applied sciences

Article

A Dynamic, Cost-Aware, Optimized Maintenance Policy for Interactive Exploration of Linked Data

Usman Akhtar [1], **Anita Sant'Anna** [2] **and Sungyoung Lee** [1,*]

[1] Department of Computer Science and Engineering, Kyung Hee University, (Global Campus), 1732, Deogyeong-daero, Giheung-gu, Yongin-si, Gyeonggi-do 17104, Korea; usman@oslab.khu.ac.kr
[2] Viniam Consulting AB, Munkalyckevägen 9 A, 302 35 Halmstad, Sweden; anita@viniam.se
* Correspondence: sylee@oslab.khu.ac.kr; Tel.: +82-31-201-2514

Received: 16 September 2019; Accepted: 7 November 2019; Published: 11 November 2019

Abstract: Vast amounts of data, especially in biomedical research, are being published as Linked Data. Being able to analyze these data sets is essential for creating new knowledge and better decision support solutions. Many of the current analytics solutions require continuous access to these data sets. However, accessing Linked Data at query time is prohibitive due to high latency in searching the content and the limited capacity of current tools to connect to these databases. To reduce this overhead cost, modern database systems maintain a cache of previously searched content. The challenge with Linked Data is that databases are constantly evolving and cached content quickly becomes outdated. To overcome this challenge, we propose a Change-Aware Maintenance Policy (CAMP) for updating cached content. We propose a Change Metric that quantifies the evolution of a Linked Dataset and determines when to update cached content. We evaluate our approach on two datasets and show that CAMP can reduce maintenance costs, improve maintenance quality and increase cache hit rates compared to standard approaches.

Keywords: linked data analytic; query performance; maintenance policy; SPARQL

1. Introduction

In the recent past, massive amounts of data are available publicly and these data are produced at an alarming rate in all domains of medical sciences, creating what we today call "Big Data". The era of big data generates new opportunities for the research community to build solutions to search and integrate life sciences data effectively. Most of these datasets have varied format or lack any format and are stored in disparate locations that are poorly linked, which hinders the operator from efficiently searching and retrieving data. Recently, Linked Data has emerged as one of the best practices to represent and connect these repositories, also allowing the exchange of information in an interoperable manner. Linked Data ontologies not only support the integration of multiple data from diverse sources but also provide an efficient way to query these datasets. Healthcare systems and recent technologies have realized the importance of acquiring and preserving big data streams for decision making. Therefore, with the evolution of big data in healthcare, the research trends have shifted from massive storage to efficient analysis of data.

Nowadays, in various healthcare disciplines, the use of large datasets is becoming popular and the data is shared on the cloud. Modern stream analytics applications [1,2] are exploiting the fusion of data streams and Linked Data publicly available on the Web. Recent investigations [3–9] have shown that the content of Linked Data is dynamic in nature and keeps on evolving. Linked Data content changes over time and some of the data stored in the local cache becomes outdated. Therefore, knowledge about what has changed in the database is important for analytics applications [10]. Analytics applications need constant updates to guarantee the quality of service in maintaining the local cache. To the extent of our knowledge, there is limited work addressing the problem of maintaining local views (or caches) up-to-date [6,10–16]. Normally a maintenance policy is needed to determine what to update and when. In the literature, three kinds of conventional maintenance policy have been reported for data analytics systems [17]: immediate (update immediately after data arrives), deferred (no execution is performed on current query evaluation) and periodic (update the local views on a regular bases). However, these conventional policies fail to effectively optimize local views for linking data due to slow response time.

In this paper, we propose a proactive maintenance policy called CAMP to update the local view by issuing the maintenance jobs during system idle time. CAMP achieved the desired query performance by doing the maintenance ahead of query evaluation time. The proposed approach works as an optimizer to accelerate the overall query processing by pre-fetching the data and storing in the cache for future queries. Therefore, instead of running repeated queries, results are provided from the local views thus improving overall response time. In the case of stream processing, the warehouse must load all the data in a real-time. Therefore, we ensure that each view reflects a consistent state even these local caches are scheduled for updates at different times. In comparison, in state-of-the-art approaches, the maintenance of local views is performed immediately whenever new data arrives. However, the overhead of these approaches is significant and it seems unfair for the system to hold up the computational resources as user have to wait until the maintenance jobs are completed. In contrast, our maintenance policy postpones until the system has a freecycle. Typically, queries issued by the Linked Data clients are repetitive and structurally similar. To deal with query similarity, we propose a query similarity metric using a distance function. The results of these similar structure queries are placed in cache to increase hit rates, while alleviating the burden on the querying endpoints.

The remainder of this paper is organized as follows—Section 2 explains the background and related work. Section 3 explains the proposed approach. The evaluation of the proposed approach is performed in Section 4 on the real datasets. This article is concluded in Section 5.

2. Background

In this section, we briefly discuss the background needed to understand Linked Data, which includes data representations and querying of the Linked Data cloud. Moreover, we also discuss an overview of related work and highlight the differences of the proposed approach with respect to state-of-the-art.

2.1. Semantic Web Technologies

The Semantic Web technology defines the interconnection of data points as an expressive representation of heterogeneous data sources [18]. As described by Tim Berners-Lee [19], the Semantic Web enables the machine in such a way that data can be searched, interpreted and reused. Linked Data is another important concept in the Semantic Web, enabling the machine to browse the Web of data, such as DBpedia (https://wiki.dbpedia.org/). Linked Data is collaboratively built from the Web corpus to represent knowledge in a structured format that greatly facilitates the sharing of information around the world. Linked Data describes the inter-connection of Web pages's *URLs (Uniform Resource Locator)* therefore facilitating the linking of the data from diverse sources. The Semantic Web is mainly composed of Knowledge Bases (KBs) such as Freebase [20], DBpedia [21] and Yago [22]. These KBs represent data as Linked Data according to a predefined schema, known as Resource Description Framework (RDF). The RDF is considered the standard representation for Linked Data, where relationships are represented in the form of triples (*subject, predicate and an object*). SPARQL (https://www.w3.org/TR/sparql11-overview/) is a widely used standard query language to retrieve and manipulate data that are stored in RDF format. SPARQL is a structured query language standardized by the W3C for querying RDF triplestores (http://www.w3.org/TR/rdf-sparql-query/). A SPARQL query can be further decomposed into Basic Graph Patterns (BGPs) and the results are represented as a hierarchical tree. The syntax of SPARQL query contains different and disjoint query types such as SELECT, CONSTRUCT, ASK and DESCRIBE. The Linked Data cloud also provides a SPARQL endpoint for their datasets. However, querying SPARQL endpoints is cumbersome. Due to network instability often the connection to these endpoints is temporarily lost, affecting query performance. These endpoints do not provide any information about dataset modification. Therefore, long-running data analytics applications must resubmit queries for keeping the local data cache up-to-date.

