CSCI 2980 Project Report

Data Migration from S-Store to BigDAWG

Yulong Tian

B01050880

Department of Computer Science, Brown University

Advisor: Prof. Stan Zdonik

Abstract

From spring 2016, I've been working with Prof. Stan Zdonik in a project about data migration from S-Store to BigDAWG polystore system. S-Store, which built on top of H-Store, is the world's first transactional streaming database system. S-Store maintains all the transactional support in a traditional relational database, while it supports streaming processing which is needed in the real-time applications. BigDAWG, which built on top of a variety of storage engines by MIT, is a polystore system provides cross-system querying, exploratory analysis, real-time decision supports and complex analytics. S-Store will serve as a front streaming engine within the BigDAWG, as well as a relational in-memory database in the relational island. In either case, the data within S-Store need to be migrated to different storage system in BigDAWG. This report first introduces these two systems, and then discusses the motivation and use cases for the migration process. Then it states the details about the decision and implementation, as well as the performance testing about the migrator implemented. In the last part of the report, I also proposed some observation and works that can be done in the future.

Table of Contents

1. Introduction **4**
1.1 S-Store and H-Store 4
1.2 BigDAWG 4
2. Motivation and Design **4**
2.1 JDBC within S-Store 4
2.2 Data migration between S-Store and BigDAWG 5
3. Implementation Details **6**
3.1 Refine JDBC implementation for S-Store 6
3.2 Modify GetSystemInfo system procedure 6
3.3 Create ExtractionRemote system procedure in S-Store 7
3.4 Execute Migration 7
4. Experiments and Analysis **8**
4.1 Case analysis 8
4.2 Migration experiment results 8
5. Future Works **10**

1. Introduction

1.1 S-Store and H-Store

H-Store is the base of S-Store. H-Store is an in memory distributed database that is optimized for OLTP (on-line transaction processing) applications. It can achieve a very high throughput with a very low latency, because it's designed to be highly distributed, over a cluster on shared-nothing nodes. H-Store is a collaboration project between MIT, Brown, Carnegie Mellon, Yale and Intel, whose final version was released in June 2016.

S-Store is built on H-Store and takes advantage of its high throughput within a low latency. Besides that, S-Store introduces push-based processing, native data structures and windows, for adding ability to deal with streaming input. S-Store regards streaming transactions as stored procedures to be executed on batch of input tuples. And additional sequence of input batch is imposed to give a strong ACID property.

1.2 BigDAWG

BigDAWG is a polystore system that designed for cross-DB querying, exploratory analysis, real-time operations, complex analytics, and prototype engines. It provides a solution to explore and analyze data stored across a variety of database systems.

One of a feature in BigDAWG is cross database querying. BigDAWG offers users a way to query multiple vertically-integrated system including traditional relational databases, column stored databases, NewSQL databases or array stored databases, without caring about the difference between those systems, such as different front-end languages. That location transparency can benefit users that they will not need to understand the details about the underlying databases in which the queries to be executed. The ideas have been implemented as different islands in the system – each island forms a front-facing abstraction to the user. Each island includes a query language, a different data model and a set of connectors or shims for its own underlying databases. For example, within relational island, there could be PostgreSQL, S-Store, and other database using relational model.

2. Motivation and Design

2.1 JDBC within S-Store

JDBC (Java Database Connectivity) is a widely used API for users to interact with their database in Java. It offers a reliable and efficient way to access the database such as Driver Manager, PreparedStatement and stored procedures. So implementation of JDBC for S-Store could benefit many users, as well as to simplify the process to let user interact with S-Store. Besides that, S-Store is served as the streaming ETL engine in BigDAWG, as well as a normal in-memory database under the relational island within BigDAWG. In either case, we need a communication interface between S-Store and other databases in BigDAWG.

Here, we choose JDBC as the connection interface within those databases. We connect S-Store and database in BigDAWG (PostgreSQL) by using JDBC. So in this way, we can extract all the metadata in S-Store by a JDBC method call, or even do the data migration via JDBC.

