An Efficient NoSQL-Based Storage Schema for Large-Scale Time Series Data

Ruizhe Ma, University of Massachusetts, Lowell, USA

Weiwei Zhou, Nanjing University of Aeronautics and Astronautics, China

Zongmin Ma, Nanjing University of Aeronautics and Astronautics, China*

iD https://orcid.org/0000-0001-7780-6473

ABSTRACT

In IoT (internet of things), most data from the connected devices change with time and have sampling intervals, which are called time-series data. It is challenging to design a time series storage model that can write massive time-series data in a short time and can query and analyze the persistent time-series data for a long time. This paper constructs the RHTSDB (Redis-HBase Time Series Database) storage model based on Redis and HBase. RHTSDB uses the memory database Redis (Remote Dictionary Server) to cache massive time-series data, providing efficient data storage and query functions. HBase is used in RHTSDB for long-term storage of time-series data to realize their persistence. The paper designs a cold and hot separation mechanism for time-series data, where the infrequently accessed cold data are stored in HBase, and the frequently accessed and latest data are stored in Redis. Experiments verify that RHTSDB has apparent advantages over Apache IoTDB and HBase in data intake and query efficiency.

KEYWORDS

HBase, Query, Redis, Storage, Time-Series Data

INTRODUCTION

With the development of the Internet of Things (IoT) (Eom & Lee, 2017), the amount of time-series data has shown explosive growth. Time-series data refers to a sequence of data points collected at fixed time intervals (Lee & Chung, 2014). Each data point is associated with a timestamp that indicates the generation time of the corresponding data. Typically, the data collected by a sensor in a particular period can be expressed as a time series $[(t_1, v_1), (t_2, v_2), ..., (t_n, v_n)]$, where v_i refers to the value collected at t_i time (Di Martino *et al*., 2019). Of course, complete time-series data can include the collection time and collection value as well as the source description information of the current collection value. For example, we need to include some measurement data information, such as the names of collection subject and collection index. Comprehensive use cases in the real world have generated a large amount of measurement data from millions or billions of different sources. Slack

collects measurement data from 4 billion unique sources at 12 million samples per second daily, for example, generating up to 12 TB of compressed data daily. It is essential to manage and process a large amount of time-series data efficiently. Unfortunately, many off-the-shelf systems cannot scale to support these workloads, which leads to the random Patchwork and vulnerability of customized solutions (Solleza, Crotty, Karumuri, Tatbul & Zdonik, 2022). For this reason, diverse time series databases are proposed to ensure efficient ingestion performance and save storage space as much as possible. Given that time-series data in applications are generally massive and redundant data containing source description information in time-series data are enormous, efficient storage and query of massive time-series data is challenging.

We identify two major categories of time series databases: which are respectively called *native time series databases* and *common time series databases* in this paper. The native time series databases are the storage systems that are developed especially for time-series data management according to their structural and usage characteristics, such as InfluxDB[1], FluteDB (Li *et al*., 2018), and Apache IoTDB (Wang *et al*., 2020 & 2023). This category of time series databases can efficiently reduce the overhead of storage space and the query delay. However, for time-series data management and processing, many other functions and operations are essential in time series databases, such as flexible aggregation, data retention, multidimensional range query, among others. While the native time series databases cannot provide full support to time-series data analysis well, mature database systems are good at dealing with relationships between data and support many unnecessary operations and guarantees for time series, increasing inefficiency and unnecessary complexity (Shafer, Sambasivan, Rowe, & Ganger, 2013). The common time series databases are the storage systems that directly apply the common databases for storing and processing time-series data. Depending on what types of databases are applied, we further identify two categories of common time series databases. The first one uses relational databases as the back end of common time series databases (e.g., (Rhea *et al*., 2017)). In recent years, NoSQL (Not only SQL) databases have attracted increasing attention from both academia and industry (Hu & Dessloch, 2015), which offer flexible data representation models and horizontal hardware scalability so that Big Data can be processed in real time (Bajaj & Bick, 2020). The second category of common time series databases uses NoSQL databases for processing time-series data (Di Martino *et al*., 2019).

NoSQL databases contain four major types of database models: *key-value stores*, *column-family stores*, *document stores*, and *graph stores* (Grolinger *et al*., 2013; Van Erven *et al*., 2019). Different types of NoSQL databases, say Redis[2] (a key-value store), HBase[3] (a column-family store), Cassandra[4] (a column-family store), MongoDB[5] (a document store), Couchbase[6] (a document store), OrientDB[7] (a graph store), have very different performances (Matallah, Belalem & Bouamrane, 2020). Among these NoSQL databases, for example, Redis (remote dictionary server) is a high-performance memory-based NoSQL database, which has excellent data writing performance and supports data persistent storage and replication of master-slave nodes (Zhou, Lu, Zhang & Qi, 2020); HBase, an open-source, distributed and versioned NoSQL database, uses Hadoop distributed file system (HDFS) to provide distributed file storage services, which has the characteristics of high availability, robust scalability and the ability to store massive data. With one of NoSQL databases as back end, some storage models for massive time-series data have been developed, for example, ModelarDB+ (Jensen, Pedersen & Thomsen, 2021) based on Cassandra, NagareDB (Calatrava, Fontal, Cucchietti & Diví-Cuesta, 2021) built on top of MongoDB, OpenTSDB[8] based on HBase, and KairosDB[9] based on Cassandra.

Selecting a specific NoSQL database for massive time-series data management is more flexible and cost-saving, where the performance of the storage model is determined by the performance of the used database models (Rinaldi *et al*., 2019). It has been demonstrated that different NoSQL databases have very different performances (Matallah, Belalem & Bouamrane, 2020). For ingestion of large-scale time-series data into the target time series database, for example, the storage consumption of HBase slowly increases along with a significant increase in data size (e.g., a 50% increase in data size requires about a 20% increase in storage consumption). However, the storage consumption of

Related Content

Emotional and Rational Components in Software Testing Service Evaluation: Antecedents and Impacts

Colin G. Onita, Jasbir S. Dhaliwaland Xihui Zhang (2022). *Journal of Database Management (pp. 1-39).*

www.irma-international.org/article/emotional-and-rational-components-in-software-testing-service-evaluation/313969

Improving Sequence Diagram Modeling Performance: A Technique Based on Chunking, Ordering, and Patterning

Thant Synand Dinesh Batra (2013). *Journal of Database Management (pp. 1-25).*

www.irma-international.org/article/improving-sequence-diagram-modeling-performance/100404

An Empirical Analysis of the Object-Oriented Database Concurrency Control Mechanism O2C2

David Olsenand Sudha Ram (1999). *Journal of Database Management (pp. 14-26).*

www.irma-international.org/article/empirical-analysis-object-oriented-database/51215

Sequential Pattern Mining from Sequential Data

Shigeaki Sakurai (2009). *Handbook of Research on Innovations in Database Technologies and Applications: Current and Future Trends (pp. 622-631).*

www.irma-international.org/chapter/sequential-pattern-mining-sequential-data/20748

Bridging Relational and NoSQL Worlds: Case Study

 (2018). *Bridging Relational and NoSQL Databases (pp. 282-310).*

www.irma-international.org/chapter/bridging-relational-and-nosql-worlds/191986