

PAPAYA: a Library for Performance Analysis of SQL-based RDF Processing Systems

Mohamed Ragab^a, Adam Satria Adidarma^{b,*} and Riccardo Tommasini^c

^a *Computer Science Inst., University of Tartu, Tartu, Estonia*

E-mail: mohamed.ragab@ut.ee

^b *Sepuluh Nopember Institute of Technology, Surabaya, Indonesia*

E-mail: adam.19051@mhs.its.ac.id

^c *LIRIS Lab INSA de Lyon, Villeurbanne, France*

E-mail: riccardo.tommasini@insa-lyon.fr

Abstract. *Prescriptive Performance Analysis (PPA)* has shown to be more useful than traditional *descriptive* and *diagnostic* analyses for making sense of Big Data (BD) frameworks' performance. In practice, when processing large (RDF) graphs on top of relational BD systems, several design decisions emerge and cannot be decided automatically, e.g., the choice of the schema, the partitioning technique, and the storage formats. PPA, and in particular ranking functions, helps enable actionable insights on performance data, leading practitioners to an easier choice of the best way to deploy BD frameworks, especially for graph processing. However, the amount of experimental work required to implement PPA is still huge. In this paper, we present PAPAYA¹, a library for implementing PPA that allows (1) preparing RDF graphs data for a processing pipeline over relational BD systems, (2) enables automatic ranking of the performance in a *user-defined* solution space of experimental dimensions; (3) allows user-defined flexible extensions in terms of systems to test and ranking methods. We showcase PAPAYA on a set of experiments based on the SparkSQL framework. PAPAYA simplifies the performance analytics of BD systems for processing large (RDF) graphs. We provide PAPAYA as a public *open-source* library under an *MIT* license that will be a catalyst for designing new research prescriptive analytical techniques for BD applications.

Keywords: Benchmarking, RDF Systems, Big Data, Apache Spark

1. Introduction

The increasing adoption of Knowledge Graphs (KGs) in industry and academia requires scalable systems for taming linked data at large volumes and velocity [1–3]. In absence of a scalable native graph system for *querying* large (RDF) graphs [4], most approaches fall back to using relational Big Data (BD) frameworks (e.g., *Apache Spark* or *Impala*) for handling large graph query workloads [5, 6]. Despite its flexibility, the relational model requires several additional *design decisions* when used for processing graphs, which cannot be decided automatically, e.g., the choice of the *schema*, the *partitioning techniques*, and the *storage formats*.

In [7, 8], we highlight the severity of the problem by showing the lack of performance replicability of BD systems for querying large (RDF) graphs. In particular, we showed that changing even just one experimental parameter, e.g., *partitioning technique* or *storage encoding*, invalidates existing optimizations in the relational representation of RDF data. We observe that issues do not lay in how the investigations were conducted but rather in the maturity of the analysis, which is limited to *descriptive* or at most *diagnostic* observations of the system behaviour. Such discussions leave much work for practitioners to transform performance observations into actionable insights [9].

*Equal Contribution with the first author. Corresponding Author mail E-mail: mohamed.ragab@ut.ee.

¹<https://github.com/DataSystemsGroupUT/PAPyA>

Later in [9], we have introduced the concept of *Bench-Ranking* as a means for enabling *Prescriptive Performance Analysis* (PPA) for processing larger RDF graphs. The PPA is an alternative to descriptive/diagnostic analyses that aims to answer the question *What should we do?* [10]. In practice, *Bench-Ranking* enables informed decision-making without neglecting the effectiveness of the performance analyses [9]. In particular, we showed how it could prescribe the best-performing combination of schema, partitioning technique, and storage format for querying large (RDF) graphs on top of SparkSQL framework [9, 11].

Our direct experience with big RDF graphs processing confirms what a well-known truth in data engineering and science project, i.e., most *time-consuming* phases is the *data preparation* [12], which accounts for 80% of the work ². In our work, we also show that the *performance analytics* can be extremely overwhelming, with the maze of performance metrics rapidly increasing with the number of system knobs. Although the Bench-Ranking methodology [9] simplifies performance analyses, a cohesive system that helps automate the intermediate steps is currently missing. In particular, the existing bench-ranking implementation was designed to show the feasibility of the approach as it does not follow any specific software engineering best practices. Thus, the adoption of our Bench-Ranking methodology may face the following challenges :

C.1 Experiment Preparation 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* (i.e., for logical schema definitions), and *Apache SparkSQL* (i.e., for physical partitioning and storage).

C.2 Portability and Usability: 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.

C.3 Flexibility and Extensibility: 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.

C.4 Complexity and Compoundness: 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 address these problems, we extend the work in [9] by designing and implementing an open-source library called *PAPAAYA* (*Python-based Approach for Performance Advanced Yield Analytics*). The main intention of this tool was to reduce our efforts while preparing the pipeline of processing large RDF KGs on Big relational engines (specifically the data preparation phase) and whilst applying the Bench-Ranking methodology for providing prescriptive analyses on the performance results. Yet, we still believe we designed PAPAAYA in a way that makes it useful and handy for practitioners to process large KGs.

The *PAPAAYA* library stems from the following objectives: (O.0) reducing the engineering work required for graph processing preparations and data loading. (O.1) reproducing existing experiments (according to user needs and convenience) for relational processing of SPARQL queries using SparkSQL. This will reduce massive efforts for building analytical pipelines from scratch for relational BD systems subject to the experiments. Moreover, PAPAAYA also aims at (O.2) automating the Bench-Ranking methods for enabling post-hoc prescriptive performance analyses described in [9]. In practice, PAPAAYA 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. Last but not least, (O.3) checking the replicability of the relational BD systems' performance for querying large (RDF) graphs within a complex experimental solution space.

The focus of this paper is to show the internals and functionality of PAPAAYA as a means for providing PPA for BD relational systems that query large (RDF) graphs. For completeness, we aim to describe PAPAAYA prescriptions with the WatDiv benchmark [13] experiments ³. In [9], we applied Bench-Ranking to the SP²B benchmark [14].

²Cleaning Big Data: Most Time-Consuming, Least Enjoyable Data Science Task, Survey Says <https://shorturl.at/wAQ47>

³Due to space limits, we diagnose the WatDiv prescriptions results and evaluation metrics in section 5 on the library GitHub page, mentioned below.

However, the best-performing configurations depend on the dataset and the query workload [9]. Thanks to PAPA YA, we can easily perform the PPA for any RDF benchmark, pointing out the performance results, and specifying the experimental configurations (see Figure 1).

Outline. Section 2 discusses the related work. Section 3 presents the necessary preliminaries to understand the paper’s content. Section 3.2 briefly introduces the *Bench-Ranking* framework concepts [9]. Section 4 presents the PAPA YA requirements alongside its framework architecture. Section 5 shows how to use PAPA YA in practice, providing examples from existing experiments [7, 9], while Section 6 concludes the paper and presents PAPA YA’s roadmap.

2. Related Work

Several tools and recommendation frameworks exist to reduce the effort required to design and execute reproducible experiments, share their results, and build pipelines for various applications [15]. For instance, the *RSPLab* provides a *test-drive* for stream reasoning engines that can be deployed on the cloud. It enables the design of experiments via a programmatic interface that allows deploying the environment, running experiments, measuring the performance, and visualizing the results. In their work outlined in [16], the authors tackled the problem of quantifying the continuous behavior of a query engine and presented two novel experimental metrics (*dief@t* and *dief@k*) that capture the performance efficiency during an elapsed period rather than a constant time. These metrics evaluate the query engine performance based on the query processing logs at various times (t) and various results (k). On another side, *gMark* [17] presents a flexible, domain-agnostic, extensible graph dataset generator driven by schemas. Additionally, it furnishes query workloads tailored to anticipated selectivity. Similarly, authors in [18] provide a data loader that facilitates generating RDF graphs in different logical relational schemas and physical partitioning options. However, this tool stops the work of data generation and data loading. This tool leaves the work of deciding the best experimental solutions for the data/knowledge engineers.

