How to Trace and Revise Identities*

Julien Gaugaz, Jakub Zakrzewski, Gianluca Demartini, and Wolfgang Nejdl

L3S Research Center, Leibniz Universität Hannover
Appelstrasse 9a, 30167 Hannover, Germany
{gaugaz,zakrzewski,demartini,nejdl}@L3S.de

Abstract. The Entity Name System (ENS) is a service aiming at providing globally unique URIs for all kinds of real-world entities such as persons, locations and products, based on descriptions of such entities. Because entity descriptions available to the ENS for deciding on entity identity—Do two entity descriptions refer to the same real-world entity?—are changing over time, the system has to revise its past decisions: One entity has been given two different URIs or two entities have been attributed the same URI. The question we have to investigate in this context is then: How do we propagate entity decision revisions to the clients which make use of the URIs provided by the ENS?

In this paper we propose a solution which relies on labelling the IDs with additional history information. These labels allow clients to locally detect deprecated URIs they are using and also merge IDs referring to the same real-world entity without needing to consult the ENS. Making update requests to the ENS only for the IDs detected as deprecated considerably reduces the number of update requests, at the cost of a decrease in uniqueness quality. We investigate how much the number of update requests decreases using ID history labelling, as well as how this impacts the uniqueness of the IDs on the client. For the experiments we use both artificially generated entity revision histories as well as a real case study based on the revision history of the Dutch and Simple English Wikipedia.

1 Introduction

Given a description of an entity as, for example, persons, locations or products, the *Entity Name System* (ENS) [4] provides globally unique URIs for such real-world entities. Because entity descriptions available to the ENS for deciding whether two entity descriptions refer to the same real-world entity (i.e., entity identity) are changing over time, the system sometimes has to revise its past decisions: One real-world entity has been given two different URIs[1] (e.g., `http://dbpedia.org/resource/Tim_Berners-Lee` and `http://data.semanticweb.org/person/tim-berners-lee`) or two entities have been

* This work is partially supported by the FP7 EU Large-Scale Integrating Project OKKAM Enabling a Web of Entities (contract no. ICT-215032).

[1] It is possible to find such duplicates by using a semantic search engine such as, for example, `http://sindice.com/search`

L. Aroyo et al. (Eds.): ESWC 2009, LNCS 5554, pp. 414–428, 2009.

attributed the same URI. For example, `http://dbpedia.org/page/Paris` could be the URI assigned to the city in France but, in DBpedia, it currently "disambiguates" (that is, refers) to 57 different entities. This could happen when not enough information is available when assigning a URI to an entity. When other entities called Paris are discovered new URIs should be assigned and the information be propagated. Our main research question in this paper is how to propagate entity decision revisions to the clients, which make use of the URIs provided by the ENS.

Providing a solution to the problem of propagating entity decision revisions is a crucial aspect of Semantic Web when the goal is to enable the Web of Entities. When software applications exchange information about entities it is essential to uniquely resolve the entities so that all the agents in the Web refer to the same real world entity using the same name, that is, the same URI. As entities can change over time, URI can be used multiple times by different sources to refer to the same entity, it is important to provide a framework for handling such identity revisions.

A lot of work have been done in the area of data provenance [9,11,6] and lineage retrieval [3]. However, today new challenges are present: with the growing size of the Semantic Web and with the spreading of semantic-aware user applications managing identity, the Web has to face the problem of scalability and efficiency. In this paper we analyse how to optimise identity management on the Web. In this paper:

- We describe a model for entity identifiers evolution on the Web (Section 2).
- We propose different labelling schemes allowing to dramatically reduce the number of identifier update requests to the Entity Name Service the client needs to perform, while preserving a reasonable uniqueness quality for its identifiers (Section 4).
- We perform extensive experiments measuring the quality of identifier uniqueness. The experiments, using both artificial and real revision history from Wikipedia, also measure the drop in identifier update requests when using the proposed labelling schemes (Section 5).
- We conclude the paper and describe possible future work in Section 6.

2 Application Setting and Scenarios

2.1 Motivation

It is known that "the Semantic Web can become an open and scalable space for publishing knowledge (in the form of RDF data) only if there will be a reliable (and trustworthy) support for the reuse of URIs" [5]. The ENS is providing such an infrastructure to the Semantic Web users.

It is also clear that in the current Web there is a proliferation of different URIs for the same real world entity. It is then necessary to provide a way of notifying the Semantic Web users of changes in the URIs. For example, a person such as Tim B. Lee has many different URIs assigned by different sources: it is

desirable for applications and/or people on the Web referring to him to use the same identifier.

