Analysis and Improvement of Optimizer for Query Processing on Graph Store

Youyang Yao, Jiaqi Li, Rong Chen
Shanghai Key Laboratory of Scalable Computing and Systems
Institute of Parallel and Distributed Systems, Shanghai Jiao Tong University
Contact:rongchen@sjtu.edu.cn

Abstract

RDF systems are widely used to store public knowledge bases and process SPARQL queries. A large number of such systems have been proposed in the recent literature to provide low latency and high throughput for concurrent query processing over large RDF data. We perform an in-depth analysis on three key components (cardinality estimation, cost model, and plan enumeration) of the query optimizer to reveal the main issues and challenges on the accuracy and performance for traditional approaches. This calls for a rethink of how to build an accurate and fast query optimizer for modern RDF systems. We introduce a type-centric approach to enhance the accuracy of cardinality estimation prominently, which naturally embeds the lineage of correlated query conditions (triple patterns) into existing type system of RDF data. The preliminary results show that our approach greatly improves the accuracy of query optimization by several orders of magnitude compared to state-of-the-art approaches and provides a better overall performance by reducing execution time or optimization time.

Keywords

Optimizer, Type-centric Estimation, Graph Store, Query Processing

ACM Reference Format:
Youyang Yao, Jiaqi Li, Rong Chen. 2018. Analysis and Improvement of Optimizer for Query Processing on Graph Store. In *APSys '18: 9th Asia-Pacific Workshop on Systems (APSys '18), August 27–28, 2018, Jeju Island, Republic of Korea.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3265723.3265729

APSys '18, August 27–28, 2018, Jeju Island, Republic of Korea

ACM ISBN 978-1-4503-6006-7/18/08…$15.00
https://doi.org/10.1145/3265723.3265729

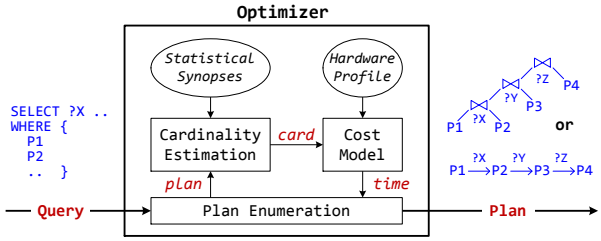

Fig. 1: The classical query optimizer architecture.

1 Introduction

Many public knowledge bases [1, 6, 7, 10, 15] are represented and stored as RDF (Resource Description Framework) graphs, where users can issue structured queries on such graphs using SPARQL. Much recent research [8, 16] has been devoted to developing scalable and high performance systems to process SPARQL queries on RDF data by using scan-join or graph-exploration approach.

The query optimization aims to find an optimal query plan such that the execution time is minimized, which is crucial for query performance. While this is one of the most studied problems in the database community, the schema-free nature of RDF data and the emerging graph-exploration scheme for query processing introduce new challenges to generate appropriate statistics, estimate pertinent results, and predict execution time accurately.

In this paper, we first investigate the three main components of the classical query optimizer architecture, as shown in Fig. 1, to reveal the main issues and challenges on the accuracy and performance for existing approaches. For cardinality estimation, the correlation among all of query conditions is crucial to the accuracy, and the statistics and estimation based on two correlated predicates [8, 11, 16] will incur tremendous error on the estimated results. For cost model, the mature model [8, 11] for scan-join approach does not apply to emerging graph-exploration approach, while the initial linear model [16] for graph-exploration approach is not competent for the performance prediction, even just on a single machine. For plan enumeration, with the performance improvement by leveraging advanced software and hardware techniques (e.g., full-history pruning and RDMA [13]),

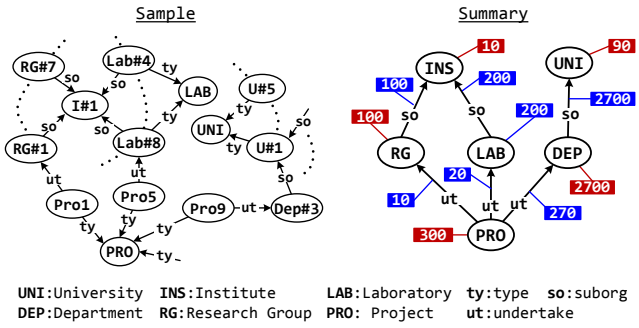

Fig. 2: A sample RDF graph (G).

the cost to enumerate the whole plan space may become a major part of the overall query time for selective (light) queries, even using dynamic programming.

We describe Wukong+P, an RDF/SPARQL-specific query optimizer for graph-exploration systems. It employs a novel type-centric approach to estimate the results (cardinality) of query accurately. Based on the observation that the same type of vertices commonly has a similar combination of predicates. Wukong+P embeds the lineage of correlated triple patterns explored so far into the predicate type and passes it on to the next graph exploration.

