Knowledge Graphs in the Age of Language Models and Neuro-Symbolic AI
A. Salatino et al. (Eds.)

doi:10.3233/SSW240022

Enabling Delayed-Answer Auctions for RDF Knowledge Graphs Monetisation

Hala SKAF-MOLLI [a,1], Pascal MOLLI [a], Luis-Daniel IBÁÑEZ [b] and Abraham BERNSTEIN [c]
[a] *University of Nantes, CNRS, LS2N, Nantes, France*
[b] *Department of Electronics and Computer Science, University of Southampton, UK*
[c] *Department of Informatics, University of Zürich, Switzerland*
ORCiD ID: Hala Skaf-Molli https://orcid.org/0000-0003-1062-6659, Pascal Molli https://orcid.org/0000-0001-8048-273X, Luis-Daniel Ibáñez https://orcid.org/0000-0001-6993-0001, Abraham Bernstein https://orcid.org/0000-0002-0128-4602

Abstract. Traditionally, querying knowledge graphs is free of charge. However, ensuring data and service availability incurs costs to knowledge graphs providers. The Delayed-Answer Auction (DAA) model has been proposed to fund the maintenance of knowledge graph endpoints by allowing customers to sponsor entities in the Knowledge Graph so query results that include them are delivered in priority. However, implementing DAA with time-to-first results acceptable for data consumers is challenging because it requires reordering results according to bid values. In this paper, we present the AuctionKG approach to enable DAA with a low impact on query execution performance. AuctionKG relies on (i) reindexing sponsored entities by bid values to ensure they are processed first and (ii) Web preemption to ensure delayed answering. Experimental results demonstrate that our approach outperforms a baseline approach to enable DAA in terms of time for first results.

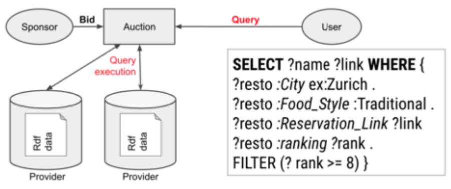
Keywords. Knowledge Graphs, Auction model, Web preemption, Monetization.

1. Introduction

Many public RDF knowledge graphs are published on the Web following the Linked Data principles [1,2]. A subset of these knowledge graphs are interconnected, as numerous IRI identifiers found in one graph also appear in multiple others [3]. These interconnected knowledge graphs form the Linked Open Data Cloud, also known as the Web of Data (WoD). The WoD can be seen as a big decentralized knowledge graph on the Web. As of September 2023, the Linked Open Data Cloud contains 1314 RDF datasets and billions of RDF triples [2].

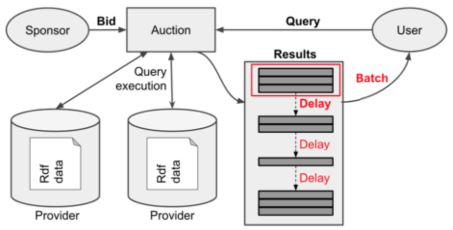
Large public knowledge graphs provide a free query SPARQL interface (a ”SPARQL” endpoint), e.g., Wikidata [4] and DBpedia [5]. Albeit free, access to these knowledge

[1]Corresponding Author: Hala Skaf-Molli, Hala.Skaf@univ-nantes.fr
[2]https://lod-cloud.net/

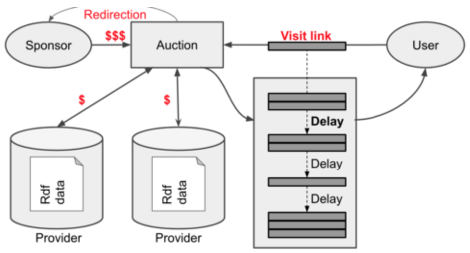


```
SELECT ?name ?link WHERE {
?resto :City ex:Zurich .
?resto :Food_Style :Traditional .
?resto :Reservation_Link ?link
?resto :ranking ?rank .
FILTER (? rank >= 8) }
```

(a) A user wants to reserve a restaurant offering traditional food with good rating



(b) A user gets a delayed answer based on bids of sponsors



(c) A user visiting link, the sponsor for that link pays the auction which distributes money among providers

Figure 1. The Delayed Auction Model as described in [6]

graphs incurs costs for the service provider. As public knowledge graphs do not generate revenue yet, the economic sustainability of such services is largely dependent on government funding. To scale the WoD vision, knowledge producers and curators must have the economic incentives to undertake the task of publishing on the Web of Data.

One possible solution is to apply the techniques that finance the World Wide Web (WWW) to the WoD. One of the biggest financial motors of the WWW is advertisement [6]. Unfortunately, the WWW advertisement techniques cannot work on the WoD, where software agents, rather than humans, query the machine-readable data and further process it on behalf of one or more human users. This means data providers cannot enforce how adverts are shown to humans, if at all.

In [6], the authors propose the Delayed-Answer Auction (DAA) model to answer the question: *How can the WoD be free and financially sustainable at the same time?*. Figure 1 illustrates the DAA model: data providers allow organizations to sponsor IRIs of entities in their datasets. When a user asks a query to the federation of data providers (Figure 1a) the auction mechanism guarantees that solutions featuring IRIs of entities with higher bids are returned earlier than those with lower bids (Figure 1b). Query results are split into N batches delivered with delays $t_1 \ldots t_N$ between them, auctioning the placement of results in the first batch to the highest bidders. Like a user clicking on a link on a Web Page, a user (or an agent acting on behalf of a user) can select an IRI from a query response to explore. If a user or agent dereferences an IRI in the query results, the sponsor of that entity pays the amount they bid, which is then distributed among data providers who contributed to the query (Figure 1c).

