Big social data provenance framework for Zero-Information Loss Key-Value Pair (KVP) Database

Asma Rani^{1,2} · Navneet Goyal² · Shashi K. Gadia³

Received: 22 April 2021 / Accepted: 4 September 2021 / Published online: 9 November 2021
© The Author(s), under exclusive licence to Springer Nature Switzerland AG 2021

Abstract

Social media has been playing a vital importance in information sharing at massive scale due to its easy access, low cost, and faster dissemination of information. Its competence to disseminate the information across a wide audience has raised a critical challenge to determine the social data provenance of digital content. *Social Data Provenance* describes the origin, derivation process, and transformations of social content throughout its lifecycle. In this paper, we present a *Big Social Data Provenance (BSDP)* Framework for key-value pair (KVP) database using the novel concept of *Zero-Information Loss Database (ZILD)*. In our proposed framework, a huge volume of social data is first fetched from the social media (Twitter's Network) through live streaming and simultaneously modelled in a KVP database by using a query-driven approach. The proposed framework is capable in capturing, storing, and querying provenance information for different query sets including select, aggregate, standing/historical, and data update (i.e., insert, delete, update) queries on *Big Social Data*. We evaluate the performance of proposed framework in terms of provenance capturing overhead for different query sets including select, aggregate, and data update queries, and average execution time for various provenance queries.

Keywords Big social data · Social media · Data provenance · Zero-information loss database · Provenance generation · Querying provenance

1 Introduction

In computing world, data are defined as the factual information in digital form used as a basis for various qualitative and quantitative analysis. The twenty-first century will be known as the century of data as it has witnessed an unprecedented growth of data in almost all domains [5,6]. With the rapid evolvement of social media and web-based communication, everyone has become more enthusiastic about sharing their thoughts, ideas, opinions and other content through a social media platform, causing an exponential growth in the size of social data [29]. Social media platforms are the major source

of unstructured data in the current times. Unstructured data is characterized by ad hoc schema, and therefore cannot be stored in SQL databases. The growth of unstructured data has led to interest in NoSQL databases, as they are much better suited due to their flexible schema. Therefore, in this paper, our *first motivation* is to build a flexible KVP data model in Apache Cassandra by using a novel query-driven approach to correlate big social data through relationships and dependencies. From the past few years, social media has become a common platform for global conversation around the world due to its giant size, vast availability, intense speed, and wide range of variant content. On the other hand, several illegitimate activities are engendered by misusing these social content through social engineering [18,19,56] to accomplish various objectives. One of the main causes behind the illegitimate activities on social media is the separation of digital content from its provenance [12]. In this paper, our *second motivation* is to explore the need of provenance information associated with the digital content published on social media and to design an efficient social data provenance framework for key-value pair (KVP) database. *Social Data Provenance* [30] involves following three dimensions, viz.

✉ Asma Rani
asma.sags@gmail.com
Navneet Goyal
goel@pilani.bits-pilani.ac.in
Shashi K. Gadia
gadia@iastate.edu

¹ Dr. B. R. Ambedkar Institute of Technology, Port Blair, India

² Birla Institute of Technology and Science, Pilani, India

³ IOWA State University, Ames, USA

“What”, “Who”, and “When”. *What* provides the description about the social media posts, *Who* describes the correlations among social media users, and *When* characterizes the evolution of users’ behaviour over time. Like data provenance [24,49], social data provenance also describes the ownership and origin of such information.

1.1 Challenges

The term “Big Data” is characterized by 7 V’s, viz. Volume, Velocity, Veracity, Variety, Variability, Visualization, and Value. Veracity of big data that is defined as quality, accuracy and truthfulness of source of data, is directly linked with data provenance. Currently, Big data and social media have become the synonymous to each other, as the major portion (over 90%) of total data in the world are produced through several social media platforms such as Twitter, Facebook, Instagram, etc. This rapidly growing large sized human-generated data is known as the *Big Social Data (BSD)* [37,47]. One of the major challenges that is usually faced by the several big social data applications is to design a flexible data model in NoSQL databases, as the traditional data modelling approaches are not suitable for correct and efficient data model design in such databases. Provenance about derivation history of big social data is usually called *Big Social Data Provenance (BSDP)* [21]. In Social Data Analytics, the credibility of an analysis generally depends upon the quality and truthiness of input data which is assured by the *Social Data Provenance* [54]. In this way, social data provenance plays a major role in clarifying opinions to avoid rumours, investigations, and explaining how and when this information is created and by whom. However, distillation of provenance information from such a huge amount of complex data is an extremely tedious task, due to its diverse formats. Barbier [3] identified some of the following issues to address the key challenges in capturing, storing and querying provenance for social data:

- Currently, no social media platform provides any provenance information to the users to identify the originators or sources of the published information.
- A wide variety of digital content including text, images, and multimedia files are dynamically generated through various social media sites. However, there is no common format of such data that is available to understand the provenance information associated with them.
- No common application programming interface (API) and architectural solutions are provided by the developers to access and manage provenance data.
- There is no widely accepted mechanism which has the potential to trace out the provenance objects from such unstructured distributed data.

In addition, several other challenges such as designing automatic provenance capturing mechanism, minimizing provenance capturing and querying overhead, different granularity levels at which provenance needs to be captured, and provenance data analysis through provenance visualizations, etc., are also explored for provenance support in big data application by different authors in [7,14,15,23]. Because of these remarkable challenges, the necessity of capturing and querying provenance information associated with social data has raised a growing interest in the era of social data analytics.

1.2 Needs of social data provenance

In computer science, provenance has been studied mainly in the following two perspectives: first is *database provenance* or *data provenance* and second is *scientific workflow provenance* or *workflow provenance* [50]. *Workflow Provenance* is a coarse-grained information that captures the information about process and entities involved in that process as a black box, while *Data Provenance* captures fine-grained information. It focuses on how any result is derived, what queries are executed, what operations are performed on data. In this paper, our main focus is on “Data Provenance”. *Social Data Provenance* describes the origin, derivation and transformations of social content throughout its lifecycle. It is also categorized in the following two categories based on its granularity level, viz. fine-grained and coarse-grained provenance [50]. Recently, social data provenance has gained a lot of attentions, as it serves different purposes such as audit trail, data discovery, update propagation, incremental maintenance, rumour identification, justification of a query result, etc. Several web-based tools [30,42] are developed to capture pre-defined provenance attributes such as name, gender, religion, location, etc., from different social networking accounts associated with a particular twitter user. Although these attributes capture complete details of a social media user, but it neither provides a provenance path nor a propagation history and updates of any social content published on a social media platform. To reconstruct and integrate provenance of messages in social media, a workflow provenance model PROV-SAID [16,51,52] based on W3C PROV data model is also proposed for a small dataset. However, most of the existing approaches are not scalable to track provenance metadata for social media efficiently. They are suitable to capture workflow provenance at coarse-grained level only. Further, as the social data are constantly changing over time, yet no any existing framework is capable to capture provenance for historical queries, which is an essential requirement of social data provenance. Therefore, the viability of such a framework becomes the necessity to engender the trust among social media users. To accomplish this, we propose a *Big Social Data Provenance (BSDP)* framework

for key-value pair database that is capable in capturing, storing and querying provenance information for different query sets including select, aggregate, standing/historical, and data update (i.e., insert, delete, update) queries on a live streamed twitter data set.

Relational databases have been the mainstay of the data community for decades starting from mid 1980's. They are ideal for structured data and predictable workload. But these databases are not scalable for handling big data which encompasses not just structured data, but also semi-structured and unstructured data. Not only SQL (NoSQL) databases have been proposed as an alternative to SQL databases to handle the challenges posed by big data, as these databases efficiently support to a low latency, horizontal scalability, efficient storage, high availability, high concurrency, and reduced operational costs [8,17,35]. Apache Cassandra is one of the most popular key-value pair (KVP) database which belongs to the NoSQL family. The key strengths of Apache Cassandra [20] are its simplicity, scalability, and a very fast streamlined NoSQL architecture in which each column is a data structure that contains a key, value, and a timestamp. It is also used in application development by Facebook, Twitter, Cloudkick, Mahalo, etc. Social media is the major source of unstructured data in the current scenario. Therefore, Apache Cassandra is a good choice to handle this extremely high volumes of unstructured data.

1.3 Research contributions

For applications those are related to auditing, security, and accountability, there is a need to restore all the operations performed on a database to produce the same result as of their previous executions. This leads to the requirement of managing all the updates (i.e., insert, delete, and update operations) without any loss of information as a provenance data. But the conventional/snapshot database systems do not maintain the history of all data objects and store only the current snapshot of data, as a result they are ill-suited for such applications. *Zero-Information Loss Database (ZILD)* [4] is a special type of database which is based on temporal database and maintains temporal data as a history of all the updates along with the complete information of operational activities performed in that database. Therefore, it is well suited for designing the provenance framework especially in capturing provenance for update, insert, delete, and historical/standing queries. In this paper, we design and develop a *Big Social Data Provenance (BSDP)* framework for key-value pair database [45]. The proposed framework is capable to answer the following questions, viz. what type of provenance data should be reconstructed from social media?, To which extent it will be useful?, How to capture this provenance data?, How and where to store provenance data?, How

to query/analyse provenance data?, etc. In summary, the main contributions of this paper are:

1. BSDP, a novel provenance solution for live streamed big social data that integrates both online and offline module and captures *fine-grained* and *coarse-grained* provenance.
2. Fine-grained provenance is captured in the form of *Provenance Path Expressions* that consists of keyspace, column family, row key and column name contributed towards each result tuple, while coarse-grained provenance is captured in form of query statements with their execution time.
3. We introduced a novel query-driven data model design methodology for Apache Cassandra.
4. Social data are constantly changing over time; however, no any existing approach is capable to capture provenance for historical queries, which is an essential requirement of social data provenance. On the contrary, our framework aims to maintain all data update (i.e., insert, delete, and update) operations without any loss of information.
5. It supports to perform historical data queries (i.e., querying a data element with a given time in the past and with a time range specified in the query statement) using User-Defined Constructs (UDCs) in CQL (Cassandra Query Language) and capturing provenance for standing/historical queries (i.e., it traces the provenance for all the result tuples of a query executed in the past).
6. Our proposed framework is developed around the novel concept of Zero Information Loss Database (ZILD) [4]. By a Zero Information Loss Database, we mean that no data value, no user, and no query and its result is ever lost. ZILDs are very useful in tracking any “data manipulations” that have taken place on social media.
7. Existing solutions for social data provenance are dedicated to a particular social media platform with limited query support and suitable for a small data set. On contrary, our framework provides a generalized provenance solution which is capable to extract real-life social data from different social media platforms through live streaming by using their supporting APIs, for instance Graph API for Facebook social graph, Twitter's Streaming API for Twitter's Network.
8. We propose different provenance generation algorithms for select, aggregate, standing, and data update queries with insert, delete and update operations. All the captured provenance is further stored in Zero-Information Loss Key-Value Pair Database (ZILKVD).

