Towards capturing and preserving changes on the Web of Data

Jürgen Umbrich, Nina Mrzelj, and Axel Polleres

Vienna University of Economics and Business, Vienna, Austria

Abstract. Existing Web archives aim to capture and preserve the changes of documents on the Web and provide data corpora of high value which are used in various areas (e.g. to optimise algorithms or to study the Zeitgeist of a generation). So far, the Web archives concentrate their efforts to capture the large Web of *documents* with periodic snapshot crawls. Little focus is drawn to preserve the continuously growing Web of Data and actually keeping track of the real frequency of changes. In this work we present our efforts to capture and archive the changes on the Web of Data. We describe our infrastructure and focus on evaluating strategies to accurately capture the changes of data and to also estimate the crawl time for a given set of URLs with the aim to optimally schedule the revising of URLs with limited resources.

1 Motivation

Existing Web archives capture the history of Web documents and preserve the Zeitgeist for different eras [12, 10]. The archived versions of the documents are of high value and used in various areas. For instance, the historical data can be used to study the evolution of the Web and algorithms can be optimised to deal with changing and growing data. The preserved data is also used by news reporters, politicians and sociologists. So far, the Web archives concentrated their efforts to capture the large Web of documents and little focus is drawn to archive the continuously growing Web of Open Data. While established projects, such as the Internet archive[1] capture some parts of the Web of Open Data sources, they do not provide a fine grained view of the history of these resources. We found only around 20 preserved versions for the content of the DBPedia URI for the city of Galway, while the Wikipedia article shows hundreds of changes in their revision.[2]

There exists first and specialised activities to provide some archives for the Web of Data such as the Dynamic Linked Data observatory project [7], which captures 90K Linked Data URIs once a week or the

[1] https://archive.org/index.php

[2] https://web.archive.org/web/*/http://dbpedia.org/resource/Galway

DBPedia project, which converts Wikipedia articles to Linked Data and releases infrequent dumps of their data [8]. Similarly, the FP-7 project Diachron aims to address in parts the challenges of preserving the evolving data with a strong focus on modelling context information and addressing quality issues. To the best of our knowledge, there exists no specific effort for an infrastructure to archive the diverse Web of Data. One challenge is to deal with heterogeneous data formats (e.g., JSON, CSV or RDF) and access mechanisms (e.g. HTTP with content negotiation or accessing SPARQL endpoints).

In this work, we propose and envision a distributed infrastructure to capture and archive the evolving Web of Data. While we provide some initial starting points to this infrastructure, which we have already implemented and started to deploy, we argue that this infrastructure should be joint effort due to the underlying complexity and the resource demands which can be solved by distributing the crawling and storage tasks. In the remainder of this work we highlight the general envisioned infrastructure in Section 2 and presents our efforts for a flexible and extensible archiver component in Section 3. We introduce various strategies and discuss our results to accurately capture the changes of documents in Section 4 and we introduce and evaluate a heuristic to estimate the crawl time for a given set of URLs in Section 5. Eventually, we conclude and present future direction in Section 6.

2 Infrastructure Overview

The necessary infrastructure to capture and preserve the data on the Web of Data should be similar to a distributed Web crawler infrastructure [10]. The components of this infrastructure should be independent and connected via a P2P network. We refer to the single unit of a web crawler as archiver which consists of the typical components of a web crawler and has the task to politely download and store the content of a set of URLs. Each archiver maintains a set of URIs of data sources and needs efficient methods to capture and store the changes in the content of the data sources. This requires an additional scheduling component which is able to adapt the re-crawling frequency of documents based on their change history. In addition, we envision that the archiver units are connected in a P2P network and that each single unit is able to correctly compute the number of URLs it can handle with the available resources, taking into account per-domain crawl delays, the number of URLs per domain and the crawl frequency of each data source. Figure 1 depicts such an infrastructure where the connections between the archiver units