The mentioned efforts aim at providing the environment that enables the practitioners to develop their experimental pipelines. Nonetheless, none of these efforts provide prescriptive performance analyses in the context of BD problems. Conversely, PAPA YA aims to cover this timely research gap in an easy and extensible approach, facilitating the building of an experimental solution space for processing large RDF KGs, hence automating PPA where possible.

3. Background

This section presents the necessary background to understand the paper’s content. We assume that the reader is familiar with the RDF data model and the SPARQL query language.

3.1. RDF Relational Processing Experimental Dimensions

Several design decisions emerge when utilizing relational BD systems for querying large (RDF) graphs, such as the relational schema, partitioning techniques, and storage formats. These experimental dimensions directly impact the performance of BD systems while querying large graphs. Intuitively, these dimensions entail different choice options (we call them dimensions’ parameters).

First, the **relational schema**, it is easy to show that we have different options on how to represent graphs as relational tables, and the choice hugely impacts the performance [5, 19]. We identify the most used ones in the literature of RDF processing [5, 6, 19, 20]: *Single Statement Table* (ST) schema that stores RDF triples in a *ternary-column* relation (subject, predicate, object), and often requires many self-joins; *Vertically Partitioned Tables* (VP) proposed by *Abadi et al.* [19] to mitigate issues of *self-joins* in ST schema proposing to use *binary* relations (subject, object) for each unique predicate in the RDF dataset; the *Property Tables* (PT) schema that prescribes clustering multiple RDF properties as *n-ary* columns table for the same *subject* to group entities that are similar in structure.

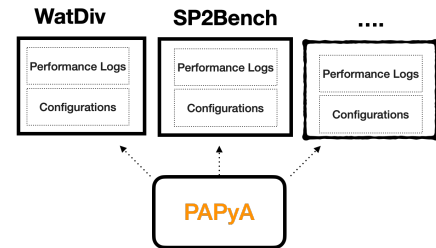


Fig. 1.: PAPA YA dynamicity.

Lastly, two more RDF relational schema advancements also emerge in the literature. The Wide Property Table (WPT) schema encodes the entire dataset into a single *denormalized* table. WPT is initially proposed for *Sempala* system by *Scätzle et.al.* [6], who also proposed another schema optimization that extended the version of the VP schema (ExtVP) [5] that pre-computes *semi-join* VP tables to reduce data shuffling.

BD platforms are designed to scale horizontally; thus, data *partitioning* is another crucial dimension for querying large graphs. However, choosing the right **partitioning** technique for RDF data is non-trivial. To this extent, we followed the indication of *Akhter et.al.* [21]. In particular, 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 data skewness which impacts parallelism.

Serializing RDF data also offers many options such as *RDF/XML*, *Turtle*, *JSON-LD*, to name a few. On the same note, BD platforms offer many options for reading/writing to various file formats and storage backends. Therefore, we need to consider the variety of **storage** formats [22]. We specifically focus on the various *Hadoop Distributed File System* (HDFS) file formats that are suitable for distributed scalable setups. In particular, HDFS supports the following row-oriented formats (e.g., *CSV* and *Avro*) and columnar formats (e.g., *ORC* and *Parquet*).

3.2. Bench-Ranking in a Nutshell

This section summarizes the concept of Bench-Ranking as a means for Prescriptive Performance Analysis. Bench-Ranking is based on *three* fundamental notions, i.e., *Configuration*, *Ranking Function*, and *Ranking Set*, defined below.

Definition 1. A configuration c is a combination of experimental dimensions. The configuration space \mathcal{C} is the Cartesian product of the possible configurations.

In [9], we consider a three-dimensional configuration space, i.e., including *relational schemas*, *partitioning techniques* and *storage formats*. Figure 2 shows the experimental space and highlights the example of the (a.ii.3) configuration, which is akin to the Single Triples (ST) schema, Subject-based Partitioning (SBP) technique, and stored in the HDFS (ORC) storage file format. This naming convention guides configurations reading in the rest of the paper results (figures and tables).

Definition 2. A ranking set \mathcal{R} is an ordered set of elements ordered by a rank score. The rank index r_i is called the index of a ranked element i within the ranking set \mathcal{R} , i.e., $\mathcal{R}[r_i]=i$. We denote with \mathcal{R}^k the leftmost (k top-ranked) subset of \mathcal{R} , and we denote with \mathcal{R}_x the ranking set calculated according to the Rank score R_x .

Definition 3. A ranking set is defined by a ranking function $f_R : \mathcal{C} \rightarrow \mathcal{R}$ that associates a rank score to every element in \mathcal{C} .

A valid example of a ranking score can be the time required for query executions by each of the selected configurations (see Table 1). The *ranking function* abstracts this notion (Definition 3). In [9], we consider a generalized version of the ranking function presented in [21], which calculates the rank scores for the configurations as follows:

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

In Equation (1), R is the *rank score* of the ranked dimension (i.e., relational schema, partitioning technique, storage format, or any other experimental dimensions). Such that, d represents the total number of parameters (options) under that dimension (for instance *five* in case of schemas, see Figure 2), $O_{dim}(r)$ denotes the number of occurrences

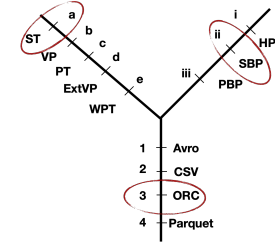


Fig. 2.: The configuration space \mathcal{C} [9].

Conf. \ Query	Q1	Q2	...	Q20	Q1	Q2	...	Q20
a.i.1	63.8	50.9	...	13.5	41 th	41 th	...	40 th
a.i.2	133.3	147.5	...	46.8	44 th	48 th	...	46 th
...
b.iii.4	24.8	25.2	...	7.5	11 th	26 th	...	32 th
...
e.ii.4	205.2	162.3	...	26.6	60 th	60 th	...	42 th

Table 1

Configuration rankings by query execution time, e.g., (a.i.1) configuration is at 41th rank in Q1 [9].

Schema	1 st	2 nd	3 rd	4 th	5 th	R
ExtVP	6	6	8	0	0	0.73
PT	6	6	5	2	1	0.68
WPT	7	3	0	0	10	0.46
ST	1	3	4	9	3	0.38
VP	0	2	3	9	6	0.26

Table 2

Example of Rank Scores [9].

of the dimension being placed at the rank r (1st, 2nd, ...). Moreover, $|Q|$ represents the total number of queries. Rank scores define the *Single-Dimensional* (SD) ranking criteria that help to provide a *high-level* view of the system performance across a set of tasks (e.g., queries in a workload) [9]. Table 2 shows a simple example of applying the above formula for computing the rank scores of the relational schema dimension. The ExtVP schema was placed in the "first" rank (six) times (i.e., performed the best in six queries), the "second" (six) times, the "third" (eight) times, the "fourth" (zero) times, and the last "fifth" also (zero) times. Thus, its overall ranking is 0.73. In contrast, the VP performed the worst with a rank score of 0.26. Intuitively, this means that the ExtVP schema in this example is the best (i.e., it has the highest rank score), and the VP schema is the worst-performing one.

Despite its generalization, Equation (1) is insufficient for ranking the configurations in a configuration set defined \mathcal{C} when it counts multiple dimensions [9]. The presence of *trade-offs* [9, 23] reduces the accuracy of these SD ranking functions. Thus, we introduced Multi-Dimensional ranking [9] by approaching Bench-Ranking as a *Multi-objective Optimization* problem. In particular, we adopt the *Pareto frontier* optimization techniques⁴, implemented using *Non-dominated Sorting Genetic Algorithm (NSGA-II)* [24], to optimize the experimental dimensions altogether and find the best-performing configuration in \mathcal{C} .

