Provenance-Aware NoSQL Databases

Anu Mary Chacko(✉), Munavar Fairooz, and S.D. Madhu Kumar

National Institute of Technology Calicut, Kozhikode, India
anu.chacko@nitc.ac.in

Abstract. NoSQL stores are very widely used for BigData Analytics. These stores are built with inherent scalability and fault tolerance. But there are not much mechanism to provide security guarantees like integrity and auditability. Provenance is a metadata which captures the details of how the data reached its current state. By way of capturing provenance it is possible to enhance the functionality of NoSQL stores to verify the integrity of results. This paper presents an approach to capture provenance of NoSQL databases using logs generated by the database. A proof of concept was implemented in MongoDB and examples are used to illustrate the use of 'Why provenance' and 'How-provenance' captured.

Keywords: Data provenance · NoSQL databases · MongoDB · MapReduce · How-provenance · Why-provenance

1 Introduction

With the growth of information technology, the volume of data has grown enormously in the last decade. This large volume of data available is usually unstructured in nature which when mined can give a lot of useful information. For meeting this requirement new paradigms of storage and analytics have evolved, and NoSQL databases is an example of the same. NoSQL stands for Not Only SQL and is an evolution of traditional Relational Databases to handle the large volume of unstructured data. NoSQL databases usually have a simple data model, support basic operations, have weak security but have high availability and scalability. The key attraction of these databases is the dynamic schema that allows the different types of unstructured data to reside in the same collection/table. Most of the NoSQL databases support only CRUD (Create, Read, Update, and Delete) operations. So when the analytic query demands more complex join or aggregation operations, analytic frameworks like MapReduce are used.

When a huge amount of data is processed, and decisions are derived based on it, users need some mechanism to ensure the credibility of decision. Data provenance can help here. Data provenance is the metadata that captures the creation and subsequent modification of the data as it is processed in and across systems.

In the case of NoSQL data analytics, data in the range of terabytes and petabytes are being processed. The credibility of the result produced by analyzing the big data is dependent on the goodness of data. Provenance is the metadata which captures the information about how a data reached its current state. Hence, provenance can be of great help in verifying and ensuring the "goodness" of data.

P. Mueller et al. (Eds.): SSCC 2016, CCIS 625, pp. 152–160, 2016.
DOI: 10.1007/978-981-10-2738-3_13

The relationship between provenance and security can be considered symbiotic [1]. To ensure security attributes like auditability or to make good security decisions users need well-captured provenance. To ensure good security practices provenance needs to be audited.

Security is a major concern for the adopters of Big Data and Cloud processing. Ensuring efficient capture and storage of provenance for data processed will provide the users a mechanism to verify and debug results obtained. Also, provenance can throw light on potential security breaches. In this paper, we evaluate an option to capture provenance of NoSQL databases and explore the uses of the collected provenance. There is no provision for capture of provenance in any of the existing NoSQL implementations. Rest of the paper is organized as follows: Sect. 2 discusses the related work in the area, Sect. 3 explores the detail of the proposed design, Sect. 4 gives the details about the implementation of proof of concept and Sect. 5 concludes the paper by exploring future work possible.

2 Related Works

Initial works in provenance was done in 1970s in the area of eScience where provenance was collected to ensure reproducibility and verification of scientific experiments. Examples of such systems are Chimera [2], MyGrid [3], PASOA [4], etc. The approach adopted for provenance capture was to redesign existing application/workflow to capture provenance.

PASS [5] (Provenance-Aware Storage System) attempts to capture provenance at the storage level. PASS is built by modifying Linux kernel to automatically deduce provenance information by observing operating system calls at read/write level. There have been significant contributions in the domain of relational databases as well.

Buneman et al. [6] categorizes database provenance as 'Why-provenance' and 'Where-provenance'. 'Why-provenance' lists all source data items that contributed to the creation of result data item. 'Where-provenance' lists the originating sources of the result. This categorization has been extended to include 'How-provenance' that explains how the individual derivations have been carried out according to the query [7].

A practical example of making a relational database provenance-aware is seen in the project done by the University of Illinois called PERM (Provenance Extension of Relational Model) [8] built by extending PostgreSQL engine. Here provenance is captured by query rewriting and is displayed along with the query results as additional columns.

