Using Rank Aggregation in Continuously Answering SPARQL Queries on Streaming and Quasi-static Linked Data

Zahmatkesh, Shima ; Della Valle, Emanuele ; Dell'Aglio, Daniele

DOI: https://doi.org/10.1145/3093742.3093926

Using Rank Aggregation in Continuously Answering SPARQL Queries on Streaming and Quasi-static Linked Data

Shima Zahmatkesh
DEIB – Politecnico di Milano
P.za Leonardo da Vinci, 32
Milan, Italy 20133
shima.zahmatkesh@polimi.it

Emanuele Della Valle
DEIB – Politecnico di Milano
P.za Leonardo da Vinci, 32
Milan, Italy 20133
emanuele.dellavalle@polimi.it

Daniele Dell'Aglio
IFI – University of Zurich
Binzmühlestrasse 14
Zurich, Switzerland
dellaglio@ifi.uzh.ch

ABSTRACT

Web applications that combine dynamic data stream with distributed background data are getting a growing attention in recent years. Answering in a timely fashion, i.e., reactiveness, is one of the most important performance indicators for those applications.

The Semantic Web community showed that RDF Stream Processing (RSP) is an adequate framework to develop this type of applications. However, RSP engines may lose their reactiveness due to the time necessary to access the background data when it is distributed over the Web. State-of-the-art RSP engines remain reactive using a local replica of the background data, but it progressively becomes stale if not updated to reflect the changes in the remote background data. For this reason, recently, the RSP community has investigated maintenance policies of the local replica that guarantee reactiveness while maximizing the freshness of the replica. Previous works simplified the problem with several assumptions.

In this paper, we investigate how to remove some of those simplification assumptions. In particular, we target a class of queries for which multiple policies may be used simultaneously and we show that rank aggregation can be effectively used to fairly consider their alternative suggestions. We provide extensive empirical evidence that rank aggregation is key to move a step forward to the practical solution of this problem in the RSP context.

CCS CONCEPTS

• **Information systems** → *Stream management*; *Rank aggregation*;

KEYWORDS

Continuous SPARQL Query Processing, Rank Aggregation, RDF stream, Linked Data

ACM Reference format:
Shima Zahmatkesh, Emanuele Della Valle, and Daniele Dell'Aglio. 2017. Using Rank Aggregation in Continuously Answering SPARQL Queries on Streaming and Quasi-static Linked Data. In *Proceedings of DEBS '17, Barcelona, Spain, June 19-23, 2017,* 10 pages.
DOI: 10.1145/3093742.3093926

DEBS '17, Barcelona, Spain

DOI: 10.1145/3093742.3093926

1 INTRODUCTION

Application domains that require to combine in a timely fashion data stream with distributed background data are getting a growing attention in recent years. For instance, in social content marketing, advertisement agencies may want to propose viral contents to influencers (e.g., users with more than 100,000 followers) when they are mentioned in micro-posts across Social Networks. Responding in timely fashion (a.k.a, being reactive) is the most important requirement in this case, because *i)* followers have few minutes of the attention span and *ii)* competitors may try to reach those influencers before us.

However, the time to access and fetch the distributed background data can be so high to put the application at risk of losing reactiveness. This is especially true when the distributed background data is *quasi static*; e.g., in the example above, the number of followers of the influencers, which are in the background data, is more likely to change when they are mentioned in the stream.

The Semantic Web community showed that RDF Stream Processing (RSP) [17] is an adequate framework to develop this type of applications: RSP engines can receive and process stream items as well as use federated SPARQL extension to access distributed background data [1].

For instance, Listing 1 shows how the example above can be declared in the context of RSP using the syntax proposed in [6]. Line 1 registers the query in the RSP engine. Line 2 describes how to construct the results. Line 3 tells the engine to open a window W on a stream S and that W is 10 minutes long and slides every minute. At each evaluation, the WHERE clause at lines 4-6 is matched against the data in the window W and in the remote service *BKG*. Lines 4 select from the window W, the number of mentions. Lines 5 ask the remote service *BKG* to select the number of followers for the users mentioned in the window. Line 6 filters out, from the results of the previous join, all those users whose number of followers is above 100,000.

```
1 REGISTER STREAM <:InfluencersToContact> AS
2 CONSTRUCT {?user a :influentialUser}
3 FROM NAMED WINDOW W ON S [RANGE 10m STEP 1m]
4 WHERE{ WINDOW W {?user :hasMentions ?mentionsNum}
5   SERVICE BKG  {?user :hasFollowers ?fCount }
6   FILTER (?fCount > 100,000)  }
```

Listing 1: Sketch of the query studied in the problem

Not surprisingly, also RSP engines may loose reactiveness when they need to access the background data distributed over the Web.

For this reason, in 2015, Dehghanzadeh et al. [5] started investigating approximate continuous query answering over streams and dynamic Linked Data sets (shortly named ACQUA in the remainder of this paper). Intuitively, the ACQUA approach proposes to keep a replica of the results of the federated SPARQL endpoint which gives access to the distributed background data. At each evaluation (in the example query, once per minute) only a subset of all the data items in the replica is refreshed according to an update policy. A *refresh budget* allows to control the number of refreshes. Keeping the refresh budget small guarantees that the RSP engine is reactive, but some data in the replica can become stale.

In 2016, Zahmatkesh et al. [19] extended ACQUA to optimize the class of queries that include the filtering of the intermediate results obtained from the federated SPARQL endpoint. The intuition is simple, it is useless to refresh data items that are not likely to pass the filter condition; it is better to focus on a *band* around the condition. For instance, for the query in Listing 1, Zahmatkesh et al. focus on the band $?fCount \in [90000, 110000]$. This new approach, first determines the data items that fall in the band (namely, the Filter Update Policy) and, then, applies one of the ACQUA policies on those data items. In the remainder of the paper we collectively name ACQUA.F the policies obtained in this way. The results of their evaluation shows that ACQUA.F policies outperform ACQUA ones, when the selectivity of the filter clause is high.

In this paper, we further investigate the ACQUA.F approach by removing the assumption that it is possible to determine a priori the band to focus on. The intuition of the proposed approach is straightforward. Instead of applying in a pipe the Filter Update Policy and one of the ACQUA policies, we let each policy express its *opinion* by ranking data items according to its criterion and, then, we use rank aggregation [8] to take fairly into account all opinions.

For this reason our research question is: *can we use rank aggregation to combine the ACQUA and the Filter Update policies so to continuously answer queries (such as the one in Listing 1) and to guarantee reactiveness while keeping the replica fresh (i.e., giving results with high accuracy)?*

In particular the contributions of this paper are the following:

- We provide empirical evidence that relaxing the ACQUA.F assumption is hard.
- We define three new policies (collectively named, ACQUA.F$^+$) that use rank aggregation to combine the Filter Update and the ACQUA policies.
- We empirically demonstrate on synthetic and real datasets that one of the new ACQUA.F$^+$ policies keeps the replica as fresh as the corresponding ACQUA.F one but without requiring to determine a priori the band to focus on.
- We empirically demonstrate that such a policy uses the same budget as the corresponding ACQUA.F policy.

This allows us to positively answer our research question and represents another significant step towards addressing the problem of combining in a timely fashion data stream with distributed background data in the RSP context.

The remainder of the paper is organized as follows. In Section 2 we introduce the relevant background information. Section 3 reviews the state of the art. Our proposed rank aggregation solutions are introduced in Section 4. Section 5 details the research hypotheses, introduces the experimental settings, reports on the evaluation of the proposed methods and discusses the practical insight we gathered. In Section 6 we review the related work and, finally, Section 7 concludes and discusses future work.