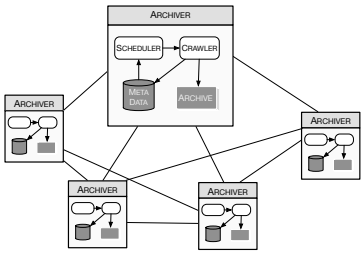


Fig. 1. Distributed archiving infrastructure.

should indicate that each unit can exchange basic information such as, the current capacity or which URLs are currently in the system. These information should then be used to decide to which units new URLs should be added and where to get the versions for a particular URL. We identify the following requirements for our envisioned infrastructure:

- **Distributed**, the infrastructure needs to be distributed since no single party can provide the necessary resources to capture the changes of the Web of Data. Clearly, a company such as Google or maybe even the Web archive has the necessary infrastructure, but we do not see in the near future any efforts to capture the changes on the Web of data. As such, preserving the Web of Data can be achieved if several parties join efforts by archiving a manageable subset of documents based on the available resource. One further advantage of the distributed architecture is that the URLs can be geographically distributed which can improve the throughput [2].
- **URL partitioning**. There is a need for a smart partitioning of the documents across the participating parties to guarantee an optimal use of the available resources. In addition, the infrastructure should ideally assign the same URL to two different parties to ensure that a document is crawled even if one party experiences some problems or a down time.
- **Crawling of different formats using different access methods** The single archiver units should be able to crawl different formats by using different access methods due to the diversity of the Web of Data. For instance, the crawling of RDF documents requires very often content negotiation to correctly access the RDF representation of a data source or in case of an SPARQL server, the crawler needs to use the SPARQL protocol and SPARQL queries to download the content. One effort in such direction which we build on is the LD-SPider framework [6] which is also used in the Dynamic Linked Data

observatory and provides mechanism to request RDF representation of data sources.

- **Adaptive (re-)scheduling component** A core requirement is an adaptive (re-)scheduling component which is able to efficiently schedule the download frequency for a data source to capture and preserve (in an ideal case) all content changes. The efficiency and quality of such an infrastructure can be measured by the accuracy of which changes are captured. Ideally, the archive preserves a snapshot of a document whenever the content change. There exists a plethora of work around rescheduling algorithms on the Web of documents such as the influential work of Cho et. al. [4, 5] or more recent work [15, 13]. However, common to all those works is the assumption that documents change according to a certain distribution such as the Poisson distribution and to the best of our knowledge such distributions could not be verified for the changes on the Web of Data [7]. As such, we have to assume that the data source on the Web of Data do not follow any particular change distribution.
- **Resource planning**: We assume that each party has only a certain amount of resources available and should be used in an optimal way. That is, that the archiver needs to correctly estimate the number of URLs that can be processed given the specifics for the available resources. In addition, the resource planning component needs to also consider the change frequency of the data sources and the current recrawling frequency. These challenges are closely related to the research area about index maintenance and freshness[11, 14] where the aim is to use the available resources to keep the content of given set of documents as current as possible. However, one important difference in our assumption is we want to estimate how many documents we can process to capture all changes. To do so, we need first to be able to compute the accurate scheduling frequency and the crawl time for a set of URLs. We will investigate the related efforts in future work but do not yet address this issue in this work.

3 Archiver, the datamonitor architecture

We started to develop the archiver component of our envisioned infrastructure and discuss the general architecture depicted in Figure 2. Our archiver, called datamonitor, consist of the following loosely coupled components:

- The **meta data** store uses currently a Postgres database and stores all vital information such as crawl logs, scheduling information and

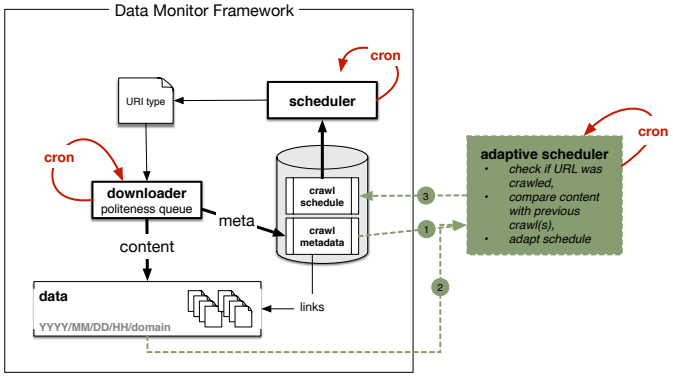


Fig. 2. Archiving infrastructure.