The foundational paper on the DAA model concentrates on its economic aspects and merely suggests a basic method to implement the DAA model: order solutions by the value of the bids on the entities they contain and then split in batches. This has the fundamental shortcoming of requiring a ‘ORDER-BY bid value’ clause to calculate the complete query answer before ordering results, leading to two practical consequences (i) potentially slowing down query processing, affecting revenue (ii) As users are likely to not consume all query results, a substantial amount of the computational work expended in producing the complete query answer is wasted.

In this paper, we introduce a novel method, AuctionKG, to tackle these challenges. AuctionKG leverages the concept of Web Preemption (a technique for suspending query execution, cf. Section 3) to support delayed query answering and re-indexes sponsored entities during bidding to ensure correct ordering when queries are executed. We claim the following contributions:

1. An efficient strategy for query execution in the context of a DAA model.
2. An experimental study comparing the time for the first results of our new method against the baseline approach suggested in [6] on a subset of the WatDiv query benchmark [7]. Our experiments show that AuctionKG is four times faster than the baseline.
3. We compare the performance difference of our method depending on the DAA batch type: fixed size versus fixed waiting time.

The rest of the paper is organized as follows: Section 2 provides background and motivation. Section 3 outlines our approach and highlights its benefits. Section 4 details our experimental study. Section 5 reviews related works. Finally, Section 6 presents conclusions and discusses future work.

2. Background and Problem Statement

Following [6], we distinguish the following actors (cf Figure 1a):

Data providers host RDF knowledge graphs and allow exclusive access to the auctioneer to manage bids.

Auctioneer hosts a SPARQL endpoint providing access to data from Data Providers and implements an auction model, *e.g.*, DAA.

Users submit SPARQL queries to the auctioneer’s endpoint.

Sponsors submit sponsorship bids to the auctioneer for entities hosted by data providers, expecting that if the bid is high enough, a result containing a sponsored entity will be delivered to users in the first batch of results.

The auctioneer acts as an intermediary between providers, users, and sponsors. It handles bids from sponsors, processes SPARQL queries from users, and delivers answers in *batches* according to the chosen auction model and current bids.

Definition 1 (Delayed Answer Auction) *Let a pair* (e,v) *be a* bid *where* e *is an RDF resource and* v *is a positive real number representing the bid’s value. Given a set of RDF Datasets* G*, a number of batches* $N > 0$*, a set of bids* S *and a SPARQL query* Q *compute the sequence of* batches *of the evaluation of* Q *on* G*,* $[\![Q]\!]_G$*:*

$$[(B_0,0),(B_1,d_1),\ldots,(B_N,d_N)]$$

```
select ?resto where {
  ?resto :type :restaurant .
  ?resto :City ex:Paris .
  ?resto :rating "A"
}
```

(a) Q_1: Query submitted by a user

```
select coalesce(?id, ?resto) where {
  ?resto :type :restaurant .
  ?resto :City ex:Paris .
  ?resto :rating "A" .
  optional {
    ?resto auction:bid ?bid .
    ?resto owl:sameAs ?id
  }
} order by ?bid
```

(b) $Bid_{?resto}(Q_1)$

Figure 2. (a) Example SPARQL query submitted by a user Q_1 (b) $Bid_{?resto}(Q_1)$: a rewriting of query Q_1 to ensure order by bid value on sponsored bindings of variable *?resto*

where $\{B_0, B_1 ..., B_N\}$ *is a partition of size N of* $[\![Q]\!]_{\mathscr{G}}$*, and* $\{d_1, ..., d_N\} > 0$ *are times measured from zero representing the delivery delay of each batch,* i.e., B_0 *is delivered at time 0,* B_1 *at time* d_1 *and so on.*

2.1. Delayed Answer Auction with Query Rewriting

A straightforward yet naive approach to implementing a DAA is to rewrite users' queries to add an *Order By* clause to ensure that sponsored entities are prioritized over non-sponsored ones, and they are delivered in bid value order.

To illustrate, consider a scenario where several data providers host and curate information about restaurants in Paris, including type of cuisine and rating, in RDF. These providers form a federation, exclusively accessed through an Auctioneer's SPARQL endpoint. Consider the following RDF triples in the Auctioneer's Knowledge Graph:

```
@prefix example: <http://example.org/>
@prefix auction: <http://auction-vocab.org/>
@prefix auctioneer: <http://auctioneer-domain.com/>

example:MyRestaurant  auction:bid 500 ;
                auction:sponsor example:sponsor1 ;
                owl:sameAs auctioneer:sponsored_resto_1 .
```

Meaning that the entity `example:MyRestaurant` has a bid of 500 units placed by sponsor `example:sponsor1`. The sameAs to the IRI `auctioneer:sponsored_resto_1` ensures that when a user consumes a result, she is redirected to the auctioneer, which proves to the corresponding Data Provider that data was accessed through it[3].

Suppose the query Q_1 of Figure 2a. Q_1 retrieves Restaurants in Paris having an A rating. To handle the DAA model, Q_1 is rewritten as $Bid_{?resto}(Q_1)$ where `?resto` is the bid variable as described in Figure 2b. $Bid_{?resto}(Q_1)$ introduces 3 additional clauses:

OPTIONAL Retrieves the bid identifier and amount for each entity if existing.