2 BACKGROUND

In this section introduces the background necessary to understand the paper. Section 2.1 introduces the key concepts of RDF Stream Processing, while Section 2.2 shortly introduce rank aggregation.

2.1 RDF Stream Processing

RDF Stream Processing (RSP) extends the RDF data model and query model considering the temporal dimension of data and evolution of data over time. In the following, we will briefly introduce the main definition that will be used in the rest of the paper. We adopt the definitions of RSP-QL [7], a reference model for RSP.

Data model. The RDF model is defined as atemporal; RSP-QL therefore needs to introduce into it the notion of time.

An *RDF stream S* is a potentially unbounded sequence of timestamped data items (d_i, t_i):

$$S = (d_1, t_1), (d_2, t_2), \ldots, (d_n, t_n), \ldots,$$

where d_i is an RDF statement, t_i the associated time instant and, for each item i, it holds $t_i \leq t_{i+1}$, i.e. stream items are in a non-decreasing time order. An RDF statement is a triple $(s, p, o) \in (I \cup B) \times (I) \times (I \cup B \cup L)$, where I is the set of IRIs, B is the set of blank nodes and L is the set of literals [14].

With *background data* RSP-QL refers to the data stored in repositories or embedded in Web pages that can be accessed through SPARQL endpoints i.e., APIs that answer SPARQL query over the Web [3]. Compared to RDF streams, background data changes very slowly over time or does not change at all.

In this case, the time dimension is pushed through the notions of time-varying and instantaneous graphs. The time-varying graph is a function that maps time instants to RDF graphs and instantaneous graph is the value of the graph at a fixed time instant t.

Query model. Time dimension also affects the query model. The main change is on the evaluation paradigm, that moves from one-time evaluation to continuous one.

A SPARQL query [16] is defined through a triple (E, DS, QF), where E denotes the algebraic expression, DS the data set and QF the query form. RSP-QL extends SPARQL to process RDF streams optionally in combination with background data. An RSP-QL query is defined by a quadruple (SE, SDS, ET, QF), where SE is an RSP-QL algebraic expression, SDS is an RSP-QL dataset, ET is the sequence of evaluation time instants, and QF is the query form.

A *time-based sliding window* $\mathbb{W}$ [7] determines a subset of the RDF stream to be taken into account at evaluation time t. $\mathbb{W}$ takes an RDF stream S as input and produces a time-varying graph $G_{\mathbb{W}}$. The sliding window $\mathbb{W}$ is defined through the parameters (ω, β, t^0): where ω and β are width and slide of windows and t^0 is the time stamp on which $\mathbb{W}$ starts to operate. Each window contains a portion of RDF statements, that can be viewed as an RDF graph.

An extended RSP-QL dataset SDS is a set composed by a default graph $\tilde{G}_0$, n named graphs $\{(u_i, \tilde{G}_i)\}$, where u_i is IRIs and m named time-based sliding window over an RDF stream $\{(u_j, \mathbb{W}_j(S_k))\}$.

An algebraic expression *SE* is a streaming graph pattern which is the extension of graph pattern expression defined by SPARQL. As in SPARQL, the evaluation of *SE* returns solution mappings. RSP-QL adds a set of *streaming operators (RStream, IStream and DStream), to transform those solution mappings in an output stream (i.e., timestamped solution mappings).

Considering I, the set of IRIs, B, the set of blank nodes, L, the set of literals, and V, the set of variables, streaming graph pattern expressions are recursively defined as follows:

- a basic graph pattern (i.e. set of triple patterns $(s, p, o) \in (I \cup B \cup V) \times (I \cup V) \times (I \cup B \cup L \cup V)$) is a graph pattern;
- let P be a graph pattern and F a built-in condition, P *FILTER* F is a graph pattern;
- let P_1 and P_2 be two graph patterns, P_1 *UNION* P_2, P_1 *JOIN* P_2 and P_1 *OPT* P_2 are graph patterns;
- let P be a graph pattern and $u \in (I \cup V)$, the expressions *SERVICE* u P, *GRAPH* u P and *WINDOW* u P are graph patterns;
- let P be a graph pattern, *RStream* P, *IStream* P and *DStream* P are streaming graph patterns.

In RSP-QL, the evaluation of graph pattern expressions produces sets of solution mappings; a solution mapping is a function that maps variables to RDF terms, i.e., $\mu : V \rightarrow (I \cup B \cup L)$. Let $dom(\mu)$ be the subset of V where μ is defined: two solution mappings μ_1 and μ_2 are **compatible** ($\mu_1 \sim \mu_2$) if the two mappings assign the same value to each variable in $dom(\mu_1) \cap dom(\mu_2)$.

2.2 Rank Aggregation

In many circumstances, there is the need to rank a list of alternative options (namely, candidates) according to multiple criteria. For instance, in many sports, the ranking of the athletes is based on the individual scores given by several judges. The problem of computing a single rank, which fairly reflects the opinion of many judges, is called *rank aggregation* [8].

Several methods exist to solve the rank aggregation problem. In this paper, we use Borda's method [4]. This method is positional. It asks each judge to rank candidates according to a numerical criterion, so that each position in the ranked list has a score. Then, the candidates are ranked by their total score, e.g., by the weighted sum of the scores given by the individual judges. We choose this method because the problem, which we address in this paper, requires to minimize the time we spend in any computation and Borda's method is computationally easy. A naïve algorithm can solve the rank aggregation problem using Borda's method in linear time and algorithms exist that can solve it in sub-linear time, e.g. the Threshold Algorithm [9].

Other methods exist to handle cases where not all the judges can give a score to all the candidates or the case where some judges are biased or even malicious. However, those methods are computationally more expensive and handle problems that do not appear in our rank aggregation scenario.

3 STATE OF THE ART

As we have already stated in Section 1, RDF Stream Processing offers solutions to integrate and process the distributed data resources on the Web. While RSP engines can receive and process stream items, they also can use federated SPARQL extension to access background data stored behind SPARQL endpoints [1]. The time to access and fetch the remote background data can be so high to put the RSP engine at risk of violating the reactiveness requirement in continuous query answering.

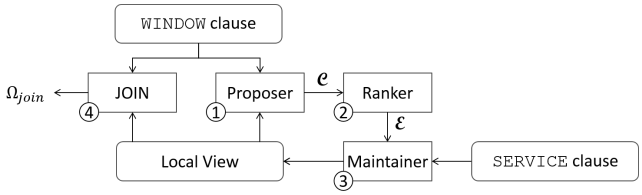


Figure 1: The framework proposed in [5] to address the problem of joining streaming and remote background data.

3.1 Approximate Continuous Query Answering in RSP

ACQUA [5] presents the first attempt to attack this problem. The intuition of the paper is straightforward: the RSP engine must avoid to access the whole remote background data at each evaluation. Instead, it uses a local replica of the background data and keeps it fresh using a maintenance policy that refreshes only a minimum subset of the replica. A maximum number of fetches (namely a **refresh budget** denoted with γ) can be given to the RSP engine to guarantee its reactiveness. If γ fetches are enough to refresh all stale data of replica the RSP engine gives correct answer, otherwise some data becomes stale and it gives an approximated answer.

Specifically, ACQUA addresses the problem of optimizing the evaluation of a class of RSP-QL queries where the streaming data is obtained by a window identified by the IRI $u_{\mathcal{S}}$, the background data is available via a SPARQL service at the URL $u_{\mathcal{B}}$ and the algebraic expression SE contains the following graph patterns:

$$(WINDOW\ u_{\mathcal{S}}\ P_{\mathcal{S}})\ JOIN\ (SERVICE\ u_{\mathcal{B}}\ P_{\mathcal{B}}),$$

where $P_{\mathcal{S}}$ and $P_{\mathcal{B}}$ are graph patterns that do not contain other SERVICE, GRAPH or WINDOW clauses.