also aggregated statistics. For instance, one crawl log entry contains information about the HTTP response header, the download time and a change state indication if the document changed compared to the last download based on a content checksum. Another table in the meta data store contains information about the next crawl time and the current crawl frequencies for each URI.

- The **scheduler** component periodically access the meta data store and retrieves the URLs for the next crawl which are then processed by the downloader component
- The **downloader** component retrieves the content of a list of supplied URLs and archives the downloaded data and collected meta data. The framework is a multi-threaded Web crawler which respects the robots.txt protocol[3] and crawl delays and uses a politeness queue to avoid overloading the servers with too many requests [7]. Our implementation provides the flexibility to use various download handlers to deal with different access mechanisms or to use particular parsers for specific content formats.

We decided that the downloader and scheduler component should run in regular intervals to make the framework more robust rather than an online system in which the downloader component would listen if new URIs should be crawled. However, this online design would require solutions to periodically persist the current state of the system to be able to recover the state in case of unexpected system crashes, e.g., due to memory exceptions or short term shutdowns. In our current design we

[3] http://www.robotstxt.org

can assure that the different components run independently and in fixed intervals and that a problem in one crawl does not affect the next crawls. We decided to start our scheduler and crawler every hour. As such, we should ideally schedule the crawl time of the URLs in such a way that each crawl can be completed in less than one hour to avoid an overlap of crawls which can cause a resource problem.

Another feature of our architecture is that the components are flexible enough to adapt to various scenarios. With the current architecture it is easy to extend our checksum based content change detection and integrate for example file format tailored algorithms or graph isomorphism check to determine more accurately changes in RDF files.

4 Rescheduling strategies

One of the core algorithms in this framework is the **adaptive scheduler** which determines the next crawl time for URLs based on the content change information and current download frequencies (see Figure 2). The literature lists several approaches to reschedule crawls for HTML data (e.g., [4, 3], which mainly assume that the change rate of documents follow a Poisson process with an average change rate of λ. However, there are no confirmed studies that the assumption of a Poisson process also holds for the sources on the Web of Data. In fact, researchers showed that the change rates of Wikipedia article do not follow a Poisson process [1]. As such, we aim on a strategy which make no assumption about the change frequency distribution.

4.1 Strategies

Next, we introduce various strategies which we evaluate in terms of how accurately they capture the changes of documents. Figure 3 depicts different strategies and how they capture or miss changes of a document. The top row shows the actual time points for the changes of a document over 20 time units with an average change rate of $\lambda = 0.5$ changes per time unit (10 changes in 20 time units). Below are two different strategies; the middle one accesses the document with a change frequency of λ and the bottom one simulates a simple adaptive strategy which increases the download frequency if we observe more changes and decreases the frequency if we observe less changes. On the side we show the numbers of changes captures and downloads performed. We can see that the middle strategy is overall better since the bottom strategy performs three more downloads to captures the same amount of changes.

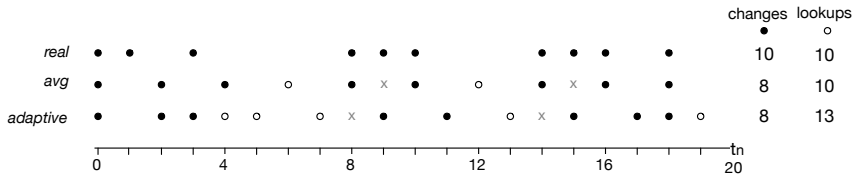


Fig. 3. Visualisation of various change strategies.