2.2. Life Sciences Linked Data

Massive amounts of data are being produced and currently available in the life sciences. However, these data are unstructured and difficult to integrate. Therefore, the Linked Data paradigm is currently suitable for publishing and connecting life science datasets to improve access and use. The Linked Data cloud contains almost 570 datasets [23] from different domains that are interlinked with each other as shown in Figure 1. A considerable portion of Linked Data is comprised of life sciences data, significant contributors include the *MEDLINE* (https://www.ncbi.nlm.nih.gov/pubmed), *Bio2RDF* [24] and *DrugBank* (https://www.drugbank.ca/). *MEDLINE* is an American national medicine bibliographic database that contains more than twenty-four million references to journals related to bio-medicine. *Bio2RDF* is a built over the Semantic Web to provide biological databases interlinked with life science data. *DrugBank* contains data related to *Bioinformatics* and *cheminformatics* with comprehensive target information. These databases contain the protein sequences that are linked to the drug entries that target the protein data.

2.3. Related Work

The issues related to the exploitation and maintenance of local views have received considerable attention in the research community over recent years. Most of the maintenance algorithms proposed in the literature are based on *eager (or deferred)* maintenance. Eager maintenance protocols update the view periodically whenever a change occurs in the cloud. Colby et al. [25] proposed deferred view maintenance to reduce the view downtime and perform incremental maintenance to keep the local views up-to-date. However, all these algorithms divide the maintenance task into smaller steps, which is not as effective as combining the small maintenance task to improve the efficiency of the overall maintenance process. Data warehouses have also used a *lazy maintenance* [26] policy that delays maintenance until downtime rather than the peak query time. Also worth mentioning are the maintenance policies for stream monitoring that examine the problem to schedule multiple continuous queries. They are able to manage incoming data streams using on deferred maintenance for updating local data caches. However, their policies mainly focus on ETL (Extract, Transform and Load) operations and they have difficulties in managing latency and accuracy when a large-scale fusion of linked datasets is given.

Semantic caching was originally proposed for Database Management System (DBMS) [27] to reduce the overhead when retrieving data over a network. The idea is to maintain has been extended to previously issued queries to facilitate the future requests. Nowadays, the semantic caching technique triplestores [28]. Godfrey et al. [29] proposed a notion of semantic overlap and introduced a caching approach that utilized client-server systems. To extend this idea, Dar et al. [27] proposed a semantic region-based caching and introduced a distance metric to update the cache where the far regions are discarded from the cache. However, these approaches can only handle *SELECT* SPARQL queries. Martin et al. [28] proposed an idea to apply cache on the SPARQL processor. The benefit of this work is to cache both the triple query result and the application result. However, this work does not consider identical and similar queries for cache replacement. To extend this work, Shu et al. [30] proposed a content-aware approach that utilized *query containment* to estimate whether the queries can be answered from the caches. The containment checking approach produces a lot of overhead, therefore this approach is not widely utilized by the Semantic Web community. Yang et al. [31] proposed an approach to decompose the query into basic graph patterns and cache intermediate results but they do not consider cache replacement to maintain the freshness of the cache. In summary, only few works have been reported that deal with the problems related to view maintenance. Moreover, the existing solutions are unable to provide proactive maintenance policy therefore reducing query time.

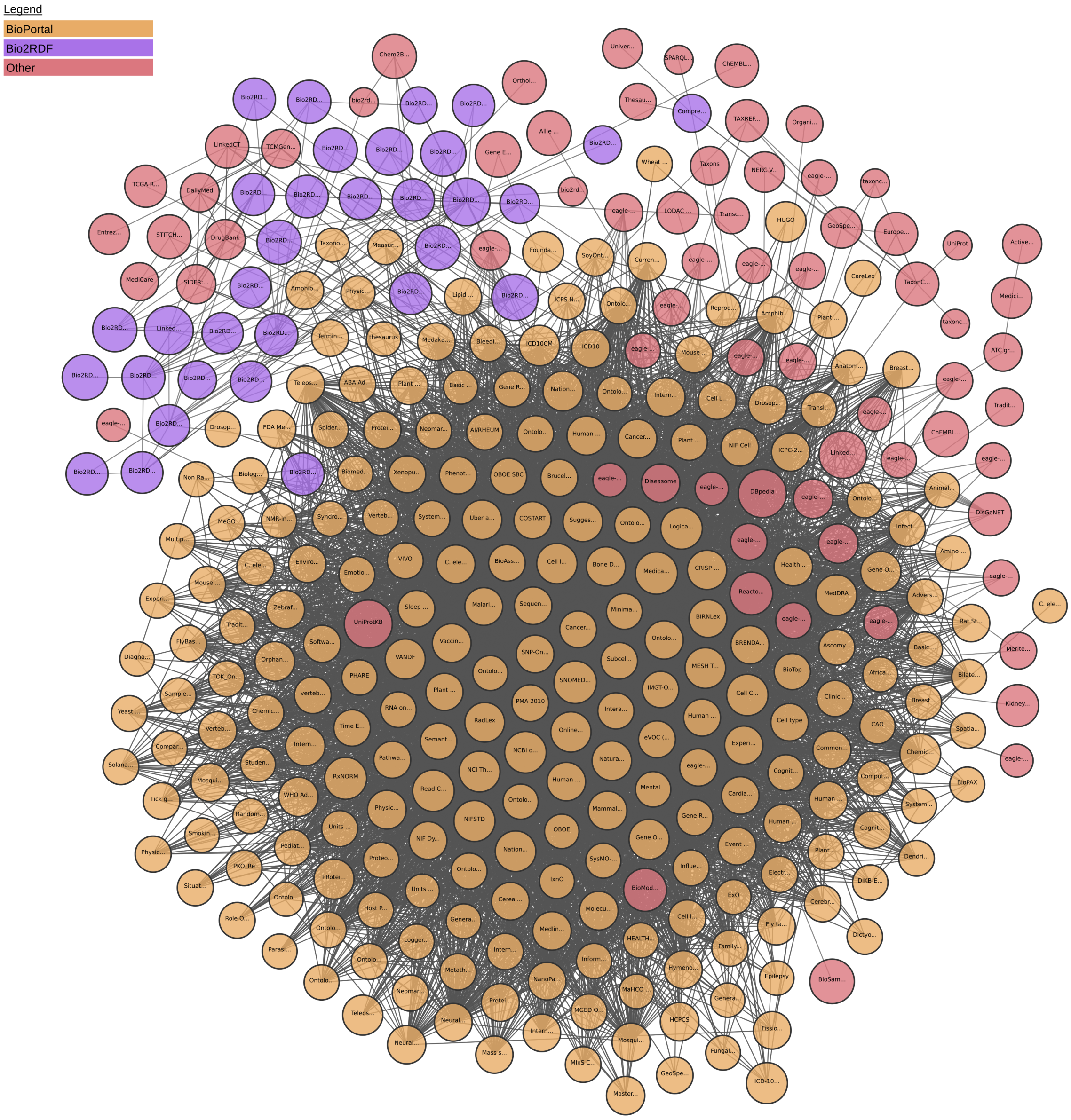


Figure 1. Showing the life science Linked Data Web.