Let us denote with $\Omega^{\mathcal{S}}$ and $\Omega^{\mathcal{B}}$ the bags of mappings returned by the WINDOW and SERVICE clauses, respectively. Let $\Omega^{\mathcal{R}}$ be the local replica of $\Omega^{\mathcal{B}}$. The maintenance process introduced in ACQUA is composed of three elements: a proposer, a ranker and a maintainer (Figure 1). (1) The **proposer** selects from $\Omega^{\mathcal{R}}$ a set C of candidate mappings for the maintenance; (2) the **ranker** orders C by using some relevancy criteria; (3) the **maintainer** refreshes the elected set $\mathcal{E}$ (the top γ elements of C) picking them from $\Omega^{\mathcal{B}}$; (4) the join operation is performed after maintenance on $\Omega^{\mathcal{S}}$ and $\Omega^{\mathcal{R}}$.

ACQUA introduces several maintenance policies. The best results are obtained combining the WSJ algorithm for the proposer and the WBM algorithm for the ranker. WSJ selects from the replica the mappings compatible with the ones returned by the WINDOW clause to generate the candidate set. WBM identifies the mappings that *i*) are going to be used in the upcoming evaluations and *ii*) allow saving future refresh operations. WBM uses the **best before time**, i.e. an estimation of the time on which one mapping in $\mathcal{R}$ would become stale, and the **remaining life time**, i.e. the number

of future evaluations that involve the mapping μ_i to assign scores and order the candidate set.

ACQUA also introduces two baseline rankers to be used in the evaluation: RND, which randomly ranks the mappings in the candidate set, and LRU, which orders C by the time of the last refresh of the mappings.

3.2 ACQUA with Filter Update Policy

ACQUA.F [19] extends the class of queries investigated by ACQUA optimizing algebraic expression SE of the following form:

$$(WINDOW\ u_{\mathcal{S}}\ P_{\mathcal{S}})\ JOIN\ ((SERVICE\ u_{\mathcal{B}}\ P_{\mathcal{B}})\ FILTER\ F),$$

where F is either $(?x < \mathcal{FT})$ or $(?x > \mathcal{FT})$, $?x$ is a variable in $P_{\mathcal{B}}$ and $\mathcal{FT}$ is the **Filtering Threshold**.

ACQUA.F introduces the Filter Update Policy for maintaining $\Omega^{\mathcal{R}}$ that contains the replica of $\Omega^{\mathcal{B}}$ (the results of the SERVICE clause). In the maintenance process, the set C of candidate mappings is selected by the WSJ proposer of ACQUA, while the Filter Update Policy acts as a ranker. For each mapping $\mu^{\mathcal{R}} \in \Omega^{\mathcal{R}}$, it computes the Filtering Distance

$$FD(\mu^{\mathcal{R}}) = |\mu^{\mathcal{R}}(?x) - \mathcal{FT}| \tag{1}$$

Then, it ranks the mappings in $\Omega^{\mathcal{R}}$ by FD and selects the top γ ones for refreshing the replica. Finally, the maintainer updates the elected mappings $\mathcal{E}$. Experiments show that the Filter Update Policy is the best policy when the FILTER clause has a high selectivity (greater than 60%) for all the tested refresh budgets.

In order to find an update policy which is less sensible to selectivity and gives better results, [19] proposed a combination of the Filter Update Policy with ACQUA policies, namely the WBM.F policy and the baseline LRU.F and RND.F policies. In the maintenance process of the combined approach, first, the proposer generates the candidate set C, then the Filtering Distance of each mapping is computed and, if the value is greater than a given Filtering Distance Threshold FDT (i.e., it falls out of a given *band*), the mapping is eliminated from the candidate set. In the next step, the update policy (from ACQUA policies) ranks the remaining mappings and selects the set $\mathcal{E}$ to be refreshed. The results of experiments show that for selectivity grater than 40%, LRU.F is the best policy, while for other selectivities, WBM.F is the best one.

As we have already pointed out in Section 1, ACQUA.F assumes that it is possible to determine a priori the band to focus on, i.e., the optimal value of the Filtering Distance Threshold. Our experimental study, in Section 5.4, shows that relaxing this assumption is hard.

4 RANK AGGREGATION SOLUTION

In this section, we introduce our proposed solution to the problem of combining in a timely fashion data stream with distributed background data in the context of RSP.

In Section 4.1, we introduce the idea of using rank aggregation to combine the ACQUA and the Filter Update Policy in order to reactively answer continuous queries while keeping the replica fresh (i.e., giving results with high accuracy). In Section 4.2, we show how to apply this idea in combining the LRU and WBM policies with the Filter Update Policy to obtain $LRU.F^+$ $WBM.F^+$, respectively. In Section 4.3, we elaborate on a different method to combine WBM and the Filter Update Policy to obtain $WBM.F^*$

4.1 Overall idea

ACQUA.F applies the Filter Update Policy and the ACQUA policies in a pipe. In this way, the *opinion* of the Filter Update Policy is more relevant than the one of ACQUA policies. This gives good result when focusing on a band around the $\mathcal{FT}$ minimizes the number of stale data. However, when the selectivity of the filter condition is low, focusing on such a band is inconvenient.

Rank aggregation was shown to be an adequate solution in similar settings where there was the need to take fairly into account the opinions of different algorithms.

In our proposed solution, we use WSJ proposer from [5] to select the candidate set C of mappings for the maintenance. As a ranker, we use rank aggregation to combine the ranking obtained by ordering the mappings in C according to the scores computed by each policy. Specifically, a weight (denoted with α) allows computing an aggregated score as follows:

$$score_{agg} = \alpha * score_{list-1} + (1 - \alpha) * score_{list-2} \tag{2}$$

The aggregated list is ordered by the score $score_{agg}$. In the next steps we follow [5], the maintenance process selects the top γ ones from ordered list to create the elected set $\mathcal{E} \subseteq C$ of mappings to be refreshed. Finally, the maintainer refreshes the mappings in set $\mathcal{E}$.

4.2 ACQUA.F$^+$ Policy

This section presents an algorithm to combine Filter Update policy with ACQUA policies, respectively, named $LRU.F^+$, and $WBM.F^+$.

In our new combined approach, the proposer selects a set C of candidate mappings for the maintenance, then the proposed policy receives as input the parameter α and the two ranked lists of mappings $C\mathcal{L}$ generated by ACQUA and Filter Update policies, and it generates a single ranked list of mappings.

Algorithm 1 shows the pseudo-code of the proposed policy. For each mapping in the candidate set C it computes the Filtering Distance as the absolute difference of the value $?x$ of mapping $\mu^{\mathcal{R}}$ and the Filtering Threshold $\mathcal{FT}$ in the query (Lines 1–3). Then, it orders the set C based on the Filtering Distance and generate the ranked list $C\mathcal{L}_f$ (Line 4). In the next step, based on the selected policy from ACQUA, the algorithm computes the score for each mapping in the candidate set C (Lines 5–7), and orders the candidate set based on the scores to generate the ranked list $C\mathcal{L}_{acqua}$ (Line 8).

For $LRU.F^+$ policy, the algorithm computes the refresh time for each mapping in the candidate set C, and generates scores based on the least recently refreshed mappings. For $WBM.F^+$ policy, for each mapping in the candidate set C, the remaining life time, the renewed best before time, and the WBM score are computed to order the candidate set.