Next, we present our strategies to decide if to increase or decrease the crawl frequency for a document based on the history of observed changes. The crawl frequency for all strategy is fixed between a minimum and maximum value.

`fix` **and** `dyn`**: Dynamic crawl frequency adaptation based on a fixed number of snapshots** This strategy determines if the crawl frequency should be changed based on a fixed number of last observed snapshots. First we determine how many observed snapshots with the same crawl frequency will be considered. We use two different methods:

`fix` uses the last two snapshots

`dyn` determines the number of snapshots based on the current crawl frequency. We use one snapshot if the frequency is higher than 2 month, 2 snapshots if it is higher than 1 month, 3 snapshots if higher than 1 weeks and 4 snapshots if higher than 1 day.

Next, we increase the frequency if the document changed in all snapshots and decrease the crawl frequency if the document did not change, otherwise we do not adapt the frequency. The actual frequency change depends again on the current frequency: We increase the frequency by a factor of 1.5 in case the old crawl frequency is higher than 1 month and use a factor of 2 otherwise. Similarly, we decrease the frequency by a factor of 1.5 in case the current crawl frequency is lower than 1 month, otherwise we decrease by a factor of 2.

`window`**: Dynamic crawl frequency adaptation based on a window of snapshots:** This strategy is similar to the previous one. However in contrast, we compute the ratio of detected changes divided by number of snapshots. We consider either the last ten snapshots or half of the available snapshots. We use the following formula to compute the new frequency based on the computed ratio: If the ratio r is higher than 0.9 we increase the frequency by dividing the current frequency by a factor of 3, if $r > 0.75$ by a factor of 2 and if $r > 0.6$ b a factor of 1.5. Similarly, we decrease the frequency by multiplying the current

frequency by a factor of 3 if $r < 0.1$, by a factor of 2 if $r < 0.25$ and by a factor of 1.5 if $r < 0.4$. The general idea behind this is that the higher the ratio the more we increase the frequency and the lower the ratio the more we decrease the frequency.

`state-1` and `state-2` **Dynamic crawl frequency adaptation based on state change probabilities** In this strategy we apply the markov chains approach to determine the likelihood to observe a change in the next download considering the past observed change. We separately maintain the knowledge about the sequence of observed states for each crawl frequency. The use this information to compute the probability that the document changes in the next download based on this state-change knowledge base. Currently, we have two variations of the markov models:

`state-1` considers only the last state of a document to compute the likelihood of the next state. Table 1 shows an example knowledge

Table 1. Example of occurrence of different states

	$n-1$	
n	0	1
0	10	50
1	40	10

base in which we store how often two change states appeared ('0' indicates no change and '1' indicates a document change). For instance, the value ($n = 1$, $n - 1 = 1$) indicates that we observed 10 times that the document changes in a row and that we observed 40 times a change after a non-change($n = 1$, $n - 1 = 0$). We use this information to determine how likely it is that the document changes in the next download: For instance considering our example table: $P(1|0) = \frac{40}{50} = 0.8$

`state-2` considers the last two change states of a document to compute the likelihood of the next change state. In contrast to `state-1`, we store the information about the occurrences of the last two change states.

Eventually, we compute the new crawl frequency based on the computed likelihood to observe a change in the next crawl. We use the same frequency in/decrease factors as in the `window` strategy.

`gold` **Gold standard** We also use a reference strategy which downloads the documents according to their overall change frequency which we

assume is know in advance. To do so, we calculated the average change frequency for each page based on the actual Wikipedia revision history.

`week` In addition, we simulate a crawl with a fixed frequency of 1 week.

4.2 Evaluation

We evaluate the performs of our strategy with two measures from Information Retrieval area, namely the recall and precision. In an ideal case we would capture all changes of a document exactly shortly after they happen. In this ideal case, we would capture all changes and would only require the same number of downloads as changes. As such, we compute the **recall** of a strategy by counting the changes we observed and divide it by the real number of changes that actually happened. The **precision** is defined as the number of observed changes divided by the number of downloads.

