A Scalable Framework for Dynamic Data Citation of Arbitrary Structured Data

Stefan Pröll[1,2] and Andreas Rauber[1]

[1] *SBA Research, Vienna, Austria*

[2] *Institute of Software Technology and Interactive Systems, Technical University of Vienna, Vienna, Austria*

Keywords: Research Data Citation, Subset Versioning, Subset Authenticity, Data Sharing.

Abstract: Sharing research data is becoming increasingly important as it enables peers to validate and reproduce data driven experiments. Also exchanging data allows scientists to reuse data in different contexts and gather new knowledge from available sources. But with increasing volume of data, researchers need to reference exact versions of datasets. Until now access to research data often based on single archives of data files where versioning and subsetting support is limited. In this paper we introduce a mechanism that allows researchers to create versioned subsets of research data which can be cited and shared in a lightweight manner. We demonstrate a prototype that supports researchers in creating subsets based on filtering and sorting source data. These subsets can be cited for later reference and reuse. The system produces evidence that allows users to verify the correctness and completeness of a subset based on cryptographic hashing. We describe a replication scenario for enabling scalable data citation in dynamic contexts.

1 INTRODUCTION

Having access to data sources is crucial for our society and recently many initiatives promote the use, reuse, sharing and open access to data. Data sharing portals[1] gain popularity and public institutions no longer hide and protect their valuable data. In the scientific domain we also face strong encouragement for sharing data and provide access to data sources which enables peers to reuse data in completely new contexts.

Research is an iterative process that requires to rerun experiments with different input data in order to verify results. Researchers often have to modify their datasets. They need to track different versions and have access to them on demand. Therefore scientists require a mechanism that allows them to work with dynamic datasets. Still they need to reference any version for later reinspection.

Sharing data without providing additional authenticity and provenance metadata for its correctness is insufficient. We need to be sure that the data is accurate, unchanged, not manipulated and complete. Evidence that allows to evaluate whether a dataset is available in the correct version, is complete and not manipulated is essential.

In this paper we present a framework that allows researchers to create, reference and cite subsets of dynamically changing data without the need for full sized data exports. We address solutions for a range of disciplines which do not yet have the tradition of using sophisticated data management tools such as large scale structured databases. Our framework for dynamic data citation allows researchers to generate and retrieve secure evidence for the integrity of their smaller-scale datasets, i.e. wide spread data formats such as CSV files.

The remainder of this paper is structured as follows. Section 2 describes the problem of dynamic data citation. Section 3 provides an overview of existing work in the area and shows its application on evolving databases which our work uses as a basis. Section 4 provides a sample use case. Section 5 provides an overview of the server side implementation and Section 6 describes a novel hashing scheme that we use for result set identification and integrity verification. Furthermore, we discuss security improvements of our approach with regards to tamper-resistance of datasets. Section 7 introduces the client component which allows researchers assembling, storing, referencing and citing datasets and sharing them with peers. Section 8 covers the query

[1] e.g. http://www.figshare.com/

Pröll S. and Rauber A..
A Scalable Framework for Dynamic Data Citation of Arbitrary Structured Data.
DOI: 10.5220/0004991802230230
In *Proceedings of 3rd International Conference on Data Management Technologies and Applications* (DATA-2014), pages 223-230
ISBN: 978-989-758-035-2

store which serves as the repository for the queries that are used to retrieve the datasets. The paper closes with a conclusion and a future outlook in Section 9.

2 MOTIVATION

Sharing data and referencing it is an important step towards a more open scientific community that actually can reproduce experiments from peers instead of relying on the paper based publication only. Classical paper publications without available data are in many cases not sufficient to reproduce experiments. Many journals now require data audits, data deposits and data descriptors for their submissions. For this reason researchers need tools that allow them tracing different versions of their datasets. Data citation deals with the question how data can be identified and thus strongly supports the culture of sharing data between scientists by allowing to precisely identify, reference, cite and credit their research data.

Researchers use various kinds of data formats as input for their experiments, e.g. CSV, JSON, XML, RDF and various office software formats. During their research scientists often need to adapt their datasets according to their requirements, they might need to update or delete values and create new subsets of their data. Each of these iterations results in a new version of the dataset. For text based data files many researchers do not use any version control software. Recently source code versioning software such as subversion[2] or git[3] have gained popularity outside the computer science departments, but still the final dataset which eventually gets published on the repository does not contain previous versions any more. Additionally datasets are mainly published as archive files on institutional Web sites where a hash string serves as a digital fingerprint. If datasets are large and consume considerable amounts of storage space, providing multiple versions with the appropriate history metadata causes a burden to the data provider. Therefore, many institutions only provide the latest version of a dataset without any previously accessible versions available.