2.2 Data migration between S-Store and BigDAWG

The target database in BigDAWG we choose here is PostgreSQL. The motivation to migrate data from S-Store to BigDAWG are:

1) S-Store serves as a front processor for BigDAWG. It accepts input from multiple sources and do ETL within it, for better integration with other database in BigDAWG. After the ETL process, the data need to be exported to the appropriate system in BigDAWG, which requires data migration.
2) S-Store is also a highly distributed in-memory relational database, which could be put under the relational island in BigDAWG. Since S-Store requires the entire data resides in the main memory, it limits the amount of data in database for a given cluster. Moreover, the high velocity streaming input could eat up memory very fast, so flush all the staled data to other relatively stable database will be very beneficial. And we can even use the migrator to achieve other task such as writing S-Store data to the disk.
3) The migrator makes cross database queries more flexible and potentially more efficient. That enables a way to run those queries, by firstly migrate all or partial data to another database, then do query in a single system, instead of combining the query results after the parallel execution on both databases. This will be extremely useful when the data is highly unbalanced (for example, suppose there is 80% data already in PostgreSQL, while only 20% data is in S-Store. This is the most common case when we are using S-Store as a ETL engine). Moreover, processed streaming

input need to go to other database in BigDAWG anyway, so why don't we just do it when those queries come?

3. Implementation Details

3.1 Refine JDBC implementation for S-Store

The old implementation of JDBC for S-Store need more refinery and testing. Actually that one is a slightly modified version from VoltDB JDBC implementation. Since I interned at VoltDB this summer, I am pretty familiar with that implementation. The problem for the old S-Store JDBC implementation are:

1) Lack of implementation for getting database metadata. Since we are going to migrate all data in S-Store to PostgreSQL, the first thing we need to figure out is the database schema. We dealt this problem by calling a system procedure, instead of implementing those methods provided by JDBC.
2) Lack of appropriate testing. That testing was a fake one: no data has been loaded into the database before doing test. So we preload the data to database, and we also add more test cases for calling stored procedures via JDBC.
3) Other minor issues such as unnecessary hardcode or missing sanity checks. We read all the implementations and refined some minor issues within that implementation to enhance code readability, flexibility and solved some other minor issues.

3.2 Modify GetSystemInfo system procedure

We need to let PostgreSQL know the incoming data's schema. The metadata within the S-Store can give us basically everything: the attributes, primary keys, indices and their types…We just need a way to extract all the metadata. That metadata is crucial for the data migration process since another database should know this before the data is actually transferred. We decided to modify the existing GetSystemInfo system procedure to extract all desired metadata out. So I went over all the metadata that S-Store has, and picked some must-have metadata such as schema info, data types and indices, and wrote extraction code within that GetSystemInfo procedure (for example, if we need the primary keys for the database, we can call @GetSystemInfo primaryKeys, and all primary keys with their table names will be extracted).

Moreover, we decided to use JDBC stored procedure calls to let other database query the metadata of S-Store. So each time the desired metadata's name will be taken as an argument, and a system procedure call will return the result. The performance of doing this way is proven good enough to let other system knows what kind of data will be transferred later, so it can create correspond schema and indices and wait for the data to come.

3.3 Create ExtractionRemote system procedure in S-Store

A very naïve idea about the migration is by SQL query. But that is too slow because each time we call it, it has to go through the process that all SQL queries need to go through: SQL parsing, optimization, plan evaluation and finally execution (which is done in EE, that is, the Execution Engine). And considering the migration process may be called very often, this is not the way we want.

A better approach is directly jump into the EE (Execution Engine) layer. By scan all the records (tuples) directly in the EE layer, we can avoid all the cost of parsing or plan evaluation time. So the idea is by creating a new system procedure in S-Store, called ExtractionRemote, to directly scan tuples in the memory. And this approach is not only much faster, but also has more expandability such as adding acknowledgement to make sure there is no data lost, and automatically retry if any data lost is found.

