The Release of S-Store System

Bikong Wang
Department of Computer Science
Brown University
Advisor: Prof. Stan Zdonik

ABSTRACT

In this paper, I describe multiple aspects that are closely related to the release of S-Store system, including dockerized S-Store system, new benchmark MIMIC2BigDAWG, dockerization of the integrated streaming processing system, verification of new S-Store scheduler and collection of stored procedures statistics. Although many other indispensable works are also finished for the final release, they will not be discussed in detail there.

1. INTRODUCTION

S-Store, built on top of H-Store, is a transactional streaming database system that features ACID transactions, dataflow ordering and exactly once processing [7]. Through inheriting most basic functionalities from H-Store and integrating many new design and features (dataflow, stream, window, trigger), S-Store owns all functionalities and features needed to guarantee strong support for both ACID and streaming. According to the experiment, S-Store can serve both OLTP and streaming applications without much sacrifice, it has demonstrated desirable performance and strong correctness guarantees compared with other state-of-the-art streaming systems [8].

The release of S-Store will further accelerate the application of streaming transactional engine in multiple fields, such as management of real-time and high-velocity data, ETL (Extract, Transform, and Load) processes, and stream processing in big data application, etc. It also helps for developing the new time-series database system for better managing the data generated via IoT and serving future big data and intelligent applications.

Moreover, the release makes S-Store completely open to the general system community, and increases accessibility of the system to new users, which will help to attract more researchers and developers to work on and contribute to the project and promise S-Store's future development.

Although the most components and core functionality of S-Store are already finished, many tasks important to the release have not completed yet and they are supposed to be done during release preparation phase. The rest of this paper documents several aspects and new features important for the final release and future development, such as verification for correctness of new S-Store scheduler, new benchmark MIMIC2BigDAWG, and dockerization of S-Store system and its application with stream generator and BigDAWG using Docker.

2. SCHEDULER VERIFICATION

The streaming transaction in S-Store can be regarded as stored procedure (SP) which operates on input stream (batch of tuples) [4]. The whole streaming workflow (DAGs) might involve multiple different stored procedures, as showed in Figure 1 from the paper [8]. Because of the dependency between data and processing order, transaction executions (TEs) should be guaranteed to be performed in proper order [4]. The execution order is not necessary to be first in and first out (FIFO), but it should always be consistent with the dataflow graph, i.e. it could be one of all possible order of the topological ordering of dataflow graph.

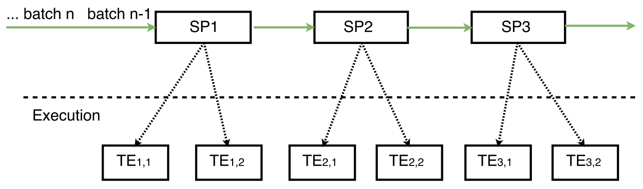

Figure 1: Dataflow Graph

The S-Store scheduler ensures the proper ordering execution of streaming transactions, and guarantees the correct batch ordering within a stored procedure and across different stored procedures, i.e. earlier batch should be executed before latter batch (identified by both the procedure and batch ID) in either one stored procedure or across multiple stored procedure. The scheduler should also guarantee that each transaction will only be executed once. The verification of scheduler is basically the test of correctness of functionalities (execution ordering guarantee and exactly once execution) that scheduler provides using both regression and unit tests.

As for implementation details, the new implemented scheduler test suite is based on TPC-DI benchmark [10], it includes the execution of single transaction and multiple transactions with and without order, and covers both single stored procedure and multiple stored procedures. The implemented scheduler test suite will fail in the case that transactions are executed out of batch order or that the same transaction has been executed repeatedly (more than once).

To make it easier for future adjustment, the new implemented test suite has been integrated as one piece of whole test suite of S-Store system, it will automatically execute when users test system or build system with test option enabled. Besides, S-Store terminal's output has been largely improved, statistics of stored procedures are collected and reorganized to better demonstrate the number of transactions actually executed within each stored procedure, so performance of the new scheduler could be evidently inferred there.