We have implemented a preliminary version of Wukong+P based on Wukong [13]. As an ongoing project, we mainly evaluate the accuracy of cardinality estimation for running a single query on LUBM [2] dataset. Compared to state-of-the-art approaches [8, 16], the preliminary results show that our approach can greatly enhance the accuracy by several orders of magnitude and improve the overall performance by up to 2.32X. The average gap between optimal and selected plans is limited to about 2%.

2 Background and Related Work

An RDF dataset is a graph (aka RDF graph) composed by triples, where a triple is formed by ⟨*subject*, *predicate*, *object*⟩. A triple can be regarded as a directed edge (*predicate*) connecting two vertices (from *subject* to *object*). Fig. 2 illustrates a part of the sample graph (G) and the summary of the whole graph. There are three categories of edges linking six types of vertices. SPARQL, a W3C recommendation, is the standard query language for RDF datasets. The major part of SPARQL queries is as follows:

Q := SELECT RD WHERE GP

where, GP is a set of *triple patterns* and RD is a *result description*. Each triple pattern (TP) is of the form ⟨*subject*, *predicate*, *object*⟩, where each of the subject, predicate and object may denote either a *variable* (e.g., ?X) or a *constant* (e.g., type). The result description contains a subset of variables in GP.

Given an RDF data graph, the SPARQL query searches on the graph for a set of subgraphs, each of which matches all

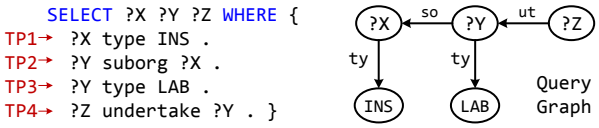

Fig. 3: A sample SPARQL query (Q).

triple patterns by binding pattern variables to values in the graph. For example, the query Q in Fig. 3 asks for all projects (?Z) that were undertaken (ut) by a laboratory which is a sub-organization (so) of an institute. The possible binding over the part of graph G in Fig. 2 is only Pro5. Note that the execution order of triple patterns (i.e., plan) will only impact the performance (much), not the result of the query. Therefore, the RDF system normally relies on the query optimizer to enumerate the valid execution orders to find an *optimal* query plan (i.e., minimal execution time) from *semantically equivalent* plan alternatives, such as PL1 and PL2 in Fig. 4. According to different plans, the optimizer estimates the *cardinality* (e.g., the number of intermediate results) of the plan based on the *statistical synopses* of RDF data and predicts the execution time of the plan with a cost model (see Fig. 1).

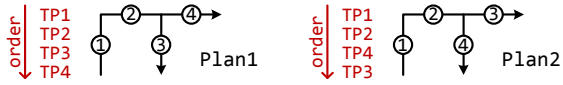

Fig. 4: Two sample plans (PL1 and PL2).

In existing state-of-the-art RDF systems, there exist two representative approaches to store RDF datasets and handle SPARQL queries: the scan-join mechanism on triple store [3, 8, 11] and the graph-exploration mechanism on graph store [4, 13, 16].

Scan-join approach. The scan-join systems, like RDF-3X [11] and TriAD [8] store RDF data as a set of triples in relational tables and leverage scan-join operations to process SPARQL queries. It entails a single range *scan* for each triple pattern to gain variable bindings and *joins* the related bindings of the individual patterns to generate the final results. For such systems, the query optimizer [8, 11] enumerates plans with different orders of join operations on triple patterns. For each plan, it estimates the size of intermediate results of join operations (i.e., the number of rows) as the cardinality and predicts the execution time based on the cost of different join operations (e.g., hash and merge join) and data transferring by network connections.

The RDF-specific statistical synopses mainly contain two types of statistics for scan and join operations respectively. First, the number of distinct 1-prefixes[1] (i.e., the number of triples with the same predicate) is used to estimate the

[1]For brevity, we skip the statistics for the number of distinct 2-prefixes, which demands incredible memory space .

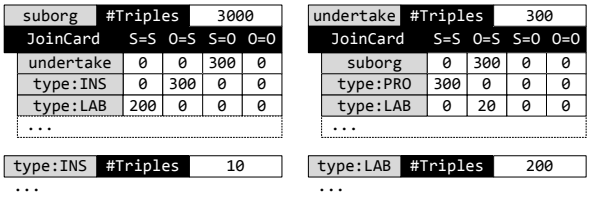

Fig. 5: A part of statistics used by scan-join systems.

