DAW: Duplicate-AWare Federated Query Processing over the Web of Data

Muhammad Saleem[1⋆], Axel-Cyrille Ngonga Ngomo[1], Josiane Xavier Parreira[2], Helena F. Deus[2], and Manfred Hauswirth[2]

[1] Universität Leipzig, IFI/AKSW, PO 100920, D-04009 Leipzig
lastname@informatik.uni-leipzig.de
[2] Digital Enterprise Research Institute, National University of Ireland, Galway
{firstname.lastname}@deri.org

Abstract. Over the last years the Web of Data has developed into a large compendium of interlinked data sets from multiple domains. Due to the decentralised architecture of this compendium, several of these datasets contain duplicated data. Yet, so far, only little attention has been paid to the effect of duplicated data on federated querying. This work presents DAW, a novel duplicate-aware approach to federated querying over the Web of Data. DAW is based on a combination of min-wise independent permutations and compact data summaries. It can be directly combined with existing federated query engines in order to achieve the same query recall values while querying fewer data sources. We extend three well-known federated query processing engines – DARQ, SPLENDID, and FedX – with DAW and compare our extensions with the original approaches. The comparison shows that DAW can greatly reduce the number of queries sent to the endpoints, while keeping high query recall values. Therefore, it can significantly improve the performance of federated query processing engines. Moreover, DAW provides a source selection mechanism that maximises the query recall, when the query processing is limited to a subset of the sources.

Keywords: federated query processing, SPARQL, min-wise independent permutations, Web of Data

1 Introduction

The emergence of the Web of Data has resulted in a large compendium of interlinked datasets from multiple domains available on the Web. The central principles underlying the architecture of these datasets include the decentralized provision of data, the reuse of URIs and vocabularies, as well as the linking of knowledge bases [2]. As a result, certain queries can only be answered by retrieving information from several data sources. This type of queries, called *federated queries*, are becoming increasingly popular within the Web of Data [1,3,8,9,12,14,21,22].

⋆ This work was carried out while the author was a research assistant in DERI.

Recently, the W3C released the SPARQL 1.1 specification which directly addresses federated queries [3]. Due to the independence of the data sources, certain pieces of information (i.e., RDF triples) can be found in multiple data sources. For example, all triples from the *DrugBank*[4] and *Neurocommons*[5] datasets can also be found in the *DERI health Care and Life Sciences Knowledge Base*[6]. We call triples that can be found in several knowledge bases *duplicates*.

While the importance of federated queries over the Web of Data has been stressed in previous work, the impact of duplicates has not yet received much attention. Recently, the work in [11] presented a benefit-based source selection strategy, where the benefit of a source is inversely proportional to the overlap between the source's data and the results already retrieved. The overlap is computed by comparing data summaries represented as Bloom filters [5]. The approach follows an "index-free" paradigm, and all the information about the sources is obtained at query time, for each triple pattern in the query.

In this paper we present DAW, a duplicate-aware approach for federated query processing over the Web of Data. Similar to [11] our approach uses sketches to estimate the overlap among sources. However, we adopt an "index-assisted" approach, where compact summaries of the sources are pre-computed and stored. DAW uses a combination of min-wise independent permutations (MIPs) [6] and triple selectivity information to estimate the overlap between the results of different sources. This information is used to rank the data sources, based on how many new query results are expected to be found. Sources that fall below a predefined threshold are discarded and not queried.

We extend three well-known federated query engines – DARQ [21], SPLENDID [8], and FedX [22] – with DAW, and compare these extensions with the original frameworks. The comparison shows that DAW requires fewer sources for each of the query's triple pattern, therefore improving query execution times. The impact on the query recall due to the overlap estimation was minimal, and in most cases the recall was not affected. Moreover, DAW provides a source selection mechanism that maximises the query recall when the query processing is limited to a subset of the sources.

The rest of this paper is organized as follows: Section 2 describes the state-of-the-art in federated query processing and different statistical synopsis approaches that can be used for approximating duplicate-free result sets. Section 3 describes our novel duplicate-aware federated query processing approach. An evaluation of DAW against existing federated query approaches is given in Section 4. Finally, Section 5 concludes our paper and presents directions for future work.

2 Related Work

In recent years, many approaches have been proposed for federated query processing for the Web of Data. Quilitz and Leser [21] propose an index-assisted

[3] `http://www.w3.org/TR/sparql11-federated-query/`
[4] `http://datahub.io/dataset/fu-berlin-drugbank`
[5] `http://neurocommons.org/page/RDF_distribution`
[6] `http://hcls.deri.org:8080/openrdf-sesame/repositories/hclskb`

federated query engine named DARQ for remote RDF data sources. DARQ combines service descriptions, query rewriting mechanisms and a cost-based optimisation approach to reduce the query processing time and the bandwidth usage. Langegger et al. [13] describe a solution similar to DARQ that relies on a mediator to keep its service descriptions up-to-date. SPLENDID [8] uses VOID[7] descriptions for data source selection along with SPARQL *ASK* queries. All of the approaches described above can be considered to be index-assisted, since they all rely in some sort of local index to guide the source selection process. Index-free approaches include FedX [22] and the Avalanche system [3]. In FedX, the source selection is performed by using *ASK* queries, while Avalanche gathers endpoints dataset statistics and bandwidth availability on the fly before the query federation. Ludwig and Tran [12] propose a hybrid query engine that assumes some incomplete knowledge about the sources to select and discover new sources at run time. A symmetric hash join is used to incrementally produce answers. Acosta et al. [1] present ANAPSID, a query engine that adapts the query execution schedulers to the SPARQL endpoints' data availability and run-time conditions.