COALESCE Ensure the return of the auctioneer corresponding to a sponsored entity when it exists. If it doesn't, the entity is not currently sponsored, and its original IRI is returned.

[3]For simplicity, we store bid information in the same named graph as the base data. This has the security issue of allowing users to query bid amounts in an attempt to game the auction. To avoid that, bid data could be stored in a different named graph, or queries involving auction related predicates forbidden.

Table 1. A sample RDF Dataset of restaurants, their locations, rankings, and reservation links

S	P	O
L'Ardoise	City	Paris
L'Ardoise	Ranking	A
L'Ardoise	rdf:type	Restaurant
Chez Louis	City	Paris
Chez Louis	Ranking	A
Chez Louis	rdf:type	Restaurant
Schweizer Schwein	City	Zurich
Schweizer Schwein	Ranking	B
Schweizer Schwein	rdf:type	Restaurant

Table 2. SPO, POS, and OSP indexes of RDF Dataset in Table 1

S	P	O
Chez Louis	City	Paris
Chez Louis	Rating	A
Chez Louis	Type	Restaurant
L'Ardoise	City	Paris
L'Ardoise	Rating	A
L'Ardoise	Type	Restaurant
Schweizer Schwein	City	Zurich
Schweizer Schwein	Rating	B
Schweizer Schwein	Type	Restaurant

P	O	S
City	Paris	Chez Louis
City	Paris	L'Ardoise
City	Zurich	Schweizer Schwein
Rating	A	Chez Louis
Rating	A	L'Ardoise
Rating	B	Schweizer Schwein
Type	Restaurant	Chez Louis
Type	Restaurant	L'Ardoise
Type	Restaurant	Schweizer Schwein

O	S	P
A	L'Ardoise	Ranking
A	Chez Louis	Ranking
B	Schweizer Schwein	Ranking
Paris	Chez Louis	City
Paris	L'Ardoise	City
Restaurant	Chez Louis	Type
Restaurant	L'Ardoise	Type
Restaurant	Schweizer Schwein	Type
Zurich	Schweizer Schwein	City

ORDER BY orders the results by bid values.

In many scenarios, it is expected that only a small amount of results will be consumed. Suppose a user retrieves 20 results of query $Q1$ on a SPARQL endpoint. The execution time of finding 20 results of query $Q1$ is less or equal to the execution time of $Q1$. If the user now retrieves 20 results of query $Q1$ on an auctioneer's SPARQL endpoint, $Q1$ is rewritten into $Bid_{?resto}(Q_1)$, and the 20 results should be ordered by bid values. In current SPARQL engines, the `OrderBy` clause requires computing the complete answer before delivering the first 20 results, yielding an execution time to obtain the first 20 results of $Q1$ greater than the execution time of $Q1$. In our experimental study (section 4), we found that on average, retrieving 20 results of Q_i is at least 10 times faster than retrieving 20 results of $Bid_v(Q_i)$.

According to controlled experiments made in [8], if the time to display search results is increased by 500ms, the revenue is reduced by 20%, making the naive solution unfeasible for commercial applications. In consequence, implementing the DAA model on current SPARQL endpoints seriously degrades the performance of users' queries. Ideally, we want the DAA model to not degrade the query performance leading to the following problem statement:

Problem 1 *Given a query Q, the problem is to reduce the execution time to deliver k results of Q with respect to the execution time to deliver k results of $Bid_v(Q)$.*

3. AuctionKG Approach

We focus on SPARQL conjunctive queries where entities may be sponsored following a DAA model. We suppose that at least one projected variable returns the URL of entities that may be sponsored. We call this projected variable the *bid variable*.

```
select ?city ?resto where {
  ?resto :type :restaurant . #tp1
  ?resto :City ?city .  # tp2
  ?resto :rating "A" . #tp3
  ?city :country :France #tp4
}
```

(a) Q_3: Query submitted by a user

```
select ?city ?resto where {
  ?resto :type :restaurant . #tp1
  ?resto :City ?city .  # tp2
  ?resto :rating "A" . #tp3
  ?city :country :France #tp4
  optional {
    ?city auction:bid ?bid .
    ?city owl:sameAs ?id }
} order by ?bid
```

(b) $Bid_{?city}(Q_3)$

Figure 3. Illustration of a query that requires forcing join order. If ?city is the bid variable, our approach requires the $tp4$ to be the first pattern evaluated to preserve order.

Given a conjunctive query Q and a bid variable v, our baseline is a bid query noted $Bid_v(Q)$, where Q is rewritten with an `optional` statement to retrieve biddings and an `order by` statement to return results following the bid order as presented in Figure 2b. The main issue with $Bid_v(Q)$, is that the time to deliver the first results of $Bid_v(Q)$ is the same as the time to compute its complete answer.

To tackle this problem, we followed a simple key idea. Given a conjunctive query Q and a bid variable v, we forced the join order of triple patterns of Q to start with a triple pattern containing v. Next, we ensured that the bid variables to return mappings following the bid order using a dedicated indexing scheme. Finally, to enable delays between ordered results, we followed the Web Preemption principle[9] that allows to suspend query execution after a quantum of time for resuming it later.

To illustrate, consider the query $Bid_{?resto}(Q_1)$ of Figure 2b, `?resto` is the bid variable. As all triple patterns of Q_1 contains the bid variable, and the join order of triple patterns of Q_1 is ensured to start with a triple pattern containing the bid variable. In this case, if all mappings of `?resto` respect the bid order then, query results follow naturally the bid order.