The information about earlier versions is valuable for several reasons. First of all, versions contribute to the provenance track of a dataset and therefore are evidence how a dataset was obtained. Even if several versions of the datasets are available, without the appropriate metadata it is not possible to understand how they have been created. This is especially true for scientific disciplines where large scale database management systems are not available that would allow tracing the creation of a dataset. Secondly, the process of creating a dataset consumes resources in the form of time and money. Thus preserving information about previous datasets is economically reasonable and other peers would benefit from the historical data. Thirdly, the attribution of datasets to their authors is not possible without referencing a precisely defined dataset.

Citing subsets of datasets is still a challenge, especially when the source data is still evolving. What is needed is a solution that allows researchers to cite arbitrary subsets of their research datasets and share them with peers without the need to copy large datasets.

We developed a prototype that supports scientists in referencing, citing and sharing their datasets in a lightweight fashion. In the scientific domain there exists a huge variety of data formats that are used for specialised research in diverse domains. The bandwidth of data formats is highly diverse and the focus of research is often directed towards highly specialised data formats used in big data processing or structured databases as they are used in large scale scientific facilities such as CERN [4] or STFC[5]. But many disciplines still rely on simpler solutions such as CSV (coma separated value) (Shafranovich, 2005). Although the format was originally only used for small scale datasets, now bigger sets also need to support proper data citation (Bertin-Mahieux et al., 2011).

CSV data recently receives more attention[6] as it is a very clear and simple data format that is widely used in different domains. Also there exist a lot of different tools that allow export and import facilities in order to transform the data into a different data format that might be more complex. Therefore we chose CSV data as a reference data type for our prototypes, but our methods can be applied to any structured data format. The system that we propose in this paper can be used to create citable subsets of versioned CSV data by only storing the queries that were used to assemble a subset. In order to preserve the knowledge how this datasets have been constructed, we need to collect dataset metadata.

Independently from the actual data format used, there are only two axiomatic methods that researchers need to apply in order to compile their datasets. They need to select those attributes of a dataset which they are interested in (projection) and they have to filter these by some criteria (selection).

[2]https://subversion.apache.org/

[3]www.git-scm.com/

[4]http://home.web.cern.ch/about/computing

[5]http://pan-data.eu/STFC

[6]http://www.w3.org/2013/csvw

For tracing the creation of a dataset, we need to store the filtering and sorting operations that were used in order to generate the dataset. If we apply these operations again on versioned data, the result set will caeteris paribus return the same result set again. We provide mechanisms that ensure dataset integrity and dataset stability, i.e. stable sorting of the dataset for further processing.

In order to create and use such datasets, we require basic data definition and manipulation languages. Due to the compatibility of relational SQL databases and CSV files, we can use the powerful features of modern relational database management systems in order to manage the data.

3 RELATED WORK

Data citation is an urgent topic (Lawrence et al., 2011), yet the citation landscape is fragmented (Parsons et al., 2010) and various approaches exist (CODATA-ICSTI, 2013). In February 2014 the Data Citation Synthesis Group published their Joint Declaration of Data Citation Principles[7] which address 8 core concepts for accessible research data. The document highlights the importance of data as a first class research object which requires the same attention with regards to citations as paper publications (principle 1). Not only stimulates data citation correct attribution among peers (principle 2), but it serves even more importantly as evidence (principle 3). Whenever a statement within a publication is based upon the foundation of data, the corresponding dataset needs to be referenced. Obviously each dataset requires a unique identifier (principle 4) that allows resolving a dataset and its accompanying materials such as metadata (principle 5). These identifiers need to be available for the long term, i.e. persistent (principle 6). Data citation needs to facilitate means for researchers to assess the data's provenance and fixity (principle 7). The guidelines encourage an interoperable design that can be applied across research domain and community boundaries.