Given the parameter α and the two ranked lists $C\mathcal{L}_f$ and $C\mathcal{L}_{acqua}$, the function *AggregateRanks* generates a single ranked list $C\mathcal{L}_{agg}$ aggregating the scores of two lists (Line 9). The set of elected mappings $\mathcal{E}$ is created by getting the top γ ones from $C\mathcal{L}_{agg}$ (Line 10). Finally, the local replica $\mathcal{R}$ is maintained by invoking the SERVICE operator and querying the remote SPARQL endpoint to get fresh mappings and replace them in $\mathcal{R}$ (Lines 11–14).

Algorithm 1: The pseudo-code of the $ACQUA.F^+$

1 **foreach** $\mu^{\mathcal{R}} \in C$ **do**
2 | $FD(\mu^{\mathcal{R}}) = |\mu^{\mathcal{R}}(?x) - \mathcal{FT}|$;
3 **end**
4 $C\mathcal{L}_f$ = order C w.r.t. the value of $FD(\mu^{\mathcal{R}})$;
5 **foreach** $\mu^{\mathcal{R}} \in C$ **do**
6 | compute the score of $\mu^{\mathcal{R}}$ based on the selected policy from ACQUA ;
7 **end**
8 $C\mathcal{L}_{acqua}$ = order C w.r.t. the generated scores;
9 $C\mathcal{L}_{agg} = AggregateRanks(\alpha, C\mathcal{L}_f, C\mathcal{L}_{acqua})$;
10 $\mathcal{E}$ = first γ mappings of $C\mathcal{L}_{agg}$;
11 **foreach** $\mu^{\mathcal{R}} \in \mathcal{E}$ **do**
12 | μ^S = ServiceOp.next(JoinVars($\mu^{\mathcal{R}}$));
13 | replace $\mu^{\mathcal{R}}$ with μ^S in $\mathcal{R}$;
14 **end**

4.3 WBM.F* Policy

This section introduces $WBM.F^*$, an improved version of $WBM.F^+$. It considers that the candidate set C in WBM algorithm has two subsets that include the "Expired" and the "Not Expired" mappings, respectively. WBM uses the refresh budget only to update the mappings from the "Expired" set.

The proposed $WBM.F^*$ algorithm computes the "Expired" and "Not Expired" lists of WBM and accordingly, generates two ranked lists ordering them based on Filter Update Policy. Finally, using rank aggregation, $WBM.F^*$ generates two ranked lists, "Expired.agg" and "Not Expired.agg". $WBM.F^*$ Policy first selects mappings from "Expired.agg" list for updating, and if there is any remaining budget, selects mappings from "Not Expired.agg" list.

Algorithm 2 shows the pseudo-code of the $WBM.F^*$ policy. For each mapping in the candidate set C, the remaining life time, the renewed best before time, and the total score according to WBM are computed (Lines 1–5), Then, the "Expired" (Exp) and "Not Expired" ($NExp$) sets based on WBM are computed (Lines 6–7). The scores of mappings are used to generate the "Expired" ($ExpL$) and "Not Expired" ($NExpL$) ranked lists (Lines 8–9).

In the next step, for each mapping in the "Expired" set (Exp), it computes the Filtering Distance as the absolute difference of the value $?x$ of mapping $\mu^{\mathcal{R}}$ and the Filtering Threshold $\mathcal{FT}$ in the query (Lines 10–12). The Filtering Distance is also computed for each mapping in the "Not Expired" set ($NExp$) (Lines 13–15). Then, it orders two sets based on the Filtering Distance (Lines 16–17) and generates the ranked lists $ExpL_f$, and $NExpL_f$. Given parameter α, and lists of mappings, the function $AggregateRanks$ generates two aggregated ranked lists: "Expired.agg" ($ExpL_{agg}$) and "Not Expired.agg ($NExpL_{agg}$) (Lines 18–19).

The set of elected mappings $\mathcal{E}$ is created by getting the top γ ones from $ExpL_{agg}list$(Line 20). If there exists any remaining refresh budget, it gets the top mappings from $NExpL_{agg}$ list (Line 21–24). Finally, the local replica $\mathcal{R}$ is maintained by invoking the SERVICE operator and querying the remote SPARQL endpoint to get fresh mappings and replace them in $\mathcal{R}$ (Line 25–28).

Algorithm 2: The pseudo-code of the $WBM.F^*$

1 **foreach** $\mu^{\mathcal{R}} \in C$ **do**
2 | compute the remaining life time of $\mu^{\mathcal{R}}$;
3 | compute the renewed best before time of $\mu^{\mathcal{R}}$;
4 | compute the score of $\mu^{\mathcal{R}}$;
5 **end**
6 Exp = possible expired mapping of C;
7 $NExp = C - ExP$;
8 $ExpL_{wbm}$ = order Exp w.r.t. the scores;
9 $NExpL_{wbm}$ = order $NExp$ w.r.t. the scores;
10 **foreach** $\mu^{\mathcal{R}} \in Exp$ **do**
11 | $FD(\mu^{\mathcal{R}}) = |\mu^{\mathcal{R}}(?x) - \mathcal{FT}|$;
12 **end**
13 **foreach** $\mu^{\mathcal{R}} \in NExp$ **do**
14 | $FD(\mu^{\mathcal{R}}) = |\mu^{\mathcal{R}}(?x) - \mathcal{FT}|$;
15 **end**
16 $ExpL_f$ = order Exp w.r.t. the value of FD;
17 $NExpL_f$ = order $NExp$ w.r.t. the value of FD;
18 $ExpL_{agg} = AggregateRanks(\alpha, ExpL_f, ExpL_{wbm})$;
19 $NExpL_{agg} = AggregateRanks(\alpha, NExpL_f, NExpL_{wbm})$;
20 $\mathcal{E}$ = first γ mappings of $ExpL_{agg}$;
21 **if** $\gamma > sizeOf(ExpL_{agg})$ **then**
22 | $\mathcal{E}'$ = first (γ - sizeOf($\mathcal{E}$)) mappings of $NExpL_{agg}$;
23 | $\mathcal{E} = \mathcal{E}$ Union $\mathcal{E}'$;
24 **end**
25 **foreach** $\mu^{\mathcal{R}} \in \mathcal{E}$ **do**
26 | μ^S = ServiceOp.next(JoinVars($\mu^{\mathcal{R}}$));
27 | replace $\mu^{\mathcal{R}}$ with μ^S in $\mathcal{R}$;
28 **end**

5 EXPERIMENTS

This section reports on the results of the experiments we ran to evaluate the proposed policies. Section 5.1 formulates the research hypotheses that we tested. Section 5.2 introduces our experimental setting made of synthetic and real datasets. Section 5.3 provides empirical evidence that relaxing the ACQUA.F assumption is hard, i.e., it is hard to determine a priori the band to focus on. Sections 5.4 and 5.5 report on the evaluation of our methods w.r.t. the research hypotheses and discusses the practical insights we gathered.

5.1 Research Hypotheses

The space of evaluation, which we explore, has five dimensions:

(1) the proposed policies;
(2) the policies that we have to compare with;
(3) the selectivity of the filtering condition (10%, 20%, ..., 90%, 75%);
(4) the refresh budget available to the policies (between 1 and 7); and
(5) the parameter α that allows controlling how the rank aggregation combines ACQUA and Filter Update policies in LRU.F$^+$, WBM.F$^+$ and WBM.F* (0.167, 0.2, 0.333, 0.5, 0.0667, 0.833);

In order to explore this vast space, we first fix the budget to a value, which is not enough to eliminate all stale data, and we tested two hypotheses:

- Hp.1 For every selectivity LRU.F$^+$, WBM.F$^+$ and WBM.F* have the same accuracy of the corresponding ACQUA.F policy, but they do not require to determine a priori the band.
- Hp.2 For every selectivity LRU.F$^+$, WBM.F$^+$ and WBM.F* are not sensible to α, i.e., the parameter α that controls the rank aggregation can be set in a wide range of values without a significant impact on the accuracy.

