

DataSpan - A Distributed Triple Store Using State Based Objects

MUNAGALA KALYAN RAM - IMT2021023, IIIT-B, India
YUKTA ARAVIND RAJAPUR - IMT2021066, IIIT-B, India
BRIJ BIDHIN DESAI - IMT2021067, IIIT-B, India

ACM Reference Format:

Munagala Kalyan Ram - IMT2021023, Yukta Aravind Rajapur - IMT2021066, and Brij Bidhin Desai - IMT2021067. 2018. DataSpan - A Distributed Triple Store Using State Based Objects. *ACM Trans. Graph.* 37, 4, Article 111 (August 2018), 3 pages. <https://doi.org/XXXXXXX.XXXXXXX>

1 INTRODUCTION

The project asks to design and implement a prototype distributed triple store by making use of state based objects. It requires implementing an application that connects **n** servers where each server is connected to a different database, i.e Postgres, MongoDB, Apache Hive, etc and each server contains the same copy of the **yago dataset**.

The client should be allowed to make updates and queries on the server of choice, independent of the other servers and on performing a merge operation, the servers should reach a common state.

This project's significance lies in coming up with an approach to ensure consistency across replicas where each replica is stored in a different database system.

2 SYSTEM ARCHITECTURE

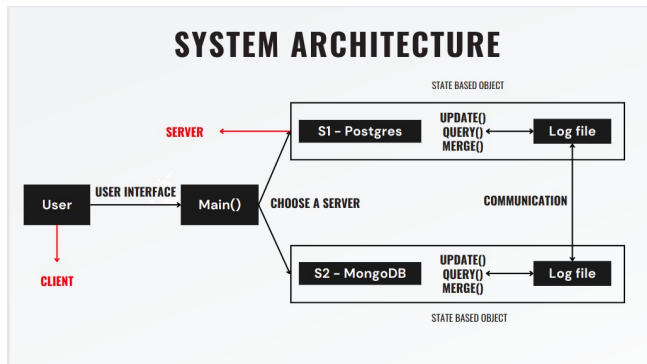


Fig. 1. System Architecture Design

Authors' Contact Information: Munagala Kalyan Ram - IMT2021023, kalyanram.munagala@iiitb.ac.in, IIIT-B, Electronic City, Bangalore, India; Yukta Aravind Rajapur - IMT2021066, yukta.rajapur@iiitb.ac.in, IIIT-B, Electronic City, Bangalore, India; Brij Bidhin Desai - IMT2021067, brijbidhin.desai@iiitb.ac.in, IIIT-B, Electronic City, Bangalore, India.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

© 2018 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM 1557-7368/2018/8-ART111
<https://doi.org/XXXXXXX.XXXXXXX>

The diagram shown in [Fig 1] describes the system architecture we used for the project. This is a client server architecture where a server is a class that connects to a database.

The class here is the state based object where it contains the **query, merge, update** methods which are implemented by utilizing the api's available to connect to the respective database through the application. Each database contains a table with the **subject, predicate, object** triple as its attributes.

All the updates done on a server are maintained in log files which contain **sequence no., subject, predicate, object, timestamp**. The timestamp is important for maintaining consistency where we utilize the **last write wins** approach that takes the update with the highest timestamp value as the latest write done on a key (subject-predicate) and the value (object) present is updated across servers involved in the merge operation.

When the client enters the user interface (terminal-based), they have the option to choose the server after which the client can make use of the server's api's, i.e the methods offered by the state based object.

3 METHODOLOGY

3.1 Methods Implemented for State Based Object

- (1) **Query (subject)** : Takes user input and creates a sql statement that is executed on the database and the output is displayed on the UI.
- (2) **Update (subject,predicate,object)** : This takes the user input and checks if a given triple is already present, else it is updated and the update is added to the log file.
- (3) **Merge (serverID)** : This takes the server to merge with as an input (remote server), then all the triples which are updated are iterated through and the latest update is taken based on the timestamp value.

3.2 Programming Languages and FrameWorks

- (1) **Programming Language** : Java
- (2) **Postgres** : Postgres [1] is installed in the system and kept running.
- (3) **MongoDB** : MongoDB [2] is installed in the system and kept running.
- (4) **Postgres Server** : Postgres JDBC and java.sql library used to implement the postgres server.
- (5) **MongoDB Server** : MongoDB java drivers are used to connect to the mongodb database.
- (6) **Apache Hive** : Hive [7] is setup along with installation of hadoop.
- (7) **JUnit** : JUnit [4] is a java library to write testing scripts.
- (8) **Hive Server** : Yet to be added

We are currently using 2 servers that are connected to MongoDB and Postgres. The final submission will have 3 servers integrated to the application where the third server will use Apache Hive.

