Strategies for Efficiently Keeping Local Linked Open Data Caches Up-To-Date

Renata Dividino[1(✉)], Thomas Gottron[1], and Ansgar Scherp[2]

[1] WeST – Institute for Web Science and Technologies,
University of Koblenz-Landau, Koblenz, Germany
dividino@uni-koblenz.de

[2] Kiel University and Leibniz Information Center for Economics, Kiel, Germany
asc@informatik.uni-kiel.de

Abstract. Quite often, Linked Open Data (LOD) applications pre-fetch data from the Web and store local copies of it in a cache for faster access at runtime. Yet, recent investigations have shown that data published and interlinked on the LOD cloud is subject to frequent changes. As the data in the cloud changes, local copies of the data need to be updated. However, due to limitations of the available computational resources (e.g., network bandwidth for fetching data, computation time) LOD applications may not be able to permanently visit all of the LOD sources at brief intervals in order to check for changes. These limitations imply the need to prioritize which data sources should be considered first for retrieving their data and synchronizing the local copy with the original data. In order to make best use of the resources available, it is vital to choose a good scheduling strategy to know when to fetch data of which data source. In this paper, we investigate different strategies proposed in the literature and evaluate them on a large-scale LOD dataset that is obtained from the LOD cloud by weekly crawls over the course of three years. We investigate two different setups: (i) in the single step setup, we evaluate the quality of update strategies for a single and isolated update of a local data cache, while (ii) the iterative progression setup involves measuring the quality of the local data cache when considering iterative updates over a longer period of time. Our evaluation indicates the effectiveness of each strategy for updating local copies of LOD sources, i. e, we demonstrate for given limitations of bandwidth, the strategies' performance in terms of data accuracy and freshness. The evaluation shows that the measures capturing change behavior of LOD sources over time are most suitable for conducting updates.

1 Introduction

Quite often, LOD applications pre-fetch data from the Web and store local copies of it in a cache or build an index over it to speed up access and search. Recent investigations [1,2,10,11,15,22] have shown that data published and interlinked on the LOD cloud is subject to frequent changes. As the data in the cloud changes, these caches or indices no longer reflect the current state of the data

M. Arenas et al. (Eds.): ISWC 2015, Part II, LNCS 9367, pp. 356–373, 2015.
DOI: 10.1007/978-3-319-25010-6_24

anymore and need to be updated. Käfer et al. [17] observed a subset of the LOD cloud over a period of 29 weeks and concluded (among others) that the data of 49.1% of the LOD sources changes. Likewise, Gottron et al. [12] observed LOD data over a period of 77 weeks and described that the accuracy of indices built over the LOD sources drops by 50% after already 10 weeks. These outcomes indicate that almost half of the LOD sources are not appropriate for long-term caching or indexing. Unquestionably, data on the LOD cloud changes and knowledge about these changes, i. e., about the change behavior of a dataset over time, is important as it affects various different LOD applications such as indexing of distributed data sources [18], searching in large graph databases [13], search optimization [19], efficient caching [8,14,24], and recommending vocabularies to Linked Data engineers [21].

LOD applications relying on data from the LOD cloud need to cope with constant data updates to be able to guarantee a certain level of quality of service. In an ideal setting, a cache or an index is kept up-to-date by continuously visiting all data sources, fetching the most recent version of the data and synchronizing the local copies with it. However, in real world scenarios LOD applications must deal with limitations of the available computational resources (e.g., network bandwidth, computation time) when fetching data from the LOD cloud. These limitations imply the necessity to prioritize which data sources should be first considered for retrieving their data. In order to make best use of the resources available, it is vital to choose a good scheduling strategy for updating local copies of the LOD data sources. While there exists research on the freshness analysis of the cached data for answering SPARQL queries in a hybrid approach such as Umbrich et al [24], to the best of our knowledge, there is no work addressing strategies to efficiently keep local copies of LOD source up-to-date.

Intuitively, a strategy dedicated to update data caches build out of data from the LOD cloud would make use of the HTTP protocol. The *Last-Modified* HTTP header field denotes when a LOD source behind this URI has been changed last. However, in a previous investigation [9], we showed that only very few LOD sources (on average only 8%) provide correct update values. Consequently, applications relying on such information are susceptible to draw wrong conclusions. Thus, this method is inappropriate for probing a LOD source for whether or not it has been changed since the last retrieval of its data. The only alternative is to actually retrieve the data from the sources and check it for changes.

In this paper, we consider update scheduling strategies for maintaining indices of web documents and metrics initially developed to capture the data changes in LOD sources and analyze their effectiveness for updating local copies of LOD sources. Scheduling strategies aim for deriving an order for data sources to when they should be visited. Consequently, the application updates its local copy by fetching data from the data sources following this order. The simplest strategy is to visit the data sources in an arbitrary but fixed order, which guarantees that the local copy of every data source is updated after a constant interval of time. Alternative strategies explore the different features provided by the data sources, e. g., their size, to assign an importance score to each data source, and

thus deriving an order. An established scheduling strategy for the Web leverages the PageRank algorithm [20], where a score of importance is given to each data source regarding its centrality in the link network with other data sources.

When considering a set of LOD sources, certainly some of them change more or less often than others [17]. For example, it is not likely that in a short time interval every LOD source changes. Thus, many sources may provide the same information during this entire interval. Therefore, is not necessary to fetch data from such sources. However, whenever data of a source changes, an update is required. Accordingly, some sources should be fetched at shorter/longer time-intervals. This implies that each LOD source could be given a different update importance, which is based on their change behaviour. We consider change metrics for LOD data sources presented in [11]. These metrics measure the change rate of a dataset based on the changes that have taken place between two points in time. Furthermore, in previous work [10], we propose the notion of dynamics of LOD datasets. The dynamics function measures the accumulated changes that have occurred within a data set in a given time interval. Even though these metrics were not directly proposed to support scheduling strategies for data updates, measures that capture the change rate or dynamics of a LOD dataset may indeed be used for conducting updates.