In addition, all the extracted social data and their provenance information are stored in a common keyspace of Apache Cassandra using a query-driven approach for fast read/write operations and efficient provenance visualization.

A case study of Twitter Social Network is given to show the feasibility and usefulness of our proposed framework in capturing, storing and querying social data provenance.

2 Related work

Social data analytics is an emerging research field that integrates social communications with data analytics. It extracts meaningful insight from extensively large data sets. It can be used to understand the user's behaviour, and to model social interactions among social media users. Big Social Data [47] is mainly characterized by 3 V's, viz. volume, velocity, and variety, where *volume* means rapidly growing social data, *velocity* is related to the dissemination of information with tremendous speed, and *variety* refers to diverse formats of social data. Nowadays, the volume, velocity, and variety of Big Social Data are facing the challenges of capturing provenance [12] and evaluating trustworthiness of social data [29]. Therefore, an efficient provenance data management system is required to trace out the provenance information through provenance capturing and querying for social data generated from various social media platforms. The importance of social data provenance in social media is also presented in [21,36,46] with several key challenges such as measuring quality and truthiness of social data, provenance storage, provenance querying, etc. [7,9,14,15,23,49,53]. Several research works are carried out to identify the suitability of NoSQL database to manage big social data with efficient storage, fast querying, and horizontal scalability [20,33]. Different approaches are proposed to model a huge volume of Twitter data set in Apache Cassandra NoSQL database for an efficient querying [11,26,40].

A provenance data model for data intensive workflows is proposed in [13] to capture provenance information for Map Reduce workflows using Kepler-Hadoop framework. The proposed provenance model is a good initiation for scientific workflows; however, it is not much efficient in terms of storage space and query execution overhead. In line with the provenance data model for scientific workflows, RAMP model is proposed in [28,39] for Generalized Map and Reduce Workflows (GMRWs) using a wrapper-based approach for provenance capturing and tracing. In this model all the transformations are either map or reduce functions rather than having one map function, followed by one reduce function. Further, HadoopProv model [2] is introduced for provenance tracking in Map Reduce workflows, where provenance tracking takes place in Map and Reduce phases separately, and construction of provenance graph is deferred at query stage, to minimize the temporal overhead. Several web-based tools [30,42] are developed to capture pre-defined provenance attributes such as name, gender, religion, location, etc., from different social networking

accounts associated with a particular twitter user. Although these attributes capture complete details of a social media user, it neither provides a provenance path nor a propagation history and updates of any social content published on a social media platform. Further, a provenance path algorithm [25] is proposed to capture provenance path of an information, to explain how this information propagates in a social network but to a few known recipients only. To reconstruct and integrate provenance of messages in social media, a workflow provenance model PROV-SAID [16,51,52] based on W3C PROV data model is proposed. Although the proposed solution identifies the posted tweets that are copied from other published tweets without giving credit to original tweeter like a retweet, it is suitable for a small dataset only. Applications of standard PROV-DM model are proposed in [27] to manage provenance data for bioinformatics workflows in a cloud computing environment using different families of NoSQL databases. A provenance framework based on algebraic structure of semirings for three specific graph algorithms is presented in [41], to compute provenance of regular path queries (RPQ) over graph database via applying annotations like labels and weight functions which is a quite complex process. A provenance model for vertex-centric graph computation and a declarative data-log based query language is presented in [38], to capture and query graph analytics provenance for both online and offline mode. Further, a provenance model for stream processing system (s2p) is proposed in [55]. Although this model is suitable to capture fine-grained (operator level) and coarse-grained (process level) provenance through online and offline parts, yet does not provide provenance support for historical queries.

To satisfy the need of Big Data Provenance, a rule-based framework for provenance identification and collection from log files is proposed in [22]. The proposed framework reduces the source code instrumentation, yet raises several questions about completeness of provenance information, as logs may not capture complete information including derivation process. Another big provenance framework is proposed in [10] for provenance collection and storage in an unstructured or semi-structured format, for scientific applications. The proposed framework is light-weighted and built on multi-layered provenance architecture that supports a wide range of provenance queries. A provenance model for Apache Cassandra, i.e. a key-value pair database, is proposed in [31,32] to capture provenance information using provenance policies. In this model, provenance querying is performed through resource expressions and a set of predefined operators. The proposed model is implemented on a small sized patient information system and uses legacy thrift APIs rather than CQL3, that makes it difficult to write a query. Various change data capture (CDC) schemes are investigated in [48] for Apache Cassandra to track modifications in source data. The logic of each scheme is implemented in Cassan-

dra by combining a Map Reduce framework with distributed computing. A layer-based architecture for provenance collection and querying in scientific applications is presented in [1], which stores semi-structured provenance documents in MongoDB in a BSON format. The proposed architecture is prominent for simple queries but not efficiently respond to complex queries.

From the available literature, it is evident that most of the existing provenance models are suitable to capture provenance for workflows at coarse-grained level only rather than fine-grained level. Secondly, some of them are not suitable to capture provenance information for a large size social media data including all types of query set. In this paper, we try to bridge this gap by designing an efficient big social data provenance framework on the top of a key-value pair database for capturing, storing and querying provenance information for different query set including select, aggregate, standing/historical, and data update (i.e., insert, delete, update) queries on live streamed big social data. Summary of different characteristics of existing provenance solutions for social data and our proposed BSDP framework is given in Table 1.

3 Proposed provenance framework

In this paper, we propose a *Big Social Data Provenance (BSDP)* Framework build upon *Zero-Information Loss Key-Value Pair Database (ZILKVD)* that efficiently captures provenance for all queries including select, aggregate, standing/historical, and data update queries with insert, delete, and update operations. ZILKVD [45] is developed based on the concept of *Zero-Information Loss Database (ZILD)* [4,43,44,46]. The proposed framework is very beneficial in tracing out the origin and derivation history of a query result. It also supports provenance querying for historical data. The major steps involved in designing the proposed provenance framework are as follows:

1. Fetching a huge volume of real-life social data from Twitter's network through live streaming by using Twitter Streaming API's.
2. To design an efficient *Key-Value Pair (KVP)* data model based upon a query-driven approach to correlate big social data through relationships and dependencies, in appropriate formats so that it makes sense for further analysis.
3. Designing *ZILKVD architecture* with data version support to maintain all insert, delete, and update operations in the form of provenance data, that will aid in Historical data queries and Standing queries.
4. Proposing following three provenance generation algorithms, viz. *SelectProv*, *AggreProv*, and *StandProv* to generate provenance information for select, aggregate,

and standing/historical queries, and to store captured provenance in ZILKVD.

5. To provide *provenance querying* support for historical data and tracing out the origin and derivation history of a query result.

In addition to the above tasks, a performance analysis of proposed provenance capturing and querying algorithms are also presented for different query sets.

3.1 Social data streaming (Twitter case study)

Over the past few years, more than 90% of total sized data are contributed by the desperate usage of various social media platforms. Several leading social media platforms such as Twitter, Facebook, Instagram, etc., are fully responsible for this mammoth data. Out of these social media platforms, Twitter is one of the most precious mines of pretty specific and publicly available pullable social data, that allow users to share their thoughts with massive audience. It is tuned for very fast communications over internet with more than 150 million active users publishing approximate 500 million tweets daily. A twitter user can either create its own tweet or can retweet the information that has already been tweeted by some other user. A twitter user can choose to follow other users also. For instance, if a user A follows user B, then user A can see B's tweets in his 'timeline'. Twitter's popularity as a massive source of information has led to research in various domains [34]. Researchers can obtain this information from twitter through publically available Twitter APIs. These APIs are categorized in the following two categories; first is *REST APIs* for conducting specific searches, reading user profile or posting new tweets, and second is *Streaming APIs* to collect a continuous stream of public information. In our framework, we are using Streaming APIs to continuously stream the tweets and related information whenever the new tweet is published as shown in Fig. 1.

Twitter provides an open standard for authorization known as Open Authentication (OAuth). This authentication mechanism allows controlled and limited access to protected information. Traditional authentication mechanism is vulnerable to theft, while OAuth mechanism provides a more secure approach without using user's username and password. By using a three-way handshaking, it allows users to grant third party access to their data. As user's password for his/her twitter account is never shared with this third-party application, therefore, user's confidence in the application is also improved. Twitter APIs can only be accessed by a twitter application using OAuth authorization mechanism. To get the authorization for accessing the protected data, user first creates a twitter application which is also known as consumer. After registering this application on twitter, a *consumer key* and a *consumer secret key* is issued to the application by

Table 1 Summary of characteristics of existing provenance solutions for Social Data

Provenance model	Data model design	Provenance granularity	Provenance capture				Provenance visualization		Application domain
			Select query	Aggregate query	Standing query	Update query	Insert query	Delete query	
RAMP [39] (2011)	No	Workflow level	Yes	Yes	No	No	No	No	Generic
Web-based tool [42] (2013)	No	Fine-grained level (pre-defined attributes of social media user)	Yes	No	No	No	No	No	Generic
Seeking provenance paths [25] (2013)	No	Fine-grained level	Yes	No	No	No	No	No	Generic
Hadoop-Prov [2] (2013)	No	Workflow Level	Yes	Yes	No	No	No	No	Generic
Millieu [10] (2013)	No	Workflow level	Yes	Yes	No	No	No	No	Generic
KVPM [32] (2013)	No	Fine-grained level (column level)	No	No	No	Yes	No	No	Application specific (patient information system)
Layer based architecture [1] (2014)	No	Workflow level	Yes	No	No	Yes	No	No	Generic
Semiring provenance [41] (2015, 2018)	No	Fine-grained level	Yes	No	No	No	No	No	Limited for three specific graph algorithms
PROV-SAID [16,51,52] (2018)	No	Workflow level	Yes	No	No	No	No	No	Application specific
SFM [30] (2019)	No	Fine-grained level (only metadata information of tweet)	Yes	No	No	No	No	No	Generic
Ariadane [38] (2019)	No	Fine-grained level	Yes	No	No	No	No	No	Vertex centric graph analysis
s2p [55] (2021)	No	Fine-grained (operator level like map, reduce, filter, window)	Yes	Yes	No	No	No	No	Generic
Proposed (BSDP) framework	Yes	Fine-grained level	Yes	Yes	Yes	Yes	Yes	Yes	Generic

Fig. 1 Twitter data streaming

twitter that will uniquely identify this application. By using these *consumer key* and *consumer secret key*, application creates a unique twitter link through which user authenticate him/herself to twitter. After verifying the user's identity, twitter issues an *OAuth verifier* to the user. Application uses this *OAuth verifier* to request an *Access Token* and *Access Token Secret* that is unique to the user. Now, twitter application authenticates the user on twitter by using these *Access Token* and *Access Token Secret*, and make API calls on behalf of the user, see Fig. 2. By using these *Access Credentials*, we fetched all the tweets related to a specific event through live streaming to design an efficient key-value pair data model as explained in Algorithm 1 (i.e., *TweetCassandra*). The two inputs to the algorithm are (1) Twitter API Access Creden-

tials, i.e. *Consumer Key* (C_k), *Consumer Secret Key* (C_{sk}), *Access Token* (A_t), *Access Token Secret* (A_{ts}); and (2) *Event name* (E) for which related tweets are required to be fetched. An efficient query-driven KVP data model in Apache Cassandra is obtained as an output of this algorithm. After successful authorization on Twitter's Network using access credentials, tweet set (T) related to input event (E) is fetched through live streaming of social data, refer to line 1. Then, on every fetched tweet (t) of tweet set, pre-processing is performed to extract the following information, viz. *User* (U) who posted the tweet, *Hashtags* (H) and *Mentioned Users* (M) in the Tweet, *Tweet Body* (T) in UTF-encoding, and other related information, etc., refer to lines 2 and 3. Simultaneously, for each *User* (U), the following related information, viz. list of user's friends (F_{rl}), list of user's followers (F_{ol}) and user's profile attributes (U_i) such as *user_name*, *screen_name*, *profile created date*, *twitter id*, *location*, etc., is also extracted from *User's Twitter profile*, refer to line 4. Similarly, *Friend Details* (F_{rld}) and *Follower Details* (F_{old}) of each user are also extracted, refer to lines 5 to 10. Finally, all the extracted information is stored in corresponding column families of Apache Cassandra in appropriate format. This information is continuously streamed and populated in different column families to build an effective query-driven KVP data model for efficient queries.