Finally, our Bench-Ranking frameworks include *two* evaluation metrics to assess the *effectiveness* of the proposed ranking criteria. In particular, we consider a ranking criterion is *good* if it does not suggest *low-performing* configurations and if it minimizes the number of contradictions within an experimental setting. When it comes to PPA, practitioners are not interested in a configuration that is the fastest at answering any specific query in a workload as long as it is never the slowest at any of the queries. To this extent, we identified two evaluation metrics [9], i.e., (i) **Conformance**, which measures the *adherence* of the *top-ranked* configurations w.r.t actual query results (see Table 1); (ii) **Coherence** which measures the level of (dis)agreement between two ranking sets that use the *same* ranking criterion across different experiments (e.g., different dataset sizes).

We calculate the conformance according to Equation (2) by positioning an element in a ranking set w.r.t the initial rank score. For instance, let's consider a ranking criterion \mathcal{R}_x with the *top-3* ranked configurations ($k = 3$) are $\mathcal{R}_x^{k=3} = \{d.ii.3, b.ii.2, \mathbf{e.ii.4}\}$, that overlap only with the *bottom-3* ranked configurations ($h = 3$) in one query Q_x , as shown in Table 1. That is, $Q_x^{h=3} = \{e.iii.1, e.iii.3, \mathbf{e.iii.4}\}$, i.e., $e.iii.4$ is in the 60th position out of 60 ranks/positions (i.e., the last rank). Thus, the Conformance of $(\mathcal{R}_x^3) = 1 - 1/(20 * 3)$, when $k = 3$, $h = 3$, and $Q = 20$.

$$A(\mathcal{R}^k) = 1 - \sum_{i=0}^{|Q|} \sum_{j=0}^k \frac{\bar{A}(i, j)}{|Q| * k}, \quad \bar{A}(i, j) = \begin{cases} 1 & \mathcal{R}^k[j] \in Q_i^i \\ 0 & \text{otherwise} \end{cases} \quad (2)$$

For coherence, we employ *Kendall's index*⁵ according to Equation (3), which counts the number of pairwise (dis)agreements between two ranking sets, Kendall's distance between two ranking sets \mathcal{R}_1 and \mathcal{R}_2 , where P is the set of *unique* pairs of distinct elements. The larger the distance, the more *dissimilar* the ranking sets are.

⁴Pareto frontier aims at finding a set of optimal solutions if no objective can be improved without sacrificing at least one other objective.

⁵Kendall is a standard measure to compare the outcomes of ranking functions.

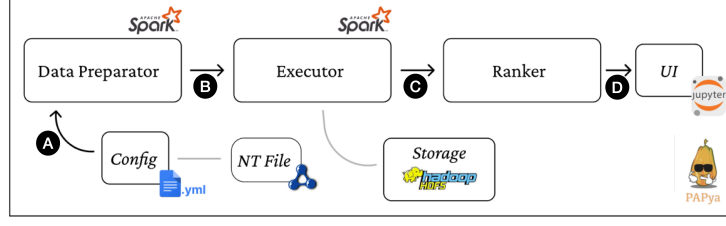


Fig. 3. Papaya architecture and workflow.

$$\bar{K}_{i,j}(\mathcal{R}1, \mathcal{R}2) = \begin{cases} 0 & \mathcal{R}1[r_i^1] = \mathcal{R}2[r_i^2] = i \wedge \mathcal{R}1[r_j^1] \\ & = \mathcal{R}2[r_j^2] = j \wedge \\ & r_i^1 - r_j^1 = r_i^2 - r_j^2 \\ 1 & \text{otherwise} \end{cases} \quad K(\mathcal{R}1, \mathcal{R}2) = \sum_{\{i,j\} \in P} \frac{\bar{K}_{i,j}(\mathcal{R}1, \mathcal{R}2)}{|P|} \quad (3)$$

We assume that ranking sets have the same number of elements. For instance, the K index between $\mathcal{R}1=(a.ii.3,a.i.3,b.ii.2)$ and $\mathcal{R}2=(a.ii.3,a.i.3,c.iii.2)$ for 100M and 500M is 0.33, i.e., one disagreement out of three comparisons.

4. PAPAAYA

In this section, we present the requirement analysis for PAPAAYA library and describe its architecture (Figure 3). We elicit PAPAAYA’s requirements based on the implementation challenges we discussed in the introduction and on the existing research efforts on benchmarking BD systems for processing and querying large RDF graphs [5, 6, 9, 23, 25]. Before delving into the requirements, it is also important to list the assumptions under which PAPAAYA is designed. We derive the following assumptions from our work on the Bench-Ranking framework [9].

A.1 Posthoc Performance Analysis of One-Time Query Processing: Bench-Ranking framework [9], wrapped in PAPAAYA, only supports “One-Time” query processing of SPARQL queries mapped into SQL and runs on top of big relational management systems (e.g., Spark-SQL). This excludes the performance analyses done for continuous query processing [16].

A.2 Query workload includes only Conjunctive SPARQL SPJ queries: The query workload includes only Conjunctive SPARQL Select-Project-Join (SPJ) queries for which there exists an SQL translation for a given schema.

A.3 Query Engine is a black box: In PAPAAYA, The query engine is treated as a black box, and thus the “pre-processing” query optimization phase is not part of the analysis. The query optimization is delegated to the Big data management systems’ optimizers (e.g., SparkSQL *Catalyst*).

4.1. Requirements

Given our assumptions, we can outline the requirements as follows:

R.1 Support for PPA: PAPAAYA shall support the necessary abstractions required to support PPA. Moreover, by default, it shall support existing Bench-Ranking techniques in [9]. (O.0)

R.2 Independence from the Key Performance Indicators (KPIs): PAPAAYA must enable PPA independently from the chosen KPI. In [9], we opted for query latency, yet one may need to analyze the performance in terms of other metrics (e.g., memory consumption). (O.2)

R.3 Independence from Experimental Dimensions: PAPAAYA must allow the definition of an arbitrary number of dimensions, i.e., allow the definition of n -dimensional configuration space. (O.3)

R.4 Usability: PAPAAYA prepares the experimental environment for processing large RDF datasets. Also, it supports decision-making, simplifying the performance data analytics. To this extent, data visualization techniques and a simplified API are both of paramount importance. (O.0)

Challenges	Requirements	PAPAAYA Solutions
C.1: Experiments Preparation	R4, R5	- Data Preparator that generates and loads graph data for distributed relational setups. - User-defined <i>YAML</i> configurations files.
C.2: Portability and Usability	R2, R3, R4, R5	- Data Preparator 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.
C.3: Flexibility and Extensibility	R1, R3, R5	- Allow adding/excluding experimental dimensions. - Allow adding new ranking algorithms. - Flexibility in shortening and enlarging the configuration space.
C.4: Compoundness and Complexity	R1, R4	- Interactive Jupyter Notebooks. - Variety of data and ranking visualizations

Table 3

Summary of challenges and requirements mapping along with PAPAAYA solutions.

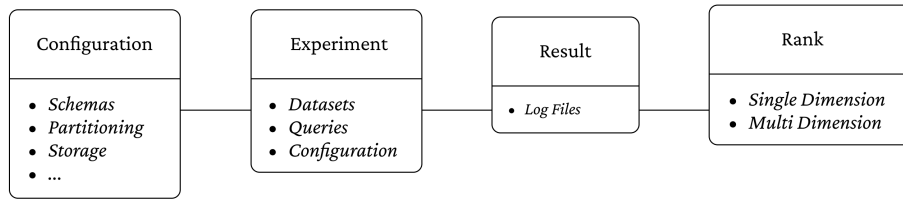


Fig. 4. PAPAAYA internal abstractions.