In this paper we present methods for the propagation of Identity Decision Revisions where such decision is taken by the ENS. In the following section we describe the context of Identity Decision Revisions and introduce different approaches for propagating such information to the interested clients.

2.2 The Entity Name System

The main goal of the ENS is to enable the Web of Entities. This is accomplished by supporting the use of globally unique IDs for entities. It is important to clarify that the information about entities (i.e., attribute names and values) stored in the ENS serves solely the purpose of distinguishing entities between themselves. The ENS is defined in [4] as: "a service which stores and makes available for reuse URIs for any type of entity in a fully decentralised and open knowledge publication space." The main functionalities of the ENS are: search for the identifier of an entity, generation of entity identifiers, matching entities present in the repository with external ones, and ranking entities by similarity to a given one. Notice that the question of how the ENS actually disambiguates entity identities is out of scope of this paper and will therefore not be discussed here.

2.3 Setting

We now describe by way of an example the possible operations that can be applied to an entity. A knowledge-based system, at some point in time, might have a wrong representation of an entity: for example, initially, a system has the knowledge that a person is called John Doe, but, after having acquired more evidence, the system knows that another entity representation, with name John J. Doe, refers actually to the same entity. In this case the two entity representations as well as their identifiers need to be *merged*. Also, the entity itself can change over time (e.g. semantic evolution, evolution of documents [1]): for example, the attribute *affiliation* of a person can change over time. For these reasons, we need to define the possible changes that the entity identifiers might undergo. The following operations can be applied on an entity:

Creation. When an entity is first encountered, a new ID is generated.

Split. When the system discovers that the same entity representation describes two different real world entities, two new IDs have to be created. For example, the system knows that a person is called "Andrea Rossi" but, at some point, it discovers that there are actually a man and a woman with the same name, in the dataset.

Merge. When two entities are matched, and recognised to be the same, they are merged, and the same has to be done to their IDs.

In the following we present a real word example where the propagation of Identity Decision Revision (IDR) is important.

Wikipedia History. We can see Wikipedia as a living information repository about entities. Therefore, it is possible to use it as a scenario of IDR propagation. Entities in Wikipedia have their own pages which can grow over time. It is also true that creation, split, and merge operations appear in this collection. New pages are created, articles about similar concepts are merged together, and articles which grow too much are usually split in two (or more) different ones. It is then easy to see the need that other pages linking to a merged/split page have of knowing about this decision. Links have to be changed and in Wikipedia one approach is to use the disambiguation pages: pages about a general concept leading to all the alternative uses of that term.

2.4 Models for Propagating Identity Decision Revisions

When we want to propagate the information of IDRs there are two main possibilities of doing it: by pushing or by pulling such information.

In the *push* model the ENS server notifies all the clients of an IDR. In this scenario the ENS must be aware of all the clients using a given URI. It is easy to see that this solution is not feasible for all clients[2] on the Semantic Web It is impossible for the ENS to keep track of which client uses which ID. More, in this case there would certainly be a problem of network traffic overload: a lot of communication would be needed for propagating IDRs.

In the *pull* model a client has to ask the ENS about changes for a given ID. In this case it is a duty of the client to decide when it is time to request for an update of the ID. That is, the client has to understand when an ID is obsolete. For example, if the client receives an e-mail which contains the ID of an entity, it as to discover that it is a newer version of an ID it already has (e.g., as a result of a split operation on the ENS). In this case it is necessary to query the ENS for an up-to-date version the ID.

It is clear that the trade-off between the two models is set by how often the IDs change. If there are few IDRs, then the push approach is more efficient as all the effort is centralised.

3 Concept Definitions for Identity Revision Management

In this section we define the concepts and relative notation which are needed for describing the approaches presented in Section 4.

In this paper we assume that the ENS, named in the following as *server*, is one central machine only. It will be future work to consider the effects of a distributed ENS platform on the identity management.

Additionally, several applications using entity identifiers are present and are asynchronously submitting entity resolution requests to the server. These can be end-user application like, e.g., e-mail clients or batch processes as web crawlers. We refer to applications that submit requests to the server as *clients*. The server

[2] The users on the Web are estimated to be 1.5 billions in 2008. Source: `http://www.internetworldstats.com/stats.htm`

will respond to requests using the knowledge available in the server's *central repository*: a container of all the up-to-date entities and identifiers. When more than one ID is up-to-date for a given deprecated ID, the ENS has to use *entity matching* [3] techniques to disambiguate between those candidates. Moreover, each client has its own *local repository* containing only the identifiers known by the client, that is, all the entities encountered up to now.