3.2 KVP data model design in Apache Cassandra

The bulk proliferation of social data has been imposing several challenges in the field of social data analytics such as efficient data model design, querying techniques, etc. But traditional data management and processing tools are incapable to handle this limitless data. Relational/SQL databases are ideal for structured data and predictable workload but not scalable for handling Big Data which encompasses not just structured data, but also semi-structured and unstruc-

Algorithm 1 *TweetCassandra*: Cassandra Data Model Design for Twitter Data Streaming

Input: Twitter API Access Credentials (A_t , A_{ts} , C_k , C_{sk}) and Twitter Event (E)

//Where A_t =Access Token, A_{ts} = Access Token Secret, C_k =Consumer Key, C_{sk} = Consumer Secret Key

Output: Cassandra Data Model (CDM)

```

1: Fetch Tweet Set  $T$  of Event  $E$ 
2: for all  $t \in T$  do
3:    $U, H, M, U_r, T, TW_i \leftarrow \text{Preprocess}(t)$ 
4:    $F_{rl}, F_{ol}, U_i \leftarrow \text{Fetch } U$  //Where  $F_{rl}$ =User's friend list,  $F_{ol}$ =User's follower list,  $U_i$ =other user information
5:   for all  $f \in F_{rl}$  do
6:      $F_{rld} \leftarrow \text{Fetch } f$  //Where  $F_{rld}$ =User's friend details
7:   end for
8:   for all  $f \in F_{ol}$  do
9:      $F_{old} \leftarrow \text{Fetch } f$  //Where  $F_{old}$ =User's follower details
10:  end for
11:   $C_{cf} \leftarrow \text{Insert (Extracted Data)}$ 
12:  Update  $C_{cf}$  //Ccf = Cassandra column-families
13: end for
14: End
  
```

Fig. 2 Open authentication process of Twitter

tured data. Social media platforms are the major source of unstructured data in the current times. Unstructured data is characterized by ad hoc schema and therefore cannot be stored in SQL databases. The growth of unstructured data has led to interest in NoSQL databases. NoSQL databases are much better suited due to their flexible schema. NoSQL represents a family of databases in which each database is quite different from others having literally nothing in common. The only commonality is that they use a data model with structure that is different from the traditional row-column relation model of RDBMSs. Graph, Document, Column-oriented, & Key-value pair are the four kinds of NoSQL databases. The basic architecture of a KVP database consists of a two-column hash table in which each row contains a unique id known as a “key”, and a “value” associated with this key. The KVP databases are a good choice to handle extremely high volumes of data in a distributed processing environment as they have a built-in redundancy, which is capable to handle the losses of storage nodes. The key strengths of KVP databases are their simplicity, scalability, and a very fast streamlined NoSQL architecture. These have the capabilities to perform an extremely fast read and write operations. Apache Cassandra is one of the most popular KVP database that comes under the ambit of NoSQL databases. It is a distributed column family store in which each column is a data structure that contains a key, a value, and a timestamp; therefore, it is also named as key-value pair column-oriented data store, see Fig. 3. The brief introduction of elementary components of information in Apache Cassandra is given below:

Fig. 3 Cassandra column, row and column-family structure

Column: Column is a smallest unit of information that contains a key, value, and timestamp.

Super Column: Super Column or composite column is a group of similar columns, or columns likely to query together with common name.

Row: A Row is a group of orderable columns, i.e., columns are stored in sorted order by their column names, with a unique row key or primary key that can uniquely identify data.

Column Family: Column Family is similar to a table in relational database but no pre-defined schema, and also provides flexibility to have different number of columns in different rows. Column families are stored in separate files on the disk.

Keyspace: Keyspace is the highest level of information in Apache Cassandra, analogous to the database in relational database, which is the set of related column families. It also maintains the information about data replication, and replication strategy on nodes.

Although Apache Cassandra is known for flexible data management to manage world's biggest datasets on clusters of several nodes deployed at different data centres, however, one of the major challenges that big social data applications face when choosing Apache Cassandra is data model design that is significantly different from traditional data model design methodologies. Traditional data model design methodology (i.e. used in relational databases) is purely a data-driven approach. On the contrary, data model design for Cassandra begins with application-specific queries, and it is purely a query-driven approach. Several SQL constructs such as data aggregation, table joins, etc., are not supported by Cassandra Query Language (CQL). Therefore, data mod-

elling in Cassandra relies on denormalization of database schema that enable a complex query to execute on a single column family only, to retrieve the required information. In this way, data duplication is common in Cassandra column families to support a variety of queries. Database schema design for big social data in Cassandra requires not only the understanding of relationships and dependencies among social data, but also the understanding of needs to access this data through a query driven approach. In this paper, we applied a query-driven methodology in KVP data model design. By a query driven, we mean designing a data model on the basis of what type of queries our database will required to support. This approach provides not only the sequence of tasks but also aids in determining what type of data will be needed and when? In our proposed framework, we designed a query-driven data model based on frequent queries required to execute on Twitter dataset. Initially, all the tweets posted by different Twitter users in the response of a particular event are fetched through Twitter's Streaming APIs. However, all such information is not being useful for our data model; therefore, only required information, viz. tweet id, tweet text, tweet published date, hashtags, user_name, screen_name, profile created date, twitter id, location, friend list, follower list, etc., is extracted from the input list of tweet objects. Simul-

Fig. 4 KVP data model in Apache Cassandra

Fig. 5 Zero-information loss KVP database architecture

taneously, pre-processing on extracted data is performed to convert them in a required format. Afterwards, all such pre-processed data is stored in different column families of Apache Cassandra. The snapshot of KVP data model design in Apache Cassandra is given in Fig. 4. Proposed data model contains a keyspace named “NewTwitter_KeySpace” that consists of 20 Column Families. The various column names of these column families with their row keys are also mentioned in Fig. 4. All 20 column families are organized on the basis of social data set fetched from the Twitter’s network to support different query sets for capturing, storing, and querying provenance. Cassandra Query Language (CQL) is used for querying and to communicate with Apache Cassandra.

3.3 ZILKVD architecture design

The proposed provenance framework for big social data is developed on top of Zero-Information Loss Key-Value Pair Database (ZILKVD). ZILKVD is designed by using the concept of Zero-Information Loss Database [4], to maintain all the insert, delete, and update operations without losing any information as a provenance data. The architecture of ZILKVD consists of following components, viz. Query Parser, Query Rewriter, Query Generator, Processing Module, and KVP Database, see Fig. 5. When user issues a query, it is sent to the Query Parser to parse the query and to identify the type of that query, i.e. Insert (I), Update (U), or Delete (D) query.

- If issued query type is an “*Insert Query*” (i.e., to insert a new row in database), then the parsed results are sent to the Query Rewriter as mentioned in step I₁ and corresponding Rewritten Insert Query (Q_i) is generated in step I₂. Here, “*valid_from*” column of this new row in corresponding column family is being set to the “*current*

date/time” and then it is sent to the KVP database for further execution.

- If issued query type is a “*Delete Query*” (i.e., to delete an existing row from the database), then the parsed results are sent to the Query Generator as mentioned in step D₁ and corresponding Update Query (Q_u) is generated in step D₂. Here, the value of “*valid_to*” column of the row to be deleted from the corresponding column family is being set to the “*current date/time*” and then it is sent to the KVP database for further execution.
- If issued query type is an “*Update Query*” (i.e., to update an existing row in database), then the parsed results are sent to both Query Generator and Processing Module in steps U_{1a} and U_{1b}, respectively. Then, in step U₂, corresponding Select Query (Q_s) generated from Query Generator is executed on KVP database, to retrieve the following information, viz. value of primary key columns of the row to be updated, old value of column before performing update, and its write time in database. This information is sent to the Processing Module in step U_{3b}, to generate corresponding Provenance Path Expression (*ProvPathExp*) in the following format, i.e., “*Key-space/Column_Family/RowKey/Update_Column_Name*”, and then sent back to the Query Generator in step U₄.
- Now, in step U₅, Query Generator generates an insert query (Q_i) to insert the following information in “*update_provenance*” column family, viz. *Query statement*, *ProvPathExp*, *old_value*, *old_value writetime* (i.e., its *valid_from* time), *new_value*, *current Date/Time*, etc., for further execution on KVP database.
- Afterwards, both the queries (i.e., generated insert query Q_i and issued update query U) are executed on KVP database in step U₅ and U₆, respectively, to maintain the complete history of data update operations. Finally, following information, viz. *Query Id*, *Query Statement*, its

time of execution, etc., are also inserted in “*query_table*” column family through an insert query executed on KVP database in step U_7 .

The high-level details of the implementation code of ZILKVD Design are given in Algorithm 2 and 3. Two inputs to Algorithm 2 are (1) A KVP Database (D_{kv}) and (2) A query Q (i.e., insert, delete or update query), and output of the algorithm is a ZILKVD database with complete history maintained.

According to algorithm 2, the issued input query Q is first parsed to retrieve the required information, i.e., parsed result R_p and to identify the query type, i.e., Q_t , refer to line 1. If Q_t is an insert query, then a corresponding rewritten insert query Q_i is generated and sent for the execution on D_{kv} , refer to lines 3 and 4. If Q_t is a delete query, then a corresponding update query Q_u is generated and sent for the execution on D_{kv} , refer to lines 6 and 7. If Q_t is an update query, then Algorithm 3, i.e., UpdateCassProv, is called, refer to line 9. The following two inputs, i.e., query Q and its parsed result R_p , are passed to the algorithm 3 and provenance path expression (P_p) of updated columns and updated “*query_table*” and “*update_provenance*” column families are obtained as outputs of the algorithm 3. According to the algorithm 3, all the required information such as KS , CF , PK , CN_u , CV_u , CT_u , etc., are retrieved from R_p , see line 1.