3. BENCHMARK MIMIC2BIGDAWG

MIMI2BigDAWG is a new benchmark created from scratch, which uses the schema and data from MIMIC II dataset [3] that is consist of heterogeneous types of data collected in the hospital. MIMIC2BigDAWG is setup on top of stream generator and BigDAWG polystore [6] aside from S-Store. The stream generator is a streaming tool with adjustable parameters, it is able to continuously send data files as tuples via specified port. BigDAWG polystore is a distributed federated storage engine, which enables data migration across each of databases (PostgreSQL, SciDB, and Accumulo) and provides functionality of unified querying to clients. From users' perspective, the BigDAWG polystore appears as a single system.

Different from other benchmarks of S-Store, the goal of the MIMIC2BigDAWG benchmark is do the real-time data ingestion, which is primarily consist of data extract, data cleaning, and data transform. As a demonstration, the benchmark is implemented to simulate a simplified streaming data migration process which migrates data from files to database system.

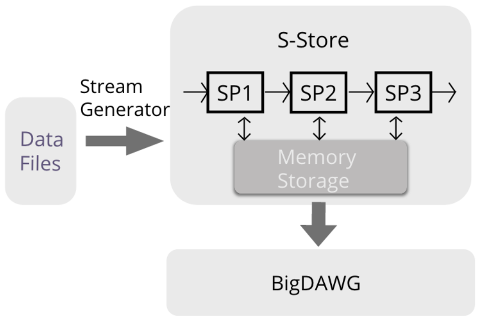

Figure 2: MIMIC2BigDAWG Data Migration

The general dataflow graph of MIMIC2BigDAWG benchmark is showed in Figure 2 [9], the whole data migration process can be separated into three stages, which include collection and dispatch of heterogeneous data from MIMIC II dataset, data processes via stored procedures (cleaning, transforming, etc.), and data route and transfer to the appropriate processing engine (Postgres) through BigDAWG. While the benchmark is initialized, it will first start preparation process, then the data will continuously flow from stream generator to the Postgres in BigDAWG via S-Store.

As to implementation, the client of stream generator is constructed inside the benchmark to receive tuples sent by stream generator server. Once tuples are received, they will go through all new implemented and predefined stored procedures in proper order controlled by scheduler, then data will be transferred and loaded into Postgres. Another thing deserve notice there is that MIMIC2BigDAWG is designed and implemented to be more straightforward and extensible for the purpose of making catch of any data flow errors caused by system much easier and making benchmark own the capability to handle more complex scenario in future.

4. DOCKERIZATION

Docker is the software container platform, which automates the deployment of application using container image. The container image is stand-alone and executable package containing source code, system libraries, runtime environment, etc [2]. Docker makes software development, shipment and deployment much easier, efficient and secure by packing software into isolated standardized units which guarantee software run completely the same no matter where it is deployed [2].

Compared with virtual machine (VM), Docker is more portable and resource efficient because OS kernel is shared by multiple containers rather than including a full copy of OS in each virtual machine. The difference between Docker and virtual machine is demonstrated in Figure 3 from [2].

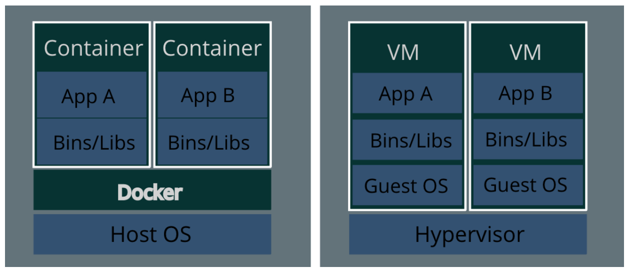

Figure 3: Docker vs Virtual Machine

Since the deployment of S-Store replies on many other software dependencies, packages and environment setups, dockerization of S-Store will save users from setting up S-Store manually. It makes S-Store much easier to use either deploying S-Store as a single system or making S-Store as a subsystem of larger integrated system [9].

Dockerfile is the script file which is used for building the Docker image, so S-Store Dockerfile is implemented and also released as a part of S-Store source code. Once user start the image building process, the S-Store Dockerfile will be used by Docker to integrates all resources and dependencies that S-Store requires, and then S-Store will automatically setup environment by instruction specified in Dockerfile. By taking advantage of Docker, S-Store could be run on any machines no matter what the operating system they use, as long as Docker are properly installed.