Clients are exchanging information about entities and their IDs as well. This means that clients are encountering new IDs as they receive new information from other clients.

As presented in Section 2.3, split and merge operations are possible on entity identifiers. We define as *Identity Revision Graph* the directed acyclic graph representing the history of one or more entity identifiers. Using the part of such a graph that it knows, a client can detect deprecated identifiers with no need of sending a request to the server.

4 Limiting the Number of Update Requests

A quick computation shows that if 50,000 clients request updates to the server for 3,000 IDs only once a day, it represents more than 1,500 requests per second, which is already significant, and it would be interesting to be able to limit this number of request. We propose to do so by trying to update only the IDs which most need to, with the idea that, as long as one ID in a client's local repository of entities refer to only one entity in this repository, it does not need an update since local uniqueness would be preserved. This is true whether the ID in question has been deprecated (split or merged) on the ENS server.

In the following we expose different techniques allowing to limit the number of update requests while preserving local uniqueness.

4.1 Lineage Preserving ID Labelling (LPID)

Wu et al. proposed in [10] a labelling scheme for transitive closure computation on Directed Acyclic Graphs (DAGs). Computing a transitive closure in a graph is used for identifying all ancestors (or descendants) of a given node in the graph. We describe hereafter the Lite version of Prime Numbers Labelling Scheme for DAGs (PLSD) proposed by Wu et al.

Considering the entity IDs evolution as a DAG, we can use the PLSD labelling scheme described in [10] to locally detect if an ID has been deprecated. As in Fig. 1(a), if an ID A is deprecated by another ID B, then there exists a directed edge $A \rightarrow B$. As a consequence the *merge* and *split* operations mentioned in Section 2.3 are modelled as illustrated in Fig. 1(b) and 1(c) respectively.

We annotate each node of the DAG such built with a pair of natural numbers: the self-label, and the ancestors-label. The self-label is a unique prime number (i.e., from all the self-labels, a self-label appears once and only once), and the ancestors-label of a vertex is the product of its self-label with the least common

[3] Also known as record linkage, or deduplication.

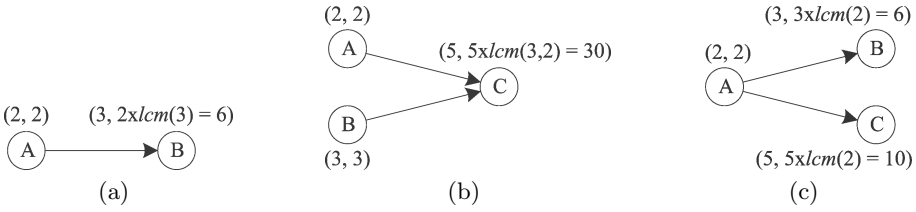


Fig. 1. (a) A is *deprecated* by B. (b) A and B are *merged* into C. (c) A is *split* into B and C.

multiple of the ancestors-labels of its ancestors (i.e., its in-component). Formally this gives:

Definition 1. *Let $G = (I, D)$ be a DAG with I a set of vertices representing IDs, and D a set of edges representing deprecation. We identify a vertex $i \in I$ by a pair $(i_{self}, i_{ancestors})$ with $i_{self}, i_{ancestors} \in \mathbb{N}^+$, where*

i) i_{self} is a prime number such that there exists a bijection between I and the set $I_{self} \equiv \{j_{self} | j \in I\}$
ii) $i_{ancestors} = i_{self} \cdot lcm(a^1_{self}, \ldots, a^n_{self})$ with $a^1, \ldots, a^n$ the ancestors (in-component) of i

We call $i = (i_{self}, i_{ancestors})$ the Lineage Preserving ID (LPID), i_{self} the self-label of i, and $i_{ancestors}$ its ancestors-label. □

Lemma 1 shows that, given two LPIDs as defined in Definition 1, the fact that the self-label of one LPID divides the ancestors-label of the other LPID implies that the former is deprecated.

Lemma 1. *Let $i = (i_{self}, i_{ancestors})$ and $j = (j_{self}, j_{ancestors})$ be two LPIDs. $j_{ancestors}/i_{self} \in \mathbb{N}$ implies that i is deprecated.* □

This approach is described in more detail in [7].

4.2 List Labelling

The LPID labelling uses an integer to encodes the ancestors' self-labels in an integer, product of those self-labels. This has the advantage to require only one division to find out whether an ID is an ancestor of another ID. On the other hand, it has the disadvantage to require the use of prime numbers as self-labels, which is costly to generate when the number of IDs is high. The obvious alternative is to use any natural number as a self-label, and store the ancestors simply in a list, separated by commas for example.