- If Q contains a “*Where Clause*” in its query statement, then value of V_{pk} is retrieved and assigned to RK to uniquely identify a row, refer to lines 2 to 4. Afterward, a corresponding select query Q_s is generated and executed to retrieve old value of column before update, and its write time in database, i.e., CV_o and CV_{owt} , respectively, refer to lines 5 and 6. Now, provenance path P_p (i.e., $KS/CF/RK/CN_u$) is generated and column family “*update_provenance*” is updated with updated values of Q_{id} , Q , P_p , CV_o , CV_{owt} , CV_u , CT_u , and current date/time, refer to lines 7 and 8.

- Similarly, if Q does not contain a “*Where Clause*,” then again Q_s is generated and executed to store all the query results in RS , refer to line 11. Now, for each result tuple r of RS , value of following parameters, i.e., V_{pk} , CV_o , CV_{owt} , etc., are retrieved and value of V_{pk} is assigned to RK . Afterwards, corresponding provenance path P_p (i.e., $KS/CF/RK/CN_u$) is generated and column family “*update_provenance*” is updated with updated values of Q_{id} , Q , P_p , CV_o , CV_{owt} , CV_u , CT_u , and current date/time, refer to lines 12 to 16.
- Finally, Q is executed and column family “*query_table*” is also updated with updated values of following parameters, i.e., Q_{id} , Q , current date/time, etc., refer to lines 19 and 20. A demonstration of above algorithms with illustrative example query 1 is given below:

Example Query 1: Update location of the user with name “DDNewsAndhra”.

Cassandra Query 1: update user_details set location = ‘Andhra’ where screen_name=‘DDNewsAndhra’;

Initially, the above Example Query 1 is passed as an input query (Q) to the Algorithm 2, where the query is parsed to identify its type (i.e., Update Query) and to retrieve the required information. Now, both query (Q) and its parsed results (R_p) are passed as inputs to the Algorithm 3. Here, the provenance path expression (i.e., ProvPathExp) of updated tuples along with the updated column families, viz. “*query_table*” and “*update_provenance*” of underlying KVP database is obtained as outputs of above algorithm. A snapshot of “*update_provenance*” column family is shown in Fig. 6.

3.4 Provenance generation algorithm

We designed and implemented three provenance generation algorithms for *select*, *aggregate*, and *standing queries*, respectively. The high level details of all the algorithms along with their illustrative example queries are given in the following subsections:

Algorithm 2 ZILKVD Design: Design ZILKVD (Zero Information Loss Key-Value Pair Database)

Input: Key-Value Pair Database (D_{kv}), Query Q (Insert/Update/Delete Query)

Output: Zero Information Loss Key-Value Pair Database with history maintained

```

1: Parsed Result ( $R_p$ ), Query Type ( $Q_t$ ) ← Query Parser ( $Q$ )
2: if  $Q_t$  is Insert-Query to insert a new row then
3:   Generate  $Q_i$  //Where  $Q_i$  is rewritten insert query with “valid_from” column set as “current date/time”
4:   Execute ( $D_{kv}$ ,  $Q_i$ )
5: else if  $Q_t$  is Delete-Query to delete an existing row then
6:   Generate  $Q_u$  //Where  $Q_u$  is corresponding update query with “valid_to” column set as “current date/time”
7:   Execute ( $D_{kv}$ ,  $Q_u$ )
8: else
9:   UpdateCassProv( $Q$ ,  $R_p$ )
10: end if
11: End

```

Algorithm 3 *UpdateCassProv(Q, R_p): ZILKVD with Update Management***Input:** Query Q (Update Query), R_p(Parsed Results)**Output:** ProvPathExp (P_p) of updated tuples, updated column families viz., “query_table”, “update_provenance”

```

1:  $KS, CF, PK, CN_u, CV_u, CT_u \leftarrow \text{Retrieve}(R_p)$  //Where  $KS=Keyspace$ ,  $CF=Column\_Family$ ,  $PK=Primary\ Key\ of\ CF$ ,
// $CN_u=Update\ column\ name$ ,  $CV_u=Update\ column\ value$ ,  $CT_u=Update$ 
//column type
2: if Q contains where clause then
3:    $\text{Retrieve } V_{pk} \leftarrow \text{Parse where clause}$  //Where  $V_{pk} = \text{Values of } PK$ 
4:    $RK \leftarrow V_{pk}$  //Where  $RK$  is RowKey
5:   Generate  $Q_s$  //  $Q_s$  is a select query corresponding to Q for retrieving old value and write time of  $CN_u$ 
6:    $CV_o, CV_{owt} \leftarrow \text{Execute } Q_s$  //Where  $CV_o = \text{Old column value}$ ,  $CV_{owt} = \text{Old column value WriteTime}$ 
7:    $P_p \leftarrow KS/CF/RK/CN_u$  //Where  $P_p$  is provenance path
8:    $\text{update\_provenance} \leftarrow \text{Insert } Q_{id}, Q, P_p, CV_o, CV_{owt}, CV_u, CT_u, \text{current date/time}$ 
9: else
10:  Generate  $Q_s$  //  $Q_s$  is a select query corresponding to Q for retrieving old value, write time of  $CN_u$  and
//values of  $PK$ 
11:   $RS \leftarrow \text{Execute } Q_s$  //RS is result set of query  $Q_s$ 
12:  for all  $r \in RS$  do
13:    Obtain  $V_{pk}, CV_o, CV_{owt} \leftarrow r$ 
14:     $RK \leftarrow V_{pk}$ 
15:     $P_p \leftarrow KS/CF/RK/CN_u$ 
16:     $\text{update\_provenance} \leftarrow \text{Insert } Q_{id}, Q, P_p, CV_o, CV_{owt}, CV_u, CT_u, \text{current date/time}$ 
17:  end for
18: end if
19: Execute  $\leftarrow Q$ 
20:  $\text{query\_table} \leftarrow \text{Insert } Q_{id}, Q, \text{current date/time}$ 
21: End

```

Operation	Query Id	Query	Column type	Old value	Old_value_writetime	New value	Provenance_path_exp	Row key	Time
update	q23	update user_details set location=' Andhra' where screen_name=' DDNewsAndhra '	VARCHAR	Andhra Pradesh	2019-12-17 08:30:01	Andhra	NewTwitter_Keyspace/user_details/'DDNewsAndhra '/Location	screen_name	2019-12-17 9:16:47
update	q25	update user_details set user_name= 'Reality Show' where screen_name=' myshowtalkies'	VARCHAR	MY SHOW MY TALKS	2019-10-17 09:07:54	Reality Show	NewTwitter_Keyspace/user_details/'myshowmytalks'/user_name	screen_name	2019-12-17 9:18:39

Fig. 6 A snapshot of “update_provenance” column family**3.4.1 Provenance generation for select queries**

Proposed framework supports to capture provenance information for select queries. The high-level details of provenance generation algorithm for select queries, i.e., “*Select-Prov*”, are given in Algorithm 4. In proposed algorithm, a select query (Q_s) and its query id (Q_{id}) are passed as inputs and a comma separated list of provenance path expression (P) for each value exists in the result tuple of a query result along with the following updated column families, viz. “*select_provenance*”, and “*query_table*” are obtained as outputs of the algorithm. Initially, Q_s is parsed and the following information, viz. KS , CF , PK , CN , etc., is retrieved from the query statement in the form of parsed result R_p , refer to lines 1 and 2. Then, a rewritten select query Q_r is generated by appending a predicate (i.e., “*valid_to*”) in the query statement, refer to line 3. The value of this predicate is being set to the *Null* to retrieve currently existing rows. Afterwards, Q_r is executed and all its result tuples are stored in record set

(RS), refer to line 4. Now, for each result tuple r of result set, a unique result tuple id is generated by using Q_{id} , refer to lines 5, 7 and 17. Initially, the value of P for all columns of each result tuple is being set to the null, refer to line 8. Then, the value of V_{pk} is retrieved from result tuple and assigned to RK , refer to lines 9 and 10. After that, for each non-key column C_i of r , provenance path expression p_i is generated (i.e., $KS/CF/RK/C_i$) and added in the corresponding r and further appended in P , refer to lines 11 to 14. A provenance path expression consists of a keyspace name, column family, row key, and column name in the following form: “*keyspace/columnfamily/rowkey/columnname*”. Provenance path expression provides a detailed provenance for each of the result tuple exists in the query result at different granularity levels, i.e., How a value in result tuple is derived. Finally, column families “*select_provenance*” and “*query_table*” are also updated, refer to lines 14 to 20. Demonstration of Algo-

Algorithm 4 *SelectProv*: Provenance Generation for Current Select Query**Input:** Query Q_s (Select Query), Q_{id} (Query Id)**Output:** ProvPathExp (p_i) with each value in result set, updated “select_provenance” and “query_table” column families

```

1:  $R_p \leftarrow \text{Parse } Q_s$ 
2:  $KS, CF, PK, CN \leftarrow \text{Retrieve}(R_p)$  //Where  $KS=Keyspace$ ,  $CF=Column\_Family$ ,  $PK=Primary\ Key\ of\ CF$ ,
// $CN=Column\ names\ in\ Q_s$ 
3: Generate  $Q_r$  // $Q_r$  is rewritten select query for retrieving values of  $Pk$  and appending
//predicate with “valid_to” as Null
4:  $RS \leftarrow \text{Execute } Q_r$ 
5:  $k=0$ 
6: for all  $r \in RS$  do
7:  $r_{tid} = Q_{id} + 't' + k$  // $r_{tid}$  is unique id of each result tuple of  $Q_r$ 
8: Set  $P = \text{Null}$ 
9: Obtain  $v_{pk}$  // $v_{pk}$ =Values of  $PK$ 
10:  $RK \leftarrow v_{pk}$  // $RK$  is Row Key
11: for  $i=1$  to  $n$  do
12:  $p_i \leftarrow KS/CF/RK/C_i$  //Where  $n=\text{number of non-key columns in } CN$ ,  $C_i \in CN$  and  $C_i$  is a non-key
//column,  $p_i$  is provenance path of  $C_i$ 
13:  $r \leftarrow \text{Add } p_i$  //Adding provenance of  $C_i$ 's value in  $r$ 
14:  $P \leftarrow \text{Append } p_i$  //P is comma separated list of provenance paths of all  $C_i$ 's in  $r$ 
15: end for
16:  $\text{select\_provenance} \leftarrow \text{Insert } r_{tid}, Q_s, P, \text{current date/time}$ 
17: Increment  $k$ 
18: end for
19:  $\text{query\_table} \leftarrow \text{Insert } Q_{id}, Q_s, \text{current date/time}$ 
20: Return  $RS$ 
21: End

```

Algorithm 4 with illustrative example queries 2 and 3 is given below:

Example Query 2: Display the location of user with Screen_Name ‘Gagan4041’.