We evaluate different strategies on a large-scale LOD dataset from the Dynamic Linked Data Observatory (DyLDO) [17] that is obtained via 149 weekly crawls in the period from May 2012 until March 2015. We investigate two different setups: (i) in the *single step* setup we evaluate the quality of update strategies for a single and isolated update of a local data cache, while (ii) the *iterative progression* setup involves measuring the quality of the local data cache when considering iterative updates over a longer period of time. Quality is measured in terms of precision and recall with respect to the gold standard, i. e., we check the correctness of data of the (updated) local copy with respect to the data actually contained in the LOD cloud. We assume that only a certain bandwidth for fetching data from the cloud is available, and we investigate the effectiveness of each strategy for different bandwidths. Therefore, in the first setup, we can observe the relation between strategies and restrictions of bandwidth (i. e., if the strategies show comparatively uniform performance over all restrictions or if better/worse performance depends on a given restriction), and use such findings as parameters for the second setup. The second setup evaluates the behavior of the strategies in a realistic scenario (e. g., a LOD search engine updating its caches). Our evaluation indicates the most effective strategies for updating local copies of LOD sources, i. e., we demonstrate for given restrictions of bandwidth, which strategy performs better in terms of data accuracy and freshness.

2 Foundations

Linked Data that is crawled from the cloud can be represented in the form of N-Quads[1]. Technically, a quad (s, p, o, c) consists of an RDF triple where s, p, and o

[1] W3C Recommendation http://www.w3.org/TR/n-quads/

correspond to the subject, predicate and object and the context c, i. e., the data source on the Web where this RDF triple was retrieved.

We assume that data from the various LOD sources is retrieved at some fix point in time t. We consider that an application visits and fetches data of LOD sources at a regular interval (say, once a week). Consequently, a LOD data source is defined by a context c and the data it provides at points in time t, i. e., the set of RDF quads $X_{c,t}$. Furthermore, we denote the size of a data set with $|X_{c,t}|$ to indicate the number of triples contained in the data set at context c at the point in time t.

In this paper, we rely on local copies of the LOD sources. Such a copy typically covers several data sources. Given a set $C = \{c_1, c_2, \ldots, c_m\}$ of contexts of interest, we can define the overall dataset as:

Definition 1. *Dataset*

$$X_t = \bigcup_{c \in C} X_{c,t} \tag{1}$$

Example 1. As a matter of example, we consider that data comes from three different data sources: *dbpedia.org*, *bbc.co.uk*, and *musicbrainz.com*, and data is retrieved at two different points in time, on May 8th, 2013 and on June 10th, 2013.

$$X_{\text{2013-05-08}} = \{X_{\text{dbpedia.org,2013-05-08}}, X_{\text{musicbrainz.com,2013-05-08}}, X_{\text{bbc.co.uk,2013-05-08}}\},$$
$$X_{\text{2013-06-10}} = \{X_{\text{dbpedia.org,2013-06-10}}, X_{\text{musicbrainz.com,2013-06-10}}, X_{\text{bbc.co.uk,2013-06-10}}\}$$

Finally, for distinct points $t_1, t_2, \ldots, t_n$ in time, we define a series of datasets over time:

Definition 2. *Series of Datasets*

$$\mathcal{X} = (X_{t_1}, X_{t_2}, \ldots, X_{t_n}) \tag{2}$$

Example 2. Our example dataset is composed by data retrieved at two different points in time (see Example 1) such that $\mathcal{X} = (X_{\text{2013-05-08}}, X_{\text{2013-06-10}})$.

3 Update Scheduling Strategies

Due to limitations such as bandwidth restrictions and the frequent data changes in the LOD cloud, LOD applications relying on data from the LOD cloud need to prioritize which data sources should be first considered in order to achieve an optimal accuracy of their local copies under the given constraints. Therefore, applications make use of a scheduling strategy for data updates. A scheduling strategy aims for deriving an order of importance for data sources based on a set of data features. In the ideal case, a strategy would derive an order such that the application would visit only the subset of LOD sources which have actually been changed. In this section, we introduce a formal specification of update functions and a set of data features used by update strategies for deriving such an order.

3.1 Data Updates

Whenever an application needs to update a local copy covering the data sources in $c \in C$, at time t_{i+1}, it would technically be sufficient to fetch the complete dataset $X_{t_{i+1}}$. However, this would imply to visit all data sources c, retrieve their most recent version of the data $X_{c,t_{i+1}}$ and integrate it into one dataset $X_{t_{i+1}}$. Due to limitations such as network bandwidth capacity for downloading data or computation time, we assume that only a certain fraction of the data can actually be retrieved fresh from the cloud and processed in a certain time interval.

Thus, applications need to apply a scheduling strategy to efficiently manage the accuracy of the data. Based on features extracted from the dataset retrieved at an earlier point in time t_i, a scheduling strategy indicates which data sources c should be visited (i. e., visit the URI c and fetch the latest version of the data made available at this URI) in the time slice between t_i and t_{i+1}. The update strategy can simply be seen as a relation:

Definition 3. *Update Strategy*

$$U \subset C \times \{1, \ldots, n\} \tag{3}$$

A tuple (c, i) in this relation indicates a point in time t_i at which data from a data source c should be updated.

Furthermore, we define the constraint of the bandwidth as a restriction to download at most up to K triples.

Definition 4. *Constraint of the Bandwidth*

$$\textit{For a given } i: \sum_{(c,i)\in U} |X_{c,t_i}| \leq K \tag{4}$$

For any given constraint of the bandwidth, it is possible to retrieve data from the sources in their order of preferences until the limit has been reached.

Example 3. Suppose our dataset has been updated the last time on May 8th, 2013 (see Example 1), and we want to again update our local copy on June 10th, 2013. However, due to limitations, the constraints of the bandwidth enables only $K = 12,000$ triples to be fetched per time slice. For such constraints, we suppose we cannot fetch all the data since $|X_{\text{dbpedia.org,2013-05-08}}| + |X_{\text{musicbrainz.com,2013-05-08}}| + |X_{\text{bbc.co.uk,2013-05-08}}| \geq 12,000$. Nevertheless, without violating these restrictions, we suppose we can entirely fetch data from the first two data sources: *dbpedia.org* and *musicbrainz.com*.