Kulkarni [9] suggests a generalized provenance model for key-value systems. The proposed system has the capability to capture tuple provenance and schema provenance. The author proposes application to explicitly select the data/collection for which provenance need to be captured. For updates, the value before modification is also captured. The scheme provides provision for a logical marker which will help in tracking set of columns as a single logical unit. Provenance queries are provided for finding the provenance information required. A proof of concept was implemented on by modifying Cassandra to make it provenance-aware.

A NoSQL metadata management tool called Wasef was proposed by Alkhaldi et al. [10]. Wasef captures provenance as one of the metadata. Provenance capture and use are quite primitive in the initial model. A proof of concept was implemented in Cassandra and evaluated.

Both KVPM and Wasef is implemented by changing the code of Cassandra to make it provenance-aware. It will be a better approach to have a generic provenance collection methodology which automatically capture provenance by observing transactions in the database. In this paper we propose a novel approach of automatic provenance collection in NoSQL database by monitoring the logs and demonstrate a proof of concept of our idea in MongoDB. The practical use of provenance collected is illustrated through an example.

3 Design

When we do analytics on data stored in NoSQL store, the primary use of provenance will be to explain unexpected results. For this purpose 'why-provenance' and 'how-provenance' will be useful. 'Why-provenance' will list all the source data tuples that contributed to the creation of result data tuple. 'How-provenance' will list the operations that caused the data tuple to reach the current value. So combining 'how-provenance' and 'why-provenance' we will have a holistic answer to how the result tuple was formed. NoSQL databases are designed to be scalable and can partition across many servers. The system scales transparently, and built-in fault tolerant mechanisms via replication, make it difficult to capture the 'Where-provenance'.

There are two approaches for capturing provenance. One approach is to redesign individual application to make it provenance-aware and the second approach is to automatically deduce provenance by observing the transformations to data. In this paper, we propose a novel approach to capture provenance by using existing logs in NoSQL database. NoSQL databases are designed with logs to enable replication of changes to ensure transparent scalability. The information in the logs can be reused to capture provenance of data transactions. As a proof of concept, MongoDB is made provenance-aware using this approach.

4 Proof of Concept - Provenance Aware MongoDB

MongoDB is an example of document-oriented datastore that captures data in the form of key-value pair. To capture provenance we use the concepts in key value provenance model proposed by Kulkarni [9].

4.1 Requirements

MongoDB supports only basic CRUD operation. Complex analytic queries are supported by built-in MapReduce Framework. So in this work the 'How Provenance' of data stored in MongoDB and 'Why Provenance' of data queried using MapReduce

is captured. Combining the 'How-provenance' with the 'Why-provenance' the holistic picture of the contributing tuples of a result can be obtained.

The requirements of the Provenance Aware MongoDB are listed below.

1. Users should be able to track tuple level and schema level provenance.
2. Users should be able to know the contributing sources for a result tuple (why-provenance) and know the operations that caused the tuple to have current value (how-provenance).
3. The solution should be generic as far as possible and should have minimal instrumentation of existing application/data store.
4. As the volume of provenance and data is huge, the user should be provided the option to select the tuple or table for which provenance needs to be captured.
5. Overheads should be minimal as far as possible.

4.2 Capturing 'How-Provenance'

To capture 'How-Provenance' system needs to capture the operations through which a tuple reached its current state. MongoDB stores humungous amount of data and provenance may not be relevant to all data. Hence, the user/programmer of the database is given flexibility to identify the document for which provenance needs to be collected. Resource for which provenance needs to be captured can be listed as 'resource expression'.

Resource expression can be written in the following formats.

If the provenance is to be tracked.

- For a particular document inside a collection -<Database/Collection/Id>.
- For a collection - <Database/Collection>(provenance is tracked for all documents for the collection).

MongoDB maintains a capped collection called Oplog for storing all operations that happened in the database so as to replicate in secondary servers. Capped Collections are fixed size collections in MongoDB where documents are retrieved in the order of insertion, and when the size gets exhausted, space for new documents are made by overwriting the oldest documents in the collection.

Oplog entries in MongoDB include the timestamp, unique id, operation name, namespace with the details of the database, collection and document affected by the operation and the new state of the document after performing the operation. Primary oplog captures all the operations that are applied on the primary node. The secondary nodes copy and apply these operations in an asynchronous process to achieve eventual consistency. Thus, oplog entries give primary information to track 'how-provenance'. In addition to the above information, details about the user executing the action need to be captured.