Cassandra Query 2: select location from user_details where screen_name=‘Gagan4041’;

Query result of the above select query contains the following two columns, viz. “LOCATION” and “LOCATION_PROVENANCE” with values “India” and “[NewTwitter_Keyspace/user_details/Gagan4041/location]”, respectively. Here, the value under the column name “LOCATION_PROVENANCE” justifies the query result, i.e., “India”. It explains that the value in result set is derived from keyspace: NewTwitter_Keyspace, column family: user_details, row key: Gagan- 40041, column: location.

Example Query 3: Display all hashtags used in the tweets posted by a user with Screen_Name ‘mkzangid’.

Cassandra Query 3: select hashtag from user_tweet_hashtag where screen_name=‘mkzangid’;

Query result of the above query is shown in Fig. 7, which shows that the user “mkzangid” used hashtag “Vikramlander” in two of his tweets with tweet id’s “1181510817377767426” and “1181512471518990342”. Provenance path expression under column “Hashtag_Provenance” shows the derivation process of the value present in result set, i.e., value “Vikramlander” in result set is derived from two different

Hashtag	Hashtag_Provenance
[Vikramlander]	[NewTwitter_Keyspace/user_tweet_hashtag/mkzangid-1181510817377767426/hashtag]
[Vikramlander]	[NewTwitter_Keyspace/user_tweet_hashtag/mkzangid-1181512471518990342/hashtag]

Fig. 7 Example query 3 result

rows with row key (composite primary key of screen_name and tweet id) “mkzangid-1181510817377767426” and “mkzangid-1181512471518990342”.

3.4.2 Provenance generation for aggregate queries

Proposed framework supports to capture provenance information for aggregate queries too. The high-level details of provenance generation algorithm for aggregate queries, i.e., “AggreProv”, are given in Algorithm 5. According to this algorithm, an Aggregate Query (Q_a) with its Query Id (Q_{id}) is passed as an input and a comma separated list of Provenance Path Expressions $pv[i]$ for each of its result tuple exists in query result is obtained as an output in Provenance Vector (pv). The provenance path expression consists of all the source rows and column names of a column family in a keyspace that contributed to generate the corresponding result tuple. All the steps of this algorithm are very similar to Algorithm 4, i.e., “SelectProv” except the concept of prove-

Algorithm 5 AggreProv: Provenance Generation for Current Aggregate Query**Input:** Query Q_a (Aggregate Query), Q_{id} (Query Id)**Output:** Comma separated list of ProvPathExps ($pv[i]$) with each value in result set, updated “select_provenance” and “query_table” column families

```

1:  $R_p \leftarrow \text{Parse } Q_a$ 
2:  $KS, CF, PK, CN \leftarrow \text{Retrieve}(R_p)$  //Where  $KS=Keyspace$ ,  $CF=Column\_Family$ ,  $PK=Primary\ Key$  of  $CF$ ,
// $CN=Column\ names\ in\ Q_a$ 
3: Generate  $Q_r$  // $Q_r$  is rewritten aggregate query for retrieving values of  $Pk$  and appending
//predicate with “valid_to” as Null
4:  $RS \leftarrow \text{Execute } Q_a$ 
5:  $RS1 \leftarrow \text{Execute } Q_r$ 
6:  $k=0$ 
7: while  $RS1 \neq \text{Null}$  do
8:    $r1 \leftarrow \text{Iterate over } RS1$ 
9:   for all  $r \in RS$  do
10:     $r_{tid} = Q_{id} + 't' + k$  // $r_{tid}$  is unique id of each result tuple of  $Q_r$ 
11:    Set  $P = \text{Null}$ 
12:    Vector  $pv = \text{Null}$  // $pv$  is a vector to store all provenance paths of  $r$ 
13:    for all  $r1 \in RS1$  till value of aggregate attribute in  $r1$  is same as in  $r$  do
14:      Obtain  $v_{pk}$  from  $r1$  // $v_{pk}$ =Values of  $PK$ 
15:       $RK \leftarrow v_{pk}$  // $RK$  is Row Key
16:      for  $i=1$  to  $n$  do
17:         $pr \leftarrow KS/CF/RK/C_i$  //Where  $n$ =number of non-key columns in  $CN$ ,  $C_i \in CN$  and  $C_i$  is a non-key
//column,  $pr$  is provenance path of  $C_i$ 
18:         $pv[i] \leftarrow \text{Append } pr$  // $pv[i]$  is comma separated list of all provenance path of  $C_i$ 's value in  $r$ 
19:         $P \leftarrow \text{Append } pr$  // $P$  is comma separated list of provenance paths of all  $C_i$ 's in  $r$ 
20:      end for
21:    end for
22:     $r \leftarrow \text{Add } pv$  // Adding provenance vector of all  $C_i$ 's value in  $r$ 
23:     $\text{select\_provenance} \leftarrow \text{Insert } r_{tid}, Q_a, P, \text{current date/time}$ 
24:    Increment  $k$ 
25:  end for
26: end while
27:  $\text{query\_table} \leftarrow \text{Insert } Q_{id}, Q_a, \text{current date/time}$ 
28: Return  $RS$ 
29: End

```

nance vector. Although, Provenance path is generated in the same way as in Algorithm 4; however, iteration is performed on all source rows that contributed to produce one result row in result set to generate $pv[i]$ of all source rows, refer to lines 13 to 21. Further, provenance of result tuples and corresponding aggregate query are stored in “select_provenance” and “query_table” column families, respectively. Demonstration of Algorithm 5 with illustrative example queries 4 and 5 is given below:

Example Query 4: Display the total no of tweets posted by a user “sunilthalia” on “08/10/2019”.

Cassandra Query 4: select count(tweet_body) from tweets_user_day where screen_name=‘sunilthalia’ and published_day=8 and published_date>=‘2019-10-08’ and published_date<‘2019-10-09’ group by screen_name allow filtering;

The above query is an example of aggregate query to retrieve the total number of tweets posted by a specific user on a given day. This aggregate query efficiently executes on “tweets_user_day” column family with composite primary key, i.e., “screen_name, published_day, and pub-

SYSTEM.COUNT(TWEET_BODY)	SYSTEM.COUNT(TWEET_BODY)_PROVENANCE
7	[NewTwitter_Keyspace/tweets_user_day/sunilthalia-8-Tue Oct 08 11:37:56 IST 2019/tweet_body, NewTwitter_Keyspace/tweets_user_day/sunilthalia-8-Tue Oct 08 11:40:12 IST 2019/tweet_body, NewTwitter_Keyspace/tweets_user_day/sunilthalia-8-Tue Oct 08 11:48:33 IST 2019/tweet_body, NewTwitter_Keyspace/tweets_user_day/....]

Fig. 8 Example query 4 result

lished_date”. Figure 8 shows the partial result of above aggregate query where the total number of tweets posted by the given user on 08/10/2019 are 7 (mentioned under the column name “SYSTEM.COUNT(TWEET_BODY)”) along with a comma separated list of provenance path expressions for all the 7 rows with the name of column families that have contributed to the result set under the column name “SYSTEM.COUNT(TWEET_BODY)_PROVENANCE”.

PUBLISHED_DAY	SYSTEM.COUNT (TWEET_BODY)	SYSTEM.COUNT(TWEET_BODY)_PROVENANCE
23	5	[NewTwitter_Keyspace/tweets_day/23-Wed Oct 23 12:35:17 IST 2019/tweet_body, NewTwitter_Keyspace/tweets_day/23-Wed Oct 23 12:38:14 IST 2019/tweet_body, NewTwitter_Keyspace/tweets_day/...]
5	34	[NewTwitter_Keyspace/tweets_day/5-Sat Oct 05 00:01:18 IST 2019/tweet_body, NewTwitter_Keyspace/tweets_day/5-Sat Oct 05 04:46:01 IST 2019/tweet_body, NewTwitter_Keyspace/tweets_day/...]
10	8	[NewTwitter_Keyspace/tweets_day/10-Thu Oct 10 01:09:32 IST 2019/tweet_body, NewTwitter_Keyspace/tweets_day/10-Thu Oct 10 07:03:30 IST 2019/tweet_body, NewTwitter_Keyspace/tweets_day/...]
30	1	[NewTwitter_Keyspace/tweets_day/30-Wed Oct 30 14:23:14 IST 2019/tweet_body]
13	2	[NewTwitter_Keyspace/tweets_day/13-Sun Oct 13 00:37:02 IST 2019/tweet_body, NewTwitter_Keyspace/tweets_day/13-Sun Oct 13 10:51:20 IST 2019/tweet_body]

Fig. 9 Example query 5 result

Example Query 5: Display the total no of tweets posted on each day in month of October, 2019.

Cassandra Query 5: select published_day, count(tweet_body) from tweets_day where published_date>='2019-10-01' and published_date<'2019-11-01' group by published_day allow filtering;

The above aggregate query executed on “tweets_day” column family with composite primary key, i.e., “published_day, published_date”, and counts the total number of tweets posted on each day of October, 2019. Partial result of above aggregate query is shown in Fig. 9, where the total number of tweets posted on each day is shown under the column name “SYSTEM.COUNT (TWEET_BODY)”, along with the tweets posted day, and a comma separated list of provenance path expression for all the rows that contributed towards aggregated result under the column name “SYSTEM.COUNT (TWEET_BODY)_PROVENANCE”.

3.4.3 Provenance generation for standing queries

Proposed framework also supports to capture provenance information for historical/standing queries using data versioning support in ZILKVD. The high-level details of provenance generation algorithm for standing queries, i.e., “StandProv”, are given in Algorithm 6, where a Standing Query (Q_{st}) along with its Time of Execution(t) is passed as an input and a comma separated list of Provenance Path Expressions (p_i) for all of its result tuple exists in query result

is obtained as an output. Initially, query Q_{st} is parsed to retrieve the following information, viz. Keyspace, Column Family, Column Names, Primary Key, etc., refer to lines 1 and 2. Afterwards, a Rewritten Select Query (Q_r) is generated to retrieve Row Key (RK), i.e., values of primary key column of column family, and each result tuple with predicate “valid_to”. The value of this predicate is set to time “ t ” (i.e., given in input) and then query Q_r is executed on database, refer to lines 3 to 7. Now, for every value in result set of Q_r , its “writetime” (time of existence in database) is compared with “ t ”. If “writetime” is less than or equal to “ t ”, then provenance path expression (p_i) is generated with corresponding source row and column contributed towards its generation and further, added in result tuple refer to lines 9 to 11,. But, if “writetime” is greater than “ t ”, then corresponding column value and provenance path are retrieved from “update_provenance” column family, refer to lines 13 to 15. At the end, the value of column and provenance path expression that are retrieved from “update_provenance” column family are updated in result set and finally, updated result set along with provenance information is obtained, refer to lines 16 to 21.

3.5 Provenance storage

In our proposed framework, all the captured provenance is stored in the following three column families of Apache Cassandra for further analysis, viz. “query_table”, “select_provenance”, and “update_provenance”, see Fig. 10.