Data We use the revision history of Wikipedia changes as our corpus to test various strategies for adapting the re-crawl frequency for documents to capture and preserve their changes. The reason for this is the already mentioned observation that the change rates of Wikipedia article do not follow any particular distribution and that the revision history of the articles span over several years, providing us a large enough time span to test our strategies.

To collect our evaluation corpus, we performed the following steps: 1) we randomly selected Wikipedia articles and gathered their revision history using the provided API.[4] Next, we filter out articles for which the revision history is shorter than 3 years and discard revisions which are declared as minor revisions. Eventually, we grouped the articles based on their average change rate which is computed by the number of revisions divided by the history time. Eventually, we use the revision histories of 2660 articles with varying average change frequencies and that provide a history of more than 3 years.. Table 2 provides the detailed overview about the distribution of documents according to their change frequencies. We aimed to select 300 documents for the various subsets.

4.3 Results

We simulated a crawl for each document over 3 years with the various strategies and present the aggregated recall and precision values for the

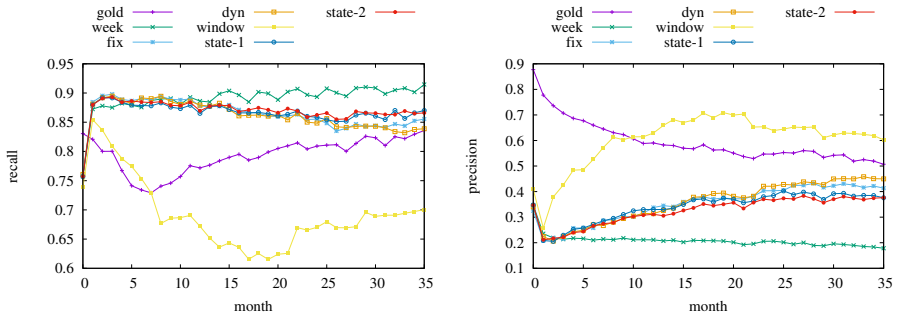
[4] http://en.wikipedia.org/w/api.php

Table 2. Overview of the dynamicity of our data corpus

`Avg. change frequency`	number of documents
$7\text{d} > \lambda > 2\text{d}$	188
$14\text{d} > \lambda > 7\text{d}$	295
$1\text{m} > \lambda > 14\text{d}$	300
$2\text{m} > \lambda > 1\text{m}$	300
$4\text{m} > \lambda > 2\text{m}$	300
$6\text{m} > \lambda > 2\text{m}$	300
$\lambda > 6\text{m}$	286
total	2660

different set of documents (see Table 2) and for all 2660 documents. We fixed the minimum crawl time to 1 day and the maximum to 6 month.

Figure 4 shows how the various strategies perform across all documents. In general, we observe that all strategy trade recall for precision or vice versa. We also see that the strategies perform overall very similar and only the `week` and `window` strategy show two extreme behaviours. The `week` strategy performs overall the best in terms of recall and worst



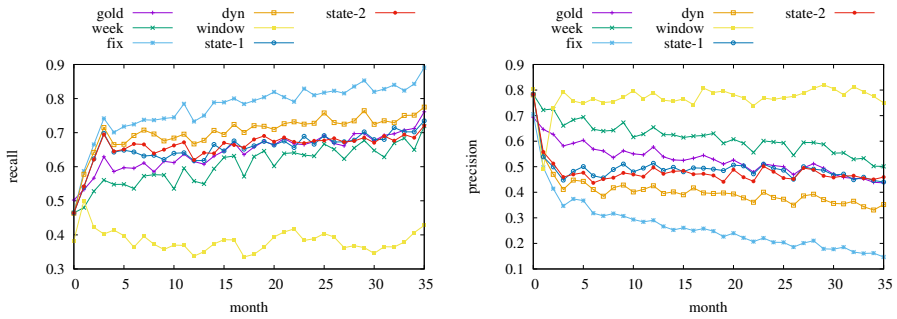
(a) Recall (b) Precision

Fig. 4. Recall and precision for all documents.

in terms of precision. This is to be expected since 92% of our documents have an average change frequency of more than 1 week and it is very likely that we capture most of the changes, however the strategy also requires a significant number of unnecessary lookups. Next, we can see that the probabilistic strategies `state-1` `state-2` are the second best in terms of recall and second worst in terms of precision. The fixed snapshot strategies `fix` and `dyn` follow closely the strategies with the markov models.