The 'how-provenance' is captured by setting up a tailable cursor to the oplog. This script runs parallel to the MongoDB process to detect any entry being made in the oplog. When a new entry comes to oplog, the resource expression file is checked to identify whether the tuple is listed for provenance capture. If so, information from Oplog like timestamp converted to ISO date time, operation type are retrieved and stored in a

separate append only provenance collection. This provenance information is augmented with user information to make provenance complete.

The following example illustrates the 'how-provenance' captured for a document.

Suppose in the MongoDB database called 'hospital' there exists a collection called 'patients'. Assume that we want to track the provenance for a particular patient, say 'P123'. In the beginning we specify resource expression as <hospital/patients/P123>

The current state of the patient record is given in Fig. 1

```
{
"_id" : "P123",
"Name" : "John",
"Doctor" : "Dr.jacob",
"Disease" : "Asthma",
"Medication" :[ "Doxil4","Laxin" ],
"Allergy" : "Sneezing "
}
```

Fig. 1. Document in MongoDB for P123

'How-provenance' for the document is given in Fig. 2.

```
{
  "_id" : "hospital.patient.P123",
  "Provenance" : [
  {       "Op_Type" : "i"
          "Operation" : "{   'Name': 'John', 'Disease' : 'Asthma',
                             'Medication': ['Doxil4' , 'Aadrone' ],
                             'Doctor' : ' Dr. James '
                         }",
          "Time" : ISODate("2015-04-29T12:56:49Z"),
          "user"  : "Dr. James",
  },
  {       "Op_Type" : "u"
          "Operation" : "{'$set': {'Medication':['Laxin]},
                                  {'Doctor': 'Dr. Jacob'}}",
          "Time" : ISODate("2015-04-29T1:57:08Z"),
          "user" : "Dr. Jacob",
  },
  {       "Op_Type" : "u"
          "Operation" : " {'$set': {'Allergy': 'Sneezing'},
                                  {'Medication:[Doxil4, Laxin]'}}",
          "Time" : ISODate("2015-04-29T32:57:16Z"),
          "user" : "Dr. Jacob",
   }
          ]
}
```

Fig. 2. How-provenance captured

The above example shows how both data and schema provenance is available for querying.

4.3 Capturing 'Why-Provenance'

The work was extended to capture MapReduce provenance in MongoDB. The provenance collected characterizes as 'Why-provenance' as it gives reason/witness for why an output was obtained.

Ikeda et al. [10] proposed a model to capture provenance in MapReduce workflows in Hadoop called RAMP (Reduce and Map Provenance). They use a wrapper-based approach to capture provenance by building wrappers for the various components of MapReduce framework like Record Reader, Mapper, Reader, and Writer. They apply an eager approach and record provenance as the workflow proceeds. The eager approach of computing provenance along with MapReduce computation introduces a lot of computational overhead and delay in job execution time. Hence a lazy approach of computing provenance was used in this work. The mapper and document writer was modified to write intermediate data into two temporary files. Mapper emits key value pairs (ki, vi). For each of the mapper output, a pair (ki, pi) is recorded in the file1 where pi is the 12-byte document id of the input document being processed. Reducer processes all the values with the same key (ki, [v1,v2,…vn]) and produces an output of the form (ki, vi). The document writer writes the key-value pair in the result collection as well as in file2. After the execution of the job, provenance file is generated by a python script, which takes the two files as input and maps input provenance with output value using the key. So the output provenance value will be (V, {p1, p2 … pn}) and will list all input documents which contributed towards output record. The algorithm to capture MapReduce Provenance is diagrammatically explained in Fig. 3.

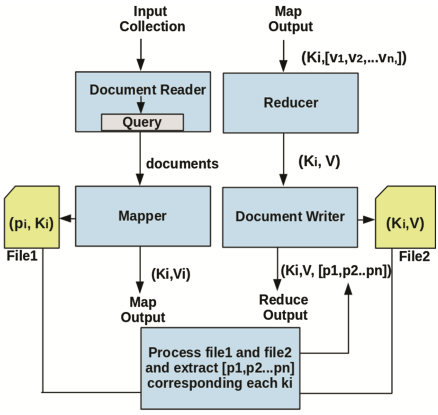

Fig. 3. Modified MapReduce to capture 'Why-Provenance'

The example given below illustrates the practical application of the 'Why-provenance.'