In a second stage of the evaluation, we fix the selectivity and we tested two more hypotheses:

- Hp.3 For every budget LRU.F$^+$, WBM.F$^+$ and WBM.F* have the same accuracy of the corresponding ACQUA.F policy.
- Hp.4 For every budget LRU.F$^+$, WBM.F$^+$ and WBM.F* are not sensible to α

It is worth to note that we do not expect LRU.F$^+$, WBM.F$^+$ and WBM.F* to outperform the corresponding ACQUA.F policy, because rank aggregation can only consider the opinions of the policies it combines. In the best case, LRU.F$^+$, WBM.F$^+$ and WBM.F* can have the same accuracy of ACQUA.F. The important point is that they no longer rely on the ACQUA.F assumption that fixing the band around $\mathcal{FT}$ is easy.

5.2 Experimental Setting

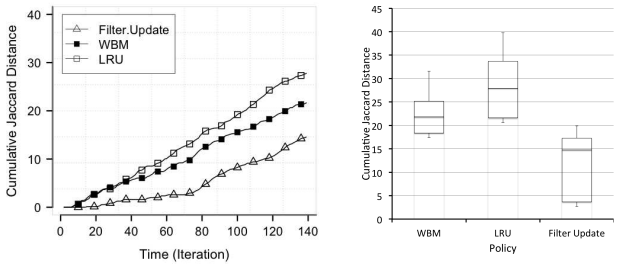
As experimental environment, we use an Intel i7 @ 1.7 GHz with 8 GB memory and a SSD disk. The operating system is Mac OS X 10.12.3 and Java 1.8.0.91 is installed on the machine. We carry out our experiments by extending the experimental setting of ACQUA.F [19] that in turn extends those presented in ACQUA [5].

The experimental datasets are composed of streaming and background data. The streaming data is a collection of tweets from 400 verified users for three hours. The background data consists of the number of followers per user collected every minute.

To control the selectivity of the filtering condition, [5] designed a set of transformations of the background data that randomly selects a specified percentage of the users and translates their time-series to be sure that it crosses the Filtering Threshold at least once during the experiment. In order to reduce the risk of bias in creating those realistic test datasets, 10 different datasets are generated for each percentage of the selectivity used in the experiments. In the remainder of the paper, we use the notation DSx% to refer to the *test case* that contains 10 datasets whose selectivity is x%.

In addition to DS10%, DS20%, ... DS90%, we also created six synthetic test cases, namely DEC40%, DEC70%, INC40%, INC70%, MIX40% and MIX70%. The percentage refers as above to the selectivity, while INC, DEC and MIX refers to how number of followers of each user evolves over time. In DEC the number of followers decreases. In INC, it always increases. In MIX, it randomly increases and decreases. In order to reduce the risk of introducing biases each synthetic test case contains 10 different datasets.

As a test query, we use the one presented in Section 1 and for each policy we run 140 iterations of the query evaluation, i.e., since the query has to be evaluated once per minute we simulated the time passing for 140 minutes.



(a) The Median over time

(b) Distribution of d_J^C at the end of the experiemnt, i.e., 140^{th} iteration

Figure 2: The two different viewpoints that can be used to illustrate the evaluation.

In order to investigate our hypotheses, we use the metric introduced in ACQUA.F [19]. We set up an Oracle that, at each iteration i, certainly provides correct answers $Ans(Oracle_i)$ and we compare its answers with the possibly erroneous ones of the query $Ans(Q_i)$. In our experiments, we use cumulative Jaccard distance at the k^{th} iteration $d_J^C(k)$ defined as:

$$d_J^C(k) = \sum_{i=1}^{k} d_J(Ans(Q_i), Ans(Oracle_i)) \tag{3}$$

where $d_J(Ans(Q_i), Ans(Oracle_i))$ is the Jaccard distance, which measure diversity of the two sets $Ans(Q_i)$ and $Ans(Oracle_i)$ at the i^{th} iteration. The lower value of cumulative Jaccard distance shows better performance of the query evaluation.

It is important to note that there are two viewpoints to show the results of the investigation of those hypotheses (Figure 2). The first viewpoint takes a time-series perspective and it allows comparing the accuracy of the various policies through the time for every evaluation. For instance, Figure 2(a) shows the medians of cumulative Jaccard distance over time for WBM, LRU and Filter Update policies when tested with DS75% and a refresh budget of 3. The plot shows that each policy has a constant behavior over time; for example, the Filter Update Policy is the best policy for each iteration.

The second viewpoint (Figure 2(b)) focuses on the distribution of the cumulative Jaccard distance at the end of the experiment (in the example at the 140^{th} iteration). It uses a box-plot [15] to highlight the median and the four quartiles of the accuracy obtained running the experiments with the 10 datasets in the DS75% test case. The box-plot shows that for first three quartiles the Filter Update policy is more accurate than all others; only the first quartile of WBM has a comparable accuracy. In this paper, we use only the second viewpoint to show the results of our experiments.

5.3 Relaxing ACQUA.F Assumption

In this experiment, we provide empirical evidence that relaxing the ACQUA.F assumption is hard. ACQUA.F assumes that it is simple to determine a priori the Filter Distance Threshold (FDT), i.e., the band around the Filtering Threshold to focus on.

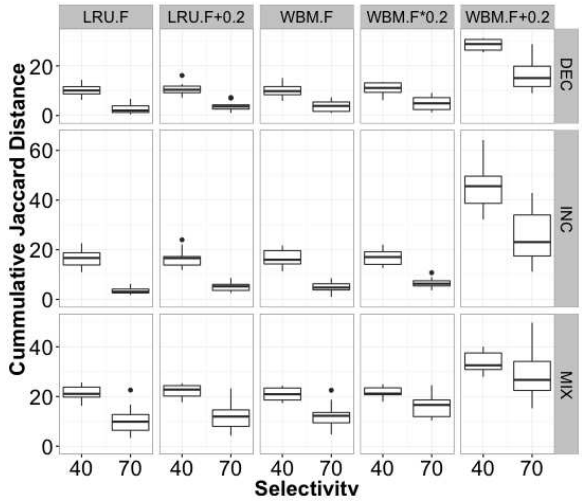


Figure 3: Result of experiment 1 that runs rank aggregation policies over synthetic datasets to compare them with existing policies for different selectivity.

To check if relaxing this assumption is easy, we test LRU.F and WBM.F policies with the DEC40%, DEC70%, INC40%, INC70%, MIX40%, and MIX70% test cases. We run each test case several times with different FDT values. The refresh budget γ is equal to 3.

Table 1: Summery of FDT value in case the policy has minimum Cumulative Jaccard Distance

	INC40%	INC70%	DEC40%	DEC70%	MIX40%	MIX70%
LRU.F	156	1000	250	625	500	500
WBM.F	500	1000	375	375	109	93

Table 1 summarizes for each policy and test case the value of FDT in which the cumulative Jaccard distance is smaller. The results show that relaxing the assumption of knowing FDT is hard. The ACQUA.F policies are sensitive to FDT and fixing a single FDT is not straightforward.

5.4 Experiment 1

In this experiment we test hypotheses Hp.1 and Hp.2 by checking the sensitivity to the filter selectivity for the proposed policies considering different value of α. Keeping the refresh budget γ equal to 3, we run experiments on both synthetic and real test cases. As explained in Section 5.2, we present the results using box-plots that capture the distribution of the cumulative Jaccard distance at the end of the experiment. The results are obtained using the 10 different datasets that are contained in each test case.