Algorithm 6 *StandProv*: Provenance Generation for Standing/Historical Query**Input:** Query Q_{st} (Standing/Historical Query), t (Time of Execution)**Output:** ProvPathExp (p_i) for each value of result set.

```

1:  $R_p \leftarrow \text{Parse } Q_{st}$ 
2:  $KS, CF, PK, CN \leftarrow \text{Retrieve}(Rp)$  //Where  $KS=Keyspace$ ,  $CF=Column\_Family$ ,  $PK=Primary\ Key\ of\ CF$ ,
// $CN=Column\ names\ in\ Q_{st}$ 
3: Generate  $Q_r$  // $Q_r$  is rewritten select query for retrieving values of  $Pk$  and appending
//predicate with “valid_to” as “t”
4:  $RS \leftarrow \text{Execute } Q_r$ 
5: for all  $r \in RS$  do
6:   Obtain  $v_{pk}$  //vpk=Values of PK
7:    $RK \leftarrow v_{pk}$  //RK is Row Key
8:   for all  $v$  in  $r$  do
9:     if writetime( $v$ ) $\leq t$  then
10:       $p_i \leftarrow KS/CF/RK/C$  //Where  $v$  is value of column  $C$  in  $r$  and  $C \in CN$ ,  $p_i$  is provenance path of  $C$ 
11:       $r \leftarrow \text{Add } p_i$  //Adding provenance of  $C_i$ 's value in  $r$ 
12:     else
13:       Generate  $Q_s$  // $Q_s$  is rewritten select query for retrieving value of  $C$  ( $v_c$ ) from update_provenance
//column family with predicate “valid_to” as “t”
14:        $RS1 \leftarrow \text{Execute } Q_s$ 
15:        $p_i, v_c \leftarrow \text{Retrieve from } RS1$  //Where  $p_i$  is provenance path of value  $v_c$  of column  $C$ 
16:        $r \leftarrow \text{Update } v$  with  $v_c$  in  $r$ 
17:        $r \leftarrow \text{Add } p_i$ 
18:     end if
19:   end for
20: end for
21: Return  $RS$ 
22: End

```

Fig. 10 Provenance storage

Provenance information of all the executed queries with their query id and time of executions is stored in “*query_table*” column family. Provenance path expressions for all the result tuples of select/aggregate queries are stored in “*select_provenance*” column family along with their query statement, result tuple id and time of executions as shown in Fig. 11. Similarly, the column family “*update_provenance*”

keeps the provenance information about all the update operations along with following attributes, i.e., *query statement*, *provenance path expression*, *old value* and *its write time*, *new value*, *column type*, and *time of update (current date/time)*, see Fig. 6. The captured provenance is used in source tracing, update tracking, and in querying historical data. Further, the

Query Id	Query	Result tupleid	Provenance_path_exp	Time
q6	select location from user_details where screen_name='Gagan4041'	q6t1	['NewTwitter_Keyspace/user_details/Gagan4041/location']	2019-12-16 05:02:12
q7	select screen_name, published_day, count(tweet_body) from tweets_user_day where published_day=8 and published_date>= '2019-10-08' and published_date< '2019-10-09' group by screen_name allow filtering	q7t1	['NewTwitter_Keyspace/tweets_user_day/barrotchetan999-8-Tue Oct 08 01:38:10 IST 2019/tweetbody']	2019-12-18 04:54:02
q7	select screen_name, published_day, count(tweet_body) from tweets_user_day where published_day=8 and published_date>= '2019-10-08' and published_date< '2019-10-09' group by screen_name allow filtering	q7t2	['NewTwitter_Keyspace/tweets_user_day/asifawan8080780-8-Tue Oct 08 01:38:10 IST 2019/tweetbody']	2019-12-18 04:54:02
q7	select screen_name, published_day, count(tweet_body) from tweets_user_day where published_day=8 and published_date>= '2019-10-08' and published_date< '2019-10-09' group by screen_name allow filtering	q7t3	['NewTwitter_Keyspace/tweets_user_day/ahmedtanzeel105-8-Tue Oct 08 01:38:10 IST 2019/tweetbody']	2019-12-18 04:54:02

Fig. 11 A snapshot of select_provenance column family

visualization of this provenance data is helpful in analysing and determining the truthiness of a query result.

3.6 Provenance querying

The proposed framework also supports querying provenance information for various purposes such as audit trail, updates tracking, source tracing, data discovery, etc. Provenance querying on captured provenance is carried out to achieve the following two objectives: first, *How any result tuple of select query is derived?*, i.e., querying provenance to know about the source of information, and second *How to track all the updates performed on a given data?*, i.e., querying provenance for historical data. Framework provides the following two column families to accomplish the above tasks, viz. “select_provenance” and “update_provenance”. Provenance path expressions for all the result tuples of select/aggregate queries along with their query statement, result tuple id and time of executions are stored in “select_provenance” column family. This provenance information is used in provenance querying to know about the source of information as shown in Fig. 11. Similarly, the column family “update_provenance” stores the provenance information about all the update operations performed along with the following parameters, i.e., query statement, provenance path expression, old value and its write time, new value, column type, and time of update (current date/time). This provenance information is used in provenance querying for historical data, see Fig. 6. In addition to above column families, one more column family, i.e., “query_table” is also used in provenance querying to obtain the information about all the queries executed till a particular

date with their time of execution. The illustrative examples of provenance querying are given below:

Example provenance Query 1: Explain how result tuple q6t1 of query q6 (as shown in Fig. 11) is derived?

The above query is executed on “select_provenance” column family to retrieve provenance path expressions for result tuple q6t1 of query q6 along with its time of execution. Here, provenance path expression of resultant tuple is “[NewTwitter_Keyspace/user_details/Gagan4041/location]” and time of query execution is “2019-12-16 05:02:34.266000+0000”. This indicates that the source keyspace name of required tuple is “NewTwitter_Keyspace”, name of column family is “user_details”, row key is “Gagan4041”, column name is “location” and time of query execution is “2019-12-1605:02:34.26600 0+0000”. Now, “user_details” column family is queried with this row key, column name and execution time to retrieve all the rows that contributed to produce the result tuple t1 of query q6 which justify the resultant tuple. However, if the source has been modified after query execution, in that case, the original source can still be devised through querying historical data. To support provenance querying for historical data, we designed the following four User-Defined CQL Constructs (UDCs), viz. “all”, “instance”, “validon now”, and “validon date”. These constructs are further categorized in the following two categories, viz. T1 (“all”, “instance”) and T2 (“validon now”, “validon date”).

The high level details of provenance querying algorithm for historical data, i.e., “QueryProv_HistData”, are given in Algorithm 7, in which an Extended Query (QE) (i.e., a CQL query with UDCs) is passed as an input and a corresponding

Algorithm 7 QueryProv_HistData: Querying Provenance for Historical Data**Input:** Query Q_E (Provenance Query with extended user-defined constructs T1,T2)**Output:** RS(Result Set) of Q_E

```

1:  $R_p, T1, T2 \leftarrow \text{Parse } Q_E$  //  $R_p$  is parsed query result,  $T1 = \text{"instance"/"all"}$ ,  $T2 = \text{"validon now"/"validon date"}$ 
2: Generate  $Q$  //  $Q$  is the corresponding query to  $Q_E$  without user-defined constructs
3:  $KS, CF, PK, CN \leftarrow \text{Retrieve } R_p$  // Where  $KS = \text{Keyspace}$ ,  $CF = \text{Column\_Family}$ ,  $PK = \text{Primary Key of CF}$ ,  $CN = \text{Column name in } Q$ 
4: if ( $T1 = \text{instance}$ ) AND ( $T2 = \text{validonnow}$ ) then
5:    $RS \leftarrow \text{Execute } Q \text{ on } CF$  // Where  $RS$  is ResultSet
6: else if ( $T1 = \text{instance}$ ) AND ( $T2 = \text{validondate}$ ) then
7:   Generate  $Q_r$  //  $Q_r$  is a rewritten select query corresponding to  $Q$  with Time  $T2$  to retrieve value of  $CN$  from
   //  $update\_provenance(UP)$  Column Family
8:    $RS \leftarrow \text{Execute } Q_r \text{ on } UP$ 
9:   if  $RS$  is Null then
10:     $RS \leftarrow \text{Execute } Q \text{ on } CF$ 
11:   end if
12: else
13:   Generate  $Q_r$  //  $Q_r$  is a rewritten select query corresponding to  $Q$  with Time  $T2$  to retrieve value of  $CN$  from
   //  $update\_provenance(UP)$  Column Family
14:    $RS \leftarrow \text{Execute } Q_r \text{ on } UP$ 
15:    $RS1 \leftarrow \text{Execute } Q \text{ on } CF$ 
16:    $RS \leftarrow \text{Append } RS1$ 
17: end if
18: Return  $RS$ 
19: End

```

Result Set (RS) of historical data are obtained as an output. In the beginning, Q_E is sent to the Query Parser to retrieve all the UDCs (T1 and T2) used in Q_E along with the CQL Query Q (i.e., CQL query without UDCs) and parsed result (R_p), refer to lines 1 and 2. In addition to this, some other information such as Keyspace Name (KS), Column Family (CF), Primary Key (PK), and Column Name (CN) associated with Q_E is also extracted from R_p , refer to line 3. Now, query Q executes on the related column families to retrieve required historical data as per the following conditions mentioned from lines 4 to 16.

- If UDC T1 and T2 are “instance” and “validon now” type constructs respectively, then query Q executes on the column families mentioned in issued query statement only, refer to lines 4 and 5.
- If UDC T1 and T2 are “instance” and “validon date” type constructs respectively, then the “write time” of current value is first fetched and compared with “validon date”. If the “write time” of current value is lesser than “validon date”, then query Q executes on the column families mentioned in issued query statement only; otherwise, it executes on “update_provenance”, refer to lines 6 to 10.
- If UDC T1 and T2 are “all” and “validon now” type constructs, respectively, then query Q executes on both “update_provenance” and the column families mentioned in issued query statement to retrieve the complete history of all the updates of a column value, refer to lines 13 to 16.
- Similarly, If UDC T1 and T2 are “all” and “validon date” type constructs respectively, then again “write

time” of current value is fetched and compared with “validon date”. If the “write time” of current value is lesser than “validon date”, then query Q executes on both “update_provenance” and the column families mentioned in issued query statement; otherwise, it executes only on “update_provenance”, refer to lines 13 to 16.

Demonstrations of Algorithm 7 with illustrative examples of provenance queries 2, 3, 4 and 5 are given below:

Example Provenance Query 2: Display all the location updates of a specific user named ‘MemeBaaaz’ till now.

Extended CQL Query Q_E : select all location from user_details where screen_name=‘MemeBaaaz’ validon now;

The above Q_E is parsed first to retrieve all the UDCs used in this extended query, i.e., “all” and “validon now”, respectively. Now, CQL query Q is executed on “user_details” and “update_provenance” column families to retrieve all the location updates of the given user “MemeBaaaz”. The query result of above provenance query is shown in Table 2

Example Provenance Query 3: Display all the location updates of a specific user named ‘MemeBaaaz’ till 23/10/2019 9:50AM.