R.5 Flexibility and Extensibility: PAPAAYA should be extensible both in terms of architecture and programming abstractions to adapt to adding/removing configurations, dimensions, workload queries, ranking methods, evaluation metrics, etc. It also should decouple data and processing abstractions to ease the integration of new components, tools, and techniques. (O.1, O.3)

4.2. Architecture, Abstractions, and Internals

This section presents the PAPAAYA's main components and shows how they fulfill the requirements. Table 3 summarizes the requirements for challenges mappings alongside the PAPAAYA solutions. PAPAAYA allows its users 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) 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, PAPAAYA includes *three* core modules depicted in Figure 3, i.e., the *Data Preparator*, the *Executor*, and the *Ranker*. Moreover, PAPAAYA relies on few core abstractions depicted in Figure 4, i.e., *Configuration*, *Experiment*, *Result*, and *Rank*. While detailing each module's functionalities, we introduce PAPAAYA workflow, which also appears in Figure 3, starting with the input is a configuration file that points to the input *N-Triple* file with the RDF graph (Figure 3 step (A)).

The first actor in the pipeline is the *Data Preparator* (DP), which prepares RDF graphs for relational processing. It takes as input a configuration file that includes experimental options of interest. The configuration file is represented by the *Configuration* abstraction (see Figure 4), which enables extensibility (R.5). Specifically, the DP allows defining an arbitrary number of dimensions with as many options as specified (R.3). In particular, it considers the dimensions specified in [9] (R.1), i.e., relational schemas, storage format, and partitioning technique. Therefore, the DP automatically prepares (i.e., *determines, generates, and loads*) the RDF graph dataset with the specified configurations that adapt with the relational processing paradigm to the storage destination (e.g., HDFS). More specifically, DP currently includes four relational schemas commonly used for RDF processing, i.e., (a) ST, (b) VP, (d) ExtVP, and (e) WPT⁶. For partitioning, DP currently supports three partitioning techniques, i.e., (i) horizontal partitioning

⁶Automating PT schema ("b" in Figure 2) generation is not yet supported by the current PAPAAYA DP

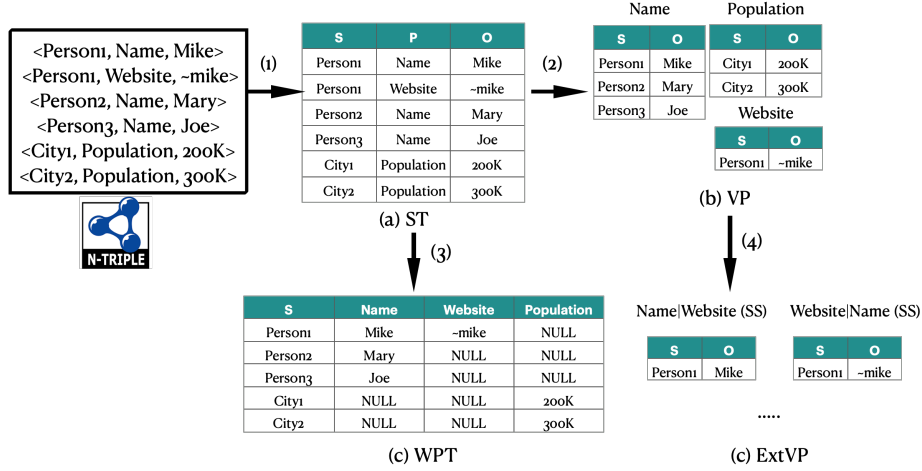


Fig. 5. RDF relational schema transformations in PAPAAYA Data Preparator.

(HP), (ii) subject-based partitioning (SBP), and predicate-based partitioning (PBP). Last but not least, DP enables storing data using four HDFS file formats (i) *CSV* or (ii) *Avro*, which are row-oriented, and (iii) *ORC* or (iv) *Parquet*, which are column-oriented storage formats [22].

The DP interface is generic, and the generated datasets are *agnostic* to the underlying relational system. The DP prepares RDF *graph* data for processing with different *relational* BD systems, especially SQL-on-Hadoop systems, e.g., SparkSQL, Hive, and Impala. Seeking scalability, the current DP implementation relies on *SparkSQL*, which allows implementation of RDF relational schema generation using the SQL transformations. Notably, *Apache Hive* or *Apache Impala* could be potential candidates for alternative implementation executors. However, SparkSQL also supports different partitioning techniques and multiple storage formats, making it ideal for our experiments [9].

Figure 5 shows a sample of schema generation in PAPAAYA DP component. First, the DP transforms the input RDF graph (*N-Triples* file(s)) into an ST table schema (i.e., Figure 5 Step (1)), and then other schemas are generated using parameterized SQL queries⁷. For instance, the VP and WPT schemas are generated using SQL queries given the ST table as a parameter (i.e., Figure 5 Step (2), and (3), respectively). While, the ExtVP schema generation relies on VP tables to exist first (i.e., Step (4) in Figure 5).

The **Executor** is the following module in PAPAAYA workflow, which is the system that is subject to experimentation (see Figure 3 step B). For instance, in [5, 7, 9, 25] the considered system is Apache SparkSQL. The executor offers an abstract API to be extended (**R.5**). In practice, it (i) starts the execution pipeline in the external system, (ii) it collects the performance logs (**R.2**), 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). The *Experiment* abstraction is defined in Figure 4. We decided to wrap the running experiments in a *SparkSQL-based* executor (*SparkExecutor*). The experiment specifications (alongside the configurations) are passed to this wrapper as parameters. It is worth mentioning that the Executor assumes the query workload is available in the form of *SQL* queries (**Assumption A.2**) running in a "One-Time" style [16] (**Assumption A.1**). In the current stage, PAPAAYA does not support SPARQL query translation nor SQL query mappings, this work is left for the future roadmap of PAPAAYA (see Section 6). It is also important to note that the current version of PAPAAYA delegates the query optimization to the executor's optimizer (**Assumption A.3**).

The results logs are then loaded by the **Ranker** component into *python Dataframes* to make them available for analysis (see Figure 3 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 **R.2**, 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 (**R.5**), we decoupled the performance analytics (e.g., ranking calculation) from its definitions and visualization. In particular, the *Rank* class

⁷ schema SQL-based transformations are kept in the DP module on PAPAAYA's GitHub repository due to space limits.

D_i	WatDiv _{mini}						WatDiv _{full}					
	\mathcal{R}_f^3	\mathcal{R}_p^3	\mathcal{R}_s^3	Pareto _O	Pareto _{Agg}	\mathcal{R}_{ta}^3	\mathcal{R}_f^3	\mathcal{R}_p^3	\mathcal{R}_s^3	Pareto _O	Pareto _{Agg}	\mathcal{R}_{ta}^3
100M	a.ii.3	a.ii.3	c.i.4	c.ii.2	a.ii.3	a.ii.3	a.ii.3	c.ii.2	d.iii.4	d.iii.2	c.ii.2	c.ii.2
	b.ii.2	a.ii.4	c.ii.2	c.i.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
	a.i.3	a.ii.2	c.i.3	b.ii.2	b.ii.2	b.ii.2	b.ii.2	c.ii.1	d.iii.3	d.iii.4	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.ii.3	a.ii.1	d.iii.4	d.iii.2	d.iii.3	d.iii.3
	a.i.3	a.ii.4	c.i.3	c.ii.2	b.ii.2	b.ii.2	a.i.3	d.iii.3	d.iii.3	d.iii.3	c.i.4	c.i.4
	c.i.4	a.ii.2	c.i.2	c.i.2	a.ii.3	a.ii.3	b.iii.2	a.ii.2	d.iii.2	c.i.4	a.iii.4	a.iii.4
500M	a.ii.3	a.ii.3	c.ii.4	c.ii.3	b.ii.2	b.ii.2	a.ii.3	c.ii.2	d.iii.2	d.ii.2	d.ii.2	c.ii.3
	a.i.3	a.ii.4	c.i.4	c.ii.4	c.ii.3	c.ii.3	a.i.3	a.iii.2	d.iii.4	c.ii.3	c.ii.3	d.ii.2
	c.i.3	a.ii.2	c.i.3	c.i.3	c.ii.4	c.ii.4	c.iii.2	a.iii.4	d.iii.1	a.iii.4	a.iii.4	a.iii.4

Table 4

WatDiv best-performing (*Top-3*) configurations according to the SD and MD ranking criteria.

abstraction reflects on the *ranking function* (Definition 3) that takes as input data elements and returns a *ranking set*. To fulfill **R.4**, 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.