We define a *last update* function lu to identify for a specific data source and a given point in time when its data was updated last:

Definition 5. *Last Update Function*

$$lu(c, i) = \text{argmax}_{j \leq i}\{(c, j) \in U\} \tag{5}$$

This function can be used recursively to identify, for instance, the update prior to the last update by $lu(c, lu(c, i) - 1)$.

Example 4. In the previous example, we updated our local copy by fetching data from *dbpedia.org* and *musicbrainz.com*, then the *last update* time of the data sources are given as:

$$lu(\textit{dbpedia.org}, \text{2013-06-10}) = lu(\textit{musicbrainz.com}, \text{2013-06-10}) = \text{2013-06-10},$$
$$lu(\textit{bbc.co.uk}, \text{2013-06-10}) = \text{2013-05-08}$$

Using the last update function lu at time t_i, we can define the aggregated data set according to an update strategy, i. e., which version of which data source is part of the current local copy. This aggregated data set X'_{t_i} is defined as:

Definition 6. *Aggregated Data Set*

$$X'_{t_i} = \bigcup_{c \in C} X_{c, t_{lu(c,i)}} \tag{6}$$

Example 5. Following Example 4, our updated dataset for June 10th, 2013 is given as: $X'_{\text{2013-06-10}} = \{X_{\text{dbpedia.org,2013-06-10}}, X_{\text{bbc.co.uk,2013-05-08}}, X_{\text{musicbrainz.com,2013-06-10}}\}$

Finally, using this notation, we can easily construct the history of a particular data source in the course of execution of an update plan over time up to time t_i:

Definition 7.

$$\mathcal{H}(c, t_i) = \{X_{c,t_j} | (c, j) \in U, t_j \leq t_i\} \tag{7}$$

Example 6. The history of our sample data source *dbpedia.org* is given as:

$$\mathcal{H}(\text{dbpedia.org}, \text{2013-06-10}) = \{X_{\text{dbpedia.org,2013-06-10}}, X_{\text{dbpedia.org,2013-05-08}}\}$$

3.2 Data Features

In the following, we present features proposed in the literature to improve the freshness of cached data. Here, we restrict ourselves to define only those features that will actually be used in our experiments. Please note that the set of features of a data source can always be extended.

Age provides the time span since the data source has been last visited and updated [4]. It captures 'how old' is the data provided by a data source:

$$f_{Age}(c, X'_{t_i}) = t_i - t_{lu(c,i)} \tag{8}$$

PageRank provides the PageRank of a data source in the overall data set at the (last known) time [20]:

$$f_{PageRank}(c, X'_{t_i}) = PR(X_{c,t_{lu(c,i)}}) \tag{9}$$

Size provides the (last known) number of triples provided by a data source:

$$f_{Size}(c, X'_{t_i}) = |X_{c,t_{lu(c,i)}}| \tag{10}$$

ChangeRatio provides the absolute number of changes of the data in a data source between the last two (known) observation points in time [7].

$$f_{Ratio}(c, X'_{t_i}) = |X_{c,t_{lu(c,i)}} \setminus X_{c,t_{lu(c,lu(c,i)-1)}}| + |X_{c,t_{lu(c,lu(c,i)-1)}} \setminus X_{c,t_{lu(c,i)}}| \tag{11}$$

ChangeRate provides the change rate between the observed data in the two (last known) points in time of a data source [11].

$$f_{Change}(c, X'_{t_i}) = \Delta(X_{c,t_{lu(c,i)}}, X_{c,t_{lu(c,lu(c,i)-1)}}) \tag{12}$$

In this case, the change rate Δ is a function (metric) to measure the change rate between two data sets. We will use two Δ functions:
Jaccard distance:

$$\Delta(X_{c,t_{lu(c,lu(c,i)-1)}}, X_{c,t_{lu(c,i)}}) = 1 - \frac{|(X_{c,t_{lu(c,lu(c,i)-1)}}) \cap (X_{c,t_{lu(c,i)}})|}{|X_{c,t_{lu(c,lu(c,i)-1)}}) \cup (X_{c,t_{lu(c,i)}})|}$$

Dice Coefficient:

$$\Delta(X_{c,t_{lu(c,lu(c,i)-1)}}, X_{c,t_{lu(c,i)}}) = 1 - \frac{2 * |(X_{c,t_{lu(c,lu(c,i)-1)}}) \cap (X_{c,t_{lu(c,i)}})|}{|X_{c,t_{lu(c,lu(c,i)-1)}})| + |(X_{c,t_{lu(c,i)}})|}$$

Dynamics measures the behavior of the data source observed over several points in time [10], where the dynamics of a data source is defined as the aggregation of absolute changes, as provided by Δ-metrics.

$$f_{Dynamic}(c, X'_{t_i}) = \sum_{i=0}^{j} \frac{\Delta(X_{c,t_{lu(c,lu(c,i)-1)}}, X_{c,t_{lu(c,i)}})}{t_{lu(c,i)}, t_{lu(c,lu(c,i)-1)}}, j \leq i.$$

3.3 Update Function

As a large number of LOD sources are available but only a limited number of sources can be fetched per run, it is required to determine which sources should be visited first. By using the vector of features of each data source, we define an update function $\rho : f \rightarrow R$, which assigns a preference score to a data source based on the observed features at time t_i.

An update strategy is defined by ranking the data sources according to their preference score in descending or ascending order, and fetching them starting

from the top ranked entry to some lower ranked entry. For instance, if we consider f_{Size} to be the feature observed at time t_i for all $c \in C$, ρ could be defined as the rank of the data sources in ascending order (from the smallest to the biggest ones).