The `window` strategy in contrast trades recall for a high precision. An interesting observation is also that our strategies seems to get better over time in terms of precision which is one of the desired effect and indicates that the adaptive rescheduling can improve the overall performance.

Next, we discuss the results for selected sets of documents. Figure 5 shows the precision and recall for documents with an average change frequency between 2 days and 1 week. We see that only the `fix` and



(a) Recall (b) Precision

Fig. 5. Recall and precision for documents with 7d $> \lambda >$ 2d.

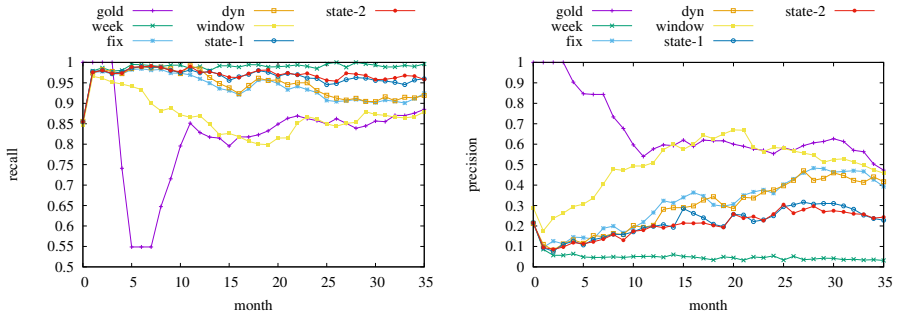
`window` strategy show extreme behaviours. The probabilistic strategies (`state-*`) are close to the performance of the actual change frequency (`gold`), indicating that with our current parameters we are able to predict with a good accuracy the actual change frequency for documents.

Figure 6 shows the results for the documents with λ between 4 and 6 month. Surprisingly, the `window` strategy performs very similar to the actual change frequency, while the other strategies achieve a high recall but with low precision. Again, we can observe that the probabilistic based heuristics perform better in terms of recall and worse in terms of precision as the fixed snapshots strategies.

We observe similar patterns for the other sets of documents and have to unfortunately omit the plots due to space limitations.

4.4 Discussion

Overall, we can conclude that so far the `window` strategy should not be considered due to its very low recall values. The main difference between the fixed snapshot and probabilistic strategies is that the former have a smaller trade off between recall and precision. However, the strategies using the markov model perform very similar for all subsets while the other



(a) Recall (b) Precision

Fig. 6. Recall and precision for documents with 6m $> \lambda >$ 4m.

strategies can have different trade-offs for different subsets. As such, we will in future work focus on studying in more detail rescheduling algorithms based on markov models and also will use them for now in our architecture.

5 Crawl resource estimation

The second experiment evaluates a heuristic to estimate the crawl time given a set of URLs and the information gathered from previous crawls. The problem we address is to compute the crawl time for a given number of URLs using a maximum number of threads. We first group the set of URLs by their domain and estimate the crawl time and number of threads we can use to crawl each domain based on the crawl delay. We use the average download time for a document for a given domain and the general crawl-delay per domain. This allows us to estimate crawl times for URLs which are not yet in our system, however, we also need to assign default times and delays which are currently set to 1sec for the domain-crawl delay and to 2 secs for the average crawl time.