Overlap estimation among data sources have been used in a number of approaches in the area of distributed and P2P information retrieval [4,10,15,18,23,24]. COSCO [10] gathers statistics about coverage and overlap from past queries and uses them to determine in which order the overlapping collections should be accessed to retrieve the most new results in the least number of collections. Bender et al. [4] describes a novelty estimator that uses Bloom filters [5] to estimate the overlap between P2P data sources. Bloom filters are also used in the BBQ strategy for benefit-based query routing over federated sources [11].

Statistical synopsis such as Min-Wise Independent Permutations (MIPs) [6], Bloom filters [5], Hash sketches [19], XSKETCH [20], fractional XSKETCH [7], and compressed Bloom filters [16] have been extensively used in the literature to provide a compacted representation of data sets. MIPs have been shown to be the provide a good tradeoff between estimation error and space requirements [15,6]. In addition, MIPs of different lengths can be compared, which can be beneficial for datasets of different sizes.

3 Duplicate-aware Federated Query Processing

In this section we present our DAW approach. DAW can be used in combination with existing federated query processing systems to enable a duplicate-aware query execution.

Given a SPARQL query q, the first step is to perform a *triple pattern-wise source selection*, i.e., to identify the set of data sources that contain relevant results for each of the triple patterns of the query. This is done by the underlying federated system. For a given triple pattern, the relevant sources are also called *capable* sources. The idea of DAW federated query processing is, for each triple

[7] `http://www.w3.org/TR/void/`

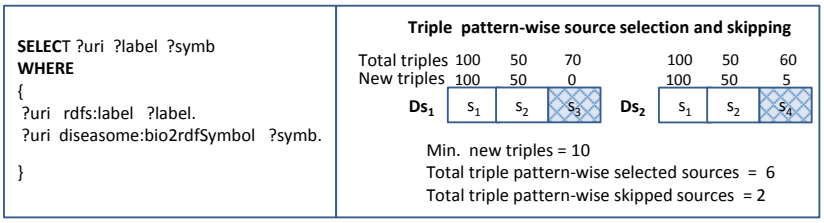


Fig. 1: Triple pattern-wise source selection and skipping example

pattern and its set of capable sources, to (i) *rank* the sources based on how much they can contribute with *new* query results, and (ii) *skip* sources which are ranked below a predefined threshold. We call these two steps *triple pattern-wise source ranking* and *triple-pattern wise source skipping*. After that, the query and the list of not skipped sources are forwarded to the underlying federated query engine. The engine generates the subqueries that are sent to the relevant SPARQL endpoints. The results of each subquery execution are then joined to generate the result set of q.

To better illustrate this, consider the example given in Figure 1, which shows a query with two triple patterns (tp_1 and tp_2), and the lists of capable sources for both patterns. For each source we show the total number of triples containing the same predicate of the triple pattern and the estimated number of new triples, i.e. triples that do not overlap with the previous sources in the list. The triple pattern-wise source ranking step orders the sources based on their contribution. As we see in the example, for the triple pattern tp_1, source S_1 is ranked first, since it is estimated to produce 100 results. S_1 is followed by S_2, which can contribute with 40 new results, considering the overlap between the two sets. S_3 is ranked last, despite having more triples than S_2. This is because our duplicated-aware estimation could not find any triple in S_3 which is not in either S_1 or S_2. In the triple-pattern wise source skipping step, S_3 will be discarded, and tp_1 will not be sent to S_3 during query execution. We can also set a threshold on the minimum number of results. For instance, by setting the threshold to 10 results, source S_4 will be skipped, since it can only contribute with 5 new results for tp_2. By applying our duplicate-aware approach – which would select S_1 and S_2 both for *tp1* and *tp2* and would skip S_3 and S_4 – we would only send subqueries to two endpoints instead of four.

Both steps are performed prior to the query execution, by using only information contained in the DAW index. The main innovation behind DAW is to avoid querying sources which would lead to duplicated results. We achieve this by extending the idea of min-wise independent permutations (MIPs) [6], which are explained in the next section.

3.1 Min-Wise Independent Permutations (MIPs)

The main rationale behind MIPs is to enable the representation of large sets as vectors of smaller magnitude and to allow the estimation of a number of set

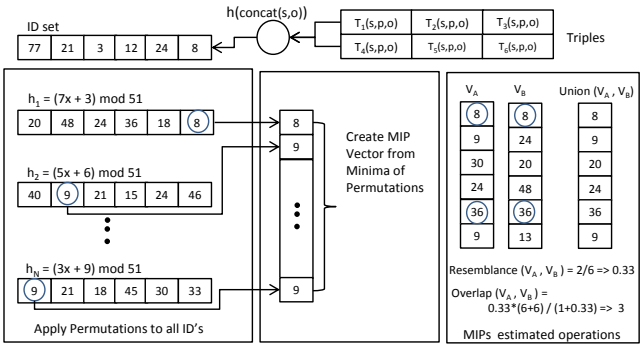


Fig. 2: Min-Wise Independent Permutations

operations, such as overlap and union, without having to compare the original sets directly. The basic assumption behind MIPs is that each element of an ordered set S has the same probability of becoming the minimum element under a random permutation. MIPs assumes an ordered set S as input and computes N random permutations of the elements. Each permutation uses a linear hash function of the form $h_i(x) := a_i * x + b_i \ mod \ U$ where U is a big prime number, x is a set element, and a_i, b_i are fixed random numbers. By ordering the set of resulting hash values, we obtain a random permutation of the elements of S. For each of the N permutations, the MIPs technique determines the minimum hash value and stores it in an N-dimensional vector, thus capturing the minimum set element under each of these random permutations. The technique is illustrated in Figure 2.