Furthermore, the bandwidth defines the amount of data that can be fetched per run. Consequently, at some point in time t_i, data of a set of data sources is updated until the bandwidth constraint has been consumed completely. For the sake of clarity, we discard a data source when its data cannot completely be fetched, i. e., when the last started fetch operation cannot be entirely executed because of the bandwidth limit being violated while reading the data. We consider that the tuple (c, i) is considered to be in the update relation U defined above, if the data from c can be entirely fetched based on the given order and the available bandwidth. Without loss of generality, we assume that the data sources are visited in a sequential order. However, it is left to the implementation to decide whether data from the different data sources should be fetched in sequential or parallel processing.

4 Evaluation

In this section, we consider the scheduling strategies described in Section 3 and analyze their effectiveness for updating local copies of the LOD sources. We evaluate these strategies on a large-scale and real world LOD dataset. Our evaluation goal is to show which of the update strategies produce better updates of the LOD sources, i. e, we demonstrate for given restrictions of bandwidth, which strategy performs better in terms of data accuracy and freshness.

4.1 Data

Our evaluation dataset is obtained from the Dynamic Linked Data Observatory (DyLDO). The DyLDO dataset has been created to monitor a fixed set of Linked Data documents (and their neighborhood) on a weekly basis[2]. Our evaluation dataset is composed of 149 weekly crawls (in the following we will refer to a crawl as a snapshot) corresponding to a period over the last three years (from May 2012 to March 2015). Furthermore, the DyLDO dataset contains various well known and large LOD sources, e. g., dbpedia.com, musicbrainz.com, and bbc.co.uk as well as less commonly known ones, e. g., advogato.org, statistics.data.gov.uk, and uefa.status.net. For more detailed information about the DyLDO dataset, we refer the reader to [17]. As we use weekly crawls obtained from the DyLDO dataset, we are only able to grab changes occurring between consecutive weeks (e. g., daily changes are not considered).

To gain a better insight into our evaluation dataset, let us first look at the evolution of the snapshots. The number of data sources per snapshot ranges between 465 and 742. On average, a snapshot is composed of 590 data sources.

[2] For sake of consistency, we use only the kernel seeds of LOD documents.

During the period studied, the number of data sources per snapshot slightly decreased, due to data sources going temporarily or permanently offline. Looking at consecutive snapshots, on average 1.05% of the data sources per snapshot are new and previously unseen (data sources birth rate), and 1.36% of the data sources disappear each week (death rate). On average, 99.3% of the data sources remain in existence between consecutive snapshots, and 37.3% of them change on a weekly basis. Taking the first snapshot as reference, only 13.9% of the data sources remain unchanged over the entire interval studied. This overview confirms prior findings [17] indicating that a high portion of the data on the LOD cloud changes.

To provide better insights into how changes are distributed over the data sources, we randomly sampled an arbitrary point in time (June 1st, 2014) and check for the distribution of triples over data sources. We observe that most of the data sources, 78.3%, are small (containing less than 1,000 triples) and they contribute only 0.5% of all triples retrieved at this point in time. The few big data sources (0.6%) that are left (up to 1,000,000 triples), contribute more than 49.2% of all triples. Furthermore, we observe that most of the changes (66.7%) take place in the data sources with more than 1,000,000 triples.

4.2 Evaluation Methodology

Ideally, scheduling strategies should prioritize an update of data sources which provide modified data. Note, that we do not consider the task of discovering new data sources for inclusion into the data cache. Rather, we want to maintain an as fresh-as-possible local copy of a fixed predefined set of LOD sources. To be able to evaluate different scheduling strategies, we use the following scenarios:

Single-Step. We evaluate the quality of update strategies for a single and isolated update of a local data cache, i. e., starting from a perfectly accurate data cache at time t_i, our goal is to measure which quality can be achieved with different update strategies at time t_{i+1}, for varying settings of bandwidth limitations.

Iterative Progression. We evaluate the evolution of the quality of a local data cache when considering iterative updates over a longer period of time, i. e., starting from a perfect data cache at time t_i, our goal is to measure how good is an update strategy in maintaining an accurate local copy at subsequent points in time $t_{i+1}, t_{i+2}, ..., t_{i+n}$ when assuming a fixed bandwidth. In our experiment, we consider four iterations.

We implemented update strategies based on rankings according to the features presented in Section 3.2:

1. *Age* updates from the last to the most recently updated data source.
2. *Size-SmallestFirst* updates from the smallest to the biggest data source.
3. *Size-BiggestFirst* updates from the biggest to the smallest data source.
4. *PageRank* updates from the highest to lowest PageRank of a data source.

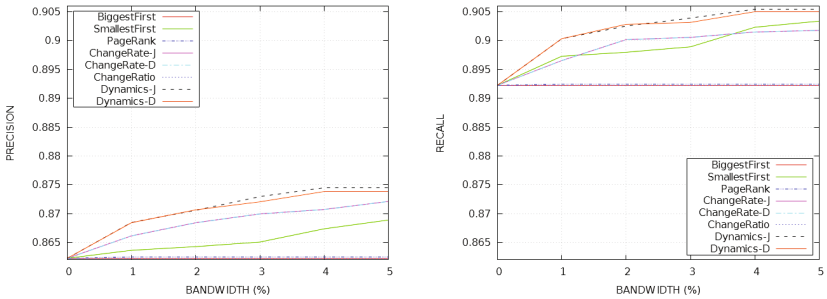
5. *ChangeRatio* updates from the most to the least changed data source based on set difference applied to the last two retrieved versions of the data.
6. *ChangeRate-J* updates from the most to the least changed data source based on Jaccard distance applied to the last two retrieved versions of the data.
7. *ChangeRate-D* updates from the most to the least changed data source based on Dice Coefficient applied to the last two retrieved versions of the data.
8. *Dynamics-J* updates from the most to the least dynamic data source based on Jaccard distance and previous observed snapshots of the data.
9. *Dynamics-D* updates from the most to the least dynamic data source based on Dice Coefficient and previous observed snapshots of the data.