Table 2 Example provenance query 2 result

Location	VALID_FROM
Meme Ki Duniya, India	Wed Oct 02 13:33:27 IST 2019
Kolkata	Wed Oct 23 08:20:18 IST 2019
Mumbai	2019-12-17 10:22:22.0

Table 3 Example provenance query 3 result

Location	Valid_from
Meme Ki Duniya, India	Wed Oct 02 13:33:27 IST 2019
Kolkata	Wed Oct 23 08:20:18 IST 2019

Extended CQL Query Q_E : select all location from user_details where screen_name='MemeBaaaz' validon 2019-10-23 09:50:16.

The query result of above provenance query 3 is shown in Table 3, i.e., all the location updates till '2019-10-23 09:50:16'.

Example Provenance Query 4: Display the current location of a specific user named 'MemeBaaaz'.

Extended CQL Query Q_E : select instance location from user_details where screen_name='MemeBaaaz' validon now.

The above example provenance query 4 generates current location of user as "Mumbai" which is valid from "2019-12-17 10:22: 22.0".

Example Provenance Query 5: Display the location of a specific user named 'MemeBaaaz' on date 23/10/2019 8:22:16AM.

Extended CQL Query Q_E : select instance location from user_details where screen_name='MemeBaaaz' validon 2019-10-23 08:22:16.

The above example provenance query 5 generates location of user on 23/10 /2019 8:22:16AM as "Kolkata" which is valid from "Wed Oct 23 08:20:18 IST 2019" to "2019-12-17 10:22:21.0".

4 Data set and evaluation

To evaluate the performance of proposed framework, all the experiments are performed on a single node Apache Cassandra Cluster on Intel i7-8700 processor @ 3.20GHz with 16GB RAM, and 1TB disk. Apache Cassandra version 3.11.3 has been used for the experiments. In the proposed framework, big social data are fetched from the Twitter's network through live streaming and modelled in Apache Cassandra. This big social data consists of around 2.4 lakh twitter users, 2.1 lakh user's friends, 1.8 lakh user's followers, and their related information such as tweet's body, tweet's id, tweeter's screen name, tweet created date, user's personal information, etc. The proposed key-value pair data model contains a keyspace named "NewTwitter_Keyspace" that consists of 20 Column Families those are used to store this huge volume of social data. On execution of each query, the provenance information is captured and stored in the following three column families, viz. "select_provenance", "update_provenance",

and "query table" that gradually increases the size of database. Java version 8 has been used as front-end programming language to interact with Cassandra, and Twitter's network. Cassandra Query Language (CQL) is used for querying and to communicate with Apache Cassandra. The performance analysis of proposed framework in terms of provenance capturing overhead and provenance query execution time for different query sets including, select, aggregate, data update and provenance queries are presented in the following subsections.

4.1 Provenance capture analysis

To perform an experimental analysis on provenance capture, several query sets of different type of queries including select, aggregate, and data update queries are executed on ZILKVD architecture. A sample set of select queries are shown in Table 4. Initially, all the queries are executed 12 times without provenance support and then, the same set of queries are again executed with provenance support. To calculate the average execution time of each query, we dropped the minimum and the maximum execution time and then taken the average of remaining 10 values. The execution performance of all the select queries in terms of average execution times is shown in Fig. 12. The average execution time of select queries with provenance support is slightly larger than the select queries without provenance support. However, it indicates that the performance overhead of most of the select queries with provenance support is very minimal in respect to the select queries without provenance support, except query

Table 4 Sample select queries

QID	Query
Q1	Find location of user with Screen_Name='Gagan4041'
Q2	Display all tweets by user with screen_name='SunilThalia'
Q3	Display all hashtags used by a user in one tweet
Q4	Display all hashtag used by a user in all tweets posted by a user
Q5	Display all tweets posted by a user on one particular day

Fig. 12 Performance of select queries without and with provenance

Table 5 Sample aggregate queries

QID	Query
Q1	Display total no. of tweets posted by a user on one particular day of a month
Q2	Display total no. of tweets posted by a user in one month
Q3	Display total tweets posted in one specific month
Q4	Display all the users with tweets count in a specific month in descending order
Q5	Display total no. of tweets posted every day of a specific month

Fig. 13 Performance of aggregate queries without and with provenance

Q8. In case of query Q8, a very large number of result tuples generates in its result set which in turns increases the execution time, as the proposed framework captures and stores the provenance for all its result tuples exists in the query result.

The proposed framework also provides the provenance support for aggregate queries with following aggregate functions such as count, max, min, etc. A sample set of aggregate queries are shown in Table 5. The performance analysis of aggregate queries in terms of average execution time with and without provenance support is also shown in Fig. 13. It indicates that the framework efficiently captures provenance for aggregate queries such as query Q1, Q2, and Q4. However, more execution time is measured for those queries in which aggregation is performed on a large number of input tuples such as query Q3, and Q5. For example, let's consider the query Q3, i.e., "count the total number of tweets posted in one month". Here, as the aggregation is performed on all the tweets of that month, which requires to capture the provenance for all such rows those are contributed to generate the result set, as a result it adds some measurable execution overhead.

Provenance capturing for data update queries is also supported by the proposed framework using ZILKVD architecture. A sample set of data update queries are shown in Table 6. The performance analysis of update queries in terms of average execution time with and without provenance support is shown in Fig. 14. It also indicates that the framework efficiently captures provenance for update queries with minimum execution overhead. The captured provenance information for update queries is stored in "update_provenance" col-

Table 6 Sample update queries

QID	Query
Q1	Update location of user with screen_name "DDNewsAndhra"
Q2	Update location of friend named "Ashutosh" of user with screen_name "Bandho"
Q3	Update url of user with screen_name "myshowmytalks"
Q4	Delete user with screen_name "DDNewsAndhra"
Q5	Insert a posted tweet in tweetset

Fig. 14 Performance of update queries without and with provenance

umn family. The following parameters such as "value_type", "old_value", "new_value", "old_value_writetime", and "provenance_path_expression", etc., are used to capture the provenance information. These parameters are further used for historical data queries, and queries executed in the past at any specific time, i.e., standing/historical queries as explained in Sect. 3.4.3.

Ultimately, the overall performance of all types of queries with and without provenance support is shown in Fig. 15. The average query execution time for "update", "select", and "aggregate" queries with and without provenance supports are summarized in Table 7. It indicates that our proposed framework is very efficient in capturing provenance information for "update", and "select" queries, while a very small overhead is measured in case of "aggregate" queries, see Fig. 16.

4.2 Provenance querying analysis

The performance analysis of querying provenance information stored in Apache Cassandra is presented in the following section. A set of different provenance queries are executed for the performance analysis of provenance querying. A sample set of provenance queries are shown in Table 8. Initially, all the provenance queries are executed 12 times. To calculate the average execution time of each query, we dropped the minimum and the maximum execution time and then taken average of remaining 10 values. The execution performance of all the queries is shown in Fig. 17. Average execution times of all the provenance queries are mentioned in milliseconds (ms). According to Fig. 17, the average execution time of provenance queries is varying from 1000ms to 1800ms. It shows that the proposed framework provides support for effi-

Fig. 15 Overall query performance without and with provenance

Table 7 Provenance performance overhead (ms)

	Update queries	Select queries	Aggregate queries
Without provenance	782	788	794
With provenance	838	866	1124

Table 8 Sample provenance queries

QID	Query
Q1	Display all the rows contributed to produce result tuple of query Q2 of Table 5
Q2	Display the row keys of all the rows those are contributed to produce result tuple t1 of query Q1 of Table 4
Q3	Display all location updates of a specific user till now
Q4	Display all location updates of a specific user till time 22/10/2019 8:00AM
Q5	Display the location of a specific user at time 22/10/2019 8:00AM

cient provenance querying for both justifying answers of a query result, and historical data queries at an accepted level of precision.

5 Applications

Proposed framework is beneficial in attempting to understand the social processes and behaviour of a social media user. Some of the application scenarios are given below:

- It is a generalized framework that provides provenance solutions for other social media platforms also. Proposed algorithm can be used in fetching social data from other social networks such as Facebook, Instagram, etc., by using their supporting APIs, for instance Graph APIs for Facebook Social Graph.
- Proposed framework is applicable for such applications where progressive user profile maintenance is required.

Fig. 16 Provenance overhead for different query set

Fig. 17 Querying provenance

For example, a social media user frequently updates his profile by adding, removing or changing his information. In such cases, our framework maintains all the data updates performed without losing any information.

- In current pandemic situation of COVID-19, where health-related data is provided by almost all the countries across the world. This data is valuable, but came in diverse format and scattered at different portals across

the Internet. BSDP can be applied to extract and analyse this data for a better understanding of current situation and in fighting against the COVID-19 pandemic.

6 Conclusion and future work

In this paper, we designed and implemented a Zero-Information Loss Key-Value Pair Database (ZILKVD) on top of which a Big Social Data Provenance (BSDP) Framework has been developed to capture and query provenance for live streamed Twitter data set. The proposed framework is capable to capture fine-grained provenance for various query sets including select, aggregate, and data update queries with insert, delete, and update operations. It also supports to capture provenance for historical/standing queries using data version support in ZILKVD. The proposed ZILKVD architecture and KVP data model leads to an adequate design methodology that provides a flexible provenance management system for social data.

The proposed framework is efficient in terms of average execution time for capturing and storing provenance for select, and data update queries. However, a small execution overhead is measured for some aggregate queries, where the aggregation is performed on a larger number of input tuples. Proposed framework supports efficient provenance querying for both justifying answers of a query result, and historical data queries at an accepted level of precision. Our provenance capturing and querying algorithms prove to be very promising, retrieving more precise information with an optimal latency.

However, our framework has the following limitations. First, proposed framework provides single-layer provenance support (i.e., tracing out direct sources that contributed to a query result) at this stage. Second, currently BSDP framework is implemented for a single node Apache Cassandra rather than for several distributed nodes in a cluster.

In the future, we plan to extend BSDP framework for multi-layer provenance support (i.e., tracing out both direct and indirect sources that contributed to a query result) by using multi-depth provenance querying. We also plan to further extend our framework for a distributed environment where data is redundantly stored across multiple nodes in a cluster.

Declarations

Conflict of Interest On behalf of all authors, the corresponding author states that there is no conflict of interest.