The Rank call allows defining additional ranking criteria (**R.5**). In addition, to fulfill **R.1**, PAPAAYA already implements SD as well as *Multi-Dimensional* (MD) criteria specified in [9].

PAPAAYA allows its users to interact with the experimental environment (**R.5**) using *Jupyter Notebook*. To facilitate the analysis, it integrates data visualization (**R.4**) 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 in Section 3.2 the notions of coherence and conformance (**R.1**). 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.

5. PAPAAYA in Practice

In this section, we explain how to use PAPAAYA in practice, showcasing its functionalities with a focus on performance data analysis, flexibility, and visualizations. In particular, we design our experiments in terms of (i) a set of SPARQL queries that we 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 [9], i.e., schema, partitioning techniques, and storage formats.

In Bench-Ranking experiments [9], we used the *SP²B* [14] benchmark datasets. In this paper, we aim to use a different benchmark to check the robustness of PAPAAYA Bench-Ranking criteria and their evaluation metrics. Thus, we present the results of experiments opting for the *WatDiv* bench-

```

1 from papa import Configuration, SparkExecutor
2 from papa import data_preparator as dp
3 #Configurations
4 confs = Configuration({
5     "schemas":["ST","VP","WPT","ExtVP"],
6     "partition":["HP,SBP","PBP"],
7     "storage":["CSV","Avro","Parquet","ORC"]})
8 #Executor
9 Q=[q1,q2,...,q20]
10 exp = dp.experiment(dataset="100M", Q, confs)
11 spe = SparkExecutor(master="local[*]")
12 res=spe.run(exp,runs=5,dataPath="hdfs:...",logsPath="hdfs:...")
13 #Bench-Ranking
14 #(1) SD-Ranking Criteria
15 schemaSDRanks = SDRank(res,dim=list(conf.keys())[0],
16 q=len(Q),d=len(list(conf.values())[0])) #schema SDRanking
17 partitioningSDRanks = ... #partitioning SDRanking
18 storageSDRanks = ... #storage SDRanking
19 #(2) MD-Ranking (Pareto)
20 paretoFronts_Q=MDRankPareto(ds="100M",dims=Q)
21 paretoFronts_Agg=MDRankPareto(ds="100M",
22     dims=[schemaSDRanks,partitioningSDRanks,storageSDRanks])
23 #Visualization
24 SDRank.plot(schemaSDRanks) #plot SDRanking for schema
25 MDRankPareto.plot(paretoFronts) #plot MD-Ranking Pareto
26 #Ranking Criteria evaluation
27 conf=Ranker.conformance(schemaSDRanks,q=20,k=3,h=45)
28 coh=Ranker.coherence(schemaSDRanks_100M,schemaSDRanks_250M)

```

Listing 1: Experiment design example in PAPAAYA.

	WatDivMini						WatDivFull					
	Conformance			Coherence			Conformance			Coherence		
	D1	D2	D3	D1-D2	D1-D3	D2-D3	D1	D2	D3	D1-D2	D1-D3	D2-D3
\mathcal{R}_s	88.33%	91.67%	93.33%	0.09	0.12	0.06	96.00%	94.00%	93.00%	0.1	0.13	0.07
\mathcal{R}_p	38.33%	13.33%	5.00%	0.14	0.14	0.14	73.00%	39.00%	36.00%	0.09	0.28	0.17
\mathcal{R}_f	63.33%	46.67%	35.00%	0.16	0.39	0.3	64.00%	32.00%	43.00%	0.14	0.25	0.15
Pareto _Q	95.00%	98.33%	95.00%	0.14	0.25	0.14	92.00%	98.00%	98.00%	0.1	0.15	0.07
Pareto _{Agg}	88.33%	73.33%	93.33%	0.16	0.25	0.2	87.00%	71.00%	76.00%	0.17	0.24	0.15
\mathcal{R}_{ta}	88.33%	73.33%	93.33%	0.2	0.24	0.17	89.00%	84.00%	76.00%	0.15	0.22	0.13

Table 5

Ranking *Coherence* (Kendall distance, the lower the better) & *Conformance* across *WatDiv* datasets (D1=100M, D2=250M, D3=500M). mark [13]⁸. *WatDiv* benchmark includes a data generator and a query workload with various graph patterns, SPARQL query shapes, and selectivities that make the analysis *non-trivial*. In our experiments, which are based on the average results of *five runs*⁹, we measure the performance of SparkSQL as a BD relational engine in terms of query *latency*. However, alternative KPIs, e.g., memory consumption, could be easily used in PAPAAYA.

Listing 1 shows a full example of PAPAAYA 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 1 lines 4-8). 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 1 line 10). An executor is defined for running the experiment along with the number of times experiments will be run (Listing 1 line 11). 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 1 line 13)¹⁰. For instance, we call the *SDRank* (Listing 1 line 15) 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 (1)). The MD-Ranking(i.e., Pareto fronts) is applied in two ways. The first one is called Pareto_Q (Listing 1 line 20), which applies the *NSGA-II* algorithm by considering the ranking sets obtained while sorting each query results individually. Using the first method, the algorithm aims at *minimizing* the query runtimes across all dimensions. The second one is called the Pareto_{Agg} (Listing 1 line 21), 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., \mathcal{R}_s , \mathcal{R}_p , and \mathcal{R}_f . The user can plot the SD rank scores and the MD Pareto ranking criterion (Listing 1 line 23). In addition, the user can evaluate the effectiveness of the ranking criterion using conformance and coherence metrics (Listing 1 line 26).

Table 4 shows 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, Table 5 provides the ranking evaluation metrics (calculated according to Equations (2) and (3)).

5.1. Rich Visualizations

To fulfill **R.4**, PAPAAYA decouples data analytics from visualizations. Meaning that the user can specify his/her visualizations of interest with the performance data. Nevertheless, PAPAAYA still provides several interactive and extensible default visualizations that help 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 6 (a-c) shows three samples of SD-ranking criteria plots. In particular, they show how many times a specific dimension's (e.g., the schema in Figure 6 (a)) alternatives/options (ST, VP, PT,...etc) achieve the highest or the lowest ranking scores. Figure 7 shows the SD ranking criteria w.r.t a simple geometrical representation (detailed in the following sections) that depicts the triangle subsided by each dimension's ranking criterion (i.e., \mathcal{R}_s , \mathcal{R}_p , and \mathcal{R}_f). The triangle sides present the trade-off 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}¹¹ results are depicted using a 3D plot in Figure 8 (a). Pareto fronts are depicted by the green shaded area of the

⁸Nonetheless, seeking conciseness, we keep *SP²B* results on the GitHub repository.

⁹Benchmarks' query workload (in SQL) and experiments runtimes: <https://datasystemsgroup.github.io/SPARKSQLRDFBenchmarking>

¹⁰It is worth noting that the performance analyses, e.g., Bench-Ranking, could start directly if the performance data (logs) are already present.

¹¹Pareto_Q cannot be visualized, i.e., as it uses more than *three* dimensions, one for each query of the workload [9].