Please note that we analyze the strategy *Age* only for the *Iterative Progression* scenario. *Age* cannot be used in the *Single Step* scenario. Since we build the follow-up copy from a perfect local copy, the feature *Age* would assign the same value to each data source.

The data features used by the strategies are extracted based on the available history information. In our experiment, the history is composed of the last four updates. In the first setup, the task to be accomplished by the strategies is to compute an update order for all data sources at the point in time t_{i+1}. For the strategies *Size* and *PageRank*, we use information about data retrieved from the last update time t_i. *ChangeRatio* and *ChangeRate* are calculated over the last two updates t_{i-1} and t_i, and *Dynamics* is calculated over the complete history for points in time t_{i-4} to t_i. For the *Iterative Progression* setup, we start with a perfect data cache at t_i. The task is to compute the updates iteratively at the next points in time t_{i+1} to t_{i+4}. In the first step, the history setup is the same as the *single-step* setup and the size of the history increases along with the iterations.

In order to make the results of the different setups comparable, and due to the fact that the iterative setup considers four iterative updates, the snapshots used in the single step evaluation are the same ones which are evaluated in the first place in the iterative evaluation setup (every fifth snapshot of the dataset). Additionally, we simulate network constraints by limiting the relative bandwidth, i. e., that only a certain ratio of triples can be fetched for updating a local copy at a given point in time. In the simulation, we stepwise increase the bandwidth constraint from 0% to 5% in intervals of 1%, from 5% to 20% in intervals of 5%, and from 20% to 100% in intervals of 20% of all available triples.

LOD sources are from time to time unavailable, i. e., some LOD sources cannot be reached by any application at a certain point in time, but may be again reachable at a later point in time. Nevertheless, the implemented strategies do not differentiate whether a source is unavailable for a period of time, or is deleted from the cloud. Whenever a LOD source is deleted or unavailable at point in time t_i, no triples are delivered and the empty set is considered for further computations.



(a) Precision (b) Recall

Fig. 1. Quality Outcomes for the Single Step Setup Setup at Low Bandwidth Level (5%).

4.3 Metrics

The quality of an update strategy is measured in terms of micro average recall and precision over the gold standard, i. e. the perfect up-to-date local copy:

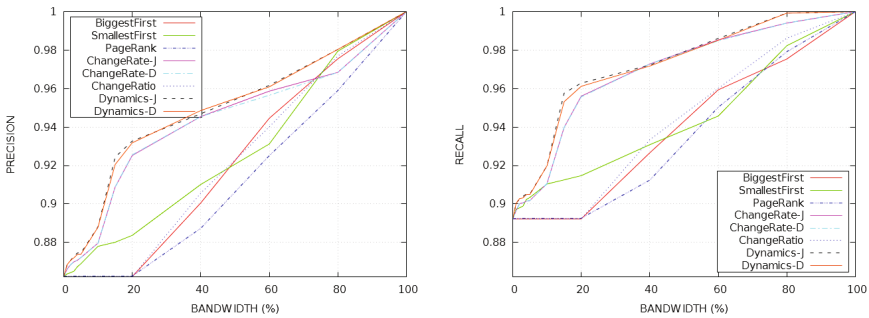
$$p_{micro}(X'_t, X_t) = \frac{\sum_{c \in C^{X'_t}} |X_{c,t} \cap X'_{c,t}|}{\sum_{c \in C^{X'_t}} |X'_{c,t}|} \tag{13}$$

$$r_{micro}(X'_t, X_t) = \frac{\sum_{c \in C^{X'_t}} |X_{c,t} \cap X'_{c,t}|}{\sum_{c \in C^{X'_t}} |X_{c,t}|} \tag{14}$$

4.4 Results

Single-Step Evaluation. Figure 2(a) and Figure 2(b) show the average precision and recall over all snapshots the *single-step* setup. The x-axis represents the different levels of constraints of relative bandwidth (in percent) and the quality in terms of precision and recall is placed on the y-axis. We observe that precision ranges from 0.862 to 1 and recall from 0.892 to 1 for bandwidth from 0% to 100% (see Figure 2(a) and Figure 2(b)). This implies that if no updates are executed (no bandwidth is available), our dataset is on average 87% correct (F measure) after one update. This value can be interpreted as the probability to get correct results when issuing an arbitrary query on the data.

Overall, the *Dynamics* strategies outperform all other strategies. First, we look at the precision curve. For very low relative (see Figure 1(a)) bandwidth (from 0% to 10%) the *Dynamics* strategies perform best, followed by the *ChangeRate* strategies. Already only with 3% available bandwidth, precision improvements is from 0.862 to 0.873 for *Dynamics*, and to 0.869 for *ChangeRate*. With 10% bandwidth the improvement gets to 0.888 for *Dynamics* and 0.879 for *ChangeRate* while the third best strategy, *SmallestFirst*, achieves 0.877 and the others strategies do not achieve scores higher than 0.862. For the higher relative bandwidths, the *ChangeRate* and *Dynamics* strategies are comparable and



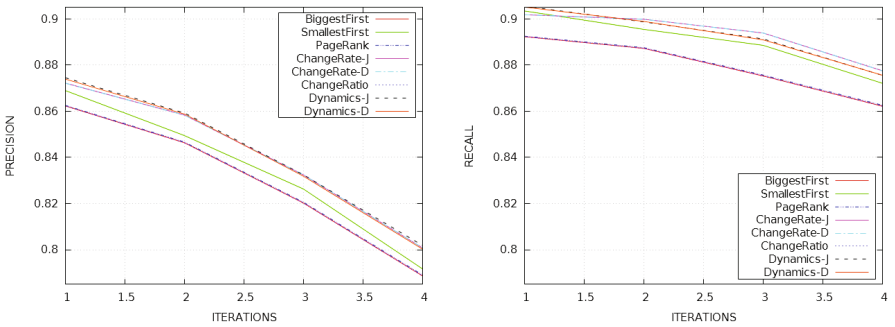
(a) Precision (b) Recall

Fig. 2. Quality Outcomes for the Single Step Setup

show only small differences in performance. Turning to recall (see Figure 1(b)), *ChangeRate* and *Dynamics* perform quite similar over the entire interval and clearly outperform all strategies and all bandwidth constraints. For even only 15% bandwidth available, the recall values are above 0.957 while all other strategies achieve at most 0.93.