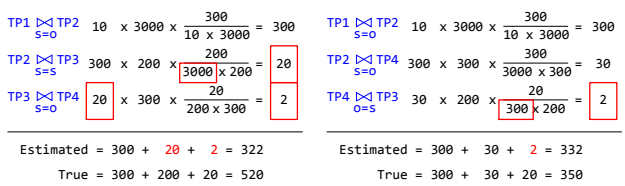

Fig. 6: The cardinality estimation on two plans in scan-join systems.

cardinality of a single triple pattern, namely the *scan* statistics. Second, the number of join partners (i.e., the number of triples for a predicate pair) is used to estimate the cardinality of the join condition on two triple patterns, namely the *join* statistics. Four cardinalities are precomputed for every predicate pairs joined on subjects or objects. Fig. 5 shows a part of statistics on the sample graph adopted by scan-join systems. For example, so's *scan* cardinality is 3000, which can be used to estimate the cardinality of TP2 ($\langle ?Y, suborg, ?X \rangle$) for Q. As only the objects of *so* triples and the subjects of ty_{INS} triples may combine together, the *so*'s *join* cardinality with ty_{INS} for $O = S$ is 300, and the rest are 0.

To estimate the cardinality of an entire query plan, the optimizer assumes *independence* among the join of triple patterns and estimates the cardinality of intermediate results gained from the join between two triple patterns *recursively*, which can be formalized as

$$Card(TP_{1,2}) = Card(TP_1) \times Card(TP_2) \times Sel(p_1, p_2, JP) \quad (1)$$

where $Card(TP_1)$ and $Card(TP_2)$ denote the cardinality of triple pattern TP_1 and TP_2, respectively. $Sel(p_1, p_2, JP)$ denotes the join selectivity of the predicate pair (p_1, p_2) associated with the triple patterns TP_1 and TP_2, which can be formalized as

$$Sel(p_1, p_2, JP) = \frac{JoinCard(p_1, p_2, JP)}{Card(p_1) \times Card(p_2)}$$

where $JoinCard(p_1, p_2, JP)$ denotes p_1's join cardinality with p_2 for the way of join partners (p_1, p_2) (e.g. $S = O$). For example (see Fig. 6), the cardinality of the first two triple patterns ($TP1$ and $TP2$) for PL1 is 300, where the cardinality of TP1 ($Card(TP_1)$) and TP2 ($Card(T_2)$) are 10 and 3000, and the join selectivity $Sel(type_{INS}, so, S = O)$ is $\frac{300}{10 \times 3000}$.

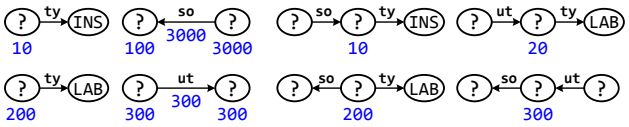

Fig. 7: A part of statistics used by graph-exploration systems.

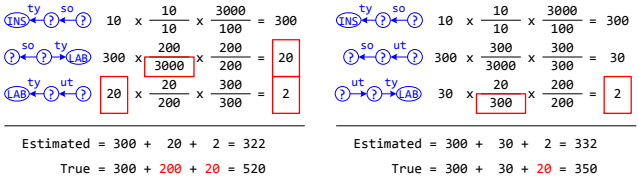

Fig. 8: The cardinality estimation on two plans in graph-exploration systems.

Based on Equation (1), the cost of an entire plan that is the sum of a specific order of triple patterns $\langle TP_1, ..., TP_N \rangle$. The cardinalities of every scan and join operations are fed into the cost model, which is briefly proportional to the cardinalities with different join operations and the cost of shipping intermediate results. The function parameters need to be measured for a given hardware setting. Finally, a bottom-up dynamic programming (DP) algorithm is used to determine the order of triple patterns that yield the minimal cost. At each DP step, the optimizer calculates the cost of the partial plan considered so far and prunes if the current branch cannot gain the minimal cost any more.

Graph-exploration approach. Instead of joining triple tables, graph exploration on the native graph form is proposed as a new primitive to process the SPARQL queries [4, 12, 13], which starts from a set of vertices and conducts a sequence of graph explorations to generate bindings for each triple pattern. The initial optimizer from Trinity.RDF [12] is inspired by the relational optimizer [11] for the scan-join approach, which models an execution plan as a graph traversal and focuses on the order of directed explorations via predicate (edge). The optimizer uses the size of intermediate results of graph exploration (i.e., the number of paths) as the cardinality and follows the proportional cost model and the DP-based plan enumeration.

The RDF-specific statistical synopses mainly contain two types of statistics associated with *predicates* for graph exploration. First, the optimizer precomputes the number of distinct subjects ($C_s(p)$), objects ($C_o(p)$), and triples ($C(p)$) for each predicate p. Second, the correlation $Cor(p_1, p_2, EP)$ between predicate pairs (p_1, p_2) is estimated to denote the number of distinct vertices with both two predicates (p_1 and p_2) as its incoming/outgoing edges (four combinations selected by EP). Fig. 7 illustrates a part of statistics on the sample graph (G) adopted by graph-exploration systems.

