Temporal Data Management and Processing with Column Oriented NoSQL Databases

Yong Hu, Group of Heterogeneous Information Systems, University of Kaiserslautern, Germany
Stefan Dessloch, Group of Heterogeneous Information Systems, University of Kaiserslautern, Germany

ABSTRACT

This article introduces how temporal data can be maintained and processed by utilizing Column-oriented NoSQL databases (CoNoSQLDBs). Although each column in a CoNoSQLDB can store multiple data versions with their corresponded timestamps, its implicit temporal interval representation can cause wrong or misleading results during temporal query processing. In consequence, the original table representation supported by CoNoSQLDBs is not suitable for storing temporal data. To maintain the temporal data in the CoNoSQLDB tables, two alternative table representations can be adopted, namely, explicit history representation (EHR) and tuple time-stamping representation (TTR) in which each tuple (data version) has an explicit temporal interval. For processing TTR, the temporal relational algebra is extended to TTRO operator model with minor modifications. For processing EHR, a novel temporal operator model called CTO is proposed. Both TTRO and CTO contain eight temporal data operators, namely, Union, Difference, Intersection, Project, Filter, Cartesian product, Theta-Join and Group by with a set of aggregation functions, such as SUM, AVG, MAX and etc. Moreover, the authors implement each temporal operator by utilizing MapReduce framework to indicate which temporal operator model is more suitable for temporal data processing in the context of CoNoSQLDBs.

Keywords: Attribute-Timestamping Model, Column-oriented NoSQL Databases, Temporal Aggregations, Temporal Data Model, Temporal Operators, Tuple-Timestamping Model

INTRODUCTION

The importance of temporal data management and processing is acknowledged by both industry and academia, e.g. the fraud detecting and tracing systems built for large online shopping systems such as Amazon and Ebay or business intelligence tools such as data warehousing and OLAP systems which allow users to store, retrieve and analyze the historical data for predicating future trend.

DOI: 10.4018/JDM.2015070103
Recently, a new type of data storage system called “Column-oriented NoSQL” database (CoNoSQLDB) has emerged. A CoNoSQLDB manages data in a structured way and stores the data which belongs to the same “column” contiguously on disk. Each tuple in a CoNoSQLDB is uniquely identified and distributed based on its row-key value. In contrast to relational database systems (RDBMSs), each column in the CoNoSQLDBs stores multiple data versions which are sorted by their corresponding timestamps. Moreover, the duration between two timestamps forms an implicit temporal interval which denotes how long a data version is valid. Well-known examples are “BigTable” (Chang, Dean & et al., 2006), which was proposed by Google in 2004 and its open-sourced counterpart “HBase” (Apache HBase).

To consume and analyze the data stored in CoNoSQLDBs, users can either write low-level programs such as the MapReduce (Dean & Ghemawat, 2004) procedures or utilize high-level languages such as Pig Latin (Apache Pig Latin) or Hive (Apache Hive). MapReduce is a parallel computing framework in which users code the desired data processing tasks in Map and Reduce functions and the framework takes charge of data partitioning, parallel task scheduling and execution and fault tolerance. Although this approach gives users enough flexibility, it imposes programming requirements and restricts optimization opportunity (as the MapReduce framework does not understand the semantics embodied in the Map and Reduce functions). Moreover, it forces manual coding of query processing logic and reduces program reusability.

Pig Latin and Hive are two high-level languages which are built on top of the MapReduce framework, where each of them includes various predefined operators. To analyze the data in a CoNoSQLDB, clients first utilize the built-in load function (specifically for CoNoSQLDBs) and denote queries either by a set of high-level operators (Pig Latin) or SQL-like scripts (Hive). Although this approach facilitates the query definition, the built-in load function will transform a CoNoSQLDB table into a first-normal-form (1NF) (Codd, 1970) by purely loading the latest data values (without containing its timestamp) and discarding the older versions. If users wish to load multiple data versions, a customized load function has to be hand-coded. Each column will then have a “set” type instead of atomic values. For example, when implementing the customized load function in Pig Latin, each column is indicated as “bag” (multi-set) type and each data version is represented as an element (type “tuple” in Pig Latin) which is further decomposed as a pair (atomic-value, timestamp). Generally, this type of table is called non-first-normal-form (NF2) (Makinouchi, 1977) or nested relations. To process NF2 in Pig Latin or Hive, users need to first flatten the nested relation to 1NF, then apply the desired data processing based on the predefined high-level operators and finally nest the 1NF relation to rebuild the nested relation. However, this approach has several pitfalls: 1) as the data volume of CoNoSQLDB is usually massive, the table reconstructing operations can heavily decrease the performance and exhaust the hardware resources; 2) the predefined high-level operators are traditional relational operators which handle only the data values without considering any temporal information. For example, to specify a temporal join processing, besides evaluating the join predicates, users also need to explicitly add a select condition to test whether two joining tuples are valid during the same period of time.

As already pointed out by the temporal database community, querying temporal data by utilizing the non-temporal operators (e.g. Pig Latin or Hive) or writing customized procedures (e.g. MapReduce programs) is usually error-prone and cumbersome. In consequence, in this paper, we study the issues of defining temporal operators for CoNoSQLDBs and several significant aspects need to be taken into account:

1. As each data version maintains a timestamp, what is its meaning in the context of CoNoSQLDBs, i.e. should it be understood as valid time or transaction time?
Accelerating Web Service Workflow Execution via Intelligent Allocation of Services to Servers
[www.igi-global.com/article/accelerating-web-service-workflow-execution/47420?camid=4v1a](www.igi-global.com/article/accelerating-web-service-workflow-execution/47420?camid=4v1a)

Managing Temporal Data
[www.igi-global.com/chapter/managing-temporal-data/20685?camid=4v1a](www.igi-global.com/chapter/managing-temporal-data/20685?camid=4v1a)

Understanding Functional Dependency
[www.igi-global.com/chapter/understanding-functional-dependency/9216?camid=4v1a](www.igi-global.com/chapter/understanding-functional-dependency/9216?camid=4v1a)