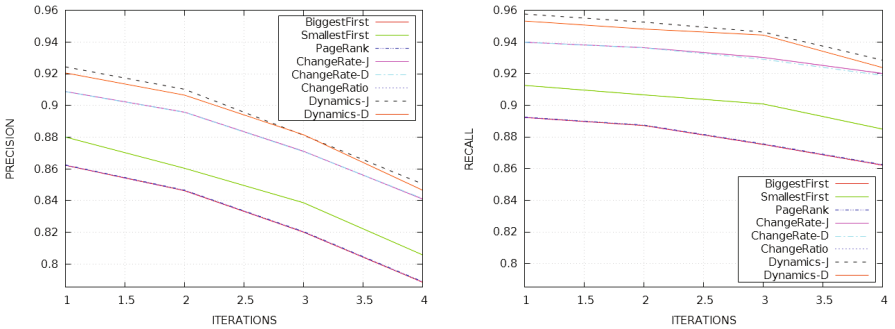
LOD sources vary in their sizes. As shown, most of the big data sources change frequently and, consequently, they are in the top ranked entries for strategies such as the *Dynamics* and *ChangeRate*. Also, some of the smaller data sources have a high change frequency. Therefore, in the ranking list provided by the *Dynamics* and *ChangeRate* strategies, a mix of big and small sources can be found in the top entries. For strategies such *BiggestFirst* and *ChangeRatio* only/most of the biggest sources are top ranked. In contrast, by the strategy *SmallestFirst* only the smallest sources are top ranked. When updating the smallest data sources first, even for a very small bandwidth, a great number of data sources can be fetched and consequently data changes can also be retrieved. This can be observed in the recall curve of the *SmallestFirst* strategy. When only low bandwidth is available, it is not possible to fetch data from big data sources since there is not enough bandwidth. This can be clearly seen for the *BiggestFirst* strategy, where updates are observed only when 20% or more is available. The more bandwidth is available, the more changes can be retrieved. Due to the mix of data sources sizes in the ranking list of the *Dynamics* and *ChangeRate* strategies, they are able to retrieve already data when only a very small bandwidth is available and overall are able to retrieve more modified data than the other strategies for all bandwidths. The others strategies narrow in quality when more bandwidth is available.

In general, the *single-step* experiments show that update strategies based on dynamics followed by change rate make best use of limited resources in terms of bandwidth. For very low relative bandwidth, the strategies based on data source dynamics tend to provide better results.



(a) Precision (b) Recall

Fig. 3. Quality Outcomes for the Iterative Progression Setup at Low Bandwidth Level (5%).

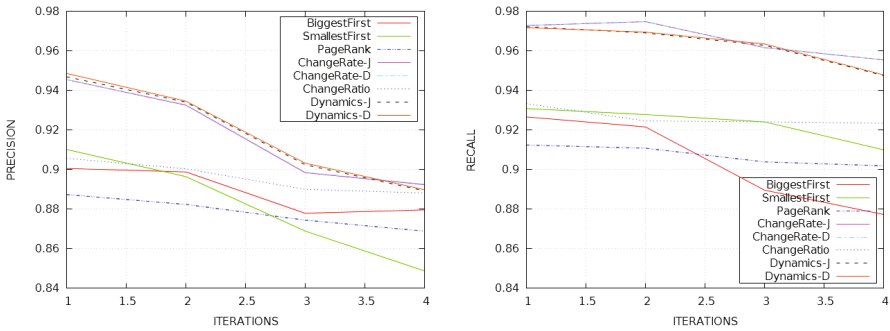


(a) Precision (b) Recall

Fig. 4. Quality Outcomes for the Iterative Progression Setup at Mid-Level Bandwidth (15%).

Iterative Progression Evaluation. In this evaluation, we look at the evolving quality when considering iterative updates. This setup simulates real use case scenarios such as of a LOD search engine continuously updating its caches. In our experiments, we look how precision and recall behave over the iterations. First, we fix the bandwidth constraints. We choose a low (5%), mid (15%), and high (40%) bandwidth which provided low, average, and good outcomes based on the previous experiments (*single-step* evaluation).

Figure 3(a) and Figure 3(b) show precision and recall for bandwidth fixed at 5%. The x-axis represents the iterations (points in time) and the y-axis the quality in terms of precision and recall. Note that quality decreases along the iterations. This is expected, since only at the first iterations the update process starts from a perfect data cache. For low relative bandwidths, the impact on the (loss of) quality of iterative updates is quite similar for all strategies. Never-



(a) Precision (b) Recall

Fig. 5. Quality Outcomes for the Iterative Progression Setup at High-Level Bandwidth (40%).

theless, the plot confirms the previous discussion that the *Dynamics* strategies followed by *ChangeRate* are the more appropriate ones, if we need to predict the next best steps (and not only the first step anymore). Nevertheless, the strategies show a uniform loss of quality.

A similar output is observed for bandwidth fixed at 15% (see Figure 4(a) and Figure 4(b)). Here again, fetching data from the source that changes more than others ensure more accurate updates. Even if we can observe that the loss of quality is comparable, the *Dynamics* strategies followed by *ChangeRate* maintain a higher level of quality after the four iterations. *Dynamics* precision and recall decreases from 0.92 to 0.846 and 0.953 to 0.929 and *ChangeRate* from 0.908 to 0.841 and 0.939 to 0.918 after the four iterations, while the quality of the other strategies are mostly lower after only one or even no iteration.