The authors of (Pröll and Rauber, 2013a) propose a model for citing evolving data from SQL databases which is based on time stamp annotated queries and versioned data. The authors describe a query centric citation approach that augments SQL queries with timing information, which can then be utilised in order to retrieve a the same data again at a later point in time. Also they present a generalised model for rendering dynamic data citable and define basic requirements for citable subsets. In (Pröll and Rauber, 2013b) the same authors applied their model on a use case and described a reference implementation. They propose several implementation strategies and describe how an existing relational database schema can be extended to support data citation. They discuss the advantages and disadvantages of several implementation variations and show how previous versions data can be retrieved from dynamically changing source data. Our work is based on these two papers and extends the model by applying a chained hashing approach which ensures data integrity of subsets. Furthermore, we apply the model to a client server infrastructure that enables researchers to create and reference arbitrary subsets from flat data files.

Fingerprinting and watermarking relational databases is a common technique to detect tampering (Li et al., 2005). The authors of (Narasimha and Tsudik, 2006) describe a method for validating the completeness and correctness of queries. In our work, we use a similar method applying a row based hash on the result set returned from a query.

4 A SCIENTIFIC USE CASE FOR CITABLE CSV FILES: SYSTEM OVERVIEW

In order to implement dynamic data citation for CSV files we use a client-server approach. The server component (see Section 5) is responsible for handling the data, managing the metadata and ensuring the integrity of the raw data. The server component is also responsible for interacting with data citation queries that return previously created subsets with the appropriate evidence of authenticity. The client (see Section 7) serves as frontend and allows users to assemble datasets. The prototype that we developed supports researchers in managing their datasets during the following exemplified experiment based on the Million Song dataset (Bertin-Mahieux et al., 2011).

4.1 Use Case Description

Music classification is a widely used method which has applications in many areas such as genre or style classification, recommender services, playlist generation etc. These systems are based on feature extraction from a potentially large set of audio files. In order to train the machine learning algorithms, specific sets of audio files and their features are required. For interoperability reasons, many tools from that domain support the CSV file format and store the fea-

[7] http://www.force11.org/datacitation

tures in this simple textual representation(Hall et al., 2009). Whenever the audio files which serve as input for such experiments change, e.g. due to an increased fidelity, detected errors in source files or newly available material, the features need extracted again and the datasets need to be updated. Researchers have to be able to differentiate between these versions on order to analyse the effects of the updated source material on their results. Therefore precise citation which considers subsets and different versions are an essential requirement. Also, researchers need tools which allow them to exchange and share data without having to deal with the overhead of managing potentially large file dumps of different versions of several subsets which they need for their work.

4.2 Supporting Automated Dynamic Data Citation

The prototype we developed allows researchers to upload their CSV data (e.g. from feature extraction application) to a Web service. The server component ingests the data and automatically transforms the data into a relational database model and bulk inserts the data, see Section 5. During the ingestion phase, the database server annotates every new record with a time stamp. As each song as a unique identifier, updates of existing song data can be detected. In this case, the server adds versioning additional metadata. No data gets overwritten or deleted, markers are used to indicate the version number and record status.

Our prototype provides a frontend which allows researchers to select specific subsets based on their personal requirements, sortings and filters. The frontend submits all selection and filtering operations to the backend, which records them in a sequential manner. The server stores adds metadata to the query such as query execution time and sorting sequences. This mechanism allows retrieving a specific version of any dataset at a later point in time. Instead of being based on specific database log file formats, this metadata remains human readable and can be utilised in any other database system in a similar fashion.

After a researcher has created a dataset, he confirms the new subset to the system. The server then iterates over the result set and computes a hash value as evidence for the integrity of the subset. The query is stored and a persistent identifier is attached to the dataset. This persistent identifier serves as a handle which can be shared with other peers and be used in publications. A resolver service may point users who enter the PID to the specific version of a dataset where it can be retrieved. Landing pages can provide additional information about the dataset such as previous versions, query text and filter terms. As the system is aware of updates and evolving data, researchers have transparent access to specific versions. There is no need of storing multiple versions of a dataset externally for the long term as the system can reproduce them on demand. As hashing methods are in place, the integrity of the datasets can be verified.