To illustrate a more complex example, suppose the query Q_3, on Figure 3b, that asks for restaurants located in a French city. Suppose the bid variable is ?city, *i.e.*, cities are sponsored, and we want results to be ordered by the values of bids on cities. In this case, our approach will force the join order of Q_3 triple patterns to start with $tp4$, *e.g* $tp4 \rightarrow tp2 \rightarrow tp3 \rightarrow tp1$.

To make this key idea work, we need to answer two questions:

1. How to ensure that any triple pattern returns its mapping following the bid order? We achieve that with a dedicated indexing scheme.
2. How to delay next results after delivering the k first (sponnsored) results? We achieve delayed results by relying on the Web Preemption principle[10].

Note that forcing join ordering on the bid variable may lead to suboptimal query execution plans, hence, we empirically measure this tradeoff in our experiments.

3.1. Re-indexing sponsored entities

Avoiding rewriting requires a SPARQL engine that ensures the *bid-aware* evaluation.

Definition 2 (bid-aware Evaluation) *Given a SPARQL query Q, a bid-aware query engine generates sponsored entities in the results of Q first and in decreasing order of bid values.*

To ensure bid-aware evaluation, when processing a SPARQL query Q, a query engine must begin by evaluating a triple pattern $TP \in Q$ that contains a sponsored result, prioritizing sponsored entities based on their bid values. It is generally challenging to guarantee this property [11]; however, with known sponsored entities, as in AuctionKG, it is feasible to initiate query execution with a triple pattern that includes a sponsored entity by enforcing the join order following the method suggested in [12].

We propose re-indexing sponsored entities at the bidding time of the auction process and forcing the join ordering to ensure bid-aware evaluation. Re-indexing ensures that sponsored entities with the highest bids are scanned first, and the enforced join orders ensure that the query engine starts the evaluation with a triple pattern encompassing sponsored entities in its results.

For an RDF graph composed of a set of RDF triples of the form <S, P, O>, AuctionKG requires four indexes SPO, POS, OSP, and PSO to ensure the property 2.

For simplicity, we present in Table 2 three indexes: SPO, POS, and OSP of the sample dataset in Table 1 (we omit prefixes for simplicity). In simple words, the re-index process **renames** entities in the index so they move to the top. It is composed of the following steps:

1. Craft a hyperlink with the auctionIRI of the auction and the bid's value. Recall from Section 2.1 that auctioneers deliver hyperlinks that redirect to them to keep track of result consumption. The bid's value is included to ensure ordering. To lexicographically order numerical bid values, a simple yet sufficient approach is to prepend to the string representation of the numerical value an alphabetical string encoding the number of digits of the bid (Z = 1 digit, Y = 2 digits, and so on, the reverse order ensures that lexicographical order returns larger numerical bids first)[4]. For instance, following from the example in Table 1, if a sponsor bids 40 units for entity `L'Ardoise` and another sponsor bids 100 units for entity `Chez Louis`, example IRIs are *http://auctioneer/aaa-Y-40-auction-entity1* and *http://auctioneer/aaa-X-100-auction-entity2*, respectively. The prefix *aaa* ensures sponsored identities are placed at the top of the index.
2. Replace the original entityIRI of the sponsored entity with the auctionIRI crafted in the previous step.
3. Insert a triple *entityIRI owl:sameAs auctionIRI* to ensure the original entity IRI can be returned upon dereference of auctionIRI by a user. Following our example, we need to state that *:aaa-Y-40-auction-entity1* is the same as *:L'Ardoise*.

We implemented the steps above as a single SPARQL Update query, for which an instance for entity `L'Ardoise` is shown in Figure 4.

Tables 2 show the index state before renaming. Table 3 shows the index of Table 1 after indexation with the query on Figure 4. The entityIRI `L'Ardoise` has been replaced with auctionIRI `aaa-Y-40-auction-entity1`. When receiving a query like the one in Figure 2a, reindexing ensures the property 2.

[4]Other encodings are possible, for example number-based: 100 = 1 digit, 010 = 2 digits, 001 = 3 digits, etc.

```
INSERT {
   auctioneer:aaa-Y-40-auction-entity1 ?p1 ?o1;
      auction:bid 40 ;
      auction:sponsor auctioneer:sponsorID ;
      owl:sameAs example:L'Ardoise .
   ?s2 ?p2 auctioneer:aaa-Y-40-auction-entity1 .
}

DELETE {
   example:L'Ardoise ?p1 ?o1 .
   ?s2 ?p2 example:L'Ardoise .
}

WHERE {
   example:L'Ardoise ?p1 ?o1 .
   ?s2 ?p2 example:L'Ardoise .
}
```

Figure 4. Example of Reindexing SPARQL Update query after a sponsorship bid. Prefixes omitted for brevity.

Table 3. Example of sponsored entities re-indexation following query in Figure 4, for index strategies SPO (left), OSP (center) and POS (right). Prefixes omitted for brevity.

