Distributed NoSQL Triple Store with Conflict Resolution and Data Consistency

SHISHIR SHAHI, Internation Institute of Information Technology Bangalore, India
VINEET PRIYEDARSHI, Internation Institute of Information Technology Bangalore, India
AKSHAY PANDEY, Internation Institute of Information Technology Bangalore, India
KESHAV GOYAL, Internation Institute of Information Technology Bangalore, India

A clear and well-documented LaTeX document is presented as an article formatted for publication by ACM in a conference proceedings or journal publication. Based on the "acmart" document class, this article presents and explains many of the common variations, as well as many of the formatting elements an author may use in the preparation of the documentation of their work.

ACM Reference Format:
Shishir Shahi, Vineet Priyedarshi, Akshay Pandey, and Keshav Goyal. 2018. Distributed NoSQL Triple Store with Conflict Resolution and Data Consistency. *ACM Trans. Graph.* 37, 4, Article 111 (August 2018), 5 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

1.1 Overview

In the era of big data, managing and querying large-scale tsv datasets efficiently has become increasingly challenging. Triple stores serve as the backbone for storing and querying tsv data, which comprises subject-predicate-object triples. This project addresses the need for a scalable and distributed solution by developing a prototype of a distributed NoSQL triple store.

The primary objective of this project is to design and implement a distributed NoSQL triple store capable of handling massive tsv datasets. The prototype utilizes a client-server model, where each server acts as both a client and a server to provide specific services to users. By leveraging state-based objects, the system ensures efficient management of tsv triples across multiple servers.

1.2 Significance of the Problem Statement

Scalability: As data volumes continue to grow, scalable solutions like NoSQL databases become essential for handling large datasets and ensuring system performance.

Data Integrity: Maintaining the integrity of triple data and ensuring consistency across distributed systems is critical.

Conflict Resolution: Effective conflict resolution mechanisms are crucial for handling concurrent updates and maintaining data coherence in distributed environments.

Flexibility and Interoperability: The project's emphasis on utilizing distinct frameworks and supporting synchronization between different systems promotes flexibility and interoperability, enabling seamless integration with existing infrastructure.

1.3 Overview of the project objectives and scope

Design and Implementation of Distributed NoSQL Triple Store: Develop a prototype system capable of storing, querying, updating, and synchronizing triples across multiple servers using state-based objects.

Utilization of Diverse Data Processing Frameworks: Implement distinct frameworks within each server, prioritizing those covered in class, to showcase flexibility and integration capabilities.

Conflict Resolution Mechanisms: Incorporate strategies for conflict resolution during merge operations, ensuring data consistency and integrity across distributed systems.

Documentation and Testing: Provide comprehensive documentation detailing system architecture, implementation details, and testing methodologies to ensure reliability and reproducibility. This report delves into the intricacies of the system architecture, methodology, implementation details, testing approach, and future directions. Through a comprehensive analysis, we aim to showcase the feasibility and potential of the proposed solution in addressing the challenges of managing tsv data in distributed environments.

2 DATA STRUCTURE

In our distributed NoSQL triple store, we utilize specific data structures for storing tuples (triples) and logs for the following reasons:

2.1 Data Structure for Storing Tuples (Triples)

2.1.1 *MongoDB Structure for Tuple Storage.*

- MongoDB stores tuples as documents in collections.
- Each document contains fields for subject, predicate, object, and timestamp.
- This structure facilitates efficient storage and retrieval of triple data in a NoSQL database.

```
{
  "subject": "Alice",
  "predicate": "likes",
  "object": "ice␣cream",
  "timestamp": ISODate("2023-05-31T14:30:00Z")
}
```

Authors' Contact Information: Shishir Shahi, Internation Institute of Information Technology Bangalore, Bengaluru, India, shishir.shahi@iiitb.ac.in; Vineet Priyedarshi, Internation Institute of Information Technology Bangalore, Bengaluru, India, vineet.priyedarshi.ac.in; Akshay Pandey, Internation Institute of Information Technology Bangalore, Bengaluru, India, akshay.pandey@iiitb.ac.in; Keshav Goyal, Internation Institute of Information Technology Bangalore, Bengaluru, India, keshav.goyal560@iiitb.ac.in.