Let $V_A = [a_1, a_2, \ldots, a_N]$ and $V_B = [b_1, b_2, \ldots, b_N]$ be the two MIPs vectors representing two ordered ID's sets S_A, S_B, respectively. An unbiased estimate of the pair-wise resemblance between the two sets, i.e. the fraction of elements that both sets share with each other, is obtained by counting the number of positions in which the two MIPs vectors have the same number and dividing this by the number of permutations N as shown in Equation 1. It can be shown that the expected error in the estimation $O(1/\sqrt{N})$ [6]. Given the resemblance and the sizes of the two set, their overlap can be estimated as shown in Equation 2. A MIPs vector representing the union of the two sets, S_A and S_B, can be created directly from the individuals MIPs vectors, V_A and V_B, by comparing the pair-wise entries, and storing the minimum of the two values in the resulting union vector (see Figure 2). A nice property of MIPs is that unions can be computed even if the two MIPs vectors have different sizes, as long as they use the same sequence of hash functions for creating their permutations. In general, if two MIPs have different sizes, we can always use the smaller number of permutations as a common denominator. This incurs in a loss of accuracy in the result MIPs, but still yields to a more flexible setting, where the different collections do not have to agree on a predefined MIPs size [15].

$$Resemblance(S_A, S_B) = \frac{|S_A \cap S_B|}{|S_A \cup S_B|} \approx \frac{|V_A \cap V_B|}{N} \tag{1}$$

$$Overlap(S_A, S_B) \approx \frac{Resemblance(V_A, V_B) \times (|S_A| + |S_B|)}{(Resemblance(V_A, V_B) + 1)} \quad (2)$$

In the DAW index, MIPs are used as follow: For a distinct predicate p belonging to a data source S, we define $T(p, S)$ as the set of all triples in S with predicate p. A MIPs vector is then created for every $T(p, S)$. First an *ID set* is generated by mapping each triple in $T(p, Sr)$ to an integer value. A triple is given in the form of subject, predicate and object tuples, i.e. $< s, p, o >$. Since all triples in $T(p, S)$ share the same predicate by definition, the mapping is done by concatenating the subject (s) and object (o) of the triple, and applying a hash function to it (Figure 2). Then, the MIPs vector is created by computing the N random permutations of each element in the *ID set* and storing their minimum value. Finally, the MIPs vector is stored and mapped to each capability of the service description, as explained in the next section.

3.2 DAW Index

In order to detect duplicate-free subqueries, DAW relies on an index which contains the following information for every distinct predicate p in a source S:

1. The total number of triples $n_S(p)$ with the predicate p in S.
2. The MIPs vector $MIPs_S(p)$ for the predicate p in S, as described in the previous section.
3. The *average subject selectivity* of p in S, $avgSbjSel_S(p)$.
4. The *average object selectivity* of p in S, $avgObjSel_S(p)$.

The average subject and object selectivities are defined as the inverse of the number of distinct subjects and objects which appears with predicate p, respectively. For example, given the following set of triples:

$$S = \{< s_1, p, o_1 >, < s_1, p, o_2 >, < s_2, p, o_1 >, < s_3, p, o_2 >\} \quad (3)$$

the $avgSbjSel_S(p)$ is equal to $\frac{1}{3}$ and the $avgObjSel_S(p)$ is $\frac{1}{2}$. These two values are used in combination with the MIPs vector to address the expressivity of SPARQL queries as explained below.

Suppose that in a given triple pattern, neither the subject nor the predicate are bound. That means the pattern is of the form $<?s, p, ?o >$, where the question mark denotes a variable. In this case, the MIPs vectors in the DAW index can be used directly to estimate the overlap among the data sources that can provide results for the pattern. This is because the MIPs vectors are created by grouping triples according to their predicate. However, if any of the subject or object is bound (for example, $< s_1, p, ?o >$), the selectivity of the pattern becomes much higher and the MIPs vectors alone are unable to address this. As a result, overlap will be overestimated. To address this issue the modify Equation 2 to account for the subject and object selectivities as follows:

$$Overlap_{tp}(S_A, S_B) \approx \frac{Resemblance(V_A, V_B) \times (|S'_A| + |S'_B|)}{(Resemblance(V_A, V_B) + 1)} \quad (4)$$

Listing 1.1: DAW index example

```
[] a sd:Service ;
     sd:endpointUrl   <http://localhost:8890/sparql> ;
     sd:capability [
      sd:predicate  diseasome:name ;
      sd:totalTriples     147  ;
      sd:avgSbjSel  ''0.0068'' ;
      sd:avgObjSel  ''0.0069'' ;
      sd:MIPs  ''-6908232 -7090543 -6892373 -7064247 ...''; ] ;
     sd:capability [
      sd:predicate  diseasome:chromosomalLocation ;
      sd:totalTtriples     160  ;
      sd:avgSbjSel  ''0.0062'' ;
      sd:avgObjSel  ''0.0072'' ;
      sd:MIPs  ''-7056448 -7056410 -6845713 -6966021 ...''; ] ;
```

where the original size of a set S_i is replaced by a value $|S_i'|$ which is given by the following equation:

$$|S_i'| = \begin{cases} |S_i| & \text{if neither subject nor object are bound,} \\ |S_i| \times avgSbjSel_S(p) & \text{if subject is bound,} \\ |S_i| \times avgObjSel_S(p) & \text{if object is bound.} \end{cases}$$

We call the set $C_S(p) = \{p, n_S(p), avgSbjsel_S(p), avgObjSel_S(p), MIPs_S(p)\}$ a *capability* of the data source. The total number of capabilities of a data source is equal to the number of distinct predicates in it.