We have decided to use these databases in the servers as they have been covered in the course and it shows that our solution can accommodate different database architectures. We choose to use java as it provides driver implementations to connect to the databases described above and is easy to use.

4 IMPLEMENTATION

This section describes how the system architecture [Fig 1] is implemented using the frameworks and programming languages mentioned in [Section 3].

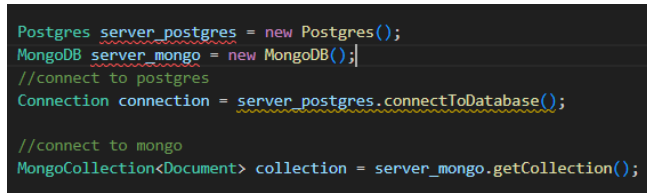
4.1 Database Setup

We first setup the mentioned databases in our systems and ensured that they are running.

We then run the postgres and mongodb scripts that stores the triples as follows :

- Postgres Server : Table with fields (SUBJECT, PREDICATE, OBJECT)
- Mongo Server : JSON object collection with field (SUBJECT, PREDICATE, OBJECT)
- Hive Server : Yet to be Added

The yago dataset is loaded to the databases using the scripts mentioned above.



```
Postgres server_postgres = new Postgres();
MongoDB server_mongo = new MongoDB();
//connect to postgres
Connection connection = server_postgres.connectToDatabase();
//connect to mongo
MongoCollection<Document> collection = server_mongo.getCollection();
```

Fig. 2. Connecting to different Servers

4.2 Main()

The main() function has the code to the user interface [Description in Section 6], initializes the servers [Fig 2] and maintain a unique id (Server ID), calls the necessary functions based on the user choices.

This function also contains the location to the Log files which are being used to maintain the updates performed on each server and maintains the attributes :

The main function also maintains a global nested map which maintains takes serverID as key and value as another map that maintains the last synced lines in the current server and the remote server.

4.3 MongoDB and Postgres Classes

```
public static void queryTriple(Connection connection, String
    ↳ subject) throws SQLException {
    String sql = "SELECT_*_FROM_sample_yago_WHERE_subject_=?
    ↳ ";
```

```
PreparedStatement statement = connection.prepareStatement
    ↳ (sql);
statement.setString(1, subject);

ResultSet resultSet = statement.executeQuery();
while (resultSet.next()) {
    String retrievedSubject = resultSet.getString("
    ↳ subject");
    String retrievedPredicate = resultSet.getString("
    ↳ predicate");
    String retrievedObject = resultSet.getString("object"
    ↳ );
    System.out.println("Subject:_" + retrievedSubject + ",
    ↳ _Predicate:_" + retrievedPredicate + ",_
    ↳ Object:_" + retrievedObject);
}
}
```

Listing 1. Query Triple Stored in Postgres with JDBC

```
public static Map<String,String[]> queryTriple(
    ↳ MongoCollection<Document> collection, String subject
    ↳ ) {

    Map<String, String[]> results= new HashMap<>();
    Bson filter = subject.isEmpty() ? null : Filters.eq("
    ↳ subject", subject);
    for (Document document : collection.find(filter)) {
        String retrievedSubject = document.getString("subject
        ↳ ");
        String retrievedPredicate = document.getString("
        ↳ predicate");
        String retrievedObject = document.getString("object")
        ↳ ;
        String [] tmp= {retrievedPredicate,retrievedObject};
        results.put(retrievedSubject, tmp);
        // System.out.println("Subject: " + retrievedSubject
        ↳ + ", Predicate: " + retrievedPredicate + ",
        ↳ Object: " + retrievedObject);
    }
    return results;
}
```

Listing 2. Query Triple Stored in MongoDB Java Drivers

Update() and Query() :

- Queries are constructed based on the user input.
- The Driver corresponding to the server translates the query into the database's internal format and communicates with the server.
- Only the update queries are written to the log file. The log file has the following format : **Sequence No, subject, predicate, object, timestamp.**

Merge() :

- It uses the global nested map defined in the main function to locate the last synced updated in the current and remote server's log files.
- Following which the remaining updates are maintained in a hashmap which will be used to implement the last-write-wins approach.
- The hashmap is maintained each triplet as HASHMAP (KEY = (SUB,PRED), VALUE = (OBJECT, TIMESTAMP)).
- - The map is updated for a key only if the timestamp is newer during the merge operation. This is where the last-write-wins concept comes into play. The merge operation is done both ways to ensure that the servers are in sync.