3. Proposed Methodology

Figure 2 illustrates the overall architecture of the proposed approach. The main core component in our proposed system architecture is the maintenance manager. When a user query registers in the system, the maintenance manager creates a view corresponding to the query and serves the request previously issued. The aim of the maintenance manager is to determine when and what to update based on the proposed Change Metric. By utilizing the change metric, our maintenance policy effectively updates the local views. The aim of the change metric is to quantify the evolution of the Linked Data and identifies the changes occurs in the cloud. This change metric utilized the maintenance policy to establish the criteria when the maintenance jobs are executed. Moreover, queries issued by the clients are structurally similar [32]. To quantify the structural similarity between queries, we propose a bottom-up query matching approach to detect changes between triple patterns occurring in consecutive queries. The result of queries is placed in a cache (or view). When similarly structured queries arrive, the request is immediately returned from the cache module.

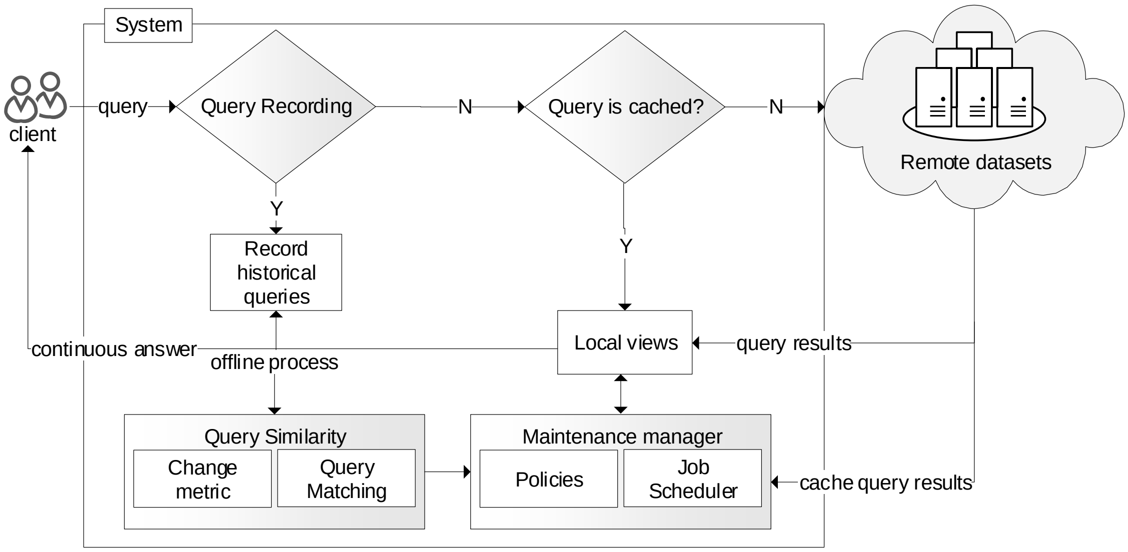


Figure 2. Overall architecture of the proposed method.

3.1. Change Metric

Due to the continuous evolution of Linked Data, the local views become outdated. We proposed a change metric that quantifies the evolution of Linked Data. The main benefit of using the change metric is to alleviate the expensive job of copying the whole data instead of only updating local views with the changed items. We utilized the Dynamic Characteristic Set (DCS), which identifies the addition and deletion of the items in the Linked Data cloud. Typically, queries issued by the end-user are repetitive and follow similar patterns that only differ in a specific element. Therefore, we use a bottom-up matching approach to find similar queries. For the structurally similar queries, we first compute the distance between the patterns, then the results of these queries are placed in a cache for future access. To improve the efficiency of our system, we schedule the view maintenance as a low priority job that combines the multiple tasks into one larger job and executes it when the system is in idle state.

We propose a change metric to quantify the evolution of the Linked Data Cloud. The change metric utilizes the Dynamic Characteristics Sets (DCS) [33], a scheme abstraction that classifies the Linked Data on the basis of the properties of *subjects* and *objects*. Moreover, DCS captures the inherent structure and relationships of the Linked Data cloud. The DCS is a combination of properties and types that are used to describe the content in Linked Data. A change at any level of Linked Data implies a change in the mapping of the *properties* and *type*.

Let X_t represent the Linked data set captured at time t, which consist of subject s, predicate p and object o, where P is the set of the properties in X_t and T are its type set. We define the DCS as a schema abstraction based on the s and p. Any change in the DCS shows new content in the Linked data has been added or removed.

Example 1. *To explain the changes in the Linked Open Data (LOD) cloud, consider the snapshots of LOD captures at time t as shown in Table 1. The changes are illustrated by DCS_1, DCS_2 , DCS_{2a} and DCS_{2b}. DCS_1 = {foaf:Person, foaf: knows} captured at X_{t_1} and DCS_1 = {foaf:Person, foaf: knows} captured at X_{t_2} remain unchanged across snapshots. From the later version of the snapshot DCS_2 was deleted and new combinations of DCS_{2a} and DCS_{2b} were observed. Thus, DCS_2 is no longer used in the LOD.*

Table 1. Rate of change in snapshots using Dynamic Characteristic Set (DCS).