It is crucial to keep the index size small to minimise the pre-processing time. On the other hand, this index must also contain sufficient information to enable an accurate source selection and duplicate-free subquery generation. Some federated query approaches such as DARQ and SPLENDID already provide the total number of triples, as well as the average selectivity values. Therefore, the storage overhead create by the DAW index depends mostly on the size of the MIPs vectors which can be adjusted to any length. In general, MIPs can provide a good estimation of the overlap between sets with a few integer in length. An example of a DAW index is given in Listing 1.1.

3.3 DAW Federated Query Processing

As explained earlier, given a SPARQL query, DAW performs the triple pattern-wise source ranking and skipping steps in order to rank the sources based on how much they can contribute with *new* query results, and skip sources which are below a given threshold. In this section we describe these two steps in detail.

Triple Pattern-wise Source Ranking: Given the heterogeneity and independence of data sources, it is expected that each source contributed differently in answering a given triple pattern, and the same result might be returned by multiple sources. Our goal is to provide a rank of the sources, according to the estimated number of new results it can contribute. By new results we mean with respect to the results already retrieved from sources ranked higher. The

source ranking step works as follows: First, as no source has been ranked yet, the algorithm chooses the largest source, as it will likely to contribute with more results. To select the next source we use the DAW index to compute the estimated overlap between the already selected source and every remaining source. The remaining source with the least amount of overlap is then chosen and ranked second. Before selecting the next source in the rank, we first need to estimate the union of the already selected sources. This is needed since we want to find out how much a source can contribute with results are not in the sources selected so far. The union can be easily estimated by applying a vector operator on the original MIPs, as explained in Section 3.1. The new union MIPs can be further combined with other MIPs to get the estimation of the union among several sets. The source ranking step continues until no more sources are left to be ranked.

Triple Pattern-wise Source Skipping: Given the rank of capable sources, the next step is to prune the rank, but skipping sources which cannot contribute with a minimum number of new results. This is done by setting a threshold, and pruning every source which falls below it. Since the total number of results depends on the triple pattern, the threshold is chosen in terms of the minimum percentage of new results a source can contribute. For instance, if the threshold is set to zero, DAW will aim at retrieving as much results as possible, while still skipping sources which cannot contribute with new results. Alternatively, the threshold can be set to higher values, in cases where the tradeoff between recall and number of sources queries is more important.

The pseudo code of the triple pattern-wise source ranking and skipping is given in Algorithm 1. It takes a triple pattern $tp_i(s, p, o)$, its list of capable sources $\mathbb{S}_i$, and the predefined threshold value as input and returned a ranked list of a subset of the capable source set R_i, $R_i \subseteq \mathbb{S}_i$ as output. The ranked list and the MPIs with the union of the selected sources are initialised with the largest source. Lines 8-14 adjust the size of the dataset to reflect the subject or object selectivities, depending on the query. Lines 15-16 estimate the overlap and number of new triples. The source with the highest amount of new triples is then selected (Lines 17-19). The triple pattern-wise source skipping is done in Line 23 and sources ranked higher than the threshold are added to the final ranked list (Line 24). The union MIPs is then updated (Line 26) and the algorithm continues until no more sources are left.

Before we present our experimental analysis of DAW it is important to note the difference between the number of triple pattern-wise sources and the number of sources (e.g. SPARQL endpoints). The total number of triple pattern-wise selected sources for a query is calculate as follow: Let $NS_i \in \{1 \dots M\}$ be the number of sources capable of answering a triple pattern tp_i where M is the number of available (physical) sources. Then, for a query q with n triple patterns, $\{tp_1, tp_2, \dots tp_n\}$, the total number of triple pattern-wise sources is the sum of the sources for individual triple patterns, i.e. $\sum_{j=1}^{n} NS_j$. In the example from Figure 1, the number of sources is 4 (s_1, s_2, s_3, s_4) but the number of triple pattern-wise sources is equal to 6.

Algorithm 1 Triple pattern source-wise ranking and skipping

Require: tp_i(s,p,o) ∈ T; $\mathbb{S}_i$; thresholdVal //triple pattern tp_i, capable data sources of tp_i; Threshold Value

```
1: rank1Source = getMaxSizeSource(S_i, tp_i) ; rnkNo = 1
2: unionMIPs = getMIPs(rank1Source, tp_i) //get MIP vector for a tp of a source
3: R_i[rnkNo] = selectedSource
4: S_i = S_i - {selectedSource}
5: rnkNo = rnkNo+1
6: while S_i ≠ ∅ do
7:    selectedSource = null; maxNewTriples =0
8:    for each S_i ∈ S_i  do
9:       MIPs = getMIPs(S_i, tp_i)
10:      if s is bound in tp_i then
11:         MIPsSetSize =MIPsSetSize*getAvgSbjSel(S_i,tp_i)
12:      else if o is bound in tp_i  then
13:         MIPsSetSize =MIPsSetSize*getAvgObjSel(S_i,tp_i)
14:      end if
15:      overlapSize = Overlap(unionMIPs,MIPs)
16:      newTriples = unionMIPsSetSize - overlapSize
17:      if newTriples > maxNewTriples then
18:         selectedSource = S_i
19:         maxNewTriples = newTriples
20:      end if
21:   end for
22:   curThresholdVal = unionMIPsSetSize / maxNewTriples
23:   if curThresholdVal ⩾ thresholdVal then
24:      R_i[rnkNo] = selectedSource
25:      selectedMIPs = getMIPs(selectedSource, tp_i)
26:      unionMIPs = Union(unionMIPs,selectedMIPs)
27:      rnkNo = rnkNo+1
28:   end if
29:   S_i = S_i - {selectedSource}
30: end while
31: return  R_i //ranked list of capable sources for tp_i
```