S	**P**	**O**
aaa-Y-40-auction-entity1	City	Paris
aaa-Y-40-auction-entity1	Rating	A
aaa-Y-40-auction-entity1	Type	Restaurant
Chez Louis	City	Paris
Chez Louis	Rating	A
Chez Louis	Type	Restaurant
Schweizer Schwein	City	Zurich
Schweizer Schwein	Rating	B
Schweizer Schwein	Type	Restaurant

P	**O**	**S**
City	Paris	aaa-Y-40-auction-entity1
City	Paris	Chez Louis
City	Zurich	Schweizer Schwein
Rating	A	aaa-Y-40-auction-entity1
Rating	A	Chez Louis
Rating	B	Schweizer Schwein
Type	Restaurant	aaa-Y-40-auction-entity1
Type	Restaurant	Chez Louis
Type	Restaurant	Schweizer Schwein

O	**S**	**P**
A	aaa-Y-40-auction-entity1	Rating
A	Chez Louis	Rating
B	Schweizer Schwein	Rating
Paris	aaa-Y-40-auction-entity1	City
Paris	Chez Louis	City
Restaurant	aaa-Y-40-auction-entity1	Type
Restaurant	Chez Louis	Type
Restaurant	Schweizer Schwein	Type
Zurich	Schweizer Schwein	City

3.2. *Web Preemption*

After ensuring bid-aware evaluation, the second step of AuctionKG is to deliver the solutions in batches with those including entities with the highest bids delivered first, and the rest after a pre-defined delay. The DAA model emphasizes the influence of the delay between batches and the probability of a user clicking on a link on a result in the batch. However, it does not specify whether a batch corresponds to a specific execution time or to a fixed number of results within the batch. Fortunately, the Web Preemption principle can handle both cases.

Web Preemption [10] is defined as the capacity of a SPARQL web server to suspend the execution of a running SPARQL query in order to resume it later. In the context of the DAA model, the SPARQL preemptable server returns results to the user after computing a fixed number of results or after a fixed amount of time. A preemptable server restarts execution from where it has been suspended when receiving a request from the user to get more results. Web preemption can be configured to deliver results either when reaching a fixed number of results or a fixed amount of time.

4. Experimental Study

This experimental study aims to answer the following questions empirically:

1. How faster AuctionKG delivers query results compared to a baseline query rewritten with an OrderBy clause?
2. What is the impact of forcing join ordering by bid variable on query execution time?
3. What is the difference in performance between delivering a fixed batch size and delivering all results collected up to a fixed waiting time?

All experimental material is available at `https://github.com/GDD-Nantes/sage-auction`.

4.1. Experimental Setup

Dataset and Queries We use the WatDiv [7] SPARQL benchmark with 10M triples. The WatDiv benchmark represents shops selling products. In the context of auctions, bids are located on product entities. Among all products (of which there are 25000), 5% are randomly sampled and assigned a random bid value between 1 and 100.

From the 12400 original diverse WatDiv queries, we removed duplicates and only kept those returning sponsored entities, yielding a workload of 222 queries featuring between 3 and 13 joins. We chose as bid variable the `product uri`, similar to the `?resto` variable in query $Bid_{?resto}(Q_1)$ of Figure 2b

Implementation For fair comparison, we used the Sage server [5] for executing our 222 queries with all approaches. Sage implements web preemption and can suspend queries after delivering several results, or after a quantum of time. We used PostgreSQL as the backend for Sage server with B+ trees and 4 indexes SPO, POS, OSP, and PSO. Concerning join ordering, Sage uses a simple strategy that sorts triple patterns by estimated cardinalities. It also avoids cartesian products (if possible) when assembling triple patterns following the sort order.

Approaches We compare the following approaches:

Basic query Rewriting (Bid): We rewrite each query in the workload following the pattern shown in Figure 2b: `OPTIONAL` and `COALESCE` clauses to retrieve bids for the bid variable and an `Order By` clause. The Order By is executed following the traditional materialize-and-sort approach. For this approach, we execute the queries until getting the complete answer, then sort and return the first 20.

[5]`http:sage.univ-nantes.fr`

Table 4. Number of results per approach

approach	**min**	**max**	**mean**
bid	2	809828	42494.527027
rename	2	809828	42494.527027
rename-force	2	809828	42494.527027
rename-force-500ms	0	20	18.150150
rename-force-top20	2	20	18.770270

Renaming (Rename) All sponsored entities are renamed as described in section 3.1. The join order is not forced to start with the bid variable, so results may be incorrect. This approach serves to measure the cost of forcing the join order. We execute queries until getting the complete answer.

Renaming (Rename-force) All sponsored entities are renamed as described in section 3.1. The join order is forced to start with the bid variable. For the rename approach, we execute the queries until getting the complete answer.

Renaming (top20) same as the Rename-force approach but retrieves only the top 20 results. This approach simulates a user consuming only the first 20 results. Unlike the Bid approach, returning the top 20 does not require materializing all possible results.

Renaming (500ms) same as the Rename-force approach, but delivers the top 20 results *or* whatever number of results produced after 500ms of execution, which may be less than 20 results. This simulates a user willing to wait at most 500ms to get at most 20 results.

We run each query in every approach 3 times and report the average total execution time in milliseconds. We also report the number of results each approach needs to produce to deliver the corresponding batches.

4.2. Experimental Results

Table 4 shows the minimal, maximal, and average number of results per approach. As expected, the maximum number of results and the average are the same for Bid, Rename and Rename-force. Comparing the average number of results between Rename-force-500ms and Rename-force-top20 shows that Rename-force-500ms returns fewer results but guarantees a fixed execution time of a batch, while Rename-force-top-20 returns always the 20 first results but in a variable time. There are 2 queries (q102 and q92) without results for the Rename-500 approach, *i.e.*, no results were found within 500ms.