ACM 1557-7368/2018/8-ART111
https://doi.org/XXXXXXX.XXXXXXX

2.1.2 *MySQL Structure for Storing Tuples(Triples).*

- MySQL uses a relational table structure to store tuples.
- The table has columns for subject, predicate, object, and timestamp.
- This structure ensures data integrity and supports SQL queries for tuple operations in a relational database system.

```
CREATE TABLE triples (
    subject VARCHAR(255),
    object VARCHAR(255),
    predicate VARCHAR(255),
    timestamp TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    PRIMARY KEY (subject, object)
);
```

2.1.3 *Hive Structure for Storing Tuples(Triples).*

- Hive uses HiveQL to define tables for storing triple values. The table schema includes columns for the triple components:

```
subject STRING,
predicate STRING,
object STRING,
timestp BIGINT
```

2.2 Data Structure for Communicating between different Systems

The "log" and "merge log" play crucial roles in facilitating communication and synchronization between different systems in a distributed NoSQL triple store:

2.2.1 *Data Structure for Logs.*

- The log maintains a record of all operations (such as updates) performed on the tuples (triples) within a specific system/server.
- When communication between systems is required, the log acts as a collection of changes made to the data.
- During synchronization, each system can refer to its log to identify the latest updates or modifications made to tuples.
- By comparing logs between systems, inconsistencies or differences in data can be identified and resolved.

2.2.2 *Merge Log.*

- The merge log specifically tracks merge events between different systems in the distributed environment.
- When a system initiates a merge operation with another system, the merge log records details such as server IDs and timestamps.
- During synchronization or merge operations, systems can refer to the merge log to determine the order of merge events and ensure chronological consistency.
- The merge log helps in resolving conflicts that may arise when merging data from multiple systems, ensuring data integrity and consistency across the distributed environment.

3 SYSTEM ARCHITECTURE

3.1 Description of Architecture:

The prototype comprises multiple servers, each hosting a distinct NoSQL database management system (DBMS) such as MongoDB and MySQL. These servers act as independent nodes capable of storing and managing triples (subject-predicate-object entities) within their respective databases.

The merge operations and conflict resolution logic within each server handle synchronization events. The merge method in the code facilitates communication and synchronization between servers by resolving conflicts based on timestamps and updating triple values accordingly.

The client interface interacts with the servers through function calls such as query, update, and merge. Users can access these services to perform operations on triples, initiate merge events, and fetch logs for auditing purposes.

3.2 Utilization of State-Based Objects:

State-based objects are represented by the Triplet class, which encapsulates the state of a triple (subject, predicate, object) along with a timestamp. Each triple in the system is managed as a state-based object, enabling efficient tracking of changes and timestamps.

The update method in MongoDBTripleStore, MySQLTripleStore and HiveTripleStore classes modifies state-based objects (triples) with new values and timestamps when updates are performed. This ensures that changes to triples are accurately recorded and tracked.

During merge operations, the merge method compares timestamps of state-based objects to resolve conflicts. If a newer update is detected, the system updates the triple with the latest value and timestamp, ensuring data consistency and integrity across distributed servers.

3.3 Client-Server Model and Services Offered:

The client interacts with server objects such as MongoDBTripleStore, HiveTripleStore and MySQLTripleStore to access services.

Users, through the client interface, can access services such as querying triples (query method), updating triple values (update method), initiating merge operations (merge method), and fetching logs (fetch_logs method) for auditing purposes. These services facilitate efficient management and synchronization of triples across the distributed NoSQL triple store prototype.

In summary, the architecture leverages state-based objects, server components, and client-server interactions to provide scalable, efficient, and consistent triple management services within a distributed NoSQL environment.

4 METHODOLOGY

4.1 Methods for State-Based Objects:

Query Method: Implemented to retrieve data based on specified subject, predicate, or object criteria from the distributed triple store. Utilizes SQL queries for SQL-based databases (e.g., MySQL, Hive) and MongoDB queries for MongoDB-based storage.

Update Method: Facilitates data modification or insertion operations in the triple store. Incorporates update queries for SQL databases and MongoDB update operations for MongoDB stores. Includes functionality for handling timestamps and maintaining data integrity.

Merge Method: Designed to merge data from multiple servers or systems, ensuring consistency and resolving conflicts. Utilizes merge logic to update existing data or insert new data based on timestamp comparisons or other criteria.

4.2 Programming Language and Framework Selection:

Python chosen as the primary programming language due to its versatility, ease of use, extensive libraries (e.g., pymongo, mysql.connector), and compatibility with both SQL and NoSQL databases.