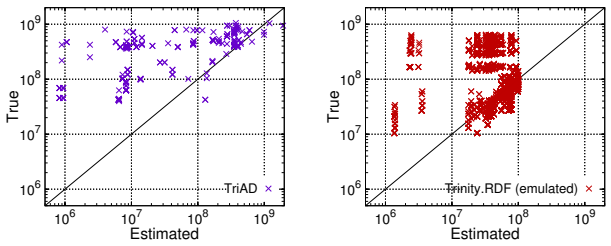

Fig. 9: A comparison btw. estimated and true cardinality for (a) TriAD and (b) Trinity.RDF using L1 on LUBM-2560.

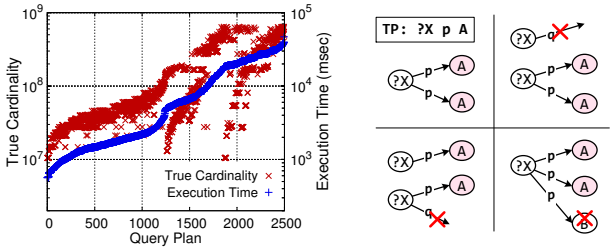

Fig. 10: (a) A comparison btw. true cardinality and execution time. (b) The different cases with the same cardinality.

The optimizer assumes the cardinality of an entire query plan as the sum of the cardinality of exploring triples with the predicate p, which is estimated as the size of the intermediate results ($R(p)$). Given the binding size of subjects (assume the source vertices) in a triple exploration ($B(s)$), $R(p)$ and $B(o)$ (assume the binding size of objects) can be formalized as

$$R(p) = B(s) \times \frac{C(p)}{C_s(p)},\ B(o) = B(s) \times \frac{C_o(p)}{C_s(p)}$$

Further, the binding size of $B(s)$ is affected by the last related triple pattern already explored. For exploring triples with the predicate p_2 from the triples with the predicate p_1, the correlated binding size $\acute{B}(v)$ can be predicted by the correlation statistics.

$$\acute{B}(v) = B(v) \times \frac{Cor(p_1, p_2, EP)}{C(p_1)}$$

For example (see Fig. 8), the cardinality of exploring the triple pattern $\langle ?Y, type, LAB \rangle$ in PL1 is 20, where the last binding size $B(?Y)$ is 300 and the correlation statistic $Cor(so, ty_{LAB}, \leftarrow\rightarrow)$ is 200. $C(so)$, $C(ty_{LAB})$, and $C_s(ty_{LAB})$ are 3000, 200, and 200 respectively.

Finally, a proportional cost model is adopted to predict the execution time, which is a linear combination of the cardinalities for computation cost and the size of bindings for communication cost. Besides, the dynamic programming algorithm is also used in plan enumeration.

3 Analysis of Query Optimization

This section presents an in-depth analysis on the accuracy and performance issues of query optimization.

Cardinality estimation. The main weakness of existing approaches is that they simply assume the independence among triple patterns and only consider at most two correlated predicates. However, we observe that *the correlation among all of triple patterns is crucial to the accuracy of cardinality estimation*. For example, consider two plans PL1 and PL2 in Fig. 4, the cardinalities in both scan-join and graph-exploration estimation are incorrect and reverses the relative result. The true cardinality of PL2 should be better. According to the statistical synopses, given 3000 vertices have the predicate so, only 200 of them also have the predicate ty_{LAB}. Consequently, given 300 vertices (?Y), the intermediate results becomes 20. But, the possibility of being a LAB is $\frac{200}{300}$, for the sub-organization of INS. Fig. 9 shows the true and estimated cardinality for different plans of a single query (L1) on LUBM-2560. The deviation in both two state-of-the-art systems, TriAD [8] (scan-join) and Trinity.RDF [8] (graph-exploration), is extremely high.[2] More importantly, this is a *fundamental* problem of approaches that estimate the cardinality based on the correlated predicates. A straightforward solution is to keep more statistics for three or more correlated predicates. However, the size of statistics would rapidly increase beyond the memory capability, and it is impossible to precompute all the dependencies between multiple predicates due to the schema-free nature of RDF data.

Cost model. The cost model for the scan-join approach has been well-studied, which thoroughly considers different join operations and the cost of data transferring. Unfortunately, it is quite difficult or impossible to attach scan-join's cost model to the graph-exploration approach directly. The initial cost model [12] for graph-exploration briefly uses a linear combination of the cardinalities for computation cost and the size of bindings for communication cost. To reveal the importance and necessity of an appropriate cost model, Fig. 10(a) compares the true cardinality and the execution time for the same plan. For brevity, we run the query (L1) on LUBM-2560 using a single machine to eliminate the communication cost. It is obvious that the cardinality alone is not sufficient to predict the execution time exactly. Fig. 10(b) further illustrate several cases with the same cardinality but different computation cost.