The difference with Definition 1, is that self-labels are ordinary natural numbers, and not restricted to prime numbers. And the definition of the ancestors-label is changed to:

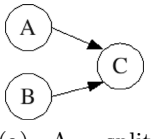
ii) $i_{ancestors} = \{a^1_{self}, \ldots, a^n_{self}\}$ is set of the self-labels of $a^1, \ldots, a^n$, the ancestors (in-component) of i

Since this labelling is equivalent to the LPID labelling, Lemma 1 can be slightly modified for list labelling: and identifier a is an ancestor of b iff $a \in b_{ancestors}$.

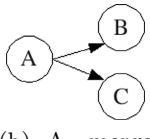
The list labelling being equivalent to the LPID labelling in terms of selection of the identifiers to ask the ENS server for un update, the only difference between the two labellings is the storage space and and the selection performance.

4.3 Merge Epochs Labelling

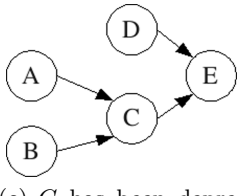
The LPID and List Labellings allow to limit identifier update requests to some identifiers being in conflict. Once the identifiers to be updated have been selected, the client has no other choice to contact the ENS server to do so. This is because the client does not know whether there was a split or not between an identifier and its descendant. Consider Figure 2(b). The client has in its repository only identifier A, and he receives identifier C. Since A has been split, the client has to ask the ENS server whether to replace A with B or with C. However, if the IDR graph is the one in Figure 2(a) and if the client knew that A has been merged with another identifier into C, it could replace A with C directly, without contacting the ENS server. In this section, we introduce an additional label, the *merge-label*, allowing the client to know whether a split occurred between any identifier and any of its descendant.



(a) A split is between A and C



(b) A merge is between A and C



(c) C has been deprecated

Fig. 2. IDR graphs. In (a) the client can replace A with C without contacting the ENS server. In (b), however, the client has to ask the ENS server to find out whether to replace A with B or C. And in (c) if the client uses the merge-label to replace A with C, it would use C which is deprecated. Whereas if the client contacted the ENS server, it would now use E which is the most up-to-date identifier for the entity considered.

As with the ancestors-label, recording which of the ancestors of an identifier can be replaced by it without asking the ENS server can be done using the *lcm* of the prime number self-labels of the ancestors, or simply the list of the ancestors. Thus, the Definitions 1 can be completed to include the merge-label as follows: the label for an identifier i is changed to $(i_{self}, i_{ancestors}, i_{merge})$, and the following definitions are added:

iii) $i_{merge} = lcm(m^1_{self}, \ldots, m^n_{self})$ with $m^k \in \{m^1, \ldots, m^s\} \equiv M$, the ancestors of i that are such that all identifiers between i (excluded) and m^k (included) have been merged with other identifiers.

Lemma 1 and its equivalent in Section 4.2 still hold for LPID Merge and List Merge labellings. For two LPID labellings $i = (i_{self}, i_{ancestors}, i_{merge})$ and $j = (j_{self}, j_{ancestors}, j_{merge})$: $i_{self}/j_{merge} \in \mathbb{N}$ implies that i can be replaced with j without risking to attribute an identifier to an entity which this identifier does not refer to. Similarly for List Merge labellings.

Drawbacks. As already mentioned, the merge labelling present the advantage to allow the client, in some cases, to replace an identifier with its descendant without contacting the ENS server. This comes at the price of two things: 1) the additional merge-label require more space to store the labelling of an identifier, and 2) the client might use deprecated identifiers, where if it contacted the ENS server, it would use the most up-to-date identifier. See Figure 2(c) for an illustration of the second point.

4.4 Baseline: No Labelling

As far as we know, since the ENS is a relatively new concept, the problem considered in this paper has not been approached before. For this reason, the baseline method which we use for comparing the above-presented techniques is the most straight forward one; that is, no labelling at all. In this case, when the client receives a description on an entity referred to by an identifier unknown to the client, the only way to make sure the new identifier does not conflict with another identifier in its repository is to simply ask the ENS server for each and every ID in its local repository.

This obviously would generate a lot of update requests to the ENS server, but it also has advantages over the labelling techniques we proposed. The first advantage is that a bigger part of the identifiers used by the client are up-to-date, since it more often asks for identifier updates. This implies that less identifiers in its repository refer to two or more entities in the world, and that less entities in its repository have more than one identifier. The second advantage is that since no labelling at all is used, the space requirement is less than when using lineage labelling.