For the frameworks:

- PyHive: Utilized for integrating Hive with Python, enabling HiveQL queries and commands within the Python codebase for interacting with Hive tables and executing Hive-specific operations.
- pymongo, mysql.connector: Python libraries used for connecting to MongoDB and MySQL databases, respectively, allowing seamless communication and data manipulation.

4.3 Server and Framework Decision-making:

Selecting three servers can offer several advantages in terms of performance, fault tolerance, and scalability. Three servers provide a scalable infrastructure that can accommodate growing data volumes and user requests. Scaling horizontally by adding more servers becomes easier, allowing the system to handle increased workload and expand storage capacity as needed.

With three servers, the system can withstand the failure of one server without experiencing significant downtime or data loss.

Redundancy in data storage and processing ensures high availability and reliability, reducing the impact of hardware or software failures.

NoSQL Triple Store (MongoDB) is selected for its flexibility, scalability, and support for storing triple data in a document-oriented format.

SQL Triple Store (MySQL) is chosen for its relational data model, SQL querying capabilities, and transaction support, suitable for structured data storage and complex query processing.

Hive is Integrated for distributed storage, structured querying (using HiveQL), and compatibility with SQL-based operations. Enables seamless integration with the existing Python-based methodology and enhances data processing capabilities.

5 IMPLEMENTATION

5.1 MongoDBTripleStore

1. `__init__`: Establishes a connection to MongoDB and initializes the collections required for storing triples and logs.
2. `query`: Retrieves triples from the MongoDB collection based on a given subject and predicate.
3. `update`: Updates or inserts triples into the MongoDB collection while also logging the update operation in a separate log collection.
4. `fetch_logs`: Gathers log entries since the last merge timestamp for a specific server, facilitating synchronization of data between servers.
5. `merge`: Integrates log entries from another server into the main MongoDB collection, ensuring data consistency across distributed systems.

5.2 MySQLTripleStore

1. `__init__`: Initializes a connection to MySQL, enabling interaction with the database.
2. `query`: Fetches triples from the MySQL table based on specified subject and predicate criteria.
3. `fetch_logs`: Retrieves log entries since the last merge timestamp for a designated server, essential for maintaining synchronized data across servers.
4. `update`: Modifies or adds triples to the MySQL table while concurrently logging the update operation in a separate log table.
5. `merge`: Combines log entries from another server into the primary MySQL table, ensuring that all servers have consistent data.

5.3 HiveTripleStore

1. `__init__`: Sets up a connection to Hive using the Beeline command-line tool, enabling interaction with the Hive database.
2. `run_command`: Executes a given query string using the Beeline command and captures the results, facilitating communication with the Hive database.
3. `query`: Retrieves triples from the Hive table based on a specified subject, enabling data retrieval for analysis or processing.
4. `update`: Modifies or adds triples to the Hive table, ensuring that the data remains accurate and up-to-date.
5. `update_with_timestamp`: Inserts or updates triples in the Hive table with a provided timestamp, allowing for precise data management based on specific time points.
6. `update_if_older`: Updates or inserts triples into the Hive table only if the provided timestamp is older than existing entries, ensuring data integrity and preventing conflicts.
7. `fetch_logs`: Gathers log entries since the last merge timestamp for a particular server, facilitating synchronization of data across distributed systems.
8. `merge`: Integrates log entries from another server into the primary Hive table, ensuring that data across servers remains consistent and synchronized.

5.4 Log Sharding and Management

- **Purpose of Log Sharding:** Sharding the log tables helps manage log data efficiently, especially in scenarios where the volume of log entries is substantial. By distributing log entries across multiple tables, we can prevent individual log tables from becoming too large, which could impact performance and scalability.

- **Implementation Details:** - Each log table is sharded based on a predefined maximum limit (‘self.lim‘). - When adding a new log entry, the system checks the size of the current log table. If it exceeds the maximum limit, a new log table is created, and the entry is added to the new table. - A new column is added to the main table to track which log table each entry belongs to. This allows for efficient retrieval and management of log data during merge operations. - When updating an existing subject-predicate tuple, the system checks if the entry exists in any sharded log table. If it does, the entry is removed from the corresponding log table before being updated in the main table. - This approach ensures that log data is distributed across multiple tables, preventing any single table from becoming a bottleneck.

Overall, the implementation of log sharding enhances the scalability and performance of the distributed NoSQL triple store. By effectively managing log data, the system can handle large volumes of updates and merge operations without experiencing significant performance degradation. In the future, we plan to further optimize the log management process and explore additional strategies for improving data synchronization and conflict resolution in distributed environments.

