Change data capture of large-scale RDF data

Jindřich Mynarz and Adam Sotona

MSD Czech Republic s.r.o., Svornosti 3321/2, 150 00 Prague 5, Czech Republic,
{name.surname}@merck.com

Abstract. We designed and implemented a method for change data capture tracking large-scale RDF datasets that change in time. The method allows efficient offline reconstruction of a view of data valid at a given time from a backlog of explicit change events. It operates on a temporal data model based on RDF and named graphs. It is built with semantic web standards and implemented via SPARQL 1.1 Update. We provide an efficient open-source implementation of the method in Halyard Bulk Update, a MapReduce application for the Halyard RDF store. This implementation allows to synchronize data via change data capture in a horizontally-scalable cluster on commodity hardware. We demonstrate the method on PUBMED, a public dataset aggregating rich metadata on biomedical literature.

Keywords: change data capture, SPARQL Update, named graphs

1 Introduction

We present a method for change data capture (CDC) tracking large-scale RDF datasets that change in time. CDC covers software design patterns to track and replicate changes from remote data sources. The method can replay incremental changes in RDF datasets that are structured according to our proposed temporal data model. We focus on efficient offline reconstruction of a view of data valid at a given time from a backlog of explicit change events.

2 Related work

There is an extensive research on versioning RDF and change detection in knowledge bases. Considering the related work, [1], surveying approaches for change propagation in RDF, gave us our point of departure. Research on CDC typically focuses on relational data, such as in [2], so it has only a high-level similarity to our work. An RDF version of PUBMED, which is our use case, was included in Bio2RDF [3], however, it supported only loading in bulk.

3 Data model

We propose a CDC data model that is based on named graphs [4]. The model represents changes as ordered sets of Snapshots of Targets. What is a Target depends on the granularity of Snapshots in the source data. In general, target is *"a stable logical entity associated with a series of different values over time."*[1] It can be a single statement, a container of statements, or a whole dataset, so long it has a unique persistent identifier, such as an IRI. Snapshot is a named graph that represents the state of a Target at some time. Snapshots are described by Change Events. Change Event has a Target, a timestamp, and a Snapshot to insert, or a Snapshot to delete, or both. Change Events to be replayed in a single synchronization run are stored in a Metadata Graph, a named graph with a known IRI. The combination of Metadata Graphs and Snapshots forms Backlog. This model is formalized as a small RDF vocabulary (Fig. 1).

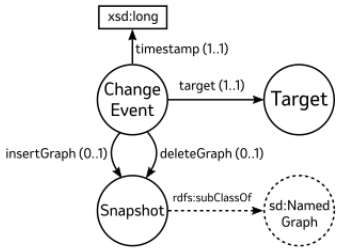

Figure 1: Backlog vocabulary

4 Replay method

In order to produce a view of a dataset valid at a given date, we replay Backlog's Change Events the timestamp of which precedes the date. For example, we replay all Change Events to get the current view. We keep the replayed Change Events to enable tracing the origin of changes and reconstructing past versions of the dataset. To replay Backlog, we filter Change Events preceding a given timestamp, order them by timestamp, and for each Change Event delete the triples from its delete graph, and insert the triples from its insert graph. Deletions are run prior to insertions to avoid retaining their intersections. Snapshot to delete need not be explicit, it may be inferred to be the Snapshot to insert of the immediately preceding Change Event for the same Target.

[1] https://clojure.org/about/state#_working_models_and_identity

5 Method implementation

We implemented the method as an extension of Halyard [5], a horizontally-scalable RDF store based on Apache HBase and Eclipse RDF4J. In order to replay Backlog efficiently we extended Halyard Bulk Update[2], a MapReduce tool that executes SPARQL Update [6] operations in parallel. The SPARQL 1.1 Update specification defines updates via set operations. Query solutions to delete or insert are first collected, deletes are removed via set difference, and inserts are added via set union. Consequently, the order of deletes and inserts within an update operation is left undefined. Order can be expressed at the level of update operations by the sequence they appear in an update request. To replay Change Events ordered by timestamp a compliant SPARQL Update engine thus needs a separate update operation for each Change Event.

Since Halyard Bulk Update diverges from the specification, it can replay Change Events in a single update operation. It applies deletes and inserts in the order the query pattern from the `WHERE` clause produces them, allowing streaming execution. The order can be overridden by the predefined `?HALYARD_TIMESTAMP_SPECIAL_VARIABLE`, which we bind to Change Events' timestamps when replaying Backlog. Explicit timestamps allow to decouple the order the update operations are applied from the order they are read. Halyard Bulk Update accepts timestamps bound to any data type that can be cast to `xsd:long`, so that both numeric data types and dates that can be represented as Unix time work.

Halyard Bulk Update allows update operations to specify how many concurrent forks they can be split into via the custom SPARQL function `halyard:forkAndFilterBy()`. The function's arguments specify the number of forks and one or more variables. Data to update is partitioned by the chosen variables' bindings among the forks. The example 1.1 shows an update operation to replay Backlog described by the `:metadata` Metadata Graph by using 30 forks. Halyard Bulk Update generates the changes for HBase in the Map phase, sorts and groups the changes in the Reduce phase, and applies them in bulk during asynchronous database compaction. The parallel replay of Backlog is deterministic when timestamps of Change Events are distinct. When Change Events of multiple Targets share the same timestamp, the Targets must be disjoint for the replay to be deterministic. Each update operation is run as a transaction with read committed isolation to avoid conflicts.