Plan enumeration. The optimization time (PLAN) has not received much attention since it is negligible compared to the execution time (EXE) in traditional systems like TriAD (see Table 1). However, with the dramatical performance

[2] Detailed experimental setup can be found in Section 6. Note that Fig. 9(a) only shows the results of the first 250 plans as the plan space of scan-join approach is extremely large.

Table 1: A comparison of optimization time and execution time (msec) for different queries on LUBM-2560.

	#TP	#PLAN	TriAD		Wukong	
			EXE	PLAN	EXE	PLAN
L1	6	2496	621	0.46	429	0.34
L2	2	4	128	0.18	102	0.01
L3	6	2496	413	0.44	629	0.31
L4	5	480	19.9	0.36	**0.06**	**0.09**
L5	2	2	2.6	0.17	0.04	0.01
L6	4	28	30.8	0.30	0.33	0.02
L7	6	2496	2,654	0.48	555	0.37

improvement by leveraging advanced software and hardware techniques, the optimization time may occupy a notable fraction of the overall query time for selective (light) queries (e.g., L4 in Table 1). Moreover, we observe that *the optimization time has nothing to do with the execution time but with the complexity (i.e., the number of triple patterns) of the query*. This is a typical *dilemma* that the heavy query demands an optimal plan by thorough optimizing but the light query demands an acceptable plan by efficient optimizing. Besides, the execution time is not known in advance.

4 Type-centric Estimation

Observations. To overcome the *fundamental* problem of approaches that estimate the cardinality based on the correlated predicates, the query optimizer needs to find an efficient way to embed the lineage of correlated triple patterns explored so far and pass it on to the next graph exploration. Fortunately, the predicate type is a perfect candidate. W3C has provided a set of unified vocabularies (as part of the RDF standard) to encode the rich semantics, where the predicate type (short for rdfs:type) provides a classification of vertices of an RDF graph into different groups. We observe that *the same type of vertices commonly has a similar combination of predicates*. For example, in Fig. 2, the instance of LAB has three predicates: so, ut, and ty_{LAB}. Therefore, the combination of predicates in triple patterns can be used to deduce the type of bindings, which may have multiple candidates with different probabilities. On the other hand, the combination of the type of bindings and the predicate in next triple pattern can also be used to deduce the number and the type of next bindings. Moreover, if the predicate type appears in the triple pattern of a given query, which is common (e.g., Q in Fig. 3), it will directly improve the overall accuracy of the estimation.

Type-based statistical synopses. Based on above observation, it makes sense that the type-centric approach uses the correlation statistics of type and predicates as the statistical synopses. Two kinds of type-based statistics are precomputed during loading RDF data: ***Type-formation statistics***: for each predicate p, Wukong+P precomputes the

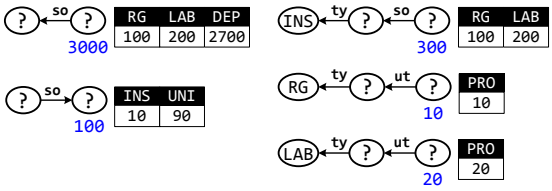

Fig. 11: A part of statistics used by Wukong+P.

type-formation array for the subjects/objects, namely $C_s(p)$/$C_o(p)$, which returns the type composition and the number of subjects/objects. In Fig. 11, $C_s(so)$ returns an array of the type composition and the number of subjects. The total number of subjects is 3000, where ty_{DEP} has 2700, ty_{LAB} has 200, and ty_{RG} has 100. These statistics are usually used to explore the first triple pattern of the plan. ***Type-derivation statistics***: for each correlated predicate p and *type*, Wukong+P precomputes the *type-derivation* array for the subjects/objects, namely $C_s(type, p)$/$C_o(type, p)$, which returns the type composition and the number of subjects/objects, when exploring the vertex (object/subject) with a deduced *type* along with a given predicate p. In Fig. 11, $C_s(ty_{INS}, so)$ returns an array of the type composition and the number of subjects. The total number of subjects is 300, where ty_{RG} has 100 and ty_{LAB} has 200. These statistics are used to explore a triple pattern with a given predicate p from vertices with a deduced *type*.