Figure 5 presents the execution times per query for each approach (encoded in color). The X-axis shows queries ordered by the execution time of the Bid approach. Each measure is a point, but we use a line-point chart to facilitate the comparison of the trends of each approach. The Y-axis features the execution time in milliseconds on a logarithmic scale. Note the diversity of complexity in our workload, with query execution times ranging between 100ms and 300s.

We observe a significant gap between Bid and Rename-force lines, evidencing the high overhead introduced by the `optional` and `order by` clauses when rewriting queries. Rename-force is slower than the bid approach in 15 queries (7% of the workload), these are the instances where forcing the join order to start with the bid variable

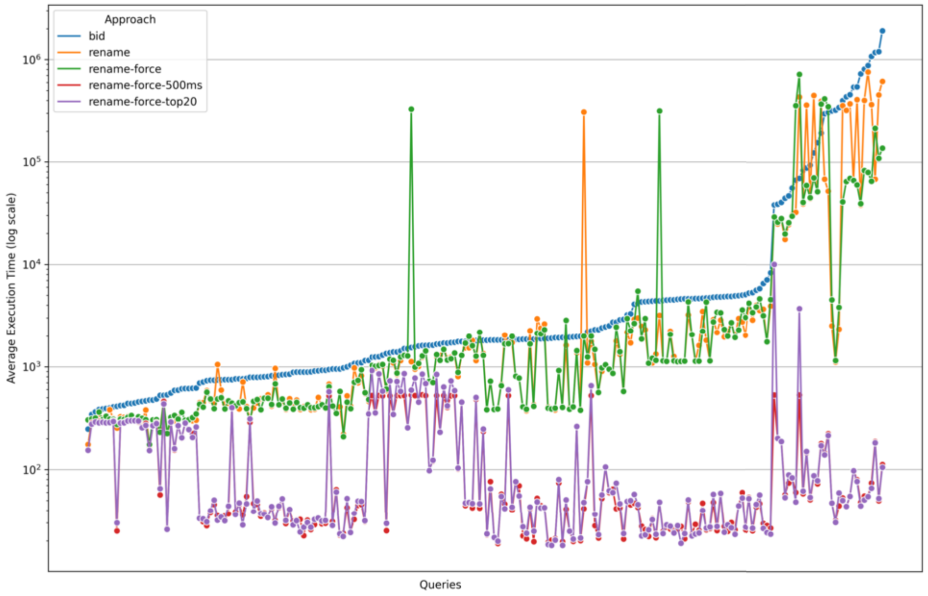


Figure 5. Execution times of all queries following the different approaches

produced suboptimal plans that were significantly worse than those produced by the Bid approach.

Comparing Rename and Rename-force allows us to evaluate the effect of forcing the join order of query execution. No approach dominates the other. We observe that most of the execution times are close, but for queries where there is a noticeable difference, it is quite significant.

Comparing Bid and rename-top20 allows us to assess the cost of the materialize-and-sort approach. Even for computing the top-20 results, the bid approach needs to produce all results, whilst Rename-top20 only computes the top20. As we can see, the performance gap can reach several orders of magnitude highlighting the intractability of the Bid approach in practice. We also observe high variations in the execution of rename-top20. Depending on the query and the join order, reaching the 20 first results depends on the selectivity of query triple patterns, *i.e.*, many intermediate results may be rejected before finding the correct ones. Compared to Rename-top20, Rename-500ms executes the query on the server for at most 500ms and at most 20 results. Consequently, Rename-500ms terminates at 500ms possibly returning less than 20 results. Overall, Rename-500ms and Rename-top20 have quite similar performances. Rename-500ms only improves query execution when Rename-top20 requires more than 500ms to compute the top20 results.

Figure 6 shows the average execution time of all approaches. This visualization further confirms our previous analysis. As Rename-top20 and Rename-500 can stop execution after a fixed amount of time or results, they dominate the Bid approach. This figure also reveals that Rename-500ms has a slightly faster execution time than Rename-top20, at the expense of completeness.

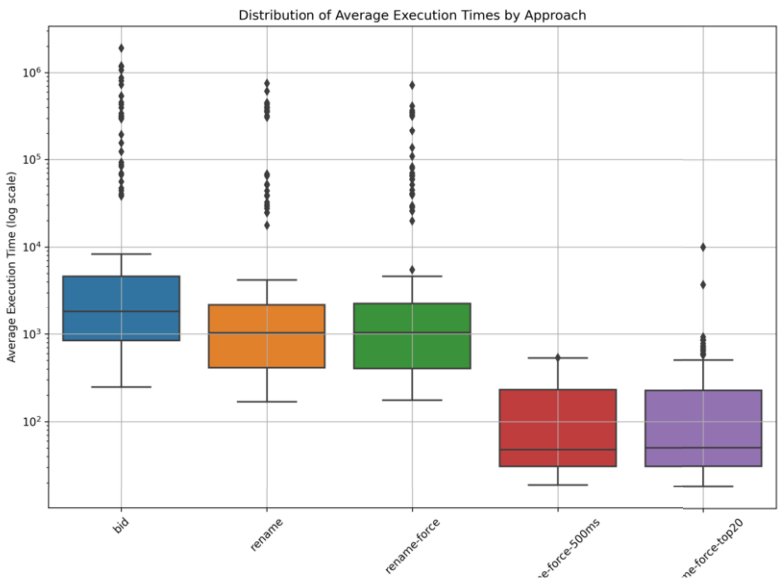


Figure 6. Average Execution times of queries per approach

5. Related Work

Data Marketplaces, also known as Data Market Platforms, are systems that mediate between data sellers and buyers and help them with their sharing, discovery, integration, and valuation problems. Data marketplaces problems can be classified as (i) Design, or how to design rules that lead to desired outcomes, for instance, auction mechanisms with certain guarantees, such as DAA and (ii) Deployment, or how to implement the market and enforce the designed rules, for instance, the efficient implementation of auction mechanisms [13,14].