In a hospital application, the collection of patient's medication bills at different times are captured in hospital database is illustrated in Fig. 4.

The total bill for a patient can be consolidated by running a MapReduce job. The output for the same is shown in Fig. 5.

The provenance captured via MapReduce (Why Provenance) and MongoDB(How Provenance) will provide a good explanation for the result as shown in Fig. 6.

Patient id	Bill Date	Prescribed Doctor	Items	Price(₹)
P127	2012-12-13 22:00:00	Dr.Jacob	{"Medicine":"Aidol7","qty":10,"price":2.5} {"Test":"MRI","qty":1,"price":1250}	1275
P133	2012-09-04 00:00:00	Dr.Ajeeb	{"Medicine":"Laxin","qty":5,"price":10} {"Medicine":"Mentol","qty":5,"price":2.5} {"Test":"Blood Test","qty":1,"price":50}	111.5
P123	2012-10-03 14:00:00	Dr.Ajeeb	{"Medicine":"Ameco7","qty":5,"price":20} {"Medicine":"Mentol","qty":5,"price":2.5} {"Test":"ECG","qty":1,"price":125}	234.5
P127	2012-12-13 22:00:00	Dr.Jacob	{"Medicine":"Aidol7","qty":10,"price":2.5}	25
P127	2012-12-13 22:00:00	Dr.Ajeeb	{"Medicine":"Demol Tab","qty":20,"price":2.5} {"Test":"ECG","qty":1,"price":250}	300
P123	2012-12-13 22:00:00	Dr.Jacob	{"Medicine":"Abeol","qty":10,"price":25}	250
P123	2012-10-04 00:00:00	Dr.Jacob	{"Medicine":"Laxin","qty":5,"price":2.5} {"Test":"ECG","qty":1,"price":125}	137.5
P133	2012-12-04 04:00:00	Dr.Ashly	{"Medicine":"Laxin","qty":5,"price":2.5} {"Medicine":"Aloxenol","qty":25,"price":25}	625
P333	2013-01-04 04:00:00	Dr.Hema	{"Medicine":"Laxin","qty":5,"price":2.5} {"Medicine":"Aloxenol","qty":25,"price":25} {"Test":"ECG","price":125}	150

Fig. 4. Hospital database

Key	Value
P127	1600
P333	150
P123	622
P133	736.5

Fig. 5. Output of the query for total bill

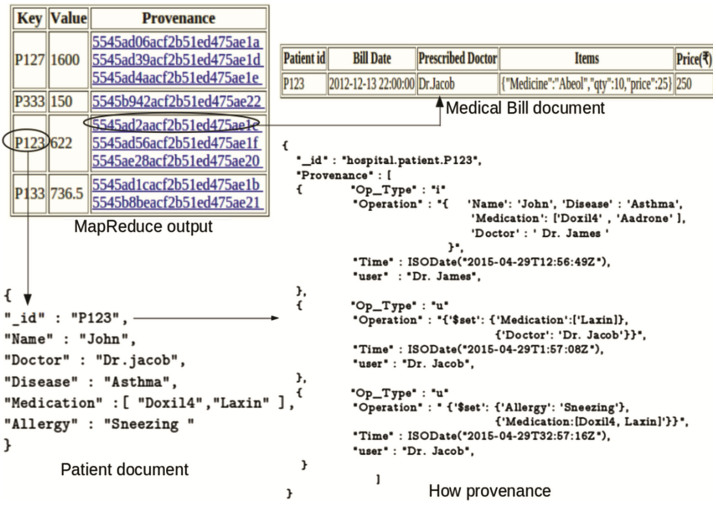

Fig. 6. Why and how-provenance combined

4.4 Performance Analysis

The experiment was conducted on a single node MongDB 2.6.1 on an Intel i3CPU Linux Laptop with 4 GB RAM.

How-Provenance' Overhead. As the strategy to capture 'how-provenance' is by running a tailable cursor parallel to MongoDB server process, there is no time overhead associated with the same. However, the storage overhead is proportional to the number of operations done on the document/collection.