5 SCALABLE BACKEND FOR DATA CITATION OF DATA

The prototype we developed provides a Web service for uploading data. Although many RDBMS natively support importing CSV files, we used a CSV parser library in order to analyse the data files and perform data cleansing, header data generation and escaping of special keywords and characters. As the content of the files is previously unknown, the database schema is generated based on the column metadata on the fly. In this simple scenario we utilise VARCHAR fields with the longest encountered field length that is gathered during the upload process. Future versions may support more specific column data types in order to increase search and indexing performance of the system. The server automatically deploys the table schema and appends columns for maintenance metadata such as the sequence in which the data was inserted and a timestamp. After the Web service has populated the database, the researcher can utilise the frontend we propose in Section 7. The server's API currently supports sorting of arbitrary columns of the dataset in either ascending or descending order. As we currently only implemented CSV data, only text based filtering is supported. More complex filtering options will be available future versions of the prototype.

The goal of our data citation approach is to reduce the overhead for data citation to a minimum for data providers and hide them transparently from researchers. We use a second database instance denoted *DCDB* that replicates the primary database denoted *DB*. The *DCDB* server implements the data citation functionality. Hence the database *DB* only contains the latest version of the records whereas the replicated database *DCDB* is used for managing historical data. Introducing a separate data citation system has several advantages. Firstly it is possible to introduce data citation without interfering with the primary database *DB*. All additional metadata that is required in order to facilitate data citation can be moved to the *DCDB* database instance. The replicated server *DCDB* implements triggers which react on updates or deletes. Any operation that alters the original table is reflected in history tables on the replicated server. Figure 1 shows an overview of the setup.

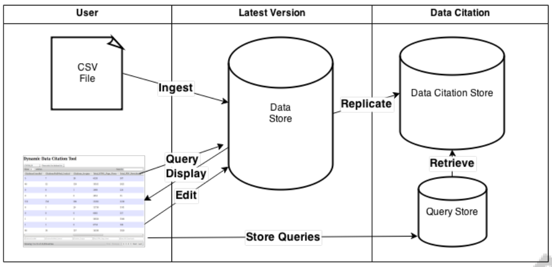

Figure 1: Replication Scheme.

The user interacts with database *DB* which handles the current state of the data. Whenever a researcher creates new subset, the query which was used is stored together with the timestamp of the query execution time in the query store. All table operations on *DB* are automatically replicated to the data citation database instance *DCDB* and annotated with versioning information. If records are updated or deleted, these changes are immediately visible at the *DB* server and replicated to the *DCDB* server instance, which maintains the historical data.

For retrieving a specific dataset, the persistent identifier allows to retrieve the query again, which then is issued against the historical data. The data citation capabilities in this setup does not cause any burden on the data server *DB* as the queries that gather historic data are only issued against the data citation server. Advantages of this scenario are horizontal scalability, which allows to introduce several data citation instances that can operate on the same data or the possibility to implement different replication strategies. Delayed replication can be used for instance to time the replication to off peak hours for further reduction of required performance of the *DB* instance.

The replicated server holds a full copy of the primary database *DB*. The overhead in terms of storage in this setup depends on the frequencies of updates on the original table in the *DB* instance. In terms of additional data the replication server automatically appends at least three columns for insertion and modification timestamps as well as the flags which indicate the record status.

6 HASHING DATASETS

Researchers may not rely the assumption that the cited data itself has not changed ever since it has been published, therefore evidence is needed. For ensuring authenticity and integrity of these archives, a checksum of the file is calculated and provided as a reference. Traditionally, checksums of datasets are based on a hash of the complete content of a data file. Tools such as md5sum or sha1sum iterate over the content of a file and compute a checksum according to a hash value. Researchers would then download the file, calculate the checksum themselves locally and compare the resulting hash with the one provided by the data publisher. The resulting hash string differs for new or changed result sets. The problem with this approach is that it requires the complete dataset as a file for each new version, hence it does not scale well for large subsets.

6.1 A Chained Approach

There are two main requirements for result set hashing: identification of content changes and recognising deviations in the sorting sequence of subsets. Several strategies exist to create a result set hash with regards to these requirements. The most obvious approach is using the complete result set as a basis for hashing. This does not scale well for huge datasets. Another approach would be to generate a hash by appending all primary keys of the result set's rows. This allows tracing the sequence, but not the content integrity. A further approach is the creation of row hashes by appending all columns of each row and computing the hash value row wise. In this case all row hashes can get sequentially appended and then serve as the basis of the hash. This approach however does not reflect the column selection that was made by the researcher.