5 Experiments

We conducted experiments aiming at comparing the different proposed labelling approaches to the baseline where no labelling is used. We compared the different approaches with respect to two main characteristics: network traffic and quality of the identifiers.

The network traffic is measured in terms of number of update requests a client addresses to the ENS server, and the size of the labelling metadata. The quality of the identifiers answers the question: how unique are the identifiers? Ideally the identifiers are globally unique, i.e. for each entity there exists exactly one identifier, and each identifier identifies exactly one entity. To measure this we used at the client level: the number of deprecated identifiers, the repository size, the number of entities per identifier, and the number of identifiers per entity.

5.1 Scenario

In our experiments, we considered the situation where a client receives semantic annotations describing different kind of entities. As the client encounters new identifiers, it first selects the identifiers in its repository which need update, according to the considered labelling scheme. The labelling scheme is one of the ones presented in Section 4: LPID, LPID Merge, List, List Merge, or the baseline without labelling. In each case it does its best to request update for all identifiers the scheme detects as deprecated. In the baseline case, since it does not have any information on which identifiers are outdated or not, it requests updates for all identifiers in its repository, plus the newly received identifiers. Once no more identifier is detected as outdated by the considered labelling scheme, it waits for a new arrival of identifiers, and repeats the process.

Before requesting for identifier updates, the merge labelling schemes perform the local replacements when applicable. Once the ENS receives an update request for identifier i, it returns one random[4] leaf identifier of the IDR graph which is a descendent of i. Howerver, for each identifier i to update, the returned up-to-date identifier is the same independently of the labelling scheme evaluated; this is important for the results to be comparable. The sets of identifiers added to the client are the same and in the same order for all evaluated labelling schemes as well.

5.2 Evaluation

In the remainder of this section we will use the following abbreviations for referring to the labelling schemes: LPID (Lineage Preserving IDentifier labelling), LPIDMerge (Lineage preserving identifier labelling with Merge labelling), List (List identifier lineage labelling), ListMerge (List identifier lineage labelling with Merge Labelling), and Classic for the Baseline without labelling.

Measuring Uniqueness. When identifiers are unique, it means there is a bijection between the set of identifiers and the set of entities. Informally this means there is exactly one identifier for each entity, and only one entity referred by each identifier. As we see unicity depends on the set of identifiers and the set of entities. We will talk about *local uniqueness* when considering the set of entities in a client's local repository and the set of identifiers related to those entities. And we will talk about *global uniqueness* when considering the set of entities in ENS server's repository and the set of identifiers related to those entities.

We will focus in this paper on global uniqueness. The reason is that we started to consider the problem of how to transmit identity decision revisions from the ENS server to the client, taking the point of view of the server, that is, global. We will then assume that each identifier in the client's repository refers to exactly

[4] In practice the ENS performs a matching operation, but since this is not the focus of this paper, we chose the identifier randomly, for the sake of simplicity.

one entity in the same repository, and devise a measure for how globally unique the identifiers in the client's repository are. This means we want to know how many entities in the world an identifier used in the client's repository refers to; and how many distinct identifiers there is in the client's repository for one entities in that repository. Those are the measures we will use in those experiments to compare the impact on uniqueness quality of the evaluated labelling schemes. We call them *number of entities per ID* and *number of IDs per entity*.

Measuring Network Traffic and ENS Server Workload. Identifier update requests makes use of two resources of concern in our scenario: 1) network bandwidth, and 2) ENS server workload. Each update request from a client needs to be sent over the network, including the identifiers to update, and if required the description of the entities whose identifier is to be updated. For each identifier update request the ENS server receives, if there is more than one leaf descendent, the server has to perform an entity matching process to find out which is the present up-to-date identifier for the requested entity.

In this paper we are not concerned with the details of the process the ENS server follows to return the up-to-date identifier. However we know that both network bandwidth consumption and ENS server workload are functions on the number of identifiers whose update is requested. This is what we will call in the remainder *number of update requests*. Note that, in practice, all identifiers needed to be updated might be sent to the server in one bulk update request. In this paper we assume that one update request concerns one identifier only.

In addition, impacting on the network bandwidth consumption, the *size of the lineage labelling metadata* will be reported.

Measuring Performance. As already mentioned, the reason why we introduced the identifier lineage labelling is to allow to reduce the number of request updates, while keeping a reasonable uniqueness quality. This is mainly a performance concern, and is therefore a central measure in this paper.

