine Static and Sequential Features for Classification. Tartu, 2017, 167 p.
112. **Mozhgan Pourmoradnasseri.** Some Problems Related to Extensions of Polytopes. Tartu, 2017, 168 p.

113. **Jaak Randmets.** Programming Languages for Secure Multi-party Computation Application Development. Tartu, 2017, 172 p.
114. **Alisa Pankova.** Efficient Multiparty Computation Secure against Covert and Active Adversaries. Tartu, 2017, 316 p.
116. **Toomas Saarsen.** On the Structure and Use of Process Models and Their Interplay. Tartu, 2017, 123 p.
121. **Kristjan Korjus.** Analyzing EEG Data and Improving Data Partitioning for Machine Learning Algorithms. Tartu, 2017, 106 p.
122. **Eno Tõnisson.** Differences between Expected Answers and the Answers Offered by Computer Algebra Systems to School Mathematics Equations. Tartu, 2017, 195 p.

DISSERTATIONES INFORMATICAE UNIVERSITATIS TARTUENSIS

1. **Abdullah Makkeh**. Applications of Optimization in Some Complex Systems. Tartu 2018, 179 p.
2. **Riivo Kikas**. Analysis of Issue and Dependency Management in Open-Source Software Projects. Tartu 2018, 115 p.
3. **Ehsan Ebrahimi**. Post-Quantum Security in the Presence of Superposition Queries. Tartu 2018, 200 p.
4. **Ilya Verenich**. Explainable Predictive Monitoring of Temporal Measures of Business Processes. Tartu 2019, 151 p.
5. **Yauhen Yakimenka**. Failure Structures of Message-Passing Algorithms in Erasure Decoding and Compressed Sensing. Tartu 2019, 134 p.
6. **Irene Teinemaa**. Predictive and Prescriptive Monitoring of Business Process Outcomes. Tartu 2019, 196 p.
7. **Mohan Liyanage.** A Framework for Mobile Web of Things. Tartu 2019, 131 p.
8. **Toomas Krips.** Improving performance of secure real-number operations. Tartu 2019, 146 p.
9. **Vijayachitra Modhukur.** Profiling of DNA methylation patterns as biomarkers of human disease. Tartu 2019, 134 p.
10. **Elena Sügis.** Integration Methods for Heterogeneous Biological Data. Tartu 2019, 250 p.
11. **Tõnis Tasa.** Bioinformatics Approaches in Personalised Pharmacotherapy. Tartu 2019, 150 p.
12. **Sulev Reisberg.** Developing Computational Solutions for Personalized Medicine. Tartu 2019, 126 p.
13. **Huishi Yin.** Using a Kano-like Model to Facilitate Open Innovation in Requirements Engineering. Tartu 2019, 129 p.
14. **Faiz Ali Shah.** Extracting Information from App Reviews to Facilitate Software Development Activities. Tartu 2020, 149 p.
15. **Adriano Augusto**. Accurate and Efficient Discovery of Process Models from Event Logs. Tartu 2020, 194 p.
16. **Karim Baghery.** Reducing Trust and Improving Security in zk-SNARKs and Commitments. Tartu 2020, 245 p.
17. **Behzad Abdolmaleki.** On Succinct Non-Interactive Zero-Knowledge Protocols Under Weaker Trust Assumptions. Tartu 2020, 209 p.
18. **Janno Siim.** Non-Interactive Shuffle Arguments. Tartu 2020, 154 p.
19. **Ilya Kuzovkin.** Understanding Information Processing in Human Brain by Interpreting Machine Learning Models. Tartu 2020, 149 p.
20. **Orlenys López Pintado.** Collaborative Business Process Execution on the Blockchain: The Caterpillar System. Tartu 2020, 170 p.
21. **Ardi Tampuu.** Neural Networks for Analyzing Biological Data. Tartu 2020, 152 p.

22. **Madis Vasser.** Testing a Computational Theory of Brain Functioning with Virtual Reality. Tartu 2020, 106 p.
23. **Ljubov Jaanuska.** Haar Wavelet Method for Vibration Analysis of Beams and Parameter Quantification. Tartu 2021, 192 p.
24. **Arnis Parsovs.** Estonian Electronic Identity Card and its Security Challenges. Tartu 2021, 214 p.
25. **Kaido Lepik.** Inferring causality between transcriptome and complex traits. Tartu 2021, 224 p.
26. **Tauno Palts.** A Model for Assessing Computational Thinking Skills. Tartu 2021, 134 p.
27. **Liis Kolberg.** Developing and applying bioinformatics tools for gene expression data interpretation. Tartu 2021, 195 p.
28. **Dmytro Fishman.** Developing a data analysis pipeline for automated protein profiling in immunology. Tartu 2021, 155 p.
29. **Ivo Kubjas.** Algebraic Approaches to Problems Arising in Decentralized Systems. Tartu 2021, 120 p.
30. **Hina Anwar.** Towards Greener Software Engineering Using Software Analytics. Tartu 2021, 186 p.
31. **Veronika Plotnikova.** FIN-DM: A Data Mining Process for the Financial Services. Tartu 2021, 197 p.
32. **Manuel Camargo.** Automated Discovery of Business Process Simulation Models From Event Logs: A Hybrid Process Mining and Deep Learning Approach. Tartu 2021, 130 p.
33. **Volodymyr Leno.** Robotic Process Mining: Accelerating the Adoption of Robotic Process Automation. Tartu 2021, 119 p.
34. **Kristjan Krips.** Privacy and Coercion-Resistance in Voting. Tartu 2022, 173 p.
35. **Elizaveta Yankovskaya.** Quality Estimation through Attention. Tartu 2022, 115 p.
36. **Mubashar Iqbal.** Reference Framework for Managing Security Risks Using Blockchain. Tartu 2022, 203 p.
37. **Jakob Mass.** Process Management for Internet of Mobile Things. Tartu 2022, 151 p.
38. **Gamal Elkoumy.** Privacy-Enhancing Technologies for Business Process Mining. Tartu 2022, 135 p.
39. **Lidia Feklistova.** Learners of an Introductory Programming MOOC: Background Variables, Engagement Patterns and Performance. Tartu 2022, 151 p.