References

1. Agrawal, R., Imran, A., Seay, C., Walker, J.: A layer based architecture for provenance in big data. In: 2014 IEEE International Conference on Big Data (Big Data), pp.1–7. IEEE (2014)
2. Akoush, S., Sohan, R., Hopper, A.: Hadoopprov: towards provenance as a first class citizen in mapreduce. In: Presented as Part of the 5th {USENIX} Workshop on the Theory and Practice of Provenance (2013)
3. Barbier, G., Feng, Z., Gundecha, P., Liu, H.: Provenance data in social media. In: Provenance Data in Social Media (2013)
4. Bhargava, G., Gadia, S.K.: Relational database systems with zero information loss. *IEEE Trans. Knowl. Data Eng.* **5**(1), 76–87 (1993)
5. Cao, L.: Data science: nature and pitfalls. *IEEE Intell. Syst.* **31**(5), 66–75 (2016)
6. Cao, L.: Data science: a comprehensive overview. *ACM Comput. Surv. (CSUR)* **50**(3), 1–42 (2017)
7. Chacko, A., Kumar, S.M.: Big data provenance research directions. In: TENCON 2017-2017 IEEE Region 10 Conference, pp. 651–656. IEEE (2017)
8. Chang, F., Dean, J., Ghemawat, S., Hsieh, W.C., Wallach, D.A., Burrows, M., Chandra, T., Fikes, A., Gruber, R.E.: Bigtable: a distributed storage system for structured data. *ACM Trans. Comput. Syst. (TOCS)* **26**(2), 1–26 (2008)
9. Che, D., Safran, M., Peng, Z.: From big data to big data mining: challenges, issues, and opportunities. In: International Conference on Database Systems for Advanced Applications, pp. 1–15. Springer (2013)
10. Cheah, Y.W., Canon, R., Plale, B., Ramakrishnan, L.: Milieu: lightweight and configurable big data provenance for science. In: 2013 IEEE International Congress on Big Data, pp. 46–53. IEEE (2013)
11. Chebotko, A., Kashlev, A., Lu, S.: A big data modeling methodology for apache cassandra. In: 2015 IEEE International Congress on Big Data, pp. 238–245. IEEE (2015)
12. Corsar, D., Markovic, M., Edwards, P.: Social media data in research: provenance challenges. In: International Provenance and Annotation Workshop, pp. 195–198. Springer (2016)
13. Crawl, D., Wang, J., Altintas, I.: Provenance for mapreduce-based data-intensive workflows. In: Proceedings of the 6th Workshop on Workflows in Support of Large-scale Science, pp. 21–30 (2011)
14. Cuzzocrea, A.: Provenance research issues and challenges in the big data era. In: 2015 IEEE 39th Annual Computer Software and Applications Conference, vol. 3, pp. 684–686. IEEE (2015)
15. Cuzzocrea, A.M.: Big data provenance: State-of-the-art analysis and emerging research challenges. In: Workshops of the EDBT/ICDT 2016 Joint Conference, EDBT/ICDT 2016, CEUR-WS, vol. 1558 (2016)
16. De Nies, T., Taxidou, I., Dimou, A., Verborgh, R., Fischer, P.M., Mannens, E., Van de Walle, R.: Towards multi-level provenance reconstruction of information diffusion on social media. In: Proceedings of the 24th ACM International on Conference on Information and Knowledge Management, pp. 1823–1826 (2015)
17. DeCandia, G., Hastorun, D., Jampani, M., Kakulapati, G., Lakshman, A., Pilchin, A., Sivasubramanian, S., Voshall, P., Vogels, W.: Dynamo: Amazon's highly available key-value store. *ACM SIGOPS Oper. Syst. Rev.* **41**(6), 205–220 (2007)
18. Fathollahi-Fard, A.M., Hajiaghahi-Keshteli, M., Tavakkoli-Moghaddam, R.: The social engineering optimizer (seo). *Eng. Appl. Artif. Intell.* **72**, 267–293 (2018)
19. Fathollahi-Fard, A.M., Ranjbar-Bourani, M., Cheikhrouhou, N., Hajiaghahi-Keshteli, M.: Novel modifications of social engineering optimizer to solve a truck scheduling problem in a cross-docking system. *Comput. Ind. Eng.* **137**, 106103 (2019)

20. Featherston, D.: *Cassandra: Principles and Application*. Department of Computer Science University of Illinois at Urbana-champaign (2010)
21. Feng, Z., Gundecha, P., Liu, H.: *Social Provenance*, pp. 2768–2772. Springer, New York (2018)
22. Ghoshal, D., Plale, B.: Provenance from log files: a bigdata problem. In: *Proceedings of the Joint EDBT/ICDT 2013 Workshops*, pp. 290–297 (2013)
23. Glavic, B.: Big data provenance: challenges and implications for benchmarking. In: *Specifying Big Data Benchmarks*, pp. 72–80. Springer (2012)
24. Glavic, B., Miller, R.J.: Reexamining some holy grails of data provenance. *TaPP* 11:3rd(2011)
25. Gundecha, P., Feng, Z., Liu, H.: Seeking provenance of information using social media. In: *Proceedings of the 22nd ACM International Conference on Information and Knowledge Management*, pp. 1691–1696 (2013)
26. Hernandez, R., Becerra, Y., Torres, J., Ayguadé, E.: Automatic query driven data modelling in cassandra. *Procedia Comput. Sci.* **51**, 2822–2826 (2015)
27. Hondo, F., Wercelens, P., da Silva, W., Castro, K., Santana, I., Walter, M.E., Araújo, A., Holanda, M., Lifschitz, S.: Data provenance management for bioinformatics workflows using nosql database systems in a cloud computing environment. In: *2017 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, pp. 1929–1934. IEEE (2017)
28. Ikeda, R., Park, H., Widom, J.: Provenance for generalized map and reduce workflows (2011)
29. Kaplan, A.M., Haenlein, M.: Users of the world, unite! the challenges and opportunities of social media. *Bus. Horizons* **53**(1), 59–68 (2010)
30. Kerchner, D., Littman, J., Peterson, C., Smallen, V., Trent, R., Wrubel, L.: *The Provenance of a Tweet* (2019)
31. Kulkarni, D.: A fine-grained access control model for key-value systems. In: *Proceedings of the Third ACM Conference on Data and Application Security and Privacy*, pp. 161–164 (2013a)
32. Kulkarni, D.: A provenance model for key-value systems. In: *Presented as Part of the 5th {USENIX} Workshop on the Theory and Practice of Provenance* (2013b)
33. Lakshman, A., Malik, P.: *Cassandra: a decentralized structured storage system*. *ACM SIGOPS Oper. Syst. Rev.* **44**(2), 35–40 (2010)
34. Li, F., Zhao, Y., Felsche, K., Xu, G., Cao, L.: *Coupling Analysis Between Twitter and Call Centre* (2015). *arXiv preprint arXiv:1509.02238*
35. Mahmood, K.: Performance comparison of nosql database cassandra and sql server for large databases. *J. Independ. Stud. Res. (JISR)* **14**(2) (2016)
36. Markovic, M., Edwards, P., Corsar, D.: A role for provenance in social computation. In: *Proceedings of the First International Workshop on Crowdsourcing the Semantic Web-CrowdSem 2013, CEUR-WS* (2013)
37. Olshannikova, E., Olsson, T., Huhtamäki, J., Kärkkäinen, H.: Conceptualizing big social data. *J. Big Data* **4**(1), 1–19 (2017)
38. Papavasileiou, V., Yocum, K., Deutsch, A.: Ariadne: Online provenance for big graph analytics. In: *Proceedings of the 2019 International Conference on Management of Data*, pp. 521–536 (2019)
39. Park, H., Ikeda, R., Widom, J.: Ramp: a system for capturing and tracing provenance in mapreduce workflows. *Proc. VLDB Endow.* **4**(12), 1351–1354 (2011)
40. Ramesh, D., Kumar, A.: Query driven implementation of twitter base using cassandra. In: *2018 International Conference on Current Trends towards Converging Technologies (ICCTCT)*, pp. 1–4. IEEE (2018)
41. Ramusat, Y., Maniu, S., Senellart, P.: Semiring provenance over graph databases. In: *10th {USENIX} Workshop on the Theory and Practice of Provenance (TaPP 2018)* (2018)
42. Ranganath, S., Gundecha, P., Liu, H.: A tool for assisting provenance search in social media. In: *Proceedings of the 22nd ACM International Conference on Information and Knowledge Management*, pp. 2517–2520 (2013)
43. Rani, A., Goyal, N., Gadia, S.K.: Data provenance for historical queries in relational database. In: *Proceedings of the 8th Annual ACM India Conference*, pp. 117–122 (2015)
44. Rani, A., Goyal, N., Gadia, S.K.: Efficient multi-depth querying on provenance of relational queries using graph database. In: *Proceedings of the 9th Annual ACM India Conference*, pp. 11–20 (2016)
45. Rani, A., Goyal, N., Gadia, S.K.: Twitter data modelling and provenance support for key-value pair databases. In: Qiao, M., Vossen, G., Wang, S., Li, L. (eds.) *Databases Theory and Applications*, pp. 87–98. Springer, Cham (2021a)
46. Rani, A., Goyal, N., K Gadia, S.: Provenance framework for twitter data using zero-information loss graph database. In: *8th ACM IKDD CODS and 26th COMAD*, pp. 74–82 (2021b)
47. Rodrigues, A.P., Chiplunkar, N.N.: Real-time twitter data analysis using hadoop ecosystem. *Cogent Eng.* **5**(1), 1534519 (2018)
48. Schmidt, F.M., Geyer, C., Schaeffer-Filho, A., DeBloch, S., Hu, Y.: Change data capture in nosql databases: a functional and performance comparison. In: *2015 IEEE Symposium on Computers and Communication (ISCC)*, pp. 562–567. IEEE (2015)
49. Senellart, P.: Provenance in databases: principles and applications. In: *Reasoning Web, Explainable Artificial Intelligence*, pp. 104–109. Springer, Cham (2019)
50. Simmhan, Y.L., Plale, B., Gannon, D.: *A Survey of Data Provenance Techniques*, vol. 47405, p. 69. Computer Science Department, Indiana University, Bloomington (2005)
51. Taxisidou, I., De Nies, T., Verborgh, R., Fischer, P.M., Mannens, E., Van de Walle, R.: Modeling information diffusion in social media as provenance with w3c prov. In: *Proceedings of the 24th International Conference on World Wide Web*, pp. 819–824 (2015)
52. Taxisidou, I., Lieber, S., Fischer, P.M., De Nies, T., Verborgh, R.: Web-scale provenance reconstruction of implicit information diffusion on social media. *Distrib. Parallel Datab.* **36**(1), 47–79 (2018)
53. Wang, J., Crawl, D., Purawat, S., Nguyen, M., Altintas, I.: Big data provenance: challenges, state of the art and opportunities. In: *2015 IEEE International Conference on Big Data (Big Data)*, pp. 2509–2516. IEEE (2015)
54. Yazici, I.M., Aktas, M.S., Gokturk, M.: A novel approach to user involved big data provenance visualization. *DBKDA* **2017**, 19 (2017)
55. Ye, Q., Lu, M.: s2p: provenance research for stream processing system. *Appl. Sci.* **11**(12), 5523 (2021)
56. Zhang, C., Fathollahi-Fard, A.M., Li, J., Tian, G., Zhang, T.: Disassembly sequence planning for intelligent manufacturing using social engineering optimizer. *Symmetry* **13**(4), 663 (2021)