There exist cases that S-Store is used as a subsystem [9], which justify the necessity of dockerization of S-Store, since such an integrated system involves too many complex packages and configurations, it becomes difficult and time consuming to manually setup all of them correctly. In our case, dockerization of S-Store with BigDAWG is more complex, it not only requires dockerization of each subsystem at the first step, but also the connection of each subsystem between corresponding containers because of the communications between either S-Store and BigDAWG or each database of BigDAWG. There S-Store is configured to join in the Docker network of BigDAWG, as showed in the Figure 4 [1], BigDAWG network bridges all components of BigDAWG and S-Store, each component makes specific ports available for communication with clients or transferring data and connecting within the system.

Considering that the complexity and flexibility of system in our case, there exist multiple different ways that users would like to use S-Store via Docker (only benchmark, interaction through terminal, etc.), which require various commands for different purpose. It becomes important to document the detailed instructions about how S-Store commands are used with Docker containers. Therefore, the document of S-Store Docker usage has been written and integrated into the S-Store document, which can either be used as a reference or help new users to quickly similar with the system [5].

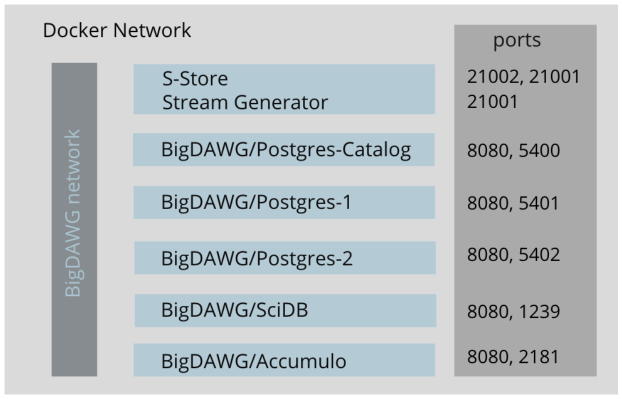

Figure 4: Dockerization of S-Store and BigDAWG

5. CONCLUSIONS

This paper gives an overview of several aspects I worked on for the final release of S-Store system. Some other works are also finished, including S-Store versions comparison and copyright updates through scripts, modification of S-Store default configurations, alteration of S-Store build scripts, and stream generator's integration into S-Store system (compilation and testing), etc., which are not given a detailed description there.

Through all works mentioned above for the S-Store release, the latest release version of S-Store is expected to be more stable and easier to use, and it should better support later projects and applications which use or build on top of S-Store system.

6. ACKNOWLEDGMENTS

I would like to thank Professor Stan Zdonik, and John Meehan for their patient guidance on this project.

7. REFERENCES

[1] BigDAWG. http://bigdawg.mit.edu.
[2] Docker. https://www.docker.com.
[3] MIMIC2. https://physionet.org/mimic2.
[4] S-Store. http://sstore.cs.brown.edu.
[5] S-Store Document. http://sstore-doc.readthedocs.io/.
[6] J. Duggan, A. J. Elmore, M. Stonebraker, M. Balazinska, B. Howe, J. Kepner, S. Madden, D. Maier, T. Mattson, and S. Zdonik. The BigDAWG Polystore System. *SIGMOD Rec.*, 44(2):11–16, Aug. 2015.
[7] J. Meehan, C. Aslantas, S. B. Zdonik, N. Tatbul, and J. Du. Data Ingestion for the Connected World. In *CIDR*, 2017.
[8] J. Meehan, N. Tatbul, S. Zdonik, C. Aslantas, U. Cetintemel, J. Du, T. Kraska, S. Madden, D. Maier, A. Pavlo, M. Stonebraker, K. Tufte, and H. Wang. S-Store: Streaming Meets Transaction Processing. *Proc. VLDB Endow.*, 8(13):2134–2145, Sept. 2015.
[9] J. Meehan, S. B. Zdonik, S. Tian, Y. Tian, N. Tatbul, A. Dziedzic, and A. J. Elmore. Integrating real-time and batch processing in a polystore. *2016 IEEE High Performance Extreme Computing Conference (HPEC)*, pages 1–7, 2016.
[10] M. Poess, T. Rabl, H.-A. Jacobsen, and B. Caufield. TPC-DI: The First Industry Benchmark for Data Integration. *Proc. VLDB Endow.*, 7(13):1367–1378, Aug. 2014.