Snapshot X_{t_1}	**Snapshot X_{t_2}**	**Status**
DCS_1 = {foaf:Person, foaf:knows}	DCS_1 = {foaf:Person, foaf:knows}	Unchanged
DCS_2 = {foaf:name, foaf:knows, dbpedia:project/media}		Deleted
	DCS_{2a} = {foaf:name, foaf:knows, dbpedia:project/social}	New
	DCS_{2b} = {foaf:name, foaf:knows, dbpedia:internProject}	New

3.2. Query Matching

We proposed an offline analysis to identify similar structure queries. The result of the previously issued queries are extracted from the log. By using query matching, we pre-fetch the results of the similar queries and place in a cache for future access. This approach lowers the burden on SPARQL endpoints. To find a similar query, we compute the *Levenshtein Distance* between two queries, normalized by the size of the query strings. We define the Distance Score as:

$$DistanceScore = \frac{Levenshtein\,(Q_1, Q_2)}{max\,(|Q_1, Q_2|)}. \tag{1}$$

In Equation (1), we define the distance score to match the triple patterns. The overall distance of the triple pattern is calculated by aggregating the individual score of the *subject*, *predicate* and an *object*. The structure base similarity is not sufficient to return a search result, as it only relies on the ordering of symbols. It is possible that two queries represent the same structure of ordering but share different content. We utilize the distance function score to determine the query matching. The query distance between the two graphs is the minimum number of edit operations, such as addition, deletion and insertion, to transform one graph to another. However, the cost is determined by the distance between triple patterns $\Delta\,(T_1, T_2)$. Complete matching is only possible in the case of bipartite graphs, where *Basic Graph Patterns (BGP)* occur with the same number as for the triple patterns. Maximum matching is determine in polynomial time. We utilize the Hungarian method [34] to obtain the individual triple patterns with the assignment of minimum cost. Therefore, the higher cost of triple matching is set to $\Delta\,(T_1, T_2) > 1$ which is considered the infinite cost, where there is no matching between the BGP elements. The score derived from the complete matching is defined as:

$$\Delta\,(BGP_1, BGP_2) = \frac{\sum\left(Triple_i, Triple_j \in M_T\right)\Delta\left(Triple_i, Triple_j\right)}{M_T}. \tag{2}$$

As shown in Equation (2), M_T assigns the cost to triples. In the case that M_T is finite, there is a valid triple pattern matching scenario that exists, and if M_T is infinite, no such scenario exists. However, real-world queries are more complex than the basic graph patterns, therefore, we use a bottom-up approach to derive the minimum cost between BGP. We continue to check the alignment of query patterns, in the case of no alignment, then the matching has an infinite cost. In the case of maximal matching, two BGPs can be matched if they are aligned and matching with finite cost can be established between all the patterns.

3.3. Maintenance Manager

In this section, we describe the maintenance manager in detail and also explain the prerequisite to efficiently update the local views. When a query is registered with the system the maintenance manager creates a view and selects the appropriate policy to keep the view up-to-date. These policies invoke the job scheduler to select the relevant stream data and store the appropriate results in the view. A maintenance policy P consists of the sequence of job needed to update the local views. To be able to keep track of all the views this module maintains hash tables of each view along with the maintenance tasks. To keep track of which version of the data is needed by the views, this module keeps track of the status of the maintenance task. The maintenance of each job is scheduled as a low priority background job and the maintenance manager combines multiple tasks to schedule as a larger job to be executed when the system is in idle state. When a maintenance job is completed, the manager removes the job from the list and releases the remaining pending tasks. Similarly, in the query execution phase, we first check whether the views used by the query contain any pending maintenance jobs. In the case of a pending task, the query requests the manager to perform the task in order to update the local views. This process is similar to on-demand maintenance, where the query waits until the maintenance process is completed.

3.4. Scheduling Updates of Linked Data

To update the view, for each transaction a maintenance task is created and accumulated in the task queue. The traditional policy is to update each job one by one regardless of the time consumption. However, in our approach we combine each small tasks into a larger maintenance job. The maintenance job will be executed only once. Our maintenance manager eliminates redundant update tasks. Therefore, reducing the size of the update stream. The maintenance job begins when the system has a freecycle. We hide the maintenance process from incoming queries to improve the performance of response time. We use CAMP [12,33] to determine when a local view needs to be updated. CAMP identifies the highly dynamic sources in the Linked Data that are frequently updated and captures these changes to updates the local views. CAMP visits the data sources based on a preference score. Highly dynamic sources should be fetched at a high priority. The preference score is based on the history of the sources. The preference score is computed as shown in Equation (3):

$$f_{PASU}(X) = \sum_{i=0} \frac{\Delta(\mathrm{X}_{\mathrm{t}_{\mathrm{lastupdate}-1}}, \mathrm{X}_{\mathrm{t}_{\mathrm{lastupdate}}})}{\mathrm{t}_{\mathrm{lastupdate(X)}} - t_{lastupdate(X)-1}}. \tag{3}$$

In Equation (3), $t_{lastupdate}$ is a function that returns the time when Linked Data was last updated. This function is recursive, as $t_{lastupdate-1}$ returns the time before the last update and Δ quantifies the amount of the changes such as addition and deletion of items. The maintenance policy indicates when the Linked Data source should be fetched. However, it is possible to retrieve the sources in the order of their assigned preference. The overall flow of the proposed algorithm is shown in Algorithm 1. The CAMP algorithm first generates the view for the incoming queries as shown in Line 1. As an input parameter, CAMP takes a query model Q and job scheduler, which indicates the time to schedule the resources. CAMP also chooses the appropriate policy to facilitate the performance of the maintenance policy P. The maintenance manager releases a policy J^r to update the local views before the query evaluation in Line 6 and 7 in a maximum execution time K_{max}.

Algorithm 1: Change-Aware Maintenance Policy (CAMP).

```
Input  :Query $Q$, Job Scheduler $H$
Output:The entire maintenance process $P$
// Creation of the maintenance process
$V^q = createView\,(Q)$
for $(i = 0; i \leq Maintenance; i++)$ do
    $P = MaintenancePolicy\,(Q, Views(V^q))$
    // Update from the most dynamic sources
    $updatecache \leftarrow J^r$;
    $J^r = P.release\,(JobScheduler)$
// Check that the recent changes occur in the LOD sources
if $f(X_i, t_i)$ recent changes then
    $updatecache \leftarrow t_{i+1}$ ;
else
    return $f(X_i, K_{max}) = update(X_i)$;
```

4. Experimental Study

This section describes the evaluation of our approach and comparison with other approaches. We first describe the setup of our experimentation and dataset used. For performance evaluation we measure the response time, maintenance cost and cache hit rates. We conducted the experiments on OpenLink Virtuoso Server 07.10 with 4xAMD A8-7650K Radeon R7 and 64bit Ubuntu 16.04.2 LTS and 32 GB of RAM with 7200 RPM. There are numerous datasets available in *Biomedicine* that originate from different sources. In our experimentation, we utilized two datasets—LinkedCT (https://www.w3.org/wiki/HCLSIG/LODD/Data) and DYLDO (http://swse.deri.org/dyldo/data/). Both of these datasets are serialized in triple format, we use NxParser (https://www.w3.org/2001/sw/wiki/NxParser) to parse these datasets. For query evaluation, we utilized the queries extracted from the logs of LinkedCT and DYLDO [35]. In our evaluation, we have utilized the queries extracted from the public available SPARQL endpoints provided by the Linked SPARQL Queries (LSQ) (https://aksw.github.io/LSQ/) datasets.