We report two durations. The first is the time it takes the client, using a given labelling scheme, to select which identifiers it needs to ask the ENS server for update. We call this the *selection time*, which is null for the baseline, since all identifiers are always updated. To be able to compare performance between the baseline and the labelling schemes, we thus need to consider also the total time it takes a client to update it's identifiers. This comprises the selection time, plus the transmission time (of the requests), plus the possible entity matching time of the ENS server, plus the transmission time back. We approximated the whole to be the selection time, plus some time constant per identifier update request. We chose arbitrarily 200 ms for this time constant, which is not an uncommon round trip time for an intercontinental internet connection. It makes it a low constant for our case, considered that it also includes the time the server needs to process the request. We call it *total matching time*.

Comparing the Labelling Schemes. Since most of the above proposed measures will yield different values for the different labelling schemes (and the

baseline), we need some measure common to all of them to be able to compare them. For this we chose the the number of new identifiers added to the client.

5.3 Datasets

We experimented with 3 different IDR graphs. The two first based on the revisions history of the Dutch and the Simple English Wikipedia datasets, and the third one artificially generated. As introduced in Section 2.3, an IDR graph is defined by a sequence of three events: identifier creations, merges and splits.

Here is how we interpreted the Wikipedia revision history in terms of those three actions. When a Wikipedia page is created, it obviously corresponds to an identifier creation. The first revision of a Wikipedia page where a redirect appears is considered as a merge between the two identifiers corresponding to the Wikipedia page and its redirect target page. And the first revision of a page where a disambiguation tag appears is interpreted as a split for this page's identifier to the ones corresponding to its targets.

The artificial IDR graph was generated with the following probability distributions. At each iteration, we first choose what action to perform[5]: creation ($p = 0.77$), split ($p = 0.08$) and merge ($p = 0.15$). If split or merge was picked, the identifier(s) to which the action applies are picked at random (uniform distribution); and the number of identifiers into which the identifier is split, or the number of identifiers to merge follow a Zipf distribution with $\sigma = 2$, a maximum of 5, and a minimum of 2.

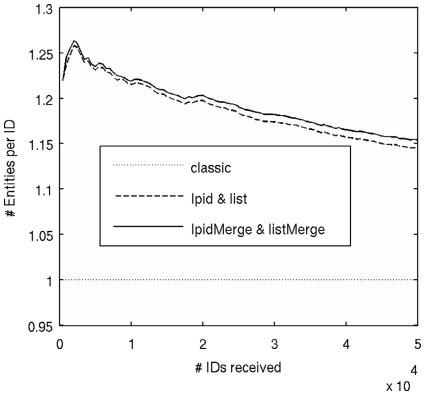
5.4 Experiment Results

We present in this section the results of our experiments. When significant difference is mentioned, this has been confirmed with a one-way ANOVA[6] with 5% of significance. For most of the graphs we present the results for each of for IDR graphs: artificially generated, simple English Wikipedia, and Dutch Wikipedia. For each we added to the client 100 times 500 (= 50,000) identifiers, and since the Dutch Wikipedia is the only having enough entities to allow it, we also tried 200 times 500 (= 100,000) identifiers for this IDR graph. Which gives us 4 different simulation for each scheme, which we note 'artificial', 'simplewiki', 'nlwiki100' and 'nlwiki200' respectively.

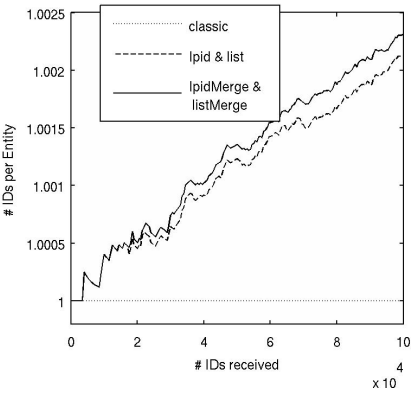
Identifiers Uniqueness. We show in Figure 3(a) the average number of entities per identifier at different steps (number of identifiers received by the client) in the simulation. The value for the classic scheme is one, as expected. For all datasets, the difference between using or not the merge-label happens to be non significant. The difference between the classic and the labelling schemes is visibly significant.

[5] Probabilities are estimated using statistics from Wikipedia articles operations.

[6] One-way ANalysis Of VAriance (ANOVA) is similar to the T-Test, but for more than two series of observations. See [8] for more details.