In order to enable result set hashing with respect to integrity and sorting we only use those columns which been matched by the projection and those rows which have been included by the selection during the filtering. These rows are used for the hash value calculation of the result set. In the next step, we calculate the checksum of the particular subset by constructing a hash chain. For each row r_n, where the subscript n denotes the sequence number of the row in the result set, the system appends the data from the projected columns and concatenates the selected values to one string per row. We denote this string of appended values as rv_n. Then we calculate the hash of each concatenated string by using a one-way cryptographic hash function denoted $h(rv_n)$. For maintaining the sequence of the results in the subset, we use a chained hash approach as in the equation in Equation 1.

$$h(rv_n) = \begin{cases} h(rv_0), & \text{if } n = 0. \\ h(h(rv_{(n-1)}) + rv_n), & \text{if } n > 0. \end{cases} \quad (1)$$

We prepend each row with the hash value of the previous row in order to reflect the sequence of the records in the result set. The system iterates in this fashion over all rows of a specific subset and sequentially computes a hash of the complete result set in the correct order. As each row (except the first rv_0) calculates its hash value based on its predecessor, the sorting that was applied to the original dataset can be maintained. Figure 2 shows this scheme.

The proposed hashing method allows to detect errors in the data, i.e. insertions, deletions or modifications. Furthermore the sequence of the data and also its alignment (e.g. the sequence of the columns) of each result set can be checked against the original result set. As we calculate the hashes individually per row, the storage demand for each row is constant (e.g. SHA1 uses 160 bits per hash). The overall checksum for the result set is computed by chaining hashes. This keeps the storage demand for the creation of the hash low as it never exceeds the length $|rv_n| + |h|$ for each hashing operation. Also the scheme is agnostic regarding the hashing algorithm used.

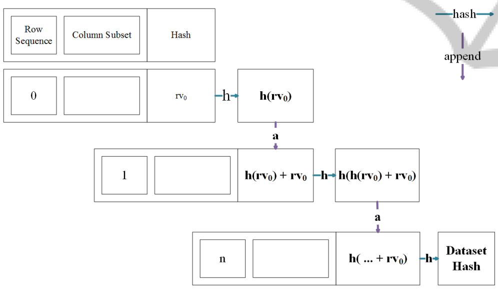

Figure 2: Hashing Scheme.

Therefore our model provides evidence that a dataset is authentic, complete and unchanged in a dynamic setting. This allows researchers to ensure that they are using the correct dataset and they can reference an explicit version of a subset with a persistent identifier. The authors of (Bakhtiari et al., 1995) present an overview of hash functions, popular hash functions which are widely used are MD5, the SHA-family of hash functions. As MD5 has been found to be vulnerable for collision attacks (Wang et al., 2004; Klima, 2005), we utilised the SHA1 hash function for generating row based hashes in our prototype implementation.

6.2 A Secure System for Storing Data Citation Metadata

The presence of hash keys alone is no guarantee for security as they can be recomputed for manipulated content. In order to harden our prototype implementation for data manipulation, we considered several mechanisms which are described in the following sections.

It is clear that the data citation database which holds the history data and their metadata requires protection from intended sabotage and unintended misuse. The same is true for the primary database. In contrast to the primary database *DB*, data is never deleted or updated from *DCDB*. Only new records along with the event type (e.g. *INSERTED*, *UPDATED* or *DELETED*) need to be inserted. Therefore the historic tables do not allow updates or deletes in a permission level.

As an additional level of security, an archival database storage mechanism needs to be deployed. The MySQL RDBMS that we use in our prototype implementation provides the ARCHIVE storage engine that does not support DELETE, REPLACE and UPDATE operations by design. This specialised storage engine only supports INSERT and SELECT statements which are the only two operations needed in this scenario. Additionally, the rows are automatically compressed upon data insertion which reduces the storage footprint of the versioned data and its metadata used for data citation. In the particular case of the ARCHIVE storage engine, it needs to be considered that this system does not support indexes, hence there is a trade-off between storage demand and data citation query performance.

6.3 Server and Client Side Dataset Validation

The queries which are stored in the query store (as described in Section 8) contain all information that is required in order to rerun the query. Hence a dataset integrity watch can be implemented by using stored procedures that periodically recompute the hash value of the datasets and compare it with the original hash value. This server side data integrity check can be provided by the API and called from the client in order to assess the integrity of a previously obtained dataset. As the hash value computation of a dataset is kept simple, it can be easily computed on the client side as well, which enhances transparency and increases the trust in research data.