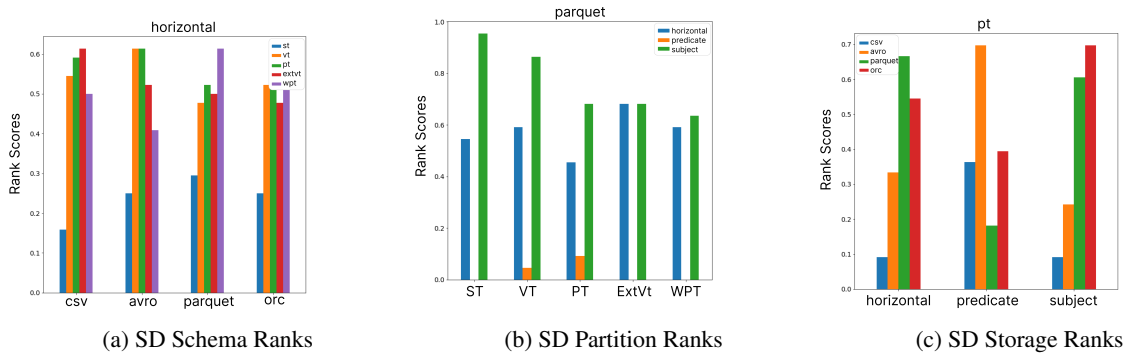


Fig. 6. Examples on SD Rank Scores over different dimensions (100M), the higher the better.

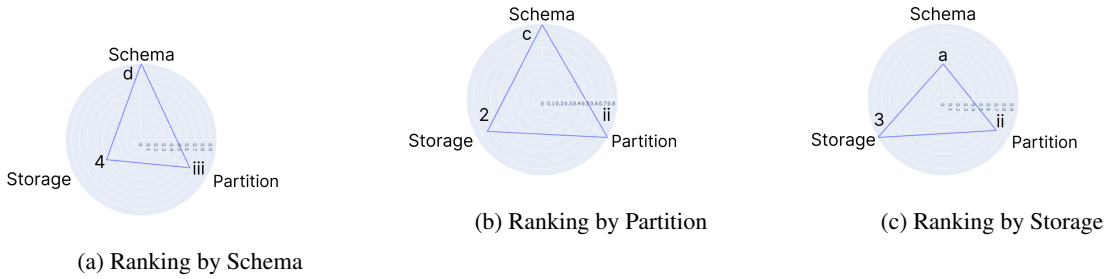


Fig. 7. Dimensions trade-offs using single-dimensional ranking (R_s , R_f , and R_p).

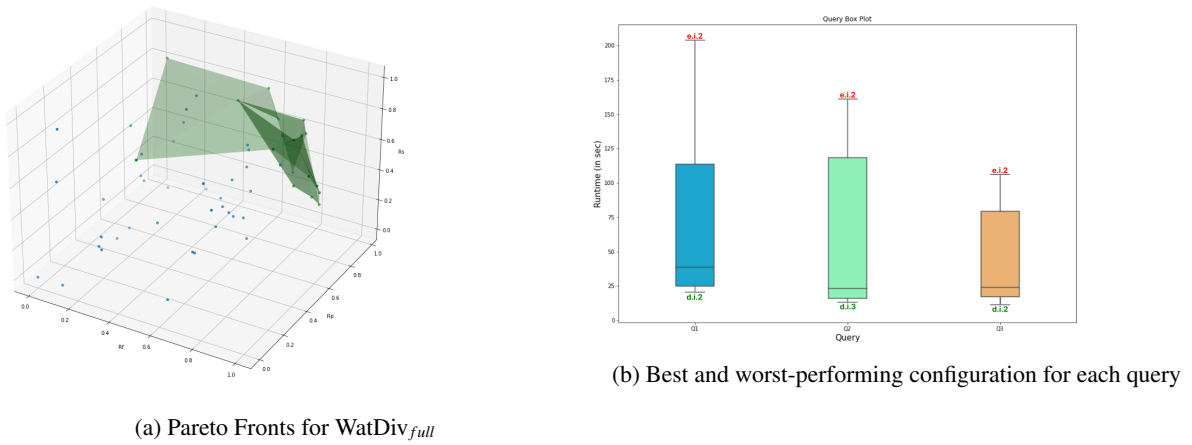


Fig. 8. Pareto Fronts, and queries best-worst configuration examples.

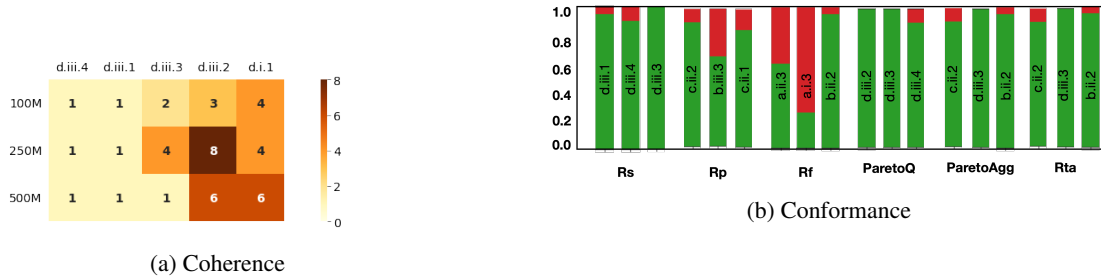


Fig. 9. Heatmap shows the coherence of the R_s criterion (Top-5 configurations) scaling from 100M to larger dataset scales. The stacked plot shows the Conformance of the top-3 ranked configurations.

three experimental dimensions of the Bench-Ranking (for WatDiv 500M triples dataset¹²). Each point of this figure represents a *solution* of rank scores (i.e., a configuration in our case).

PAPAAYA visualizations allow explaining the conformance and coherence results using simple plots. For instance, Figure 9 (a) shows the coherence of the top-5 ranked configurations of the \mathcal{R}_s criterion in the 100M dataset while scaling to the larger datasets, i.e., 250M and 500M. PAPAAYA 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 1). The green color represents the level of conformance, and the red depicts a configuration that is performing worse than the h worst rankings. Thus, this may explain why \mathcal{R}_p and \mathcal{R}_f criteria have low conformance results in Table 4, while the other criteria have relatively higher conformance values.

Practitioners can also use PAPAAYA visualizations for *fine-grained* ranking details. For instance, showing the best and worst configurations for each query (as shown in Figure 8 (b) for example of *three* queries of the WatDiv workload). Such detailed visualizations could help the user rationalize the final prescriptions of PAPAAYA.

5.2. PAPAAYA Flexibility & Extensibility

Adding/Removing Configurations/Queries.

To show an example of the extensibility of PAPAAYA, 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 PAPAAYA 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 the left part of Table 4). The configurations' exclusion and inclusion are specified easily from the *YAML* configuration file (as shown in Listing 2 lines 7-12), i.e., PAPAAYA considers only the specified configurations and ranks accordingly.

The right side of Table 4 shows the top-ranked *three* configurations according to the specified configurations. 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 (\mathcal{R}_s) criterion. In *WatDiv_{mini}*, excluding the Predicate partitioning ('*iii*'), the *subject-based* partitioning ('*ii*') completely dominates (one exception) in the \mathcal{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., \mathcal{R}_p , \mathcal{R}_s , Pareto_{Agg/Q}, and \mathcal{R}_{1a} .