The *Linked Clinical Trials (LinkedCT)* dataset [36] is a Linked Data representation of the open dataset *ClinicalTrials.gov*. The original dataset is published in XML format and the main benefit of its linked representation is that it facilitates SPARQL queries. This dataset contains information about governmental and privately funded clinical trials in approximately 9.8 million triples. This dataset also contains links to external datasets such as *DBpedia* and *Bio2RDF.org* via SPARQL endpoints..

The Dynamic Linked Data Observatory ***DYLDO*** [10,37] monitors the evolution of Linked Data over time. It collects snapshots of the Web of Data using 149 weekly crawls of the Linked Data Cloud. The average size of a snapshot is about 1.35 GB. The data collected during three years adds up to approximately 36 GB. DYLDO includes various well known sources such as *DBpedia.org*, *identi.ca* and *DBtropes.org*.

4.1. Baseline Approaches

We have evaluated our approach against *Eager Maintenance*, *Time To Live (TTL)*, *PageRank*, *Size*, *ChangeRatio* and *ChangeRate*.

Eager Maintenance: In eager maintenance, all the updating jobs are performed immediately after the new data arrives. Eager maintenance updates all the materialized views immediately after the query evaluation and each update query has to wait until view maintenance is done. The cost of the view maintenance is normally high [38].

Time To Live (TTL): TTL is a fixed threshold—the age at which the view must be refreshed. Most of the maintenance manager use *TTL* to captures how old the views from the last query evaluation are. TTL is an important feature that illustrates when the data source was last visited and updated [39].

$$P_{TTL}(X_{t,c}) = t_i - t_{last}(X_{t,c}) \tag{4}$$

As shown in Equation (4), TTL is used by the scheduling strategies to fetch the data source c in time t_i, whereas, t_{last} is the last update time.

PageRank (PR): Updates the local views based on the ranking of the Linked Data sources. The ranking is calculated based on the number of incoming links and the sources are pre-fetched from the highest PageRank [40]. It is represented as:

$$P_{PR}(X_{t,c}) = PR(X_{c_{highest}, t_{update}}) \tag{5}$$

Size: In this policy, *Size* is determined by checking the Linked Data sources. The priority is given to the largest data source [41]. The formula for *size* is as follows:

$$P_{size}(X_{t,c}) = |X_{c_{max}, t_{update}}| \tag{6}$$

ChangeRatio: In this policy, *ChangeRatio* counts how many items were updated based on the last known time period [42]. This metric is useful for storing the change history and number of detected changes of the Linked Data as shown in Equation (7).

$$P_{ChangeRatio} = \sum_{i=no.of changes} |X_{c,t_{lastupdate}}| \tag{7}$$

ChangeRate: In this maintenance policy, the local views are updated from the sources with the most to the least changes [15] observed at last known points. It is represented as follows:

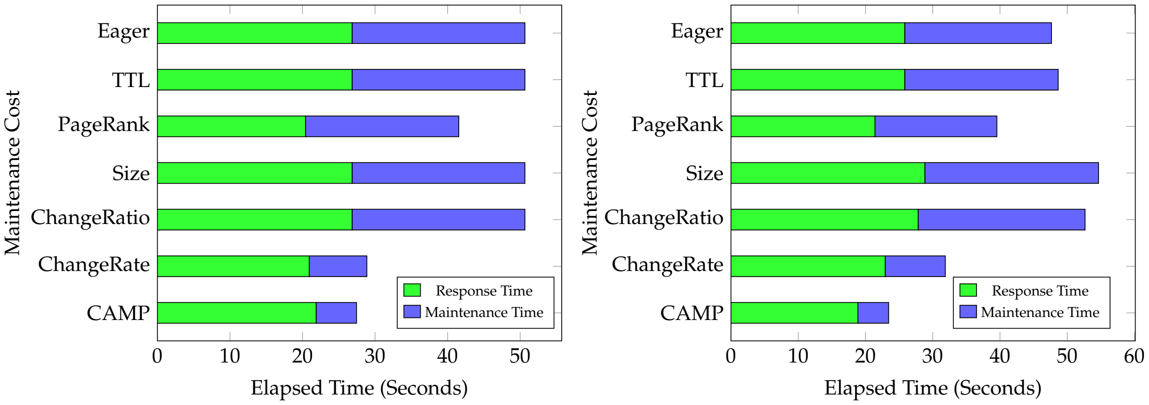
$$P_{ChangeRatio} = \Delta \left(X_{c,t_{lastupdate}}, X_{c,t_{lastupdate-1}} \right) \tag{8}$$

4.2. Performance Evaluation

To evaluate the effectiveness of CAMP, we compared our proposed approach with the state-of-the-art approaches to find out the quality of the maintenance performed by existing strategies. We evaluated on the basis of the maintenance cost, quality and cache hit rates.

4.2.1. Maintenance Cost

We define the maintenance cost as the time taken to perform maintenance operations. Figure 3 shows the maintenance cost for each policy on each dataset. Our proposed approach, CAMP, performed the maintenance job in the background when the system was idle and enough resources were available. Therefore, the overhead of our proposed approach was completely hidden from the user and measured the required system time in order to perform the offline maintenance task. The maintenance cost was measured as a sum of the response time and total time. Our approach produced a lower elapsed time of 5 s as the query does not pay for the cost of maintaining the view. Compared with *Eager Maintenance* where the query has to wait until the maintenance job is completed, the average response time was 15 s. However, PageRank, Size and TTL did not consider maintenance cost while updating the local views and these strategies performed worst as the query had to wait until the maintenance jobs were completed. Similarly, in the case of the ChangeRatio and ChangeRate, these policies use the periodic update function that keeps on tracking the changed occurred in the Linked Data cloud. As these policies often run in the background, they produced high latency.



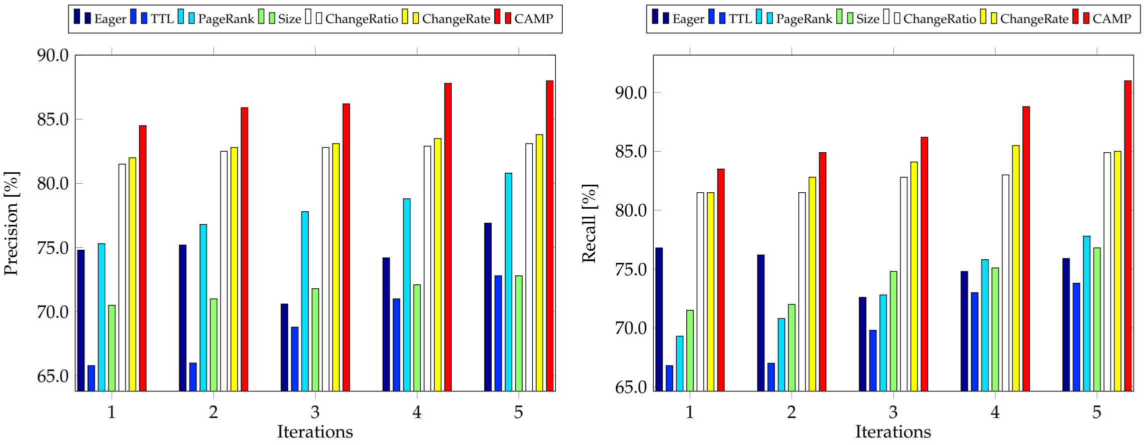
(a) LinkedCT Dataset

(b) DYLDO Dataset

Figure 3. Maintenance cost: Showing the comparison with other state-of-the-art approaches on (**a**) LinkedCT and (**b**) DYLDO datasets.