7 A FRONTEND FOR DATASET ASSEMBLY

We developed a simple browser based frontend for dynamic tabular data which documents the steps applied during the creation of the dataset. The structure of the tables does not need to be known in advance as the table configuration can be loaded dynamically. This renders our approach flexible as it can be applied to any data format that can be represented in tables. The client submits requests to the Web service, which then queries the database and retrieves the appropriate result set. All computationally intensive operations such as filtering and sorting are moved towards the server side. Therefore the client becomes very lightweight. Researchers can use the frontend for browsing and creating even complex subsets from large source data sources. As the API is generic, the client can be replaced anytime by domain specific approaches. Plugins for specialised data editors can be implemented that transparently hide the communication with the server, as long as the requests are in compatible with the API of the Web service.

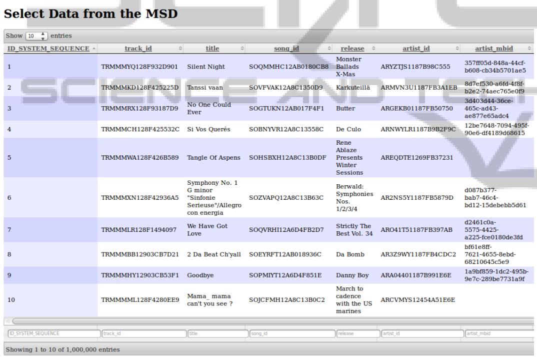

Figure 3: The Frontend for Creating Subsets.

The server generates SQL queries based on the filter criteria and applies the appropriate sorting transparently to the user. The researcher can confirm each filtering step and therefore apply different filter combinations and sortings that are applied sequentially on the dataset. Each of these operations is traced on the server side and stored in the query store, see Section 8. When the user is finished with compiling the dataset, the data can be exported as a CSV file, as JSON or other formats. The server computes the hash as described in Section 6 and attaches a persistent identifier to the query. This identifier can be used later for retrieving the same data. Updates of the data on a record level need to be reflected in the data citation database instance. Hence the CSV data either needs a unique primary key (e.g. the *track_id* in the million song dataset) or a frontend needs to be used, which allows utilising the automatically generated record sequence number. Updates can be detected by an altered hash key of the set. If a query which already exists in the query store delivers a different hash, the query gets a new persistent identifier assigned and constitutes a new version of the same set.

8 AN ALL-PURPOSE QUERY STORE

The query store collects the queries that have been used in order to create a dataset, thus preserving the information about the construction of subsets of data. Whenever a researcher uses the frontend for assembling a dataset, a query object is instantiated. This object maintains a list of all operations the researcher executed in their appropriate sequence. Each query can handle multiple filters with arbitrary filter properties and it maintains the sorting direction for each of the database columns that have been involved in the query. This knowledge is important in order to preserve the sorting sequence of the dataset, which needs to be preserved whenever a technology change forces a migration to a new data store. Figure 4 shows an ER diagram of the query store.

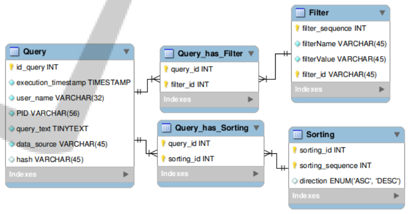

Figure 4: The Query Store Holds the Metadata.

The information about the queries can be seen as provenance data as it describes how a dataset has been constructed. Each query contains a timestamp that allows to map the query to a specific state of the database, hence only those records are fetched which have been valid during the execution time of the query. Additionally a persistent identifier such as a DOI (Paskin, 2010) can be assigned to each dataset. This allows scientists to reference a specific version of a dataset e.g. in their publication.

9 CONCLUSIONS AND OUTLOOK

In this paper we presented a framework and a prototype implementation enabling dynamic data citation for a general purpose data format. We chose the CSV format as it is used across domain boundaries, simplistic yet flexible and therefore highly popular increasingly in settings involving larger volumes of data and in dynamic data that is released in subsequent batches and integrated across versions of updated data. We thus deem it essential to facilitate convenient and transparent citation capabilities for such types of data. We presented the steps necessary for scientists to create citable subset of dynamic CSV data. We proposed a solution which consists of a server and a client component. The server side is responsible for data management, versioning, data security and citation facilities. It exposes an API via a Web service for filtering, sorting and creating datasets of arbitrary complexity that can be queried by clients. Users can upload their datasets via a Web service to the server which automatically migrates the file into a relational database.