Precision and recall under a relative high bandwidth (fixed at 40%) is shown in Figure 5(a) and Figure 5(b). The recall values of the *Dynamics* strategies and *ChangeRate* almost not change over the iterations (above of 0.947). The precision values decrease with a maximum of 0.889. Over all iterations, these strategies outperforms all the others even when only one-step update is applied. Interestingly, at this bandwidth level, the strategies which fetch data from the big data sources first show good performance since—up to that bandwidth level—it is possible to load a big data source entirely. As most changes concentrate in the big data sources, they are also able to fetch most of the changes. For instance, precision and recall of the *ChangeRatio* strategy reaches values of 0.888 and 0.923, respectively, after the iterations.

Overall, the results of this experiment setup confirms the discussion from the previous one, i. e, the strategies based on the dynamics features followed by the ones based on change rate are the more appropriate ones if we need to predict (iterative) updates. Certainly, the more bandwidth is available, the more changes can be grabbed (and therefore the rate of quality lost over the iterations is lower for all strategies). Still even after four iterations, *Dynamics* strategies

(followed by *ChangeRate*) were able to better maintain an up-to-date local copy for all different bandwidth levels. From our experiments and for low relative bandwidth, these strategies could definitely better support applications to fetch the most changed data (and thus to avoid to fetch unchanged data) than the other strategies.

5 Related Work

The evolution of the Web has been observed in [5] in order to obtain implications of changes on incremental Web crawlers. Incremental crawlers update local data collections if they recognize influencing changes. Likewise, the dynamics of Web pages is empirically analyzed in [3,7] with a dedicated focus on the update frequencies of search engine indices. Estimations for changes of data items and elements are proposed in [6]. Such estimations are used if the history of changes is incomplete, e.g., it is known that a Web page has changed but it is not known how often it has changed in a certain period.

Various related work have investigated the characteristics of the LOD cloud. Their goal is to apply these characteristics for the purpose of different applications such as query recommendation and indices updates. Some works conducted structural analysis of the LOD cloud such as [1,2,15] in order to obtain statistical insights into the characteristics of the data. In addition, there is related work on analyzing the LOD cloud in order to verify its compliance with established guidelines and best practices how to model and publish data as Linked Data [16,22]. Other works as by Neumann et al. [19] analyze LOD in order to obtain statistics like its distribution in the network.

Among those works that are dedicated on the study of the Linked Data dynamics, with a dedicated focus on the update frequencies of LOD search engine indices, Umbrich et al. [23] compare the dynamics of Linked Data and the dynamics of Linked datasets with HTML documents on the Web. Their change detection uses (i) HTTP metadata monitoring (HTTP headers including timestamps and ETags), (ii) content monitoring, and (iii) active notification of datasets. These three detection mechanisms are compared by several aspects like costs, reliability, and scalability of the mechanism. Similar to our approach, the content monitoring applies a syntactic comparison of the dataset content, i.e., a comparison of RDF triples (but ignoring inference). Change detection is a binary function which is activated whenever changes are found. In our evaluation, we consider more complex change metrics to allow fine-grained ranking.

The importance of caching for efficient querying linked data is analyzed by Hartig et al. [14]. Query execution is based on traversing RDF links to discover data that might be relevant for a query during the query execution itself. Data is cached and it is used for further queries. Caching show some beneficial impact to improve the completeness of the results. Additionally, Umbrich et al. [24] proposed a hybrid approach for answering SPARQL queries, i. e, deciding which parts of a query are suitable for local/remote execution. The authors estimate the freshness of materialized data using the notion of coherence for triple patterns against the live engine. Dehghanzadeh et al. [8] extend this approach by

extending the statistics of cardinality estimation techniques that are used in the join query processing phase.

The Dynamic Linked Data Observatory is a monitoring framework to analyze dynamics of Linked Data [17]. Snapshots of the Web of data are regularly collected and then compared in order to detect and categorize changes. Using these snapshots, the authors study the availability of documents, the links being added to the documents, and the schema signature of documents involving predicates and values for rdf:type and determine their change rate. Motivated by this work, Dividino et al. [11] analyzed the changes on the usage of the vocabulary terms in the DyLDO dataset. The authors show that the combination of vocabulary terms appearing in the LOD documents changes considerably.

6 Conclusion

In this paper, we propose and evaluate scheduling strategies for updating on a large-scale LOD dataset that is obtained from the cloud by weekly crawls over the course of three years. In a first setup, where we evaluate the quality of update strategies for a single and isolated update of a local data cache, we observe that update strategies based on dynamics or change rate make best use of limited resources in terms of bandwidth. For very low relative bandwidth, the strategies based on data source dynamics provide better results. Already only with 15% available bandwidth, we observed improvements of precision and recall for dynamics from 0.862 to 0.924 and from 0.892 to 0.957, respectively.

In a second evaluation setup, we evaluate the behavior of the strategies in a realistic scenario (e. g., a LOD search engine updating its caches) which involves measuring the quality of the local data cache when considering iterative updates over a longer period of time. Overall the results of this experiment setup confirms the discussion from the previous one. Especially for low relative bandwidth, update strategies based on dynamics or change rate are more appropriate to support applications to fetch the most changed data (and thus to avoid to fetch unchanged data) than the others strategies.

In future work, we plan to investigate the impact on the performance when combining different update strategies. We also intend to consider further evaluation setups such as the cold start setup, i. e., we measure how good is an update strategy starting from an empty cache and considering iterative updates over a longer period of time. At last, we mentioned in this paper that the implemented strategies do not differentiate whether a source is unavailable for a period of time or is deleted from the cloud. Therefore, we plan to extend these strategies to consider the availability of the LOD sources over time.

Acknowledgments. The research leading to these results has received funding from the European Community's Seventh Framework Programme under grant agreement n° 610928.