With such flexibility, PAPAAYA also provides several dynamic views on the ranking criteria. For example, Table 6 shows the SD ranking of the schema dimension by 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,

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 6: Schemas global ranking across various configurations.

```

1 # (1) Full-WatDiv Configurations
2 dimensions:
3   schemas: ["st", "vp", "pt", "extvp", "wpt"]
4   partition: ["horizontal", "subject", "predicate"]
5   storage: ["Avro", "CSV", "ORC", "Parquet"]
6   query: 20
7 # (2) Mini-WatDiv Configurations
8 dimensions:
9   schemas: ["st", "vp", "pt"]
10  partition: ["horizontal", "subject"]
11  storage: ["csv", "orc", "parquet"]
12  query: 10
13 # (3) WatDiv without Partitioning (i.e., Centralized)
14 dimensions:
15  schemas: ["st", "vp", "pt", "extvp", "wpt"]
16  partition: null
17  storage: ["Avro", "CSV", "ORC", "Parquet"]
18  query: 20

```

Listing 2: *YAML* configuration file for various experiments in PAPAAYA.

¹²Due to space limits, we keep other Pareto figures on the PAPAAYA GitHub page

D_i	\mathcal{R}_s	\mathcal{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 7

Best-performing configurations, **excluding the partitioning** dimension.

i.e., "PBP!/PBP", "CSV!/CSV", respectively. For instance, the PT schema global ranking is interestingly oscillating with those changes in the available configurations.

Adding/Removing Full Experimental Dimension. PAPAAYA' 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 2 lines 13-18). In [26], we run experiments on SparkSQL with different relational schemas and storage backends yet without data partitioning. Table 7 shows the best-performing (top-3) configurations in WatDiv experiments when excluding the partitioning dimension. Table 8 shows the conformance and coherence metrics' results for the various ranking criteria¹³.

Adding Ranking Criterion. PAPAAYA abstractions enable users to plug in a new ranking criterion besides the already existing ones (i.e., the abstract Rank class, Section 4). 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 10, 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, $0 < R \leq 1$). Equation (1) defines the blue triangle area; we call it ranking by triangle area (R_{ta}).

$$TriangleArea(R_{ta}) = \frac{1}{2} \sin(120) * (R_f * R_p + R_s * R_p + R_f * R_s) \quad (4)$$

The formula (Cf. Equation (4)) 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* triangle is

Metric	D	\mathcal{R}_s	\mathcal{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 8

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

```

1 from papaya import Rank
2 #SD Ranker (Implementation of Equation (1))
3 class SD_Ranking (Rank):
4     ...
5 #MD Ranker (Pareto-NSGA2)
6 class MD_Ranking (Rank):
7     ...
8 #Add Triangle_Area as a new ranking criterion.
9 class RtaCriterion (Rank):
10    def calculate_rta(self):
11        r_scores = SDRank(...).calculateRank()
12        rta_scores = []
13        for i in range(len(r_scores)):
14            rs = r_scores[0][i]
15            rp = r_scores[1][i]
16            rf = r_scores[2][i]
17            # RTA Formula (Equation 4)
18            y = (math.sin(math.radians(120)))/2
19            outer_triangle_area = y * (1+1+1)
20            rta = y * (rf*rp + rs*rp + rf*rs)
21            rta_scores.append(rta)
22        return rta_scores
23    def plot(self):# plots (Figure 7)
24        ...

```

Listing 3: Plugin the Triangle-Area as new Ranking criterion.

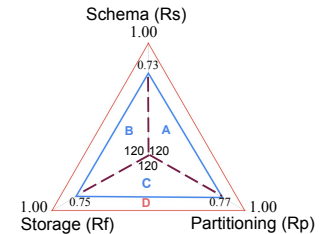
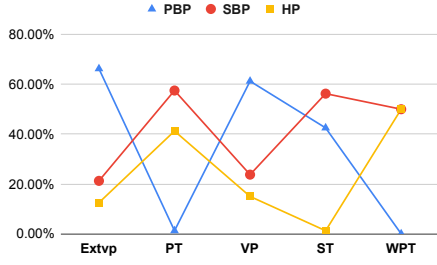
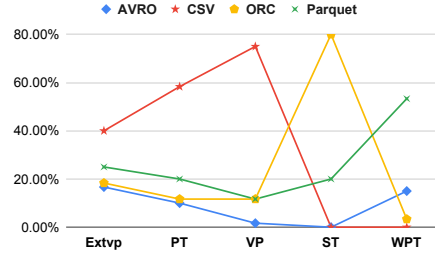


Fig. 10.: Triangle Area criterion.

¹³Notably, the R_p and R_{ta} criteria cannot be calculated when excluding the partitioning dimension.



(a) Impact of partitioning on the schema performance



(b) Impact of storage on the schema performance

Fig. 12. Schema Replicability across changing partitioning or storage formats.

$R_{ta} = \frac{1}{2} \sin(120)(0.75 * 0.771 + 0.73 * 0.77 + 0.75 * 0.73) = 0.73$. In addition to the SD and MD ranking criteria classes, Listing 3 shows how to extend PAPAAYA 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 PAPAAYA abstractions enable extending new ranking algorithms/criteria. Table 4 shows the top-5 best-performing 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 5 also shows 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.

5.3. Checking Performance Replicability

PAPAAYA 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 12 (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.

PAPAAYA can also check the performance replicability by comparing two configurations as discussed in [7]. For instance, PAPAAYA 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 [5, 6].

Table 9 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 evident in 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. Conversely, we observe significant

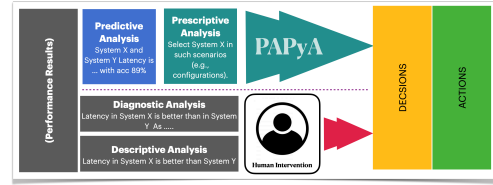


Fig. 11.: Performance analysis methodology, and how PAPAAYA reduces human intervention in BD analytics.

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

Table 9: The *replicability* of schema advancements (i.e., WPT, ExtVP) VS. baselines (i.e., PT, VP), WatDiv 500M dataset.

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. PAPAYA enables showing those *trade-offs* of considering alternative storage file formats and partitioning techniques alongside the experiments' query evaluation.

Table 10 provides a concise overview of the objectives, delineating the challenges encountered during their pursuit, and outlines the set of requirements necessary for accomplishing these objectives.

6. Conclusion and Roadmap

This paper presents PAPAYA, an extensible library that reduces the efforts needed to analyze the performance of BD systems used for processing large (RDF) graphs. PAPAYA implements the performance analytics methods adopted in [5, 6, 26] including an novel approach for prescriptive performance analytics we presented in [9].

Inspired by *Gartner's* analysis methodology [27], Figure 11 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 5.3). 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 12). On the other side, PAPAYA aims to reduce 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, PAPAYA still provides several descriptive analytics and visualizations on the performance to explain the final decisions given by PAPAYA. PAPAYA also aims to reduce the engineering work required for building an analytical pipeline for processing large RDF graphs. In particular, PAPAYA prepares, generates, and loads data ready for big relational RDF graph analytics.

PAPAYA 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, PAPAYA 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 PAPAYA as an open-source library under MIT license and published at a persistent URI. PAPAYA's GitHub repository includes tutorials and documentation on how to use the library.

As a maintenance plan, PAPAYA's roadmap includes:

1. Covering the phase of query evaluation into PAPAYA pipeline. In particular, we plan to provide native support of SPARQL by incorporating native triple stores for query evaluation.
2. Incorporating SPARQL into SQL translation for a given schema, i.e., query translation is a schema-dependent task. This can be approached using advancements of *R2RML* mapping tools (e.g., *OnTop*) [28].
3. Wrapping other *SQL-on-Hadoop* executors to PAPAYA; thus, the performance of the engines could also be compared as well as enabling benchmarking of other KG data models (e.g., property graphs [29]) in PAPAYA.
4. Using orchestration tools (such as *Apache Airflow*) to monitor the PAPAYA pipelines.
5. Integrating PAPAYA with tools like *gmark* [17], which generates graphs and workloads, and *ESPRESSO* [30], which enables search and query functionalities over personal online datastores as well as personal KGs.