4.2.2. Maintenance Quality

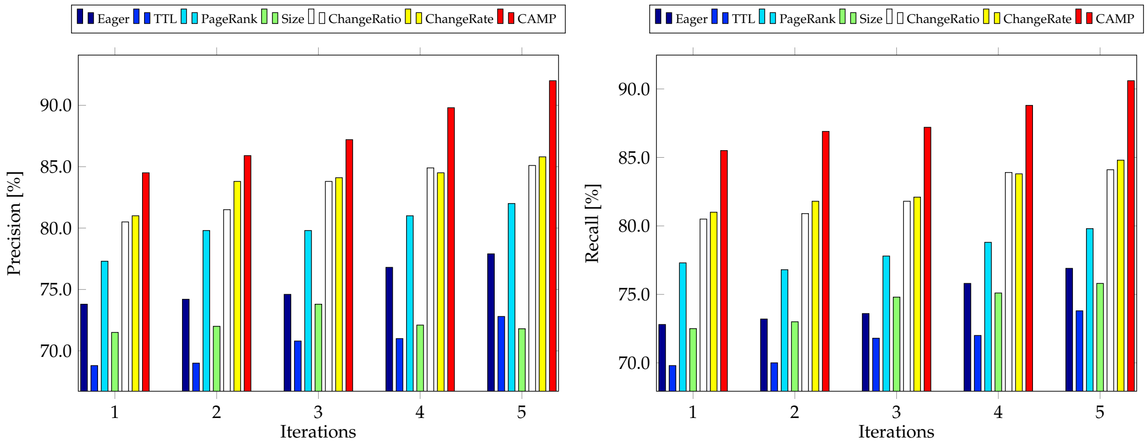
In this evaluation, we utilized the quality of the updates performed by the maintenance policies under consideration. We started with the perfect cache and assumed that a change occurs in the cloud. Due to this change the local views become outdated. Therefore, our main goal was to check the quality of the updates performed by the state-of-the-art strategies as shown in Figure 4. And we evaluated how the existing strategies perform to update the local views. We used the precision and recall as an evaluation metric to measure the quality of the updates results are shown in Figure 4. In this setup, all the existing strategies showed a uniform loss in the quality of update. CAMP outperformed all other strategies, achieving 91% (precision) and 89% (recall) accuracy in the datasets. We observed that our proposed strategy only updates the relevant data sources with less overhead and delay. Strategies like *Pagerank*, *Size* and *TTL* performed worst because these strategies were executing irrelevant queries. ChangeRatio and ChangeRate only captured changed items and their efficiency degraded with each iteration over time.



(a) Precision on LinkedCT Dataset

(b) Recall on LinkedCT Dataset

Figure 4. *Cont.*



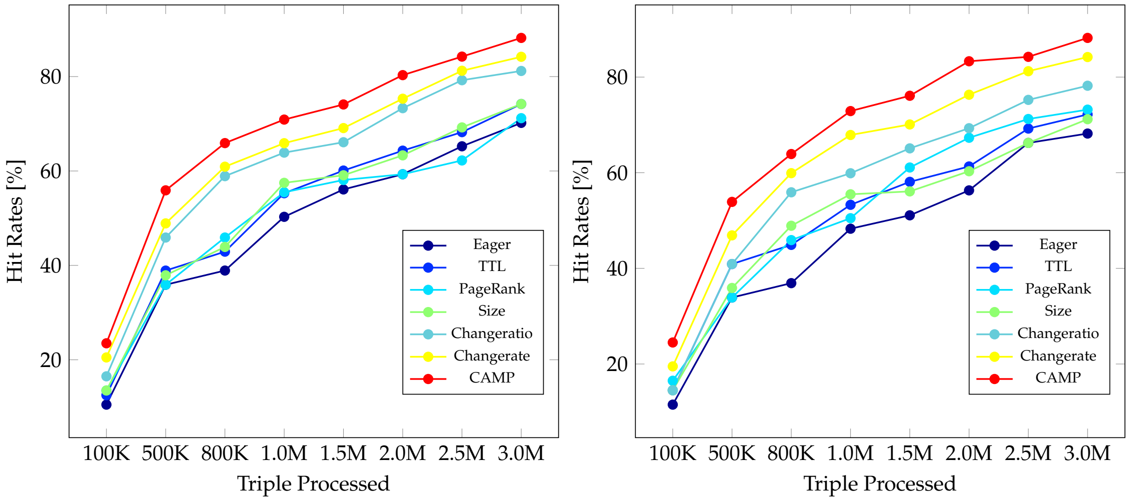
(c) Precision on DYLDO Dataset

(d) Recall on DYLDO Dataset

Figure 4. Quality of updates performed by the proposed approach as compared to the existing approaches.

4.2.3. Cache Hit Rates

As the cache had limited space; it was advantageous to replace the cache with more valuable content to improve query performance (i.e., cache hit rate). In this evaluation, we measured the performance of query times in terms of better hit rates. We evaluated both of the datasets, LinkedCT and DYLDO. Figure 5 shows the hit rates achieved by the existing approaches. We utilized the access logs and based on the maintenance policy, each approach replaced the cache to keep it up-to-date. CAMP replaced the cache based on the access frequency and more frequent access queries were placed in a cache for future access. On average, CAMP achieved 82% hit rates as compared to the Eager (77%) and ChangeRate (70%). *PageRank*, *Eager*, *TTL* and *Size* performed worst in term of cache hit rates.



(a) Hit Rate on LinkedCT Dataset

(b) Hit Rate on DYLDO Dataset

Figure 5. Hit Rate achieved by CAMP as compared to existing approaches.