The Semantic Web community has produced marketplaces that are either built upon Semantic technologies or intend to support the sale of Linked Data [15,16,17,18]. However, efforts are mostly focused on the transformation, integration, and description of data using semantic technologies to enable discovery and matchmaking between sellers and buyers through SPARQL queries. [16] supports the assignment of pre-defined pricing schemes to complete datasets, but not for fine-grained mechanisms such as DAA.

For query-answering on the Web of Data, there are no slots to auction, and agents can reorder received results at will. DAA overcomes this issue by replacing slots with time-to-deliver results, including sponsored entities as the scarce resource up for grabs. Several other auction models have been developed for selling data, but most of them with different pricing schemes that are not time-sensitive. [19] constructs a Data Marketplace for selling data to improve the accuracy of prediction tasks. Instead of receiving a query as input, the auctioneer receives from buyers a Machine Learning Model and a bid stating how much they are willing to pay for data that increases the accuracy of the model. [20] considers a general scenario where buyers submit bids to get some data in order to maximize a utility function. In this context, they use delays to protect auctioneers from strategic behavior such as the submission of artificially low bids in an attempt to influence future prices downward. To counter this, auction winners are computed on an epoch instead of individually (called an *Epoch-Shield*), and introducing a time delay to avoid strategic buyers fine-tuning their strategies by submitting multiple bids, without hurting truthful buyers (called a *Time-shield*). This approach acts upon users' bidding for query results and complements DAA, which acts upon users' bidding for entities to appear on query answers.

Our approach is related to the problem of efficient execution of general SPARQL top-k queries as defined in [11,21,9]. Top-k queries return the top-k results ordered by a user-defined scoring function. In the context of DAA, the scoring function is not defined by the users, but by the auctioneer. This allows the auctioneer to build the required dedicated index for ensuring bid ordering independently of SPARQL operators implemented in the engine, as long as they respect index order.

[21] proposes an approach to approximate SPARQL top-k join processing tailored to the case where the ranking function can only be computed at runtime and where frequent data updates make the computation of statistics on the ranking function infeasible. They propose to learn score distributions in a pay-as-you-go manner at runtime, having score statistics with constant space complexity and a computation complexity bounded by the result size. They achieve time savings of up to 65%, at the expense of 10-15% recall loss. An approximated model is not suitable for a DAA, as not delivering results in the right order would mean the Auctioneer violates the contract with the Sponsor. Fortunately, bid value is a ranking function that is available offline, making our approach feasible.

SPARQL-RANK [11] introduces the idea of having dedicated physical operators such as RankJoin to provide early termination of queries supported by sorted access by ranking function. Sorted access enables the early termination of top-k queries because, after a certain point, remaining mappings are guaranteed to not be part of the top-k results. This approach tackles the general case when the ordering expression may be complex and not known in advance. In the DAA context, the ordering of entities is simple and known in advance. This allows to build sorted access to sponsored entities.

The approach outlined in [9] combines Web Preemption with top-k query processing. It proposes a preemptable top-k operator whose overhead does not depend on k. It demonstrates a reduction in query execution time by up to 60% and a decrease in the amount of data transferred by a factor of 100. However, this technique only ensures *early pruning* rather than *early termination* as SPARQL-RANK. In our context, a preemptable top-k operator requires scanning all the values of a bid variable before delivering ordered results. Thanks to its sorted access to sponsored entities, AuctionKG scans only required values to deliver the answer.

6. Conclusion

In this paper, we presented AuctionKG, an approach combining reindexing and Web Preemption to implement the delayed auction model efficiently. Experiments demonstrate that the simple rewriting approach seriously degrades performance making the DAA model intractable in practice. Thanks to AuctionKG, we can deliver the first results in less than 1 second making the DAA approach usable in practice and contributing to the foundations of a sustainable economic model for the Web of Data.

In future work, other techniques are needed to ensure sorted access by bid order of triple patterns, as renaming does not work with a dictionary-like index. A further challenging problem is to consider more than one bid variable, *i.e.*, on query Q_3 of Figure 3b, not only considering `?city` but also `?resto`. How to return first the city with the highest sponsorship bid together with the restaurant with the highest sponsorship bid?

Acknowledgements

This work is supported by the French Labex CominLabs project MiKroloG, and the French ANR project MeKaNo (`ANR-22-CE23-0021`). Luis-Daniel Ibáñez was supported by the European Union's Horizon Europe research and innovation actions under Grant Agreement Nº 101093216 (UPCAST project). We thank students of the ALMA Master at the University of Nantes for their contributions to earlier versions of this research.

References

[1] Bizer C, Heath T, Berners-Lee T. Linked data: The story so far. In: Semantic services, interoperability and web applications: emerging concepts. IGI global; 2011. p. 205-27.

[2] Bizer C, Vidal ME, Skaf-Molli H. Linked Open Data. In: Encyclopedia of Database Systems; 2017. Available from: `https://hal.science/hal-02076911`.