6 TESTING

In our testing methodology, we strategically partitioned the extensive Yago dataset into multiple subsets, each containing approximately 10^4 entities. By analyzing subsets of this size, we aimed to strike a balance between computational efficiency and dataset coverage, ensuring that our experiments captured meaningful insights while remaining tractable in terms of computational resources. Through iterative testing and analysis of these subsets, we gained valuable insights into the dataset's characteristics, uncovering patterns, anomalies, and trends that contribute to a deeper understanding of its underlying structure and content. This approach allowed us to effectively explore the vast Yago dataset and derive meaningful conclusions.

During testing we also created a new function called UpdateMergeLog() which updates the mergeLog table whenever two servers merge. It updates the serverID and timestamp on both server's mergelog table.

7 ADVANCED FEATURES

We will clear the log table if all the server are in sync and no data is left to be committed.

Clearing the log table upon completion of the merger operation and achieving synchronization in a distributed database system offers several benefits. It optimizes storage usage by freeing up space occupied by potentially large log data, thereby enhancing database performance and reducing storage costs. Simplified maintenance is achieved as administrators no longer need to manage the log table separately, streamlining workflows. Enhanced security is ensured by minimizing the risk of unauthorized access to sensitive operation data stored in the log table. Additionally, clarity and simplicity are provided as the deletion of the log table signifies successful synchronization and maintains a clean database structure. This approach also reduces redundancy, improves scalability, and ensures that only essential data is retained, contributing to the overall efficiency and effectiveness of the distributed database system.

8 USER INTERFACE

The user interface developed for the prototype is a terminal-based interface designed for user-friendliness and efficiency. Upon launching the interface, users are presented with an overview of available options, including selecting the database for the transaction and specifying the type of operation to be performed. The interface guides users through the interaction process, prompting them to choose the desired database from a list of available options. Once the database is selected, users are then prompted to specify the type of operation they wish to perform, such as merging, updating, or inserting data.

9 RESULTS AND EVALUATION

The development of a Distributed NoSQL Triple Store with Conflict Resolution and Data Consistency prototype marks a significant advancement in addressing the challenges associated with managing large-scale TSV datasets in distributed environments. The project aimed to design and implement a scalable solution capable of handling massive volumes of triple data while ensuring data integrity, consistency, and efficient conflict resolution.

It is worth noting that while the report provides an overview of the project's objectives and methodologies, the detailed results and evaluations will be presented during the viva. This decision was made to facilitate a more interactive and comprehensive discussion of the prototype's performance, where we will showcase the results and demonstrate the effectiveness of the implemented solution.

10 CONCLUSION AND CHALLENGES FACED

Key Findings and Achievements: The project successfully addresses the challenge of managing and querying large-scale TSV datasets efficiently by developing a prototype of a distributed NoSQL triple store. Leveraging a client-server model and state-based objects, the system demonstrates scalability, data integrity, conflict resolution, and flexibility in handling triple data across distributed servers. Key achievements include the development of server components hosting distinct NoSQL database management systems, implementation of merge operations and conflict resolution logic, utilization of state-based objects for efficient tracking of triple changes, and offering various services through the client interface for querying, updating, merging, and auditing triple data.

Challenges Faced: .

(1) **Merge Conflicts:** One of the primary challenges encountered during the project was effectively managing merge conflicts that arose when synchronizing data across distributed servers. Conflicting updates to the same triple from different servers required careful resolution to maintain data consistency and integrity. Developing robust conflict resolution strategies and mechanisms to handle conflicting updates was crucial to ensuring the reliability of the distributed NoSQL triple store.

(2) **Debugging Merge Operations:** Debugging merge operations posed another significant challenge, particularly when dealing with complex conflicts or unexpected behaviors during the merging process. Identifying the root causes of merge-related issues and troubleshooting them effectively required thorough understanding of the system architecture, data synchronization mechanisms, and interactions between client and server components. Debugging tools and techniques were essential for diagnosing and resolving merge-related errors and inconsistencies efficiently.

11 WORK SPLIT

- Akshay handled MongoDB implementation and contributed to the report.
- Keshav was responsible for Hive integration and related code development.
- Shishir worked on UI design, SQL integration, and frontend development.
- Vineet focused on testing, debugging (merge operations) and contributed to the project report.

Received 20 February 2007; revised 12 March 2009; accepted 5 June 2009