5. Conclusions

Often, Linked Data applications pre-fetch data and place it in a cache for future access. Due to the continuous evolution of the Linked Data Cloud, the local cache becomes outdated. In this paper, we proposed a maintenance policy that performs the local cache update jobs before query evaluation. The proposed maintenance policy utilizes a change metric together with a query similarity measure to identify and update changed items. Most of the queries issued by the Linked Data client are similar in structure. Therefore, instead of running the queries repeatedly, we pre-fetched the results of these queries to improve cache hit rates. For the maintenance jobs, we combined the smaller task into one job to reduce resource utilization. We compared the effectiveness of our proposed approach to state-of-the-art approaches, namely Eager, *TTL*, *PageRank*, *Size*, *ChangeRatio* and *ChangeRate*. The proposed approach outperformed the existing policies in terms of lower maintenance cost, higher maintenance quality and better cache hit rates. In the future, we will investigate the benefit of our proposed approach to accelerate the query time in real-world data analytics application.

Author Contributions: U.A. was the principal researcher of this study. He proposed and formulated the idea, designed and performed the experiments and wrote the paper; A.S.A. edited the manuscript for language and clarity; S.L. provided advisory comments, remarks, and financial aid for the paper.

Funding: This research was supported by the Ministry of Science and ICT (MSIT), Korea, under the Information Technology Research Center (ITRC) support programme (IITP-2017-0-01629) supervised by the Institute of Information & Communications Technology Planning & Evaluation (IITP). This work was supported by an IITP grant funded by the Korean government (MSIT) (no.2017-0-00655) and NRF-2016K1A3A7A03951968.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

CAMP	Change-Aware Maintenance Policy
KBs	Knowledge Bases
LOD	Linked Open Data
DCS	Dynamic Characteristics Set
SPARQL	SPARQL Protocol and RDF Query Language
TTL	Time To Live
PR	Page Rank
DYLDO	Dynamic Linked Data Observatory

References

1. Saggi, M.K.; Jain, S. A survey towards an integration of big data analytics to big insights for value-creation. *Inf. Process. Manag.* **2018**, *54*, 758–790. [CrossRef]
2. Khan, S.; Liu, X.; Shakil, K.A.; Alam, M. A survey on scholarly data: From big data perspective. *Inf. Process. Manag.* **2017**, *53*, 923–944. [CrossRef]
3. Fernández, J.D.; Umbrich, J.; Polleres, A.; Knuth, M. Evaluating query and storage strategies for RDF archives. In Proceedings of the 12th International Conference on Semantic Systems, Leipzig, Germany, 12–15 September 2016; ACM: New York, NY, USA, 2016; pp. 41–48.
4. Gottron, T.; Knauf, M.; Scherp, A. Analysis of schema structures in the Linked Open Data graph based on unique subject URIs, pay-level domains, and vocabulary usage. *Distrib. Parallel Databases* **2015**, *33*, 515–553. [CrossRef]
5. Cho, J.; Garcia-Molina, H. Estimating frequency of change. *ACM Trans. Internet Technol. (TOIT)* **2003**, *3*, 256–290. [CrossRef]
6. Akhtar, U.; Amin, M.B.; Lee, S. Evaluating scheduling strategies in LOD based application. In Proceedings of the 2017 19th Asia-Pacific Network Operations and Management Symposium (APNOMS), Seoul, Korea, 27–29 September 2017; pp. 255–258.
7. Bizer, C.; Heath, T.; Berners-Lee, T. Linked data-the story so far. *Int. J. Semant. Web Inf. Syst.* **2009**, *5*, 1–22. [CrossRef]
8. Konrath, M.; Gottron, T.; Staab, S.; Scherp, A. Schemex—Efficient construction of a data catalogue by stream-based indexing of linked data. *Web Semant. Sci. Serv. Agents World Wide Web* **2012**, *16*, 52–58. [CrossRef]
9. Gottron, T. Measuring the Accuracy of Linked Data Indices. *arXiv* **2016**, arXiv:1603.06068.
10. Käfer, T.; Abdelrahman, A.; Umbrich, J.; O'Byrne, P.; Hogan, A. Observing linked data dynamics. In Proceedings of the Extended Semantic Web Conference, Montpellier, France, 26–30 May 2013; Springer: Berlin/Heidelberg, Germnay, 2013; pp. 213–227.