The data is annotated with extra metadata such as original sequence of insertion, timestamps and the row hash. The client component is a simple browser based frontend which allows scientists to create citable subsets from the previously uploaded datasets. The frontend transmits each sorting or filtering operation to the server component which stores them in the query store. When the user concludes the creation of a dataset, the server rewrites the filtering and sorting information into a single SQL query and appends timing metadata. A persistent identifier can be assigned to the query and serves as reference information for the specific subset.

We presented a novel hashing scheme which allows verifying the integrity of the data and providing result sets of provably correct sorting sequences. The hashing mechanism is based on row based hashes and concatenated row hashes. For enhancing the scalability, we introduced a new replication scheme, which allows separating the live system from the data citation instance.

In future revisions of our prototype we will integrate support for several interfaces that are natively used by scientists for assembling datasets. We will develop plugins for various data editors that transparently hide the provenance data collection for creating secure datasets. Furthermore, we will develop prototypes and tools for a much broader range of data formats, hence enabling stable and secure data citation within diverse fields of research.

ACKNOWLEDGEMENTS

Part of this work was supported by the projects APARSEN, TIMBUS and SCAPE, partially funded by the EU under the FP7 contracts 269977, 269940 and 270137.

REFERENCES

Bakhtiari, S., Safavi-Naini, R., Pieprzyk, J., et al. (1995). Cryptographic hash functions: A survey. *Centre for Computer Security Research, Department of Computer Science, University of Wollongong, Australie*.

Bertin-Mahieux, T., Ellis, D. P., Whitman, B., and Lamere, P. (2011). The million song dataset. In *Proceedings of the 12th International Conference on Music Information Retrieval (ISMIR 2011)*.

CODATA-ICSTI (2013). Out of cite, out of mind: The current state of practice, policy, and technology for the citation of data. CODATA-ICSTI Task Group on Data Citation Standards and Practices.

Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., and Witten, I. H. (2009). The weka data mining software: An update. *SIGKDD Explor. Newsl.*, 11(1):10–18.

Klima, V. (2005). Finding md5 collisions on a notebook pc using multi-message modifications. *IACR Cryptology ePrint Archive*, 2005:102.

Lawrence, B., Jones, C., Matthews, B., Pepler, S., and Callaghan, S. (2011). Citation and peer review of data: Moving towards formal data publication. *International Journal of Digital Curation*, 6(2):4–37.

Li, Y., Swarup, V., and Jajodia, S. (2005). Fingerprinting relational databases: Schemes and specialties. *Dependable and Secure Computing, IEEE Transactions on*, 2(1):34–45.

Narasimha, M. and Tsudik, G. (2006). Authentication of outsourced databases using signature aggregation and chaining. In Lee, M., Tan, K.-L., and Wuwongse, V., editors, *Database Systems for Advanced Applications*, volume 3882 of *Lecture Notes in Computer Science*, pages 420–436. Springer Berlin Heidelberg.

Parsons, M. A., Duerr, R., and Minster, J.-B. (2010). Data citation and peer review. *Eos, Transactions American Geophysical Union*, 91(34):297–298.

Paskin, N. (2010). Digital Object Identifier (DOI) System. *Encyclopedia of library and information sciences*, 3:1586–1592.

Pröll, S. and Rauber, A. (2013a). Citable by Design - A Model for Making Data in Dynamic Environments Citable. In *2nd International Conference on Data Management Technologies and Applications (DATA2013)*, Reykjavik, Iceland.

Pröll, S. and Rauber, A. (2013b). Scalable Data Citation in Dynamic, Large Databases: Model and Reference Implementation. In *IEEE International Conference on Big Data 2013 (IEEE BigData 2013)*, Santa Clara, CA, USA.

Shafranovich, Y. (2005). Common Format and MIME Type for Comma-Separated Values (CSV) Files. RFC 4180.

Wang, X., Feng, D., Lai, X., and Yu, H. (2004). Collisions for Hash Functions MD4, MD5, HAVAL-128 and RIPEMD.