Type-centric cardinality estimation. Wukong+P still uses the size of intermediate results of graph exploration (i.e., the number of paths) as the cardinality and estimate the cardinality for each exploration along with triple patterns of the plan, which can be formalized as

$$Card_{res}(TP_k) = \sum_{i=1}^{n} \{Card_{ty}(ty_{(k-1,i)}) \times \frac{\sum C_{s/o}(ty_{(k-1,i)},\ p_k)[j]}{C_s(ty_{(k-1,i)})}\} \quad (2)$$

$$= \sum_{i=1}^{n} Card_{ty}(ty_{(k,i)}) \quad (3)$$

$$Card_{res}(start) = \begin{cases} \sum_{i=1}^{n} C_s(p_1)[i] & start == s_1 \\ \sum_{i=1}^{n} C_o(p_1)[i] & start == o_1 \\ 1 & start == const \end{cases} \quad (4)$$

where $Card_{res}(TP_k)$ denotes the result cardinality of the kth triple pattern, and $Card_{ty}(ty_{(k,i)})$ denotes the cardinality of the ith type in the result of the kth exploration. $C_s(ty_{(k-1,i)})$ denotes the number of vertices having such type from statistics.

For the first triple pattern of the plan, there exist two kinds of start points we can choose. For starting from a variable, the cardinality is estimated using Equation (4) by looking up the *type-formation statistics* ($C_s(p)$ or $C_o(p)$). For starting from a constant, the type of the constant is required for

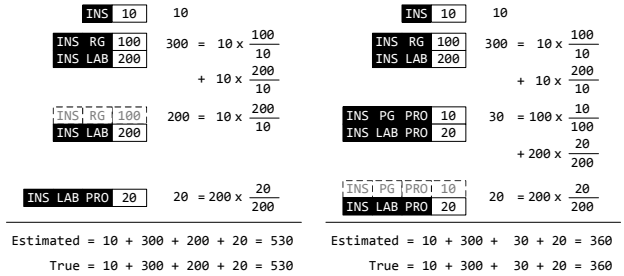

Fig. 12: The estimated and true results of two plans in Wukong+P.

the further estimation, and can be obtained from RDF data. For the following exploration, the *type-derivation statistics* is used to estimate the result cardinality of the next bindings (see Equation (2)).

Intuitively, $\frac{C_{s/o}(ty_{(k-1,i)},\ p_k)[j]}{C_s(ty_{(k-1,i)})}$ estimates the number of bindings with the type in $C_{s/o}(ty_{(k-1,i)},\ p_k)[j]$ explored from the vertices with the type $ty_{(k-1,i)}$. By multiplying the cardinality of the source vertex with the type $ty_{(k-1,i)}$, we can derive the type cardinality of the target vertex with the specific type in $C_{s/o}(ty_{(k-1,i)}, p_k)[j]$. Finally, the Equation (2) can be deduced to the Equation (3) by combining the bindings with the same type.

Fig. 12 shows an example for the cardinality estimation in Wukong+P. For PL1, the cardinality of ?X in the first exploration could be obtained directly from $C_s(ty_{INS})$. Next, the triples with the predicate *so* are explored from ?X, the result cardinality in this exploration is able to estimate using Equation (2) based on *type-derivation statistics* from Fig. 11. The total number of ty_{INS} is 10. The *type-derivation statistics* show that the total number of ty_{RG} and ty_{LAB} derived from ty_{INS} along with *so* edge is 300. Thus, each ty_{INS} has $\frac{100}{10}$ vertices with the type ty_{RG} and $\frac{200}{10}$ vertices with the type ty_{LAB}. Therefore, the result cardinality in this exploration is 300 using Equation (3) to combine the type cardinalities. In the third exploration, all ty_{RG} is pruned due to not matching the triple pattern. In the last exploration, the predicate *ut* is explored and the final result cardinality is pruned to 20.

As we can see, the third exploration in PL1 drops the 100 of 300 results, which matches with the case in real query processing. But both scan-join and graph-exploration systems severe under-estimate the prune ratio as $\frac{200}{3000}$. This is because Wukong+P embed the dependencies into the predicate *type*, while existing systems only consider two correlated predicates.

No type and multiple types. In rare cases, some vertices may have no *type*. Wukong+P will assign a new virtual type to them according to the combination of their predicates. On the other hand, some vertices may have multiple *types*. Wukong+P will also assign a new virtual type to them according to the combination of their *types*. The virtual type will be calculated separately, except that the type cardinality of this virtual type will not be pruned if the given *type* in the triple pattern belongs to the virtual type.

Table 2: A summary of RDF datasets.

Dataset	#Triples	#Subjects	#Objects	#Predicates
LUBM-2560	352 M	55 M	41 M	17
LUBM-10240	1,410 M	222 M	165 M	17

5 Future Work

Wukong+P is still an ongoing project, we plan to solve the following issues in future:

Cost model. We need a new cost model for graph-exploration systems to consider new features [13], including key-value lookups, full-history pruning, in-place and fork-join execution mode. Inspired by symbolic execution, we plan to feed the complete history of estimated cardinalities and other information (e.g., path pruning) as the history table (intermediate results) into a mimic graph exploration, which provides the detailed behavior of query processing. Furthermore, unlike prior work [12], Wukong+P expects to predict an absolute execution time instead of a relative number. Thus, we will make a tool to measure (only once) the parameters of the cost model for a given hardware and benchmark setting.