Acknowledgments. We acknowledge support from the European Social Fund via IT Academy programme and the European Regional Development Funds via the Mobilitas Plus programme (grant MOBTT75).

References

- [1] R. Tommasini, M. Ragab and et.al., A first step towards a streaming linked data life-cycle, in: *International Semantic Web Conference*, 2020.

Objectives	Challenges	Requirements
O.0	C.1, C.2	R.2-R.5
O.1	C.2, C.3	R.1-R.5
O.2	C.2, C.3, C.4	R.1-R.5
O.3	C.2, C.3, C.4	R.1-R.5

Table 10: Summary of Objectives, Challenges, and Requirements.

- [2] R. Tommasini, M. Ragab, A. Falcetta, E. Della Valle and S. Sakr, Bootstrapping the Publication of Linked Data Streams.. 1
- [3] M.R. Moawad, H.M. O. Mokhtar and H.T. Al Feel, On-the-fly academic linked data integration, in: *Proceedings of the International Conference on Compute and Data Analysis*, 2017, pp. 114–122. 2
- [4] S. Sakr, A. Bonifati, H. Voigt, A. Iosup, K. Ammar, R. Angles, W. Aref, M. Arenas, M. Besta, P.A. Boncz et al., The future is big graphs: a community view on graph processing systems, *Communications of the ACM* **64**(9) (2021), 62–71. 3
- [5] A. Schätzle, M. Przyjacieli-Zablocki, S. Skilevic and G. Lausen, S2RDF: RDF querying with SPARQL on spark, *Proceedings of the VLDB Endowment* **9**(10) (2016), 804–815. 4
- [6] A. Schätzle, M. Przyjacieli-Zablocki, A. Neu and G. Lausen, Sempala: Interactive SPARQL query processing on hadoop, in: *ISWC*, 2014. 5
- [7] M. Ragab, R. Tommasini, F.M. Awaysheh and J.C. Ramos, 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. 6
- [8] M. Ragab, R. Tommasini and S. Sakr, Comparing Schema Advancements for Distributed RDF Querying Using SparkSQL, in: *Proceedings of the ISWC 2020 Demos and Industry Tracks*, CEUR Workshop Proceedings, Vol. 2721, CEUR-WS.org, 2020, pp. 30–34. 7
- [9] 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)*, IEEE Computer Society, Los Alamitos, CA, USA, 2021, pp. 241–251. doi:10.1109/BigData52589.2021.9671277. 8
- [10] K. Lepenioti, A. Bousedekis, D. Apostolou and G. Mentzas, Prescriptive analytics: Literature review and research challenges, *International Journal of Information Management* **50** (2020), 57–70. 9
- [11] M. Ragab, Towards Prescriptive Analyses of Querying Large Knowledge Graphs, in: *New Trends in Database and Information Systems - ADBIS 2022 Short Papers, Doctoral Consortium and Workshops: DOING, K-GALS, MADEISD, MegaData, SWODCH, Turin, Italy, September 5-8, 2022, Proceedings*, S. Chiusano, T. Cerquitelli, R. Wrembel, K. Nørsvåg, B. Catania, G. Vargas-Solar and E. Zumpano, eds, Communications in Computer and Information Science, Vol. 1652, Springer, 2022, pp. 639–647. doi:10.1007/978-3-031-15743-1_59. 10
- [12] N. Sambasivan, S. Kapania, H. Highfill, D. Akrong, P.K. Paritosh and L. Aroyo, "Everyone wants to do the model work, not the data work": Data Cascades in High-Stakes AI, in: *CHI '21: CHI Conference on Human Factors in Computing Systems, Virtual Event / Yokohama, Japan, May 8-13, 2021*, Y. Kitamura, A. Quigley, K. Isbister, T. Igarashi, P. Bjørn and S.M. Drucker, eds, ACM, 2021, pp. 39:1–39:15. doi:10.1145/3411764.3445518. 11
- [13] G. Aluç, O. Hartig, M.T. Özsu and K. Daudjee, Diversified stress testing of RDF data management systems, in: *International Semantic Web Conference*, Springer, 2014, pp. 197–212. 12
- [14] M.S. et.al., SP²Bench: A SPARQL Performance Benchmark, in: *ICDE 2009*, Y.E. Ioannidis, D.L. Lee and R.T. Ng, eds, 2009, pp. 222–233. 13
- [15] M.R. Moawad, M.M.M.Z.A. Maher, A. Awad and S. Sakr, Minaret: A recommendation framework for scientific reviewers, in: *the 22nd International Conference on Extending Database Technology (EDBT)*, 2019. 14
- [16] M. Acosta, M.-E. Vidal and Y. Sure-Vetter, Diefficiency metrics: measuring the continuous efficiency of query processing approaches, in: *International Semantic Web Conference*, 2017, pp. 3–19. 15
- [17] G. Bagan, A. Bonifati, R. Ciucanu, G.H. Fletcher, A. Lemay and N. Advokaat, gMark: Schema-driven generation of graphs and queries, *IEEE Transactions on Knowledge and Data Engineering* **29**(4) (2016), 856–869. 16
- [18] V.A.A. Ayala and G. Lausen, A Flexible N-Triples Loader for Hadoop., in: *International Semantic Web Conference (P&D/Industry/BlueSky)*, 2018. 17
- [19] D.J. Abadi, A. Marcus, S.R. Madden and K. Hollenbach, Scalable semantic web data management using vertical partitioning, in: *VLDB*, 2007. 18
- [20] I. Abdelaziz, R. Harbi, Z. Khayyat and P. Kalnis, A survey and experimental comparison of distributed SPARQL engines for very large RDF data, *Proceedings of the VLDB Endowment* (2017). 19
- [21] A. Akhter, A.-C.N. Ngonga and M. Saleem, An empirical evaluation of RDF graph partitioning techniques, in: *European Knowledge Acquisition Workshop*, 2018. 20
- [22] T. Ivanov and M. Pergolesi, The impact of columnar file formats on SQL-on-hadoop engine performance: A study on ORC and Parquet, *Concurrency and Computation: Practice and Experience* (2019), e5523. 21
- [23] M. Ragab, R. Tommasini, S. Eyvazov and S. Sakr, 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. 22
- [24] K. Deb, A. Pratap, S. Agarwal and T. Meyarivan, A fast and elitist multiobjective genetic algorithm: NSGA-II, *IEEE Transactions on Evolutionary Computation* **6**(2) (2002), 182–197. doi:10.1109/4235.996017. 23
- [25] M. Cossu, M. Färber and G. Lausen, Prost: Distributed execution of SparQL queries using mixed partitioning strategies, *arXiv preprint arXiv:1802.05898* (2018). 24
- [26] M. Ragab, R. Tommasini and S. Sakr, Benchmarking Spark-SQL under Alliterative RDF Relational Storage Backends., in: *QuWeDa@ ISWC*, 2019, pp. 67–82. 25
- [27] J. Hagerty, Planning Guide for Data and Analytics, 2017, [Online; accessed 4-Sep-2021]. 26
- [28] M. Belcao, E. Falzone, E. Bionda and E.D. Valle, Chimera: A Bridge Between Big Data Analytics and Semantic Technologies, in: *International Semantic Web Conference*, Springer, 2021, pp. 463–479. 27
- [29] M. Ragab, Large Scale Querying and Processing for Property Graphs, in: *DOLAP@EDBT/ICDT 2020, Copenhagen, Denmark, March 30, 2020*, 2020. 28
- [30] M. Ragab, Y. Savateev, R. Moosaei, T. Tiropanis, A. Poulouvassilis, A. Chapman and G. Roussos, ESPRESSO: A Framework for Empowering Search on Decentralized Web, in: *International Conference on Web Information Systems Engineering*, Springer, 2023, pp. 360–375. 29