Figure 3 shows the results when using the synthetic datasets. Each column shows the results related to one policy and for two levels of selectivity (40% and 70%). Each row groups the experiments run using the same test case (DEC, INC, and MIX). To show the value of α used in the rank aggregation policies, we opt for the notation $< policy >< \alpha >$. For example $LRU.F^+0.2$ in the second column refers to $LRU.F^+$ policy when using $\alpha = 0.2$.

The results of the first two columns show that $LRU.F^+$ with α equal to 0.2 is comparable to LRU.F for all synthetic test cases (i.e., DEC, INC, and MIX). The results of columns three to five show that $WBM.F^*$ with $\alpha = 0.2$ is comparable to WBM.F, while $WBM.F^+$ with $\alpha = 0.2$ is worst than WBM.F.

Overall the results allow us to say that, w.r.t. the synthetic test cases, Hp.1 is verified only by $LRU.F^+$ and $WBM.F^*$. In other words, the best approach for combining WBM and Filter Update policies is considering the "Expired" and "Not Expired" lists of WBM in rank aggregation algorithm (see Algorithm 2).

Figure 4 shows the obtained results over the test cases DS10%, DS20%, ..., and DS90%. Each column shows the results related to one policy for different selectivities. The first three columns show the results of Filter Update, ACQUA and ACQUA.F policies, respectively. Columns four to eight show the results for our proposed rank aggregation policies for five different values of α.

Figure 4(a) compares the proposed $LRU.F^+$ policy with Filter Update, LRU, and LRU.F. Independently from the selectivity, $LRU.F^+$ with $\alpha = 0.167$ is as accurate as LRU.F and remains better than Filter Update and LRU. This verifies Hp.1 w.r.t. $LRU.F^+$ on real data. The results of column four to eight also show that $LRU.F^+$ little sensitive to α in the range 0.167 and 0.5, while for larger α it becomes less accurate than LRU.F. This verifies Hp.2 w.r.t. $LRU.F^+$.

The experiments on the synthetic and the real test cases shows that $LRU.F^+$ can have practical value, because it works for all selectivities and for a wide range of values of α.

Figure 4(b) allows comparing the proposed $WBM.F^+$ with different value of α with Filter Update, WBM, and WBM.F. The box-plots show that $WBM.F^+$ is less accurate than WBM.F, and Filter Update, and it is little sensitive to α in the range 0.167 and 0.5. Therefore, Hp.1 is not verified for $WBM.F^+$, but Hp.2 is.

From a practical perspective, we learned that merging the two lists of "Expired" and "Not Expired" mappings in the WBM algorithm can badly affect the result. $WBM.F^+$ is of no practical usage.

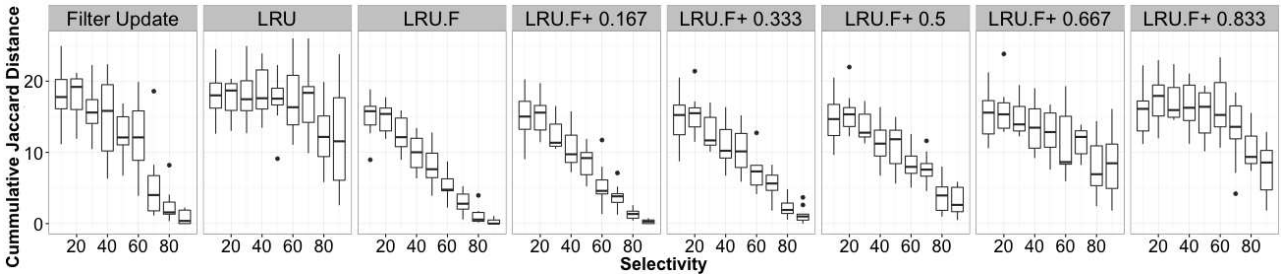
Figure 4(c) allows comparing the proposed $WBM.F^*$ with different values of α with Filter, WBM, and WBM.F. $WBM.F^*$ is as accurate as WBM.F for selectivities smaller than 60% and it is little sensitive to α. Accordingly, Hp.2 is verified w.r.t. $WBM.F^*$, but Hp.1 is only partially verified for low selectivities.

From a practical perspective, it is worth observing that $WBM.F^*$ with $\alpha = 0.167$ is: *i*) always better than WBM (i.e., the best policy in ACQUA) and *ii*) better than $LRU.F^+$ for low selectivity. Having to choose a policy, $LRU.F^+$ is the one that on average gives the best accuracy, but having the possibility to estimate the selectivity at run time, it would be better to use $WBM.F^*$ for low selectivities (<60%) and $LRU.F^+$ for high selectivities (≥60%).

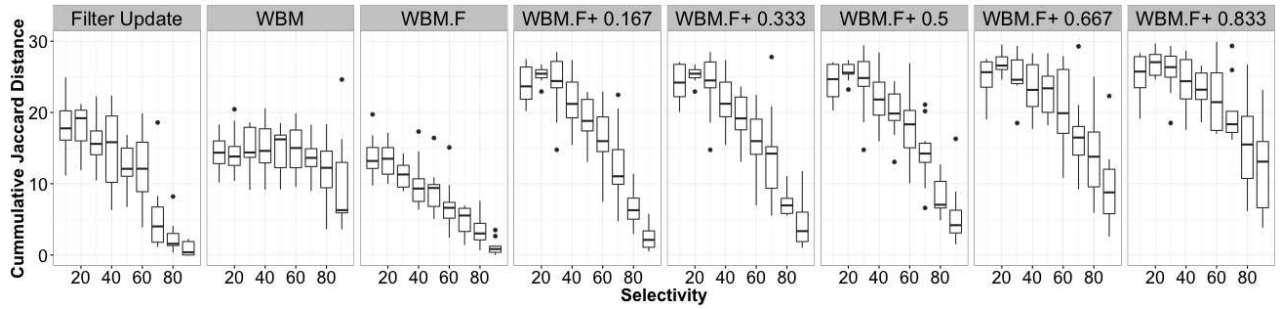
5.5 Experiment 2

In this experiment we test Hp.3 and Hp.4 by investigating the sensitivity to the refresh budget γ for the proposed policies and for different values of α.

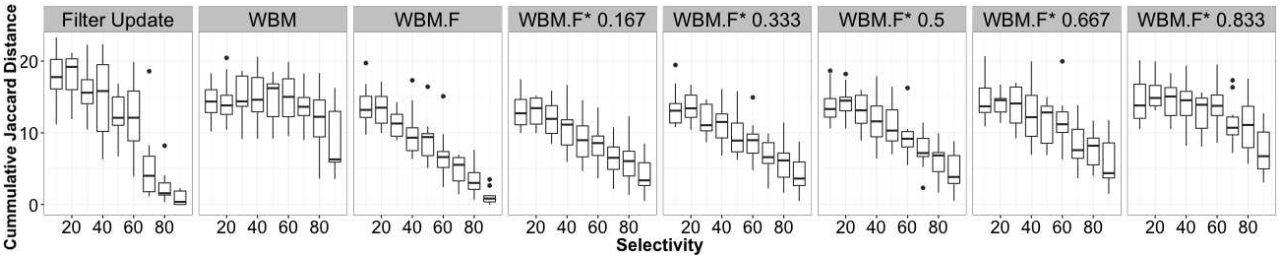
We run experiments using a subset of the test cases introduced in Section 5.2. For synthetic data, the experiments run for refresh budget 3 and 5 over the DEC70%, INC70%, and MIX70% test cases.



(a) $LRU.F^+$



(b) $WBM.F^+$



(c) $WBM.F^*$

Figure 4: Result of experiment 1 that runs rank aggregation policies over real datasets to compare them with existing policies for different selectivity.

For the real data the refresh budget varies from to 1 to 7 and the experiments run over DS75% test case.