Plan enumeration. As shown in Table 1, the optimization time may become the bottleneck for light queries (e.g., L4). Currently, Wukong+P will try to estimate one plan selected by some heuristics (e.g., start from the constant of a triple pattern) first and then use a fixed threshold to classify the query into two kinds, heavy or light. Wukong+P will enumerate all of the plans to find the optimal one for heavy queries and directly choose the first plan for light queries. We plan to use a budget-based mechanism for plan enumeration. The optimization budget is the user-defined rate of the least execution time predicted so far. Wukong+P will enumerate and predict plans till the budget runs out or the optimal plan has been found.

6 Preliminary Results

Experimental Setup. All evaluations were conducted on a 24-core Intel machine, which has two 12-core Xeon E5-2650 v4 processors and 128GB DRAM. We dedicate one processor to run up to 10 worker threads, and use a single thread to generate requests and perform query optimization. We use 2 datasets of Leigh University Benchmark (LUBM) [2] (see Table 2) and the queries were widely used by many RDF systems [5, 8, 13, 14, 16, 17]. We compare the accuracy and

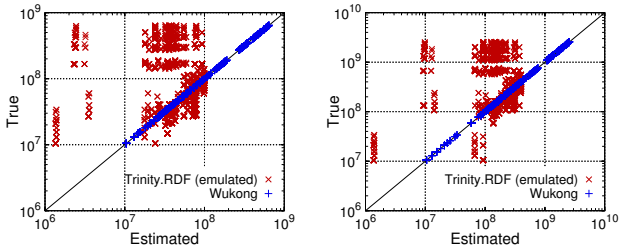

Fig. 13: A comparison of cardinality estimation between Wukong+P and Trinity.RDF using L1 on (a) LUBM-2560 and (b) LUBM-10240.

Table 3: Q-error for cardinality estimation.

LUBM-2560	Median	90th	95th	MAX
TriAD	6.810	56.584	81.099	474.262
Trinity.RDF	1.785	12.928	18.755	255.952
Wukong+P	1.001	1.002	1.002	1.004

performance of Wukong+P against two state-of-the-art approaches, TriAD [8] and Trinity.RDF [12]. Since the source code of Trinity.RDF is not available, we implemented its query optimizer on Wukong [13]. To measure the accuracy of cardinality estimation across different systems, we use the *q-error* [9], which denotes the factor by which an estimated cardinality differs from the true one. The closer the *q-error* is to 1, and the more accurate the optimizer is.

Cardinality estimation. We first compare the estimated cardinality between Trinity.RDF and Wukong+P using all plans (2496) of L1 on LUBM-2560 and LUBM-10240. The sum of the intermediate results in each graph exploration is recorded as the true cardinality. As shown in Fig. 13, the estimated cardinalities in Trinity.RDF extremely deviate from the ideal line, where the estimated and true cardinality are equal. The estimated cardinalities of Wukong+P are very close to the ideal result on both two datasets, which confirms the benefits of the type-centric approach. Table 3 shows the median (50th), 90th, 95th and max (100th) percentiles of the *q-error* for the cardinality estimation using all plans of L1 on LUBM-2560. Wukong+P can achieve negligible errors thanks to the type-centric approach. The max *q-error* is still quite close to the optimal value. For TriAD and Trinity.RDF, both of them produce very high *q-error* (up to 474X deviation).

Query performance. To further study the effectiveness of Wukong+P, we compare the selected and optimal execution time using LUBM-2560. For Wukong, we execute all possible plans for all queries to find the optimal plan (OPT) with the least execution time. Note that we can not achieve the optimal plan for TriAD since it is not very stable for repeated execution. As shown in Table 4, Wukong+P can choose the optimal or near-optimal plan in most cases. For L3 and L7,

Table 4: A comparison of optimal and selected performance (msec) on LUBM-2560. EXE|PLAN denote the execution and optimization time. OSDI16 denotes the execution time published in original Wukong paper [13].