5 TESTING

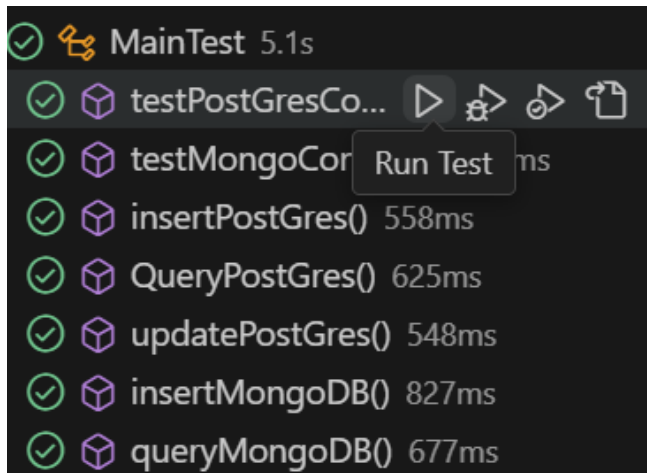


Fig. 3. JUnit Testing on update, query, connection

5.1 Testing Approach

The testing of the application is done by using JUnit and manual testing.

- **JUnit** : This library is currently used to test the update, query and connection operations of the Postgres and MongoDB servers as seen in [Fig 3].
- **Manual Testing** : The merge operation is tested by manually making updates to the each server through the UI and the log files are checked after the merge operation to check if the last-write-wins approach is implemented correctly.

5.2 Dataset Description

The YAGO dataset is a comprehensive knowledge base that organizes human knowledge into structured data. It includes information about entities and their relationships sourced from sources like Wikipedia and WordNet. The data is structured as triples of subject-predicate-object.

A sampled version of the yago dataset is being used to perform the testing by updating the object value of various keys (subject-predicate) and performing merge.

5.3 Issues

We are currently facing issues with testing the merge operation using JUnit and we have not tested on 3 servers. We will look to make shift from manual testing to JUnit and make more test cases to cover a wider range of user interactions.

6 USER INTERFACE

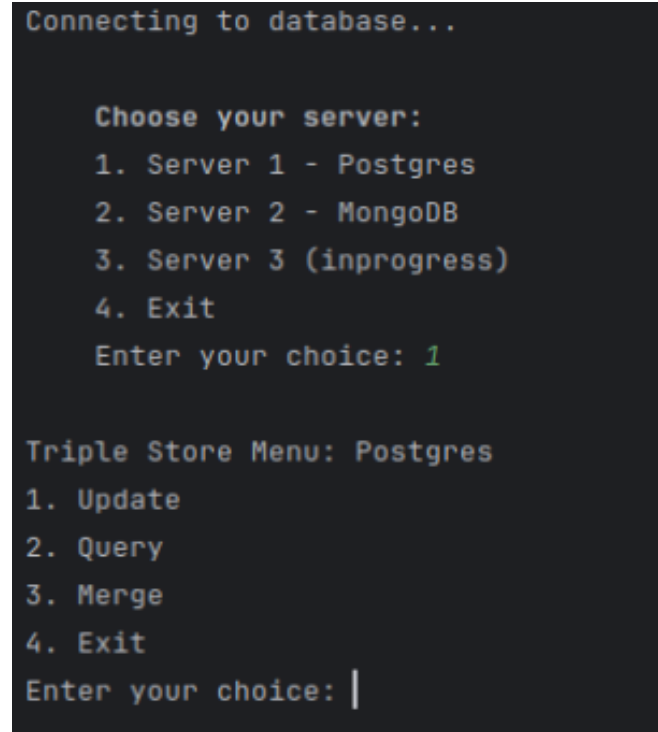


Fig. 4. Terminal Based User Interface

We have implemented a terminal based user interface [Fig 4] . The user is provided with options to choose a server of his choice.

After picking the server, the user can perform the following operations independent of the other servers **update**, **query**, **merge**, **exit**. On exit, the user can once again choose the server to execute further operations.

REFERENCES

- [1] Postgres Installation: <https://www.digitalocean.com/community/tutorials/how-to-install-and-use-postgresql-on-ubuntu-20-04>
- [2] MongoDB Installation: <https://www.digitalocean.com/community/tutorials/how-to-install-mongodb-on-ubuntu-18-04-source>
- [3] Java Maven Repository : <https://mvnrepository.com/>
- [4] JUnit - Java Library For Testing : <https://www.javatpoint.com/junit-tutorial>
- [5] Yago Dataset: <https://www.mpi-inf.mpg.de/departments/databases-and-information-systems/research/yago-naga/yago/downloads/>
- [6] Hadoop Installation: <https://medium.com/@vikassharma555/hadoop-installation-on-windows-wsl-2-on-ubuntu-20-04-lts-single-node-d604729ea0ca>
- [7] Apache Hive Download : <https://hive.apache.org/general/downloads/>
- [8] Working with MongoDB in Java: <https://www.baeldung.com/java-mongodb>