We compute the number of maximum threads to crawl the URLs for one domain by dividing the average crawl time c by the domain-delay d ($\mathbf{threads} = \frac{t}{d}$). Figure 7 shows an example of a crawl process for 6 documents with a given domain-delay (on the top axis) and an average document crawl time which is more than three times the delay. The resulting number of threads in this scenario is 3 and the process would be as follows: Thread 1 (t1) starts to crawl doc1 and after the delay time another thread can crawl the next document (t2 and doc2), and so on. Once we have the number of threads we compute the total

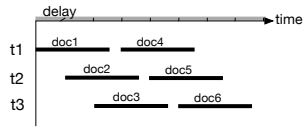


Fig. 7. Crawling URLs of one domain with multiple threads.

crawl time as follows $\textbf{crawltime} = \frac{docs*t}{threads} + d \times (threads - 1)$ We map the estimation of the total crawl size in this multi-threaded scenario to a bin packing problem and apply a first-fit bin-packing algorithms [9]. The bin size corresponds to the crawl time and we distribute the urls per domain over the bins. We can derive the total number of required threads and total crawl time once the algorithm terminates. Next, we briefly describing the algorithm: Initially, we set the size of the bins to the maximum crawl times over all domains. Next, we process the domain groups in descending order by the number of threads and assign each domain to the bin which minimises the bin size or create a new bin in case the domain does not fit in any bin (that is that the current size of the bin plus the crawl time for the domain would exceed the total bin size). After we processed all domains we sum up the maximum number of domain threads per bin to compute the total number of threads needed to crawl all URLs. If the total number of threads exceeds our maximum number of thread[5] we rerun the whole process and decrease the maximum number of threads that can be assigned to one domain. As such, in each round we slightly increase the overall crawl time since we have to assign less threads to each domain. Once we find a packing for which the total number of threads is less or equals our maximum number, we take the bin size (the longest crawl time for a domain) as the estimated crawl time.

5.1 Evaluation

We performed first experiments which our crawl time estimation algorithms based on 1388 crawls with crawl times ranging from less than 30 minutes to over 2 hours. We evaluate by how much we over or underestimated the actual crawl time. Table 3 shows the average under or overestimation fraction for the various crawls grouped by their frequency. We can see that our current heuristic tends to overestimate the crawls by a large factor. In contrast, the underestimation is very small considering the overall crawl times. For instance, we underestimate the crawls which

[5] The maximum number of threads depends on the available hardware

take around 60 mins by around 10 minutes. Overall, the results are not entirely satisfying and we will investigate improved algorithms in future work. However, the results are already encouraging and actual usable for the overall scheduling of URLs based on their re-crawl frequency with the goal to schedule the URLs in such a way that each crawl should take around one hour. The results show that we tend to underestimate the crawls resulting in overlaps of an average of 10 minutes for two consecutive crawls. While overestimating the crawl time does not cause trouble for the scheduling. However, due to the high overestimation we could add more URLs to the crawls and do currently not optimally use the available resources.

Table 3. Results of crawl time estimation

Actual crawl time	Overestimate (crawls)	Underestimate (crawls)	total crawls
$t < 30min$	78% (202)	-16% (154)	356
$30min < t < 60min$	134% (25)	-18% (46)	71
$60min < t < 120min$	638% (19)	-11% (51)	70
$120min < t < 180min$	1.3% (15)	-1% (215)	230
$180min < t < 240min$	- (0)	-36% (4)	4
$t > 240min$	- (0)	-44% (28)	28

6 Conclusion

We presented in this seminal work our vision of an infrastructure to preserve and archive the changes on the Web of Data and outlined some core requirements for such an infrastructure. In addition, we presented our efforts in developing and researching the core component of this infrastructure and conducted two initial experiments. The first experiments evaluated various heuristics to reschedule the crawl frequency of URLs to accurately capture the changes of the data source. Our findings indicated that a strategy based on state-change transitions probabilities provide promising results and indications to concentrate more efforts on this in the future. The second experiment evaluated a simple first-fit bin packing algorithm to estimated the crawl time for a set of URLs based on their average domain download and crawl-delay times. Our results show that the average underestimation is very low, however the approach tends to overestimate the crawl time by large.