Why-Provenance'/MapReduce Provenance Overhead. We ran MapReduce on MongoDB on various randomly generated datasets of different sizes. The time given is cumulative time for processing MapReduce workflow and running the python script to generate provenance. Time overhead is about 70–73 %. Space overhead depends on the documents that are processed, as for every input document 12 bytes are required to track the document id. In the 260 MB dataset used, space overhead is very large as we used more than 700000 documents in the dataset. Experiment results are graphically represented in Fig. 7.

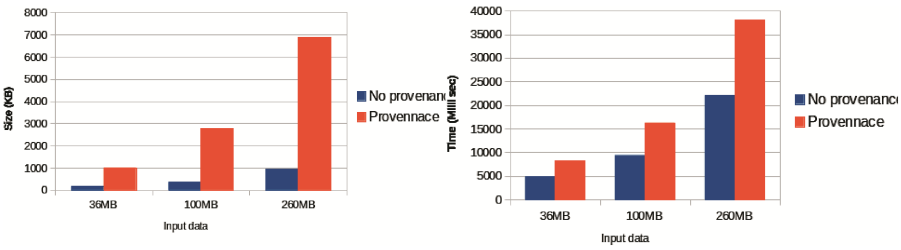

Fig. 7. Space overhead and time overhead for why-provenance against data size

5 Conclusion and Future Work

Provenance is a metadata that helps us audit the security of our systems and ensure that our system is trustworthy. In this era of data explosion, big data analytics is extensively used for decision making. In this context, capturing provenance is critical as it provides a mechanism to ensure trustworthiness of result. In this paper, the option of capturing of the provenance of NoSQL database by using system logs is explored. The proof of concept was demonstrated by building a basic prototype in MongoDB and capturing 'how-provenance' and 'why-provenance' of queries.

Using our approach, any application built on top of MongoDB can capture provenance of all database operations without adding any code for the same. The user is given the flexibility of selecting documents for which provenance need to be tackled via resource expression.

The work is based on the assumption of fault tolerance that is built into NoSQL databases via replication mechanism. In this basic prototype, we have explored the type of provenance and their uses. As future extension of this work, storage and performance optimizations to make the provenance model more usable can be explored. Currently an append-only table is used to capture provenance so that provenance remains immutable. Securing the provenance with integrity and confidentiality guarantees will be an interesting line of work.

References

1. McDaniel, P.: Data provenance and security. J. IEEE Secur. Priv. **9**(2), 83–85 (2011)
2. Foster, I.,Vöckler, J., Wilde M., Zhao, Y.: Chimera: a virtual data system for representing, querying, and automating data derivation. In: Proceedings of the 14th Conference on Scientific and Statistical Database Management (2002)
3. Ikeda, R., Salihoglu, S., Widom, J.: Provenance- based refresh in data-oriented workflows. In: Proceedings of the 20th ACM International Conference on Information and Knowledge Management (2011)
4. Moreau, L., Groth, P., Miles, S., Vazquez, J., Ibbotson, J., Jiang, S., Munroe, S., Rana, O., Schreiber, A., Tan, V., Varga, L.: The provenance of electronic data. Commun. ACM **51**(4), 52–58 (2008)
5. Muniswamy-Reddy, K., Holland, D., Braun, U., Seltzer, M.: Provenance-aware storage systems. In: Proceedings of the 2006 USENIX Annual Technical Conference, Boston, June 2006
6. Glavic, B., Dittrich, K.R.: Data provenance: a categorization of existing approaches. In: Proceedings of the 12th GI Conference on Datenbanksysteme in Buisness, Technologie and Web (BTW) (2007)
7. Cheney, J., Chiticariu, L., Tan, W.-C.: Provenance in databases: why, where and how. Found. Trends Databases **1**(4), 379–474 (2009)
8. Galvic, B.: Perm: efficient provenance support for relational databases. Ph.D. thesis, University of Zurich (2010)
9. Kulkarni, D.: A provenance model for key-value systems. In: TaPP 2013 Proceedings of the 5th USENIX Workshop on the Theory and Practice of Provenance (2013)
10. Park, H., Ikeda, R., Widom, J.: RAMP: a system for capturing and tracing provenance in MapReduce workflows. In: International Conference on Very Large Data Bases, pp. 1351–1354 (2011)