We choose to fix selectivity to 70% for the synthetic data and 75% for the real data, because, according to the results reported in Section 5.4, this is the smallest value of selectivity for which $LRU.F^+$, $WBM.F^+$, and $WBM.F^*$ have comparable accuracy.

Figure 5 shows the results obtained using the synthetic test cases. The first two columns allow asserting that the result of LRU.F and $LRU.F^+$ with $\alpha = 0.2$ are comparable. The columns three to five show that $WBM.F^*$ and $WBM.F^+$ with $\alpha = 0.2$ are worst than WBM.F. In particular, $WBM.F^+$ is worst for both budget, whereas $WBM.F^*$ seems unable to use additional budget (i.e., the accuracy with budget 5 is similar to the accuracy with budget 3). Therefore, Hp.3 is verified for $LRU.F^+$, but not for $WBM.F^+$ and $WBM.F^*$.

This observation provides an additional insight on $WBM.F^+$ and $WBM.F^*$. In discussing Hp.1 in Section 5.4, we note that $WBM.F^*$ is more accurate than $WBM.F^+$ for budget 3, but here we discover that apparently giving more budget to $WBM.F^*$ does not turn in more accurate results.

Turning to real data (see Figure 6) confirms the insight we gathered using synthetic data: $LRU.F^+$ is comparable with LRU.F (Figure 6(a)) while $WBM.F^+$ and $WBM.F^*$ are worst than WBM.F. (Figures 6(b) and 6(c)). $WBM.F^*$ is not able to use all the budget when it is greater than 3. On the contrary $WBM.F^+$, given a high budget, becomes comparable to WBM.F. Therefore Hp.3 is verified for $LRU.F^+$, partially verified for $WBM.F^+$ for budget greater than 5, and not verified for $WBM.F^*$.

As a marginal note, Hp.4 (sensitivity to α) is verified for all policies that use rank aggregation.

From a practical perspective, this analysis confirms that if one has to chose a policy $LRU.F^+$ is on average the best one. $WBM.F^*$ is a perfect solution only when the available budget is very low.

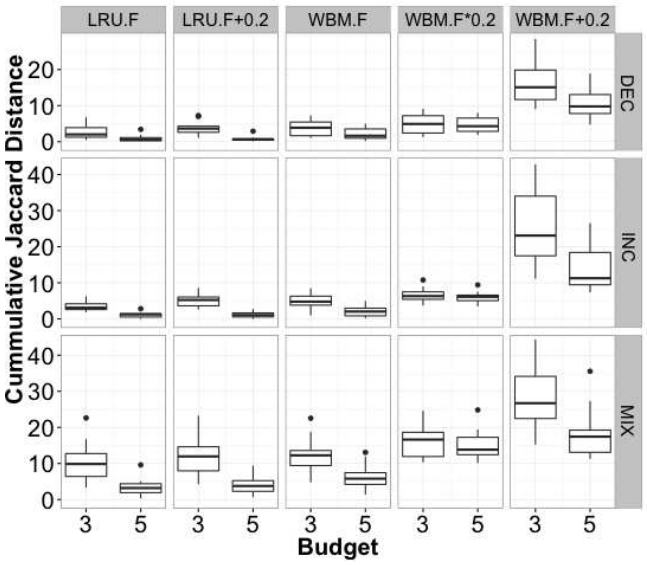


Figure 5: Result of experiment 2 that runs rank aggregation policies over synthetic datasets to compare them with existing policies for different refresh budget.

6 RELATED WORK

Many systems locally replicate data sources, which takes time to access, to improve their performance and availability. To get accurate answer, a maintenance process is needed to keep the local replicas fresh. Extensive studies exist about optimization and maintenance process in database community [2, 11, 13, 18]. However, those works still do not consider the problem of combing streaming data with distributed background data.

The only follow up work of ACQUA that we are aware of is [10]. It studied the maintenance process for a class of queries that extends the 1:1 join relationship of ACQUA work to M:N join, but that does not include FILTER clauses. It models the join between streams and background data as a bipartite graph.

7 CONCLUSIONS AND FUTURE WORK

In this work, we studied the problem of continuously evaluating queries that need to access distributed background data while processing a data stream. The time to access and fetch the distributed background data can be so high to put the application at risk of losing reactiveness.

RDF Stream Processing (RSP) is an adequate framework to study this problem because it can process data streams and it can access distributed background data using federated SPARQL. The ACQUA work proposed an approach to i) keep a replica of the results of the federated SPARQL endpoint and ii) use various maintenance policies to refresh such a replica. ACQUA.F, which is an extension of ACQUA, considers the class of queries that also includes a FILTER clause.

In this paper, we further investigate the ACQUA.F approach by removing the assumption that it is possible to determine a priori the band to focus on. We proposed new policies that use rank aggregation. Those new policies let each ACQUA.F policy express its opinion by ranking data items according to its own criterion and, then, aggregate those ranks to take fairly into account all opinions.

To study our research question, we formulate four hypotheses. In Hypotheses Hp.1 and Hp.3, we test if our proposed policies have the same accuracy of the corresponding ACQUA.F policies, without determining a priori the band to focus on. In Hypotheses Hp.2, and Hp.4, we test if the proposed policies are sensible to α. The results are reported in Table 2.

Table 2: Summary of the verification of the hypotheses w.r.t. $LRU.F^+$, $WBM.F^+$, and $WBM.F^*$.

	measuring	varying	$LRU.F^+$	$WBM.F^+$	$WBM.F^*$
Hp.1	accuracy	selectivity	✓		<60%
Hp.2	sensitivity to α	selectivity, α	✓	✓	✓
Hp.3	accuracy	budget	✓	> 5	
Hp.4	sensitivity to α	budget, α	✓	✓	✓

The results of experiment 1 (about Hp.1, and Hp.2) show that $LRU.F^+$ policy has the same accuracy of the LRU.F policy for every selectivity, and $WBM.F^*$ policy is comparable to WBM.F policy for low selectivity. They also show that the proposed policies are not sensible to α and $\alpha \in [0.167, 0.5]$ is acceptable for every selectivity.

The results of experiment 2 (about Hp.3, and Hp.4) show that $LRU.F^+$ policy has the same accuracy of the LRU.F policy for every budget, and $WBM.F^+$ policy is comparable to WBM.F policy for high value of budget. They also show that $WBM.F^*$ is not able to use all the budget, and even increasing the budget, the error does not go below a given minimum. Moreover, the proposed policies are not sensible to α.

In our future work, we intend to further optimize the approach by *i*) combining more than two policies, and *ii*) dynamically determine the best α to combine the opinions of the different policies, e.g., by dynamically estimating the selectivity of the filtering condition.

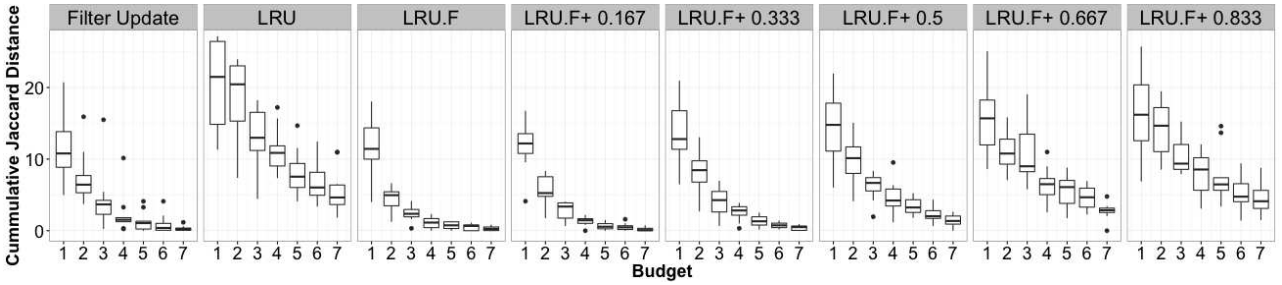
We also want to study the static optimization of pushing the FILTER clause(s) into the SERVICE clause and keeping a cache of recent results instead of a full replica.