4 Experimental Evaluation

In this section we present an experimental evaluation of the DAW approach. We first describe the experimental setup, followed by the evaluation results. All data used in this evaluation can be found at the project web page.[8]

4.1 Experimental Setup

Datasets: For our experiments, we used four different datasets. The Diseasome dataset contains diseases and disease genes linked by disease-gene associations.

[8] `https://sites.google.com/site/DAWfederation/`

Dataset	Number Triples	Dataset Size (MB)	Index Size (MB)	Index. Gen. Time (sec)	Discrepancy	No. Duplicated Slices	Duplicate Slice ID
Diseasome	91,122	18.6	0.17	4	1,500	1	10
Publication	234,405	39.0	0.24	6	2,500	1	10
Geo	1,900,006	274.1	1.63	133	50,000	2	5,8
Movie	3,579,616	448.9	1.66	201	100,000	1	2

Table 1: Overview of the datasets used in the experiments

EP	CPU(GHz)	RAM	Hard Disk
1	2.2, i3	4GB	300 GB
2	2.9, i7	16 GB	256 GB SSD
3	2.6, i5	4 GB	150 GB
4	2.53, i5	4 GB	300 GB
5	2.3, i5	4 GB	500 GB
6	2.53, i5	4 GB	300 GB
7	2.9, i7	8 GB	450 GB
8	2.6, i5	8 GB	400 GB
9	2.6, i5	8 GB	400 GB
10	2.9, i7	16 GB	500 GB

Table 2: SPARQL endpoints specification

Dataset	STP	S-1	S-2	P-1	P-2	P-3	Total
Diseasome	5	5	5	4	5	2	26
Geo	5	5	5	-	-	-	15
Movie	5	-	-	-	-	-	5
Publication	5	5	5	7	7	4	33
Total	20	15	15	11	12	6	79

Table 3: Distribution of query types across datasets

The Publication dataset is the Semantic Web Dog Food dataset and contains information on publications, venues and authors of publications. The Geo dataset resulted from retrieving the portion of triples from DBpedia that maps resources to their geo-coordinates. Finally, the Movie dataset is the RDF version of IMDB and contains amongst others a large number of actors, movies and directors. To simulate a federated scenario with fragmented datasets distributed across several sources, we partitioned each dataset in 10 slices and distributed the slices across 10 data sources (one slice per data source). Each data source is a Virtuoso-2012-08-02 SPARQL endpoint with the specifications given in Table 2.

To distribute the data across our 10 endpoints we defined a *discrepancy factor*, which controls the maximal size difference between the different slices.

$$discrepancy = \max_{1 \leq i \leq M} |L_i| - \min_{1 \leq j \leq M} |L_j|, \tag{5}$$

where L_i stands for the ith slice. The data is first partitioned randomly among the slices in a way that $\sum_i |L_i| = D$ and $\forall i \forall j \; i \neq j \rightarrow ||L_i| - |L_j|| \leq discrepancy$. None of the existing benchmarks for federated query processing addresses the data duplication issue. Therefore, in order to add duplicates among slices, we randomly selected a number of slices and duplicated their contents across all remaining slices. For the DAW index, we use MIPs vectors of different sizes to better reflect the number of triples per predicate in each source. The sizes were chosen in a way that the overall index size is kept small. Table 1 presents an overview of the datasets, including the total number of triples and total size, the size of the DAW index, the index generation time, the discrepancy value among the 10 slices, the number of slices that were duplicated and their corresponding ID.

Queries: We used three types of queries in our experiments: Single triple patterns queries (STP), star-shaped queries (S-1, S-2), and path-shaped queries (P-1, P-2, P-3). Single triple pattern (STP) queries consist of exactly one triple pattern in the query. Star-shaped and path-shaped queries are defined as in [9]. A S-k star-shaped query has one variable as subject and k joins, i.e., (k+1) triple patterns. An example of a S-1 star-shaped query is given in Figure 1. A P-k path-shaped query is generated by using the object of one triple pattern as subject in the next triple pattern, and it also contains (k+1) triple patterns. Previous work has shown that these query shapes are the most common shapes found in real-world RDF queries [17]. Our benchmark data consisted of 79 queries as shown in Table 3. Some query shapes could not be used on certain datasets due to the topology of the underlying ontology. For example, P-1 queries could not be sent to the Geo dataset since it only contained object properties. Each type a query was executed we used a random resource as subject or object, depending on the query type. The predicates of all queries are fixed.