11. Umbrich, J.; Karnstedt, M.; Hogan, A.; Parreira, J.X. Hybrid SPARQL queries: Fresh vs. fast results. In Proceedings of the International Semantic Web Conference, Boston, MA, USA, 11–15 November 2012; Springer: Berlin/Heidelberg, Germnay, 2012; pp. 608–624.
12. Nishioka, C.; Scherp, A. Keeping linked open data caches up-to-date by predicting the life-time of RDF triples. In Proceedings of the International Conference on Web Intelligence, Leipzig, Germany, 23–26 August 2017; ACM: New York, NY, USA, 2017; pp. 73–80.
13. Dividino, R.; Gottron, T.; Scherp, A. Strategies for efficiently keeping local linked open data caches up-to-date. In Proceedings of the International Semantic Web Conference, Bethlehem, PA, USA, 11–15 October 2015; Springer: Berlin/Heidelberg, Germnay, 2015; pp. 356–373.
14. Dividino, R.; Scherp, A.; Gröner, G.; Grotton, T. Change-a-LOD: Does the schema on the linked data cloud change or not? In Proceedings of the Fourth International Conference on Consuming Linked Data-Volume 1034, Sydney, Australia, 22 October 2013; pp. 87–98.
15. Knuth, M.; Hartig, O.; Sack, H. Scheduling Refresh Queries for Keeping Results from a SPARQL Endpoint Up-to-Date (Extended Version). *arXiv* **2016**, arXiv:1608.08130.
16. Nishioka, C.; Scherp, A. Temporal patterns and periodicity of entity dynamics in the Linked Open Data cloud. In Proceedings of the 8th International Conference on Knowledge Capture, Palisades, NY, USA, 7–10 October 2015; ACM: New York, NY, USA, 2015; p. 22.
17. Fotopoulou, E.; Zafeiropoulos, A.; Papaspyros, D.; Hasapis, P.; Tsiolis, G.; Bouras, T.; Mouzakitis, S.; Zanetti, N. Linked data analytics in interdisciplinary studies: The health impact of air pollution in urban areas. *IEEE Access* **2015**, *4*, 149–164. [CrossRef]
18. Berners-Lee, T.J. *Information Management: A Proposal*; Technical Report; 1989. Available online: http://cds.cern.ch/record/369245/files/dd-89-001.pdf (accessed on 11 November 2019).
19. Berners-Lee, T.; Hendler, J.; Lassila, O. The semantic web. *Sci. Am.* **2001**, *284*, 34–43. [CrossRef]
20. Bollacker, K.; Evans, C.; Paritosh, P.; Sturge, T.; Taylor, J. Freebase: A collaboratively created graph database for structuring human knowledge. In Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data, Vancouver, BC, Canada, 9–12 June 2008; ACM: New York, NY, USA, 2008; pp. 1247–1250.
21. Lehmann, J.; Isele, R.; Jakob, M.; Jentzsch, A.; Kontokostas, D.; Mendes, P.N.; Hellmann, S.; Morsey, M.; Van Kleef, P.; Auer, S.; et al. DBpedia—A large-scale, multilingual knowledge base extracted from Wikipedia. *Semant. Web* **2015**, *6*, 167–195.
22. Suchanek, F.M.; Kasneci, G.; Weikum, G. Yago: A core of semantic knowledge. In Proceedings of the 16th International Conference on World Wide Web, Banff, AB, Canada, 8–12 May 2007; ACM: New York, NY, USA, 2007; pp. 697–706.
23. Bizer, C.; Heath, T.; Berners-Lee, T. Linked data: The story so far. In *Semantic Services, Interoperability and Web Applications: Emerging Concepts*; IGI Global: Hershey, PA, USA, 2011; pp. 205–227.
24. Belleau, F.; Nolin, M.A.; Tourigny, N.; Rigault, P.; Morissette, J. Bio2RDF: Towards a mashup to build bioinformatics knowledge systems. *J. Biomed. Inform.* **2008**, *41*, 706–716. [CrossRef] [PubMed]
25. Colby, L.S.; Griffin, T.; Libkin, L.; Mumick, I.S.; Trickey, H. Algorithms for deferred view maintenance. In Proceedings of the ACM SIGMOD Record, Montreal, QC, Canada, 4–6 June 1996; ACM: New York, NY, USA, 1996; Volume 25, pp. 469–480.
26. Zhou, J.; Larson, P.A.; Elmongui, H.G. Lazy maintenance of materialized views. In Proceedings of the 33rd International Conference on Very Large Data Bases, Vienna, Austria, 23–27 September 2007; VLDB Endowment: New York, NY, USA, 2007; pp. 231–242.
27. Dar, S.; Franklin, M.J.; Jonsson, B.T.; Srivastava, D.; Tan, M. Semantic data caching and replacement. In Proceedings of the VLDB, Mumbai (Bombay), India, 3–6 September 1996; Volume 96, pp. 330–341.
28. Martin, M.; Unbehauen, J.; Auer, S. Improving the performance of semantic web applications with SPARQL query caching. In Proceedings of the Extended Semantic Web Conference, Crete, Greece, 30 May–2 June 2010; Springer: Berlin/Heidelberg, Germnay, 2010; pp. 304–318.
29. Godfrey, P.; Gryz, J. Answering queries by semantic caches. In Proceedings of the International Conference on Database and Expert Systems Applications, Florence, Italy, 30 August–3 September 1999; Springer: Berlin/Heidelberg, Germnay, 1999; pp. 485–498.

30. Shu, Y.; Compton, M.; Müller, H.; Taylor, K. Towards content-aware SPARQL query caching for semantic web applications. In Proceedings of the International Conference on Web Information Systems Engineering, Nanjing, China, 13–15 October 2013; Springer: Berlin/Heidelberg, Germnay, 2013; pp. 320–329.
31. Yang, M.; Wu, G. Caching intermediate result of SPARQL queries. In Proceedings of the 20th International Conference Companion on World Wide Web, Hyderabad, India, 28 March–1 April 2011; ACM: New York, NY, USA, 2011; pp. 159–160.
32. Dividino, R.Q.; Gröner, G. Which of the following SPARQL Queries are Similar? Why? In Proceedings of the LD4IE@ ISWC, Sydney, Australia, 2013. Available online: https://pdfs.semanticscholar.org/b200/a1c95c53f66090d900eb203b8dd277469004.pdf (accessed on 11 November 2019)
33. Akhtar, U.; Razzaq, M.A.; Rehman, U.U.; Amin, M.B.; Khan, W.A.; Huh, E.N.; Lee, S. Change-Aware Scheduling for Effectively Updating Linked Open Data Caches. *IEEE Access* **2018**, *6*, 65862–65873. [CrossRef]
34. Lorey, J.; Naumann, F. Caching and prefetching strategies for SPARQL queries. In Proceedings of the Extended Semantic Web Conference, Montpellier, France, 26–30 May 2013; Springer: Berlin/Heidelberg, Germnay, 2013; pp. 46–65.
35. Saleem, M.; Ali, M.I.; Hogan, A.; Mehmood, Q.; Ngomo, A.C.N. LSQ: The linked SPARQL queries dataset. In Proceedings of the International Semantic Web Conference, Bethlehem, PA, USA, 11–15 October 2015; Springer: Berlin/Heidelberg, Germnay, 2015; pp. 261–269.
36. Hassanzadeh, O.; Miller, R.J. Automatic curation of clinical trials data in LinkedCT. In Proceedings of the International Semantic Web Conference, Bethlehem, PA, USA, 11–15 October 2015; Springer: Berlin/Heidelberg, Germnay, 2015; pp. 270–278.
37. Käfer, T.; Umbrich, J.; Hogan, A.; Polleres, A. Towards a Dynamic Linked Data Observatory. *LDOW at WWW*. 2012. Available online: https://ai.wu.ac.at/~polleres/publications/kaef-etal-2012LDOW.pdf (accessed on 25 October 2019).
38. Zhuge, Y.; Garcia-Molina, H.; Hammer, J.; Widom, J. View maintenance in a warehousing environment. In Proceedings of the ACM SIGMOD Record, San Jose, CA, USA, 22–25 May 1995; ACM: New York, NY, USA, 1995; Volume 24, pp. 316–327.
39. Cho, J.; Garcia-Molina, H. Synchronizing a database to improve freshness. In Proceedings of the ACM Sigmod Record, Dallas, TX, USA, 15–18 May 2000; ACM: New York, NY, USA, 2000; Volume 29, pp. 117–128.
40. Page, L.; Brin, S.; Motwani, R.; Winograd, T. *The PageRank Citation Ranking: Bringing Order to the Web*; Technical report; Stanford InfoLab: Standford, CA, USA 1999.
41. Harchol-Balter, M.; Schroeder, B.; Bansal, N.; Agrawal, M. Size-based scheduling to improve web performance. *ACM Trans. Comput. Syst. (TOCS)* **2003**, *21*, 207–233. [CrossRef]
42. Cho, J.; Ntoulas, A. Effective change detection using sampling. In Proceedings of the 28th International Conference on Very Large Data Bases, Hong Kong, China, 20–23 August 2002; VLDB Endowment: New York, USA 2002; pp. 514–525.