References

1. Alexander, K., Cyganiak, R., Hausenblas, M., Zhao, J.: Describing linked datasets - on the design and usage of void, the 'vocabulary of interlinked datasets'. In: LDOW. CEUR (2009)
2. Auer, S., Demter, J., Martin, M., Lehmann, J.: LODStats – an extensible framework for high-performance dataset analytics. In: ten Teije, A., Völker, J., Handschuh, S., Stuckenschmidt, H., d'Acquin, M., Nikolov, A., Aussenac-Gilles, N., Hernandez, N. (eds.) EKAW 2012. LNCS, vol. 7603, pp. 353–362. Springer, Heidelberg (2012)
3. Brewington, B.E., Cybenko, G.: How dynamic is the Web? Computer Networks (2000)
4. Cho, J., Garcia-Molina, H.: Synchronizing a database to improve freshness. In: SIGMOD (2000)
5. Cho, J., Garcia-Molina, H.: The evolution of the web and implications for an incremental crawler. In: VLDB, VLDB 2000. Morgan Kaufmann Publishers Inc. (2000)
6. Cho, J., Garcia-Molina, H.: Estimating frequency of change. ACM Trans. Internet Technol. (2003)
7. Cho, J., Ntoulas, A.: Effective change detection using sampling. In: Proceedings of the 28th International Conference on Very Large Data Bases, VLDB. VLDB Endowment (2002)
8. Dehghanzadeh, S., Parreira, J.X., Karnstedt, M., Umbrich, J., Hauswirth, M., Decker, S.: Optimizing SPARQL query processing on dynamic and static data based on query time/freshness requirements using materialization. In: Supnithi, T., Yamaguchi, T., Pan, J.Z., Wuwongse, V., Buranarach, M. (eds.) JIST 2014. LNCS, vol. 8943, pp. 257–270. Springer, Heidelberg (2015)
9. Dividino, R., Kramer, A., Gottron, T.: An investigation of HTTP header information for detecting changes of linked open data sources. In: Presutti, V., Blomqvist, E., Troncy, R., Sack, H., Papadakis, I., Tordai, A. (eds.) ESWC Satellite Events 2014. LNCS, vol. 8798, pp. 199–203. Springer, Heidelberg (2014)
10. Dividino, R., Gottron, T., Scherp, A., Gröner, G.: From changes to dynamics: dynamics analysis of linked open data sources. In: PROFILES. CEUR (2014)
11. Dividino, R., Scherp, A., Groner, G., Grotton, T.: Change-a-lod: does the schema on the linked data cloud change or not? In: COLD. CEUR (2013)
12. Gottron, T., Gottron, C.: Perplexity of index models over evolving linked data. In: Presutti, V., d'Amato, C., Gandon, F., d'Aquin, M., Staab, S., Tordai, A. (eds.) ESWC 2014. LNCS, vol. 8465, pp. 161–175. Springer, Heidelberg (2014)
13. Gottron, T., Scherp, A., Krayer, B., Peters, A.: Lodatio: using a schema-level index to support users infinding relevant sources of linked data. In: KCAP. ACM (2013)
14. Hartig, O.: Zero-knowledge query planning for an iterator implementation of link traversal based query execution. In: Antoniou, G., Grobelnik, M., Simperl, E., Parsia, B., Plexousakis, D., De Leenheer, P., Pan, J. (eds.) ESWC 2011, Part I. LNCS, vol. 6643, pp. 154–169. Springer, Heidelberg (2011)
15. Hausenblas, M., Halb, W., Raimond, Y., Feigenbaum, L., Ayers, D.: SCOVO: using statistics on the web of data. In: Aroyo, L., Traverso, P., Ciravegna, F., Cimiano, P., Heath, T., Hyvönen, E., Mizoguchi, R., Oren, E., Sabou, M., Simperl, E. (eds.) ESWC 2009. LNCS, vol. 5554, pp. 708–722. Springer, Heidelberg (2009)
16. Hogan, A., Umbrich, J., Harth, A., Cyganiak, R., Polleres, A., Decker, S.: An empirical survey of linked data conformance. J. Web Sem. (2012)

17. Käfer, T., Abdelrahman, A., Umbrich, J., O'Byrne, P., Hogan, A.: Observing linked data dynamics. In: Cimiano, P., Corcho, O., Presutti, V., Hollink, L., Rudolph, S. (eds.) ESWC 2013. LNCS, vol. 7882, pp. 213–227. Springer, Heidelberg (2013)
18. Konrath, M., Gottron, T., Staab, S., Scherp, A.: Schemex - efficient construction of a data catalogue by stream-based indexing of linked data. J. Web Sem. (2012)
19. Neumann, T., Moerkotte, G.: Characteristic sets: accurate cardinality estimation for RDF queries with multiple joins. In: ICDE. IEEE Computer Society (2011)
20. Page, L., Brin, S., Motwani, R., Winograd, T.: The pagerank citation ranking: Bringing order to the web. Technical Report 1999–66, Stanford InfoLab, November 1999. previous number = SIDL-WP-1999-0120, http://ilpubs.stanford.edu:8090/422/
21. Schaible, J., Gottron, T., Scheglmann, S., Scherp, A.: Lover: support for modeling data using linked open vocabularies. In: EDBT/ICDT 2013 Workshops. EDBT. ACM (2013)
22. Schmachtenberg, M., Bizer, C., Paulheim, H.: Adoption of the linked data best practices in different topical domains. In: Mika, P., Tudorache, T., Bernstein, A., Welty, C., Knoblock, C., Vrandečić, D., Groth, P., Noy, N., Janowicz, K., Goble, C. (eds.) ISWC 2014, Part I. LNCS, vol. 8796, pp. 245–260. Springer, Heidelberg (2014)
23. Umbrich, J., Hausenblas, M., Hogan, A., Polleres, A., Decker, S.: Towards dataset dynamics: change frequency of linked open data sources. In: LDOW. CEUR (2010)
24. Parreira, J.X., Umbrich, J., Karnstedt, M., Hogan, A.: Hybrid SPARQL queries: fresh vs. fast results. In: Cudré-Mauroux, P., Heflin, J., Sirin, E., Tudorache, T., Euzenat, J., Hauswirth, M., Parreira, J.X., Hendler, J., Schreiber, G., Bernstein, A., Blomqvist, E. (eds.) ISWC 2012, Part I. LNCS, vol. 7649, pp. 608–624. Springer, Heidelberg (2012)