Federated Query Engines: We implemented our DAW approach on top of three different federated query engines: DARQ [21], SPLENDID [8], and FedX [22]. Both DARQ and SPLENDID already provide an index with some of the statistics needed in DAW. Therefore, we only needed to extend this index. For FedX, which is index-free, we added an index similar to the one in DARQ with our DAW extension. The underlying query execution mechanism remained the same.

Metrics: We compared the three federated approaches against their DAW extensions. For each query type we measured (i) the average number of triple pattern-wise sources that were skipped, (ii) the average recall, and (iii) the average query execution time. We did not consider the number of endpoints requests, as it depends on a number of factors, such as join type, block and buffer size, that vary across the different federated query processors. The threshold was initially set to zero, in order to maximise recall while querying fewer sources. All experiments were carried out in a machine with a 2.53GHz i5 processor, 4 GB RAM, and 500 GB hard disk. Experiments were carried out in a local network, so the network costs were negligible. After the first warm up run, each query type was executed 10 times and results were averaged.

4.2 Experimental Results

Triple Pattern-Wise Source Skipping: Table 4 shows the number of capable triple pattern-wise sources that were skipped by our approach, for each query type, as well as the recall. The total number of triple pattern-wise sources selected by the original systems is shown in brackets. The threshold was set to zero, which means that only sources that were estimated to returned no new results were pruned. We can see that DAW can effectively reduce the total triple pattern-wise selected sources, thus enable fewer subqueries federation. The highest gain was in the Diseasome dataset, where 214 sources were skipped in the DARQ approach, without affecting the recall. This corresponds to a decrease on

Listing 1.2: A Single Triple Pattern (STP) query example

```
SELECT ?title WHERE
{ www2008-paper:103 pub:title ?title. }
```

Dataset	STP	S-1	S-2	P-1	P-2	P-3	Total	Recall
Diseasome	14(35)	30(77)	40(107)	35(65)	65(125)	30(50)	214(459)	100%
Geo	22(40)	23(55)	37(101)	-	-	-	82(196)	99.99%
Movie	22(38)	-	-	-	-	-	22(38)	100%
Publication	9(30)	10(37)	15(86)	14(60)	21(120)	32(102)	101(435)	100%
Total	67(143)	63(169)	92(294)	49(125)	86(245)	62(152)	419(1128)	-

(a) DARQ

Dataset	STP	S-1	S-2	P-1	P-2	P-3	Total	Recall
Diseasome	7(28)	30(77)	40(107)	35(65)	65(125)	30(50)	207(452)	100%
Geo	19(37)	23(55)	37(101)	-	-	-	79(193)	99.99%
Movie	15(31)	-	-	-	-	-	15(31)	100%
Publication	3(24)	10(37)	15(86)	14(60)	21(120)	32(102)	95(429)	100%
Total	44(120)	63(169)	92(294)	49(125)	86(245)	62(152)	396(1105)	-

(b) FedX and SPLENDID

Table 4: Distribution of the triple pattern-wise source skipped by DAW extensions for threshold value 0

the number of queried sources from 459 to 245. In other words, a full recall was achieved by querying only 53% of the available triple pattern-wise sources. In all cases except in the Geo dataset, the recall was not affected and all relevant results were retrieved. In the Geo dataset, the DAW index incorrectly pruned a small number of relevant sources, but the recall was still 99.99%. That means that DAW can deliver the same query results while querying much fewer sources. The source selection methods from FedX and SPLENDID return the same set of sources, therefore the number of skipped sources was the same for both. Moreover, they both use SPARQL ASK queries in the selection mechanisms, which leads to a better performance for STP queries. For example, consider the STP query given in Listing 1.2 where both the subject and predicate are bound. It is likely that a WWW2008 paper with id 103 is found in only one data source but the property `pub:title` may be found in every source. As a result, FedX and SPLENDID will only select a single capable source while DARQ will select all sources containing that predicate.

Query Execution Time: For each dataset and query type, we measured the average query execution time in each of the federated query approaches and also in their DAW extension. Again, the threshold was set to zero and the average was over 10 queries. Figures 3, 4, and 5 show the results. We can see that DAW improves the query performance for most of the cases. For three of the datasets, Diseasome, Geo and Movie, DAW improved the query execution times of all federated systems tested, for all query types. The query performance in the Diseasome dataset showed the highest improvements. This is due to the large number of triple pattern-wise sources that were pruned. We can also see that