Additionally, we intend to broaden the class of queries by exploring queries with multiple FILTER clauses and queries that contain a ranking clause [12] that involves variables present both in the WINDOW and in the SERVICE clauses.

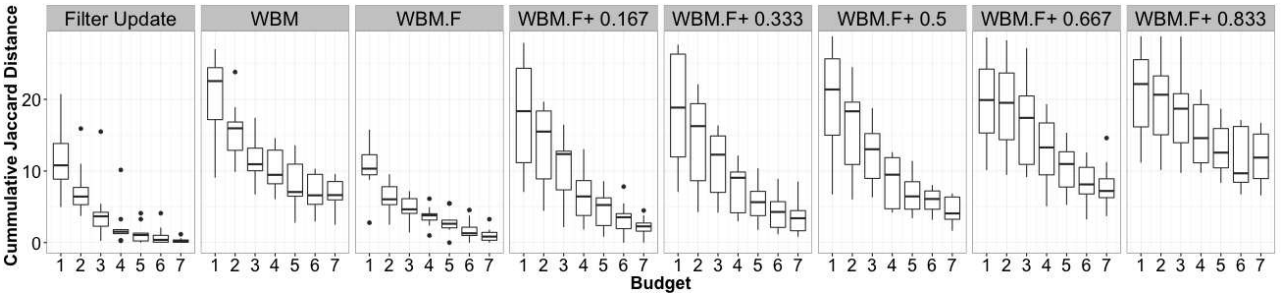
We also intend to study the effect of different trends in the data. For future work we would like to use data from other domains (e.g., sensor networks) with various characteristics such as flat/fast growing/shrinking of data rate and positive/negative correlation between growing/shrinking of data rate and changes in data.

REFERENCES

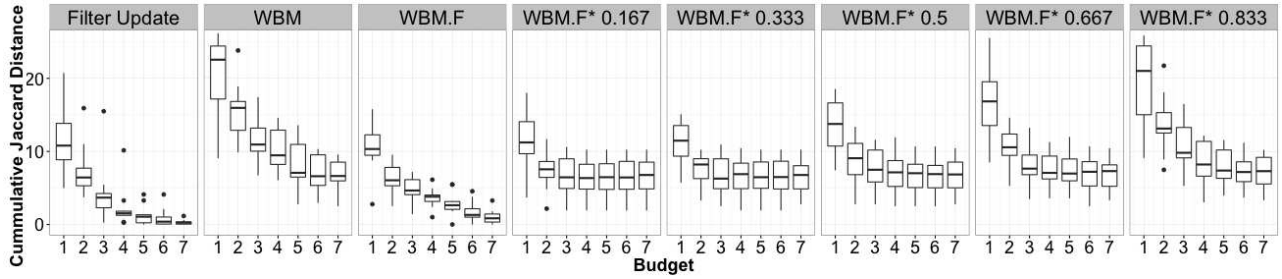
[1] Carlos Buil Aranda, Marcelo Arenas, Óscar Corcho, and Axel Polleres. 2013. Federating queries in SPARQL 1.1: Syntax, semantics and evaluation. *J. Web Sem.* 18, 1 (2013), 1–17.
[2] Shivnath Babu, Kamesh Munagala, Jennifer Widom, and Rajeev Motwani. 2005. Adaptive Caching for Continuous Queries. In *ICDE*. IEEE Computer Society, 118–129.
[3] Kendall Grant Clark, Lee Feigenbaum, and Elias Torres. 2008. SPARQL Protocol for RDF (W3C Recommendation 15 January 2008). *World Wide Web Consortium* (2008).
[4] Jean C de Borda. 1781. Mémoire sur les élections au scrutin. (1781).



(a) $LRU.F^+$



(b) $WBM.F^+$



(c) $WBM.F^*$

Figure 6: Result of experiment 2 that runs rank aggregation policies over real datasets to compare them with existing policies for different refresh Budget.

[5] Soheila Dehghanzadeh, Daniele Dell'Aglio, Shen Gao, Emanuele Della Valle, Alessandra Mileo, and Abraham Bernstein. 2015. Approximate Continuous Query Answering over Streams and Dynamic Linked Data Sets. In *ICWE (LNCS)*, Vol. 9114. Springer, 307–325.

[6] Daniele Dell'Aglio, Jean-Paul Calbimonte, Emanuele Della Valle, and Óscar Corcho. 2015. Towards a Unified Language for RDF Stream Query Processing. In *ESWC (Satellite Events) (LNCS)*, Vol. 9341. Springer, 353–363.

[7] Daniele Dell'Aglio, Emanuele Della Valle, Jean-Paul Calbimonte, and Óscar Corcho. 2014. RSP-QL Semantics: A Unifying Query Model to Explain Heterogeneity of RDF Stream Processing Systems. *Int. J. Semantic Web Inf. Syst.* 10, 4 (2014), 17–44.

[8] Cynthia Dwork, Ravi Kumar, Moni Naor, and D. Sivakumar. 2001. Rank aggregation methods for the Web. In *WWW*. ACM, 613–622.

[9] Ronald Fagin, Amnon Lotem, and Moni Naor. 2001. Optimal Aggregation Algorithms for Middleware. In *PODS*. ACM.

[10] Shen Gao, Daniele Dell'Aglio, Soheila Dehghanzadeh, Abraham Bernstein, Emanuele Della Valle, and Alessandra Mileo. 2016. Planning Ahead: Stream-Driven Linked-Data Access Under Update-Budget Constraints. In *International Semantic Web Conference (1) (LNCS)*, Vol. 9981. 252–270.

[11] Hongfei Guo, Per-Åke Larson, and Raghu Ramakrishnan. 2005. Caching with 'Good Enough' Currency, Consistency, and Completeness. In *VLDB*. ACM, 457–468.

[12] Ihab F. Ilyas, George Beskales, and Mohamed A. Soliman. 2008. A survey of top-k query processing techniques in relational database systems. *ACM Comput. Surv.* 40, 4 (2008), 11:1–11:58.

[13] Alexandros Labrinidis and Nick Roussopoulos. 2004. Exploring the tradeoff between performance and data freshness in database-driven Web servers. *VLDB J.* 13, 3 (2004), 240–255.

[14] Ora Lassila and Ralph R Swick. 1999. Resource description framework (RDF) model and syntax specification. *W3C* (1999).

[15] Robert McGill, John W Tukey, and Wayne A Larsen. 1978. Variations of box plots. *The American Statistician* 32, 1 (1978), 12–16.

[16] Jorge Pérez, Marcelo Arenas, and Claudio Gutierrez. 2009. Semantics and complexity of SPARQL. *ACM Trans. Database Syst.* 34, 3 (2009), 16:1–16:45.

[17] Emanuele Della Valle, Daniele Dell'Aglio, and Alessandro Margara. 2016. Taming velocity and variety simultaneously in big data with stream reasoning: tutorial. In *DEBS*. ACM, 394–401.

[18] Stratis Viglas, Jeffrey F. Naughton, and Josef Burger. 2003. Maximizing the Output Rate of Multi-Way Join Queries over Streaming Information Sources. In *VLDB*. Morgan Kaufmann, 285–296.

[19] Shima Zahmatkesh, Emanuele Della Valle, and Daniele Dell'Aglio. 2016. When a FILTER Makes the Difference in Continuously Answering SPARQL Queries on Streaming and Quasi-Static Linked Data. In *ICWE (LNCS)*, Vol. 9671. Springer, 299–316.