Our future work will concentrate in improving both heuristics with the goal of developing algorithms to optimally schedule and reschedule

the crawl time of URLs so that the system is using the available resources in an optimal way and is able to capture the changes of the source on the Web of Data as timely and accurately as possible.

Acknowledgements This work was partially funded by the "Jubiläumsfond der Stadt Wien 2014".

References

1. Rodrigo Almeida, Barzan Mozafari, and Junghoo Cho. On the evolution of wikipedia. In *Proceedings of the First International Conference on Weblogs and Social Media, ICWSM 2007, Boulder, Colorado, USA, March 26-28, 2007*, 2007.
2. B. Barla Cambazoglu, Vassilis Plachouras, Flavio Junqueira, and Luca Telloli. On the feasibility of geographically distributed web crawling. In *Proceedings of the 3rd International Conference on Scalable Information Systems*, InfoScale '08, pages 31:1–31:10, ICST, Brussels, Belgium, Belgium, 2008. ICST.
3. Carlos Castillo, Mauricio Marín, M. Andrea Rodríguez, and Ricardo A. Baeza-Yates. Scheduling algorithms for web crawling. In *(WebMedia & LA-Web 2004), 12-15 October 2004, Ribeirao Preto-SP, Brazil*, pages 10–17, 2004.
4. Junghoo Cho and Hector Garcia-Molina. The evolution of the web and implications for an incremental crawler. In *VLDB 2000, September 10-14, 2000, Cairo, Egypt*, pages 200–209, 2000.
5. Junghoo Cho and Hector Garcia-Molina. Estimating frequency of change. *ACM Trans. Internet Techn.*, 3(3):256–290, 2003.
6. Robert Isele, Jürgen Umbrich, Christian Bizer, and Andreas Harth. LDspider: An open-source crawling framework for the Web of Linked Data. In *Posters&Demos at International Semantic Web Conference (ISWC)*. 2010.
7. Tobias Käfer, Ahmed Abdelrahman, Jürgen Umbrich, Patrick O'Byrne, and Aidan Hogan. Observing Linked Data dynamics. In *Extended Semantic Web Conference (ESWC)*, pages 213–227. Springer, 2013.
8. Jens Lehmann, Robert Isele, Max Jakob, Anja Jentzsch, Dimitris Kontokostas, Pablo N. Mendes, Sebastian Hellmann, Mohamed Morsey, Patrick van Kleef, Sören Auer, and Christian Bizer. Dbpedia - A large-scale, multilingual knowledge base extracted from wikipedia. *Semantic Web*, 6(2):167–195, 2015.
9. Rhyd Lewis. A general-purpose hill-climbing method for order independent minimum grouping problems: A case study in graph colouring and bin packing. *Computers & OR*, 36(7):2295–2310, 2009.
10. Julien Masanès. *Web archiving*. Springer, 2006.
11. Yadu Nagar and Niraj Singhal. Article: A users search history based approach to manage revisit frequency of an incremental crawler. *International Journal of Computer Applications*, 63(3):18–22, February 2013. Full text available.
12. Jinfang Niu. An overview of web archiving. *D-Lib Magazine*, 18(3/4), 2012.
13. Kira Radinsky and Paul N. Bennett. Predicting content change on the web. In *Sixth ACM International Conference on Web Search and Data Mining, WSDM 2013, Rome, Italy, February 4-8, 2013*, pages 415–424, 2013.
14. Marc Spaniol, Dimitar Denev, Arturas Mazeika, Gerhard Weikum, and Pierre Senellart. Data quality in web archiving. In *Proceedings of the 3rd ACM Workshop on Information Credibility on the Web, WICOW 2008, Madrid, Spain, April 20, 2009*, pages 19–26, 2009.
15. Qingzhao Tan and Prasenjit Mitra. Clustering-based incremental web crawling. *ACM Trans. Inf. Syst.*, 28(4):17, 2010.