(a) Number of Entities per Identifier for the artificial IDR graph



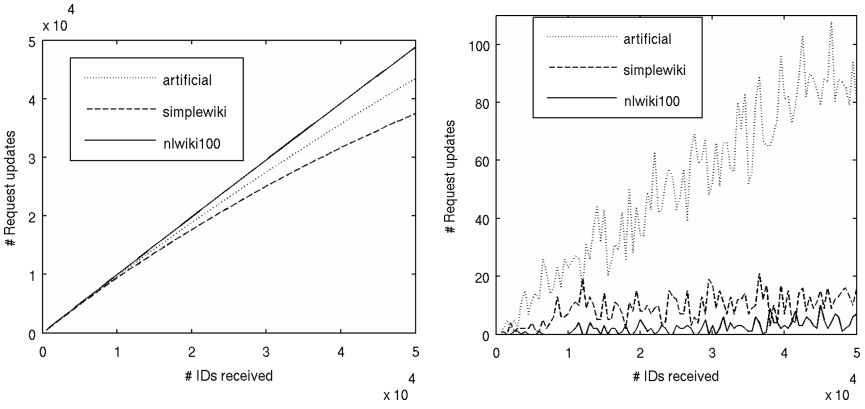
(b) Number of Identifiers per Entity for Dutch Wikipedia IDR graph with 100,000 IDs received

Fig. 3. Uniqueness Quality Measurements

In Figure 3(b) we present the average number of identifiers per entity for each iteration step. Whereas for the classic scheme we have the expected values of one, it is significantly different from the others, but merge-labelling makes a significant difference only for the IDR graphs of Simple English and Dutch Wikipedia on 100,000 identifiers.

We note that the number of entities per identifier tends to decrease over time, which is a good sign in the sense that it converges to the ideal case of one entity existing on the ENS server per identifier in the client's repository. However the number of identifiers per entity tends to augment over time, and this more rapidly while using merge-labels as not using them. This is problematic, even though it is easier to detect duplicate entities in the client's repository than reverse an accidental merge of two distinct entities because they erroneously shared a same identifier. Note however that, even if the differences are significant, the values are still quite low, which indicates that the problem would occur rarely.

Network Traffic and ENS Server Workload. We present in Figure 4 the number of identifiers the client is requesting the ENS server for update for the baseline, as well as for the labelling schemes without merge-labels (LPID and List). The merge labelling schemes are similar to the ones without, except that, as expected, the number of updates is reduced from a factor 2 approximately. Because the values for the baseline are so huge compared to the labelling schemes, we present the plots per scheme instead of per dataset as for the other measures. In addition to the obvious significant explosion of updates of the classic scheme compared to using lineage labelling—approximately 50 to 500 more than with lineage labelling—the one-way ANOVA revealed that using merge-labels or not does not change significantly the number of update requests.



(a) Classic Scheme (b) LPID & List Schemes

Fig. 4. Number of ID update requests per iteration of 500 IDs added to the client

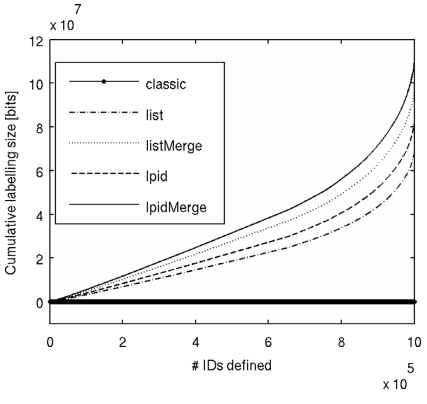
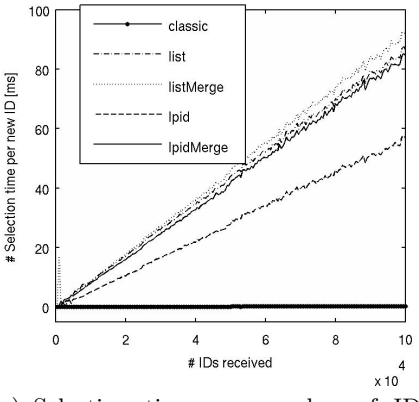


Fig. 5. Size of the labelling as the number of IDs grows on the Artificial IDR graph

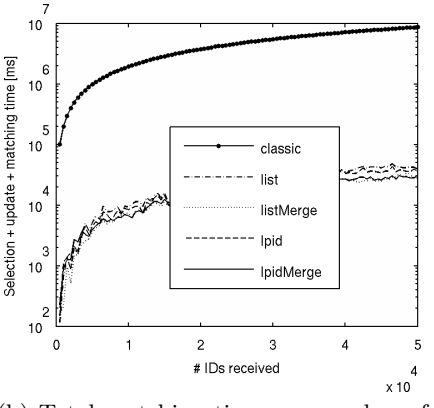
Figure 5 present the cumulative size of the lineage labelling for all the identifier schemes on the artificial IDR graph. For example, the total size needed to store the lineage labelling for one million IDs using the LPIDMerge scheme is 13 MBytes (109,124,363 bits). Since the baseline is not using any labelling, its size is zero for all datasets. Additionally, we notice that the list labelling scheme uses less space than the LPID one, and naturally, adding merge-labels augments the size of the labelling. All differences are statistically significant, and similar for other IDR graphs.