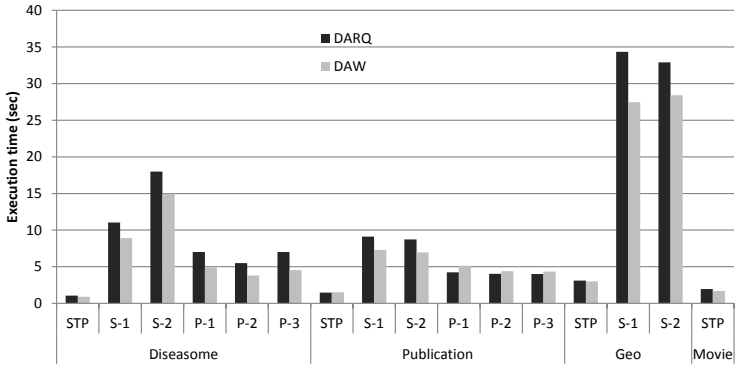


Fig. 3: Query execution time of DARQ and its DAW extension

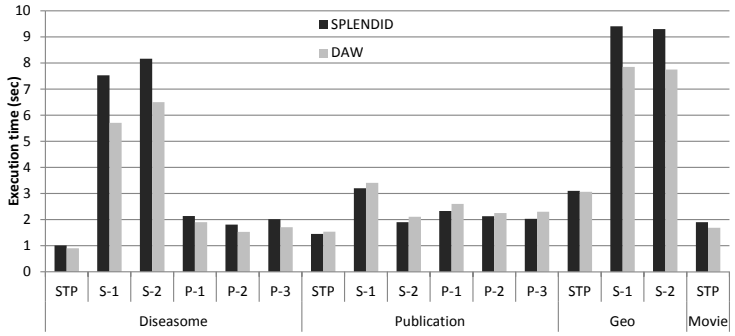


Fig. 4: Query execution time of SPLENDID and its DAW extension

if the number of skipped sources is low – as for the Publication dataset – the overhead in computing the sources overlap can be higher than the execution time saved by querying fewer sources, so the overall query execution time is worse. The overall performance is summarised in Table 5. We were able to improve the query execution time in DARQ by 16.46%, the SPLENDID by 11.11%, and FedX by 9.76%. For the Diseasome dataset, the improvement for the DARQ approach was 23.34%. These are averaged values across all datasets and query types. DAW led to a performance gain for most of the settings. We expect that in a setup with larger datasets and higher overlap, DAW can lead to even better improvements.

Number of Queried Sources vs. Query Recall: The evaluation presented so far focused on achieving full recall, and only discarded sources that the DAW index estimated to contribute with no new results. We have shown that the estimation given by our algorithm is quite accurate, as only 0.01% of the results in one dataset were missing. There might be cases, however, where full recall is not crucial and the query processing budget is limited. Here, the goal is to retrieve as many results as possible by querying only a subset of capable sources. Standard federated query processing approaches are only able to identify the

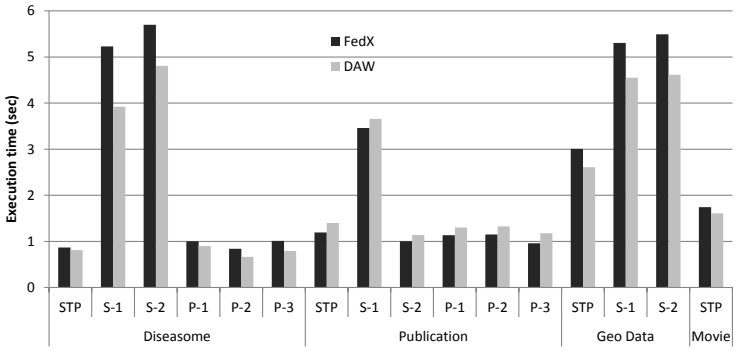


Fig. 5: Query execution time of FedX and its DAW extension

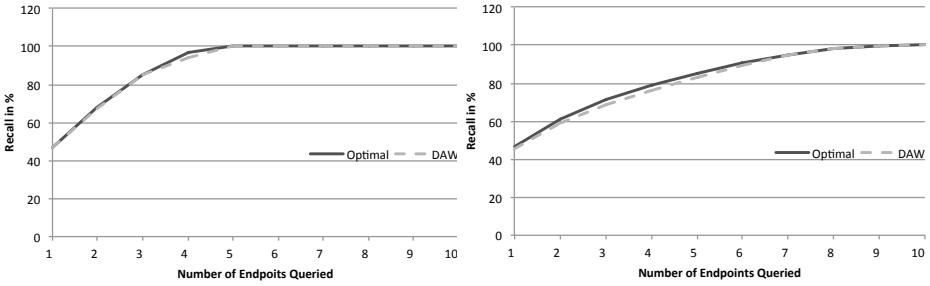
	Diseasome		Publication		Geo Data		Movie		Overall	
	Exe.time	Gain	Exe.time	Gain	Exe.time	Gain	Exe.time	Gain	Exe.time	Gain
DARQ	8.27		5.26		23.44		1.96		9.59	
DAW	6.34	23.34	4.94	6.14	19.62	16.31	1.68	13.88	8.01	16.46
SPLENDID	3.78		2.18		7.27		1.90		3.71	
DAW	3.04	19.48	2.38	-8.94	6.22	14.40	1.68	11.16	3.30	11.11
FedX	2.44		1.48		4.60		1.74		2.44	
DAW	1.98	18.79	1.67	-12.38	3.92	14.71	1.61	7.59	2.20	9.76

Table 5: Overall performance evaluation. *Exe.time* is the average execution time in seconds. *Gain* is the percentage in the performance improvement

set of capable sources. They are not able to compare the contribution of the sources in order to identify which subset yields to a better recall. With DAW, an approximation of this contribution is provided by the ranking step. For any given threshold, DAW is able to provide the subset of capable sources that will deliver the best recall for that number of sources. To demonstrate this, we computed the query recall for different threshold values for the DAW DARQ extension. We ran each of the STP queries 10 times on the Diseasome and Publication datasets and averaged the results. We varied the threshold value in order to limit the query to a fixed number of endpoints and we computed the query recall based on the DAW source selection. We compared it with the optimal duplicate-aware approach, where sources were manually selected to maximise the recall. The results are show in Figure 6. We can see that, in both cases, the source selection given by DAW is very close to the optimal case. Moreover, our experiment demonstrates the great potential in using source ranking for federated query processing. For the Diseasome dataset, by querying only 3 out of the 10 endpoints, DAW is able to retrieve 80% of the query results. A full recall is achieved with only 6 endpoints. This naturally depends on the degree of overlap, but nevertheless it shows promising results that should be further explored.