[3] Hitzler P. A review of the semantic web field. Communications of the ACM. 2021;64(2):76-83.

[4] Vrandečić D, Krötzsch M. Wikidata: a free collaborative knowledgebase. Communications of the ACM. 2014;57(10):78-85.

[5] Lehmann J, al. Dbpedia–a large-scale, multilingual knowledge base extracted from wikipedia. Semantic web. 2015;6(2):167-95.

[6] Grubenmann T, Bernstein A, Moor D, Seuken S. Financing the Web of Data with Delayed-Answer Auctions. In: Champin P, Gandon FL, Lalmas M, Ipeirotis PG, editors. Proceedings of the 2018 World Wide Web Conference on World Wide Web, WWW 2018, Lyon, France, April 23-27, 2018. ACM; 2018. p. 1033-42. Available from: `https://doi.org/10.1145/3178876.3186002`.

[7] Gao L, Golab L, Özsu MT, Aluç G. Stream WatDiv: A Streaming RDF Benchmark. In: Groppe S, Gruenwald L, editors. Proceedings of the International Workshop on Semantic Big Data, SBD@SIGMOD 2018, Houston, TX, USA, June 10, 2018. ACM; 2018. p. 3:1-3:6. Available from: `https://doi.org/10.1145/3208352.3208355`.

[8] Kohavi R, Longbotham R, Sommerfield D, Henne RM. Controlled experiments on the web: survey and practical guide. Data mining and knowledge discovery. 2009;18(1):140-81.

[9] Aimonier-Davat J, Skaf-Molli H, Molli P. Processing SPARQL TOP-k Queries Online with Web Preemption. In: QuWeDa'22: 6th Workshop on Storing, Querying and Benchmarking Knowledge Graphs - Co-located at ISWC'22; 2022. .

[10] Minier T, Skaf-Molli H, Molli P. SaGe: Web Preemption for Public SPARQL Query Services. In: Liu L, White RW, Mantrach A, Silvestri F, McAuley JJ, Baeza-Yates RS, et al., editors. The World Wide Web Conference, WWW 2019, San Francisco, CA, USA, May 13-17, 2019. ACM; 2019. p. 1268-78. Available from: `https://doi.org/10.1145/3308558.3313652`.

[11] Magliacane S, Bozzon A, Della Valle E. Efficient Execution of Top-K SPARQL Queries. In: Cudré-Mauroux P, Heflin J, Sirin E, Tudorache T, Euzenat J, Hauswirth M, et al., editors. The Semantic Web – ISWC 2012. Berlin, Heidelberg: Springer Berlin Heidelberg; 2012. p. 344-60.

[12] Aimonier-Davat J, Skaf-Molli H, Molli P, Dang MH, Nédelec B. Join Ordering of SPARQL Property Path Queries. In: Extended Semantic Web Conference. Hersonissos, Greece: LNCS; 2023. Available from: `https://hal.science/hal-04151371`.

[13] Fernandez RC, Subramaniam P, Franklin MJ. Data market platforms: trading data assets to solve data problems. Proc VLDB Endow. 2020 jul;13(12):1933–1947. Available from: `https://doi.org/10.14778/3407790.3407800`.

[14] Azcoitia SA, Laoutaris N. A Survey of Data Marketplaces and Their Business Models. SIGMOD Rec. 2022 nov;51(3):18–29. Available from: https://doi.org/10.1145/3572751.3572755.

[15] Jacob B, Ortiz J. data.world: A Platform for Global-Scale Semantic Publishing. In: The Semantic Web – ISWC 2017 – Posters and Demo Track; 2017. .

[16] Roman D, Paniagua J, Tarasova T, Georgiev G, Sukhobok D, Nikolov N, et al. proDataMarket: A Data Marketplace for Monetizing Linked Data. In: The Semantic Web – ISWC 2017 – Posters and Demo Track; 2017. .

[17] Pomp A, Paulus A, Burgdorf A, Meisen T. A Semantic Data Marketplace for Easy Data Sharing within a Smart City. In: Proceedings of the 30th ACM International Conference on Information & Knowledge Management. CIKM '21. New York, NY, USA: Association for Computing Machinery; 2021. p. 4774–4778. Available from: https://doi.org/10.1145/3459637.3481995.

[18] Hamed N, Gaglione A, Gluhak A, Rana O, Perera C. Query Interface for Smart City Internet of Things Data Marketplaces: A Case Study. ACM Trans Internet Things. 2023 sep;4(3). Available from: https://doi.org/10.1145/3609336.

[19] Agarwal A, Dahleh M, Sarkar T. A Marketplace for Data: An Algorithmic Solution. In: Proceedings of the 2019 ACM Conference on Economics and Computation. EC '19. New York, NY, USA: Association for Computing Machinery; 2019. p. 701–726. Available from: https://doi.org/10.1145/3328526.3329589.

[20] Castro Fernandez R. Protecting Data Markets from Strategic Buyers. In: Proceedings of the 2022 International Conference on Management of Data. SIGMOD '22. New York, NY, USA: Association for Computing Machinery; 2022. p. 1755–1769. Available from: https://doi.org/10.1145/3514221.3517855.

[21] Wagner A, Bicer V, Tran T. Pay-as-you-go Approximate Join Top-k Processing for the Web of Data. In: Presutti V, d'Amato C, Gandon F, d'Aquin M, Staab S, Tordai A, editors. The Semantic Web: Trends and Challenges. Cham: Springer International Publishing; 2014. p. 130-45.