More interestingly, the size of the lineage labelling schemes augments almost exponentially. In our case the biggest labelling is of 4,500 bits for one identifier in the artificial IDR graph, which has a bit more than one million identifiers. This is still reasonable, but this size always increases over time, and this will be even more pronounced as new identifiers will be created and new split and merge operations will occur.

Performance. Figure 6(a) shows the selection time. The main observation we can make regarding those figures, is that LPID selects the identifiers to be updated significantly faster than LPIDMerge, and the two List labellings. The classic is trivially different from the others since it is null, and all other labelling perform similarly. This analysis is confirmed by one-way ANOVA.



(a) Selection time per number of ID added to the client's repository on the Dutch Wikipedia IDR graph and 100,000 identifiers added to the client.



(b) Total matching time per number of identifier update request on the artificial IDR graph.

Fig. 6. Performance measurements

In Figure 6(b), we see the total matching times, i.e. the total time needed to update a clients repository, in average, for one identifier to update. It shows clearly that the classic scheme is much slower. This is mainly due to the very high number of identifier update requests it generates. Also, there is not a significant difference between the other schemes. This is confirmed by a one-way ANOVA.

6 Conclusions and Further Work

We proposed to use identifier lineage labelling, and experiments confirmed that they allow to reduce the number of identifier update requests to the ENS server, while keeping an acceptable quality of uniqueness at the level of client's local repository. We also proposed merge-labels to further reduce the number of update requests by allowing the clients to replace an identifier by one of its descendants when this does not decrease the local uniqueness quality of the identifiers on the client's repository, which was also confirmed by the experiments. In the future we need to address the exponential growth of the identifier labelling. To reduce the size of the labelling, we intend to investigate the use bloom filters [2] to summarise list of IDs. As well, avoiding to duplicate an ID in the merge-label and in the ancestors-label should also allow to reduce the size

of merge-labelling, without solving the problem of exponential growth. The use of time-to-live (TTLs) associated to IDs indicating to the client when to request an update seems to be a promising approach as well.

References

1. Berberich, K., Bedathur, S.J., Neumann, T., Weikum, G.: A time machine for text search. In: SIGIR, pp. 519–526 (2007)
2. Bloom, B.H.: Space/time trade-offs in hash coding with allowable errors. Commun. ACM 13(7), 422–426 (1970)
3. Bose, R., Frew, J.: Lineage retrieval for scientific data processing: a survey. ACM Comput. Surv. 37(1), 1–28 (2005)
4. Bouquet, P., Stoermer, H., Bazzanella, B.: An Entity Name System ('ENS') for the Semantic Web. In: Bechhofer, S., Hauswirth, M., Hoffmann, J., Koubarakis, M. (eds.) ESWC 2008. LNCS, vol. 5021, pp. 258–272. Springer, Heidelberg (2008)
5. Bouquet, P., Stoermer, H., Niederée, C., Maña, A.: Entity name system: The backbone of an open and scalable web of data. In: ICSC, pp. 554–561. IEEE Computer Society Press, Los Alamitos (2008)
6. Cui, Y., Widom, J.: Lineage tracing for general data warehouse transformations. VLDB J. 12(1), 41–58 (2003)
7. Gaugaz, J., Demartini, G.: Entity identifiers for lineage preservation. In: IRSW, Tenerife, Spain (June 2008)
8. Miller Jr., R.G.: Beyond ANOVA: Basics of Applied Statistics. Texts in Statistical Science Series. Chapman & Hall/CRC, Boca Raton (1997)
9. Simmhan, Y.L., Plale, B., Gannon, D.: A survey of data provenance in e-science. SIGMOD Rec. 34(3), 31–36 (2005)
10. Wu, G., Zhang, K., Liu, C., Li, J.: Adapting Prime Number Labeling Scheme for Directed Acyclic Graphs. In: Li Lee, M., Tan, K.-L., Wuwongse, V. (eds.) DASFAA 2006. LNCS, vol. 3882, pp. 787–796. Springer, Heidelberg (2006)
11. Zhao, J., Goble, C.A., Stevens, R., Bechhofer, S.: Semantically linking and browsing provenance logs for e-science. In: Bouzeghoub, M., Goble, C.A., Kashyap, V., Spaccapietra, S. (eds.) ICSNW 2004. LNCS, vol. 3226, pp. 158–176. Springer, Heidelberg (2004)