We choose the second approach. Credit to Adam Dziedzic at University of Chicago, most of work is done by him. We can also choose the output data format such as PostgreSQL binary, SciDB binary or CSV, if we need a readable output.

3.4 Execute Migration

We decided to use JDBC to get the metadata from S-Store and let PostgreSQL get ready (create same tables and indices) for the migration. And we also use JDBC prepared statement to call the ExtractionRemote, taking the desired table name, the coordinator in S-Store, and desired output format as parameters. And we can get an output file (preferable binaries) from calling that procedure. After that, we can directly import that binary file to PostgreSQL, which is already supported.

A major concern about the process above is: the intermediate file is actually written to the disk. That wastes all the effort of S-Store to keeping the data in the memory, and we can only start to load after all the binaries has been written. That makes the migration process

taking too much time than we want. So we turned to remove that intermediate file and migrate data from S-Store to PostgreSQL directly, without any writing or reading from the disk. We used pipe to achieve this, by writing binaries to buffer and let PostgreSQL consume the data, as long as the two sides has agreed on the data format to be transferred. The migration performance dramatically increased by using this approach.

4. Experiments and Analysis

4.1 Case analysis

The migration enables the system to choose from doing query parallel or migrate data first and then execute query in single system. For example, consider a query need to be executed on S-Store side, but that query also need data which is already stored in PostgreSQL. Some possible plans are as follows:

1) Cross-system query

   We let required data remains in PostgreSQL. So that query need to be executed in cross-system way in a single time, which could be very expensive. Moreover, since the data in S-Store changes frequently because of the incoming streaming, we may need to run the same query again in a very short time – which makes this approach more expensive.

2) Replication

   The basic idea is cache PostgreSQL data in S-Store. Its performance depends on how often the data in PostgreSQL changes – because we need to make sure out cache is always up to date.

3) Migration

   We can migrate the data to S-Store. So in that way we need to afford the data migration cost, but after that we can just run the query locally in S-Store.

4.2 Migration experiment results

In order to evaluate the performance of the migration, we inducted an initial experiment that compares two query plan for a UNION query (Fig 1):

1) Migrate data from S-Store to PostgreSQL before doing query, and then execute the query exclusively in PostgreSQL.

2) Executing the query in parallel and UNION the result in the relational island within BigDAWG.

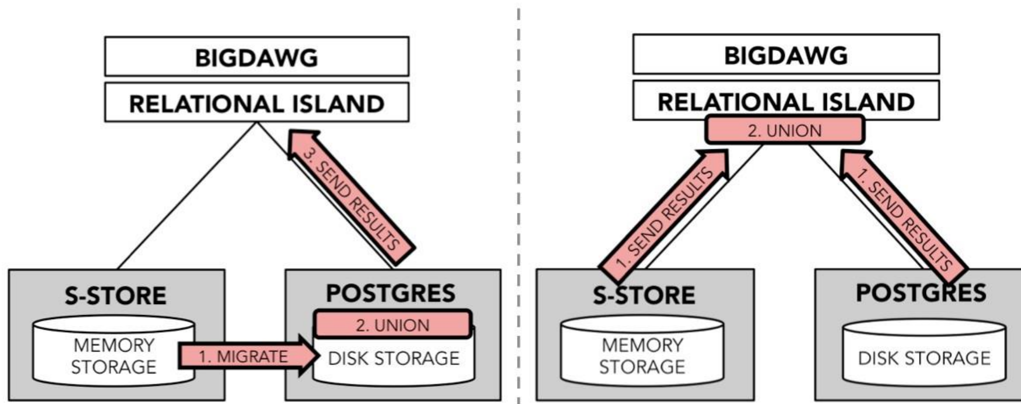

Fig 1. Migrate and query single system VS UNION sub-result in relational island

Here are some configurations about that experiment. We used an Intel XEON processor with 40 cores @2.20GHz, using S-Store in single-node mode. We are using the binary to binary migration described above, and queries are both executed via JDBC. The total number of tuples is 200,000, while the ratio of which were stored is S-Store or PostgreSQL is a variable.