6 Change data capture for PUBMED

Our primary use case for CDC is PUBMED. PUBMED[3] is a dataset maintained by the National Library of Medicine that aggregates rich metadata on vast

[2] https://merck.github.io/Halyard/tools#Halyard_Bulk_Update
[3] https://www.ncbi.nlm.nih.gov/pubmed

Listing 1.1 SPARQL Update operation to replay Backlog

```
PREFIX :        <http://example.com/>
PREFIX halyard: <http://merck.github.io/Halyard/ns#>
PREFIX backlog: <https://data.gin.merck.com/vocabulary/backlog#>

DELETE { ?deleteS ?deleteP ?deleteO . }
INSERT { ?insertS ?insertP ?insertO . }
WHERE {
  GRAPH :metadata {
    ?change backlog:timestamp ?HALYARD_TIMESTAMP_SPECIAL_VARIABLE .
    FILTER halyard:forkAndFilterBy(30, ?change)
  }
  {
    GRAPH :metadata { ?change backlog:insertGraph ?insertGraph . }
    GRAPH ?insertGraph { ?insertS ?insertP ?insertO . }
  } UNION {
    GRAPH :metadata { ?change backlog:deleteGraph ?deleteGraph . }
    GRAPH ?deleteGraph { ?deleteS ?deleteP ?deleteO . }
  }
}
```

amounts of biomedical literature. PUBMED data is distributed in XML via an FTP server. Each year a baseline snapshot of the data is produced and followed by incremental updates published daily.

PUBMED represents updates explicitly as versions of articles, so change detection is not needed. We treat the articles as Targets of Change Events and their versions as Snapshots. Snapshots for a Target, especially the consecutive ones, may contain overlapping RDF triples, which can be removed to save storage space. However, in our case computing the intersections of Snapshots is more expensive than the extra storage. Articles to delete, specified by the XML element `DeleteCitation`, are represented as Change Events with no inserts and the maximum timestamp for their date, causing their preceding Snapshot to be deleted.

Each day we pull PUBMED updates, transform them to RDF, and replay them. Since PUBMED is available in XML, we use XSLT to transform it to RDF, as it is a native XML processing language. As our temporal data model requires named graphs, we opted for the TriX RDF syntax [7], which is a verbose low-level data format, but unlike RDF/XML it allows named graphs. As of January 2019, the current reconciled RDF version of PUBMED we produce contains 2.2 billion triples, while its backlog amounts to 2.8 billion quads. Using an Amazon Web Services EMR cluster with 8 i3.2xlarge (8 CPU cores, 60 GB RAM) instances plus 2 i3.2xlarge instances as task nodes, the complete replay of this dataset takes 9 hours and costs $30. The incremental daily replay runs in minutes. We

use the RDF version of PUBMED for various tasks, such as custom business intelligence queries or training text classifiers on abstracts.

7 Conclusion

We designed and implemented a standards-based method for CDC of large-scale RDF datasets. The method is implemented by a SPARQL Update operation run by Halyard Bulk Update, a MapReduce application for the Halyard RDF store. Its implementation can replay large datasets, such as PUBMED with billions of RDF triples, in a horizontally-scalable cluster with commodity hardware. A future work to consider is how to make post-processing data, such as deduplication of PUBMED authors, work with the proposed temporal data model.

References

1. Seaborne, A., Davis, I.: Supporting change propagation in RDF. In: Proceedings of the W3C workshop - RDF next steps., Palo Alto, CA, USA (2010).

2. Das, S., Botev, C., Surlaker, K., Ghosh, B., Varadarajan, B., Nagaraj, S., Zhang, D., Gao, L., Westerman, J., Ganti, P., Shkolnik, B., Topiwala, S., Pachev, A., Somasundaram, N., Subramaniam, S.: All aboard the databus!: Linkedin's scalable consistent change data capture platform. In: Proceedings of the third ACM symposium on cloud computing. pp. 18:1–18:14. ACM, New York, NY, USA (2012). https://doi.org/10.1145/2391229.2391247.

3. Belleau, F., Nolin, M.-A., Tourigny, N., Rigault, P., Morissette, J.: Bio2iRDF: Towards a mashup to build bioinformatics knowledge systems. Journal of Biomedical Informatics. 41, 706–716 (2008). https://doi.org/https://doi.org/10.1016/j.jbi.2008.03.004.

4. Carroll, J.J., Bizer, C., Hayes, P.J., Stickler, P.: Named graphs, provenance and trust. In: Proceedings of the 14th international conference on world wide web. pp. 613–622., Chiba, Japan (2005).

5. Sotona, A., Negru, S.: How to feed Apache HBase with petabytes of RDF data: An extremely scalable RDF store based on Eclipse RDF4J. In: Kawamura, T. and Paulheim, H. (eds.) Proceedings of the ISWC 2016 posters & demonstrations track., Kobe, Japan (2016).

6. Gearon, P., Passant, A., Polleres, A. eds: SPARQL 1.1 update. (2013).

7. Carroll, J.J., Stickler, P.: TriX: RDF triples in XML. Hewlett-Packard (2004).