	TriAD	Wukong			
		OPT	OSDI16	Trinity.RDF	Wukong+P
L1	621|0.46	427	754	429|0.34	**428**|1.42
L2	128|0.18	102	103	102|0.01	**102**|0.01
L3	413|0.44	270	274	629|0.31	**270**|1.06
L4	19.9|0.36	0.06	0.09	0.06|0.09	0.06|**0.01**
L5	2.6|0.17	0.04	0.05	0.04|0.01	0.04|**0.01**
L6	30.8|0.30	0.26	0.29	0.33|0.02	0.29|**0.01**
L7	2,654|0.48	288	608	555|0.37	**299**|0.84
GM	104.8|0.32	8.01	10.78	10.28|0.07	8.18|0.07

Wukong+P outperforms Trinity.RDF due to the accurate cardinality estimation. For L1, Trinity.RDF can also choose the near-optimal plan even using a very inaccurate estimator, since the minimal cost plan coincidentally has low execution time. However, the second minimal cost plan has nearly 3X slowdown performance. For light queries (L4, L5 and L6), Wukong+P can also provide comparable performance with near-optimal plans even only enumerate a single plan, thanks to the heuristic mechanism. As expected, the plan time is much smaller than that of Trinity.RDF. Therefore, the overall performance in Wukong+P is still better than Trinity.RDF.

Comparing with the number in original Wukong paper [13] (OSDI16), which uses a heuristic method to choose the plan manually, Wukong+P can still outperform it, especially for heavy queries (e.g., L1 and L7), which have large plan space. Hence, the statistic-based optimizer is critical to improve the performance for RDF systems.

7 Conclusion

This paper proposes a novel type-centric approach for the cardinality estimation, which embeds the lineage of correlated query conditions into the predicate type. The preliminary results show that Wukong+P can improve the accuracy of query optimization by several orders of magnitude compared to state-of-the-art approaches.

Acknowledgments

We sincerely thank the anonymous reviewers for their insightful suggestions and Kai Zeng for sharing his experience to design and implement optimizer for Trinity.RDF. This work is supported in part by the National Natural Science Foundation of China (No. 61772335), the National Key Research & Development Program (No. 2016YFB1000500).

References

[1] DBpedia's SPARQL Benchmark. http://aksw.org/Projects/DBPSB.

[2] SWAT Projects - the Lehigh University Benchmark (LUBM). http://swat.cse.lehigh.edu/projects/lubm/.

[3] Abadi, D. J., Marcus, A., Madden, S. R., and Hollenbach, K. Sw-store: a vertically partitioned dbms for semantic web data management. *Proc. VLDB Endow.* (2009).

[4] Abdelaziz, I., Harbi, R., Salihoglu, S., Kalnis, P., and Mamoulis, N. Spartex: A vertex-centric framework for rdf data analytics. *Proc. VLDB Endow. 8*, 12 (Aug. 2015), 1880–1883.

[5] Atre, M., Chaoji, V., Zaki, M. J., and Hendler, J. A. Matrix "bit" loaded: A scalable lightweight join query processor for rdf data. In *Proc. WWW* (2010).

[6] Bio2RDF Consortium. Bio2rdf: Linked data for the life science. http://bio2rdf.org/, 2014.

[7] Google Inc. Introducing the knowledge graph: things, not strings. https://googleblog.blogspot.co.uk/2012/05/introducing-knowledge-graph-things-not.html, 2012.

[8] Gurajada, S., Seufert, S., Miliaraki, I., and Theobald, M. Triad: A distributed shared-nothing rdf engine based on asynchronous message passing. In *Proc. SIGMOD* (2014).

[9] Leis, V., Gubichev, A., Mirchev, A., Boncz, P., Kemper, A., and Neumann, T. How good are query optimizers, really? *Proc. VLDB Endow. 9*, 3 (Nov. 2015), 204–215.

[10] National Center for Biotechnology Information. Pubchemrdf. https://pubchem.ncbi.nlm.nih.gov/rdf/, 2014.

[11] Neumann, T., and Weikum, G. Rdf-3x: A risc-style engine for rdf. *Proc. VLDB Endow.* (2008).

[12] Shao, B., Wang, H., and Li, Y. Trinity: A distributed graph engine on a memory cloud. In *Proc. SIGMOD* (2013).

[13] Shi, J., Yao, Y., Chen, R., Chen, H., and Li, F. Fast and concurrent rdf queries with rdma-based distributed graph exploration. In *Proc. OSDI* (2016).

[14] Wang, S., Lou, C., Chen, R., and Chen, H. Fast and concurrent rdf queries using rdma-assisted gpu graph exploration. In *Proc. USENIX ATC* (2018).

[15] Wu, W., Li, H., Wang, H., and Zhu, K. Q. Probase: A probabilistic taxonomy for text understanding. In *Proc. SIGMOD* (2012), pp. 481–492.

[16] Zeng, K., Yang, J., Wang, H., Shao, B., and Wang, Z. A distributed graph engine for web scale rdf data. In *Proc. VLDB* (2013).

[17] Zhang, Y., Chen, R., and Chen, H. Sub-millisecond stateful stream querying over fast-evolving linked data. In *Proc. SOSP* (2017).