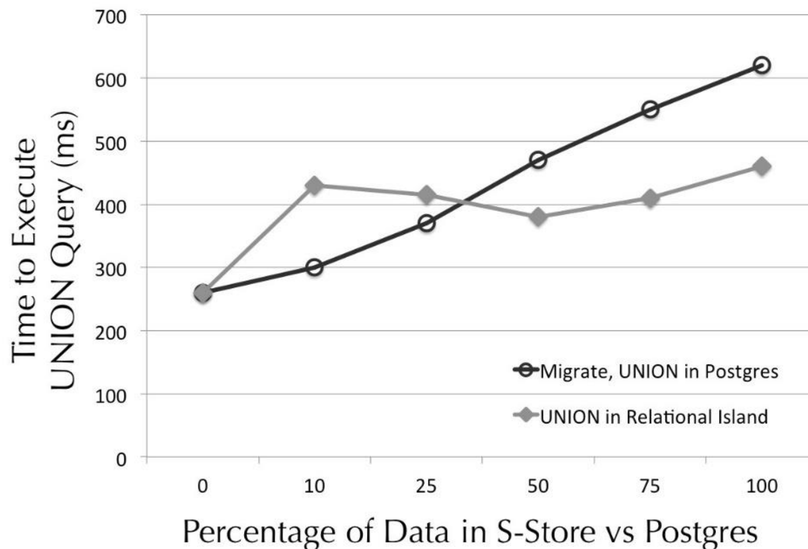

Fig 2. Comparison for two UNION plans

The result is as Fig. 2. As we can see, the cost of these two plans depend on the amount of data within S-Store vs PostgreSQL. The cost of migration increasing nearly linearly, because that depends on how much binaries we need to transfer. It seems like the migration plan is more efficient when less than 30% of data is in S-Store, and the biggest gain of doing migration happens when S-Store holds about 10% data. But we need to consider that since S-Store served as a streaming front engine for BigDAWG, only initial migration will be expensive. Taking continuous streaming input will just require migrating part of the data (usually much less than 30%), which proves the migration process we implemented is very useful.

Moreover, just as we expected, if the data we need to migrate is too much (> 30%), then the cross-database query becomes better. By the way, the cross-database querying achieves its best performance when we have a 50-50 distribution, which balances the workload of two databases.

5. Future Works

We can refine the migration process further, such as:

1) Add acknowledgement in both sides, so sender database can detect any data lost and resend all binaries again automatically;
2) Get metadata directly by calling JDBC methods, which can call the system procedure implicitly;
3) Make the data migration process not a one-time transferring. It will be better if we can migrate only part of the data as we needed;
4) Do more experiment with other databases to verify the performance gain and determine a general threshold for the efficiency guarantee to migrate before doing query.
5) Add more test cases about the system procedures we implemented, as well as other functionality we added during the migration process.

Reference

[1] A. Elmore et al., “A Demonstration of the BigDAWG Polystore System,” VLDB, Aug 2015.

[2] J. Meehan et al., “S-Store: Streaming Meets Transaction Processing,” arXiv:1503.01143, Mar 2015.

[3] J. Meehan et al., “Integrating Real-Time and Batch Processing in a Polystore,” IEEE-HPEC 2016.

[4] R. Kallman et al., "H-Store: a High-Performance, Distributed Main Memory Transaction Processing System," Proc. VLDB Endow., vol. 1, iss. 2, pp. 1496-1499, 2008.

[5] S. Tian., “Project Report: Integrate S-Store with BigDAWG,” 2016.

[6] http://sstore.cs.brown.edu/index.html/

[7] http://istc-bigdata.org/index.php/tag/bigdawg/

[8] http://hstore.cs.brown.edu/