5 Conclusion and Future Work

In this paper we presented DAW, an approach for duplicate-aware federated query over the Web of Data. DAW combines min-wise independent permuta-



(a) Diseasome (b) Publication

Fig. 6: Recall for varied number of endpoints queried

tions with selectivity values to estimate the number of duplicate-free results. This estimation is used to first rank triple pattern-wise sources, based on their contribution, and to skip sources that contribute with little or no new results. DAW can be directly combined with existing index-assisted federated query processing systems, in order to improve the query execution. We evaluated our approach against DARQ, SPLENDID and FedX – three well known federated systems. The evaluation shows that by using the DAW extension the query execution times were improved in most of the cases, while recall was marginally affected. Moreover, DAW is suitable for maximising the recall for a fixed number of queried sources.

We will look at extending our index to further reduce the query execution time, for instance, by pre-computing some of the overlap statistics, based on query logs. The effect of different MIPs sizes and threshold values to find the optimal trade-off between execution time and recall will also explored, as well as different data partition methods.

6 Acknowledgments

This work has been supported by the European Commission under Contract No. FP720117287661 (GAMBAS), FP7-Granatum: RE7098, FP7-GeoKnow Grant No. 318159, by Science Foundation Ireland under Grant No. SFI/08/CE/I1380 (Lion-II) and Grant No. SFI/12/RC/2289 (INSIGHT).

References

1. M. Acosta, M. Vidal, T. Lampo, J. Castillo, and E. Ruckhaus. Anapsid: an adaptive query processing engine for sparql endpoints. In *ISWC*, pages 18–34, 2011.
2. S. Auer, J. Lehmann, and A.-C. Ngonga Ngomo. Introduction to linked data and its lifecycle on the web. In *RW*, pages 1–75, 2011.
3. C. Basca and A. Bernstein. Avalanche: putting the spirit of the web back into semantic web querying. In *SSWS*, pages 64–79, November 2010.

4. M. Bender, S. Michel, P. Triantafillou, G. Weikum, and C. Zimmer. Improving collection selection with overlap awareness in p2p search engines. In *SIGIR*, pages 67–74, 2005.
5. B. H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, 13(7):422–426, July 1970.
6. A. Z. Broder, M. Charikar, A. M. Frieze, and M. Mitzenmacher. Min-wise independent permutations. *IJCSS*, 60:327–336, 1998.
7. N. Drukh, N. Polyzotis, and Y. Matias. Fractional xsketch synopses for xml databases. In *XSym*, pages 189–203, 2004.
8. O. Görlitz and S. Staab. Splendid: Sparql endpoint federation exploiting void descriptions. In *COLD, ISWC*, 2011.
9. A. Harth, K. Hose, M. Karnstedt, A. Polleres, K.-U. Sattler, and J. Umbrich. Data summaries for on-demand queries over linked data. In *WWW*, pages 411–420, 2010.
10. T. Hernandez and S. Kambhampati. Improving text collection selection with coverage and overlap statistics. In *WWW (Special interest tracks and posters)*, pages 1128–1129, 2005.
11. K. Hose and R. Schenkel. Towards benefit-based rdf source selection for sparql queries. In *SWIM*, page 2, 2012.
12. G. Ladwig and T. Tran. Linked data query processing strategies. In *ISWC*, pages 453–469, 2010.
13. A. Langegger, W. Wöß, and M. Blöchl. A semantic web middleware for virtual data integration on the web. In *ESWC*, pages 493–507, 2008.
14. Y. Li and J. Heflin. Using reformulation trees to optimize queries over distributed heterogeneous sources. In *ISWC*, pages 502–517, 2010.
15. S. Michel, M. Bender, P. Triantafillou, and G. Weikum. Iqn routing: Integrating quality and novelty in p2p querying and ranking. In *EDBT*, pages 149–166, 2006.
16. M. Mitzenmacher. Compressed bloom filters. *IEEE/ACM Trans. Netw.*, 10(5):604–612, Oct. 2002.
17. M. Morsey, J. Lehmann, S. Auer, and A.-C. N. Ngomo. Dbpedia sparql benchmark: performance assessment with real queries on real data. In *ISWC*, pages 454–469, 2011.
18. Z. Nie, S. Kambhampati, and T. Hernandez. Bibfinder/statminer: Effectively mining and using coverage and overlap statistics in data integration. In *VLDB*, pages 1097–1100, 2003.
19. N. Ntarmos, P. Triantafillou, and G. Weikum. Distributed hash sketches: Scalable, efficient, and accurate cardinality estimation for distributed multisets. *ACM Trans. Comput. Syst.*, 27, 2009.
20. N. Polyzotis and M. Garofalakis. Statistical synopses for graph-structured xml databases. In *SIGMOD*, pages 358–369, 2002.
21. B. Quilitz and U. Leser. Querying distributed rdf data sources with sparql. In *ESWC*, pages 524–538, 2008.
22. A. Schwarte, P. Haase, K. Hose, R. Schenkel, and M. Schmidt. Fedx: Optimization techniques for federated query processing on linked data. In *ISWC*, Lecture Notes in Computer Science, pages 601–616. 2011.
23. M. Shokouhi and J. Zobel. Federated text retrieval from uncooperative overlapped collections. In *SIGIR*, pages 495–502, 2007.
24. L. Si and J. P. Callan. Relevant document distribution estimation method for resource selection. In *SIGIR*, pages 298–305, 2003.