Analysis: Data Ingestion for the Connected World https://people.csail.mit.edu/tatbul/publications/sstore_cidr17.pdf

Anwesha Saha

October 2, 2023

1 What is the problem this paper is solving?

The slowness in traditional data ingestion or ETL (Extract, Transform and Load) methods in Big Data applications does not allow modern systems to perform timely actions and make real time decisions. The given paper tries to solve this problem by building a new architecture for ETL that takes advantage of the push-based nature of a stream processing system.

2 Why is it important?

Traditional ETL is constructed as a pipeline of batch processes due to which read and write actions are very slow. Previous applications like data warehouses were not sensitive to the latency which current applications like IoT are. It therefore becomes mandatory to devise a system that accurately models the real world in order to accommodate real-time decision making.

3 Why is it challenging?

The task is challenging because there are a number of requirements for successful creation of such a system that are:

1. The system should be able to process data fast since the value of data decreases drastically with time
2. The order of the data has to be maintained without having long waiting hours for data to become available
3. Time-series data increases exponentially very fast which makes storing data expensive. Hence there needs to be a decision making system that stores only selective data and processes data as it arrives

4 Describe the paper gist in 1-2 sentences.

The paper tries to find a solution for data ingestion in recent big data applications that are time sensitive by using a stream based approach and designing an architecture comprising of four main components for - data collection, streaming ETL, OLAP backend, and a data migrator. For this, it uses Kafka for data collection, S-Store for Streaming ETL and Postgres and BigDAWG for OLAP Backend and Migration, and proves via experiments and by comparison with existing systems with similar aims, why the suggested architecture best solves the problem undertaken.

5 What is important to remember? What did we learn?

The same approach for ingestion does not fit all use cases. For example, conventionally since time sensitive applications were not as common, we did not require a stream based ingestion mechanism. With the suggested solution and architecture, we not only aim to solve problems in the IoT domain but also better performance conventional ingestion applications used by various enterprises. To suggest such an improvement, there is a requirement to also maintain the properties that a previous batch system processing architecture maintained. The proposed system would have to maintain correctness and predictability of results, scale with the number of incoming data sources and process data with least delay. To uphold these properties, we divide our requirements into three parts - ETL requirements, Streaming requirements and Infrastructure requirements. We use Kafka for data collection, S-Store for Streaming ETL and Postgres and BigDAWG for OLAP Backend and Migration to develop the architecture for the proposed solution. We prove the efficiency of our system and selected methods via experiment and also state the limitations visible from it. The paper further tries to better the proposed solution by addressing these limitations - specifically relational databases and suggests a time series ETL for future work. The paper also discusses some alternative approaches to solving the same or similar problems and how it differs from the proposed architecture.

In conclusion, the paper does solve the problem it initially addressed but also admits that there are better ways to compare the actual results for finding the true efficacy of the proposed architecture and allows the scope for improvement based on the solution suggested.

6 Solution description: Explain how the solution works

The solution consists of four primary components: data collection, streaming ETL, OLAP backend, and a data migrator to provide a reliable connection between the ETL and OLAP components.

1. Data collection is assumed to be from heterogeneous distributed systems and therefore requires it to be scalable and fault tolerant as well as be responsible for assigning logical batches of tuples for consumption by the ETL engine. Apache Kafka is used for this since it is highly scalable and reliable publish-subscribe system with the capability to handle multiple requests simultaneously. It serves as the messaging queue for individual tuples, each of which are routed to the appropriate dataflow graph within the streaming ETL engine. Batching is performed by Kafka.

2. Streaming ETL needs to have the ability to cache data from the data warehouse since constant lookup is expensive. The data post cleaning and transformation remains in the Streaming ETL engine till the data is ready to be migrated either through data warehouse pulling the data or the ETL pushing the data to the warehouse. For this purpose, the streaming ETL engine must be scalable to support expanding amounts of data, and fault-tolerant to ensure that results are recoverable in the event of a failure.

   S-Store is used for Streaming ETL since it has state management and models dataflow graphs as a series of transactions, each of which ensure a consistent view of the modified state upon commit. S-Store is built on top of the main-memory OLTP system H-Store and integrates streaming functionality such as streams, windows, triggers, and data flows. It provides three fundamental guarantees which together are exclusively available in S-Store: ACID transactions, dataflow ordering, and exactly once processing. It uses relational tables and user defined stored procedures as its transactional operations. Each stored procedure is defined using a mixture of Javaand SQL allowing for flexibility. Incoming data is batched in order to improve performance.

3. OLAP backend has a query processor along with one or more OLAP engines. Each OLAP engine contains its own data warehouse, and delta data warehouse which stores updated data that is periodically merged with the full data warehouse.

   Postgres was chosen as a backend database. To address migration issues, built-in data migrator provided by the BigDAWG polystore was chosen. S-Store is integrated with BigDAWG so migration between S-Store and any OLAP system supported by the polystore is easy to implement and efficient and is transactional.

4. Durable migration is required for moving the data between the OLAP backend and the Streaming ETL that has to be done when no other intensive operation is being performed. For this, ACID state management is crucial for a Streaming ETL component, and therefore the migration mechanism should fully support ACID transactions. The migration is done either by a Push or a Pull depending on use case since both methods have proven advantage and disadvantages.

The architecture handles migration by using the postgres database with BigDAWG. Using two-phase commit and maintaining open transactions on both S-Store and Postgres until both sides of the migration have completed, BigDAWG is able to ensure that a batch of data is either fully migrated or completely rolled back, thereby avoiding data loss or duplication issues. This satisfies the requirement that data is only written once and no data is lost.

7 Describe the experimental setup

The experimental setup tries to find the ideal frequency with which the data should be migrated to a data warehouse for a streaming ETL system. This can be done in two ways -

1. ingestion engine periodically pushes the data to the warehouse
2. the warehouse pulls the data from the ingestion engine when required

The experiment was run on an IntelR CoreTMi7 machine with 8 virtual cores and 8 GB of memory. S-Store, BigDAWG, and Postgres were run on a single node. S-Store was run in single-partition mode, and Postgres used the default settings. Batches were composed of a single tuple per batch.

- In the setup - for the streaming ETL engine- S Store the operations to ingest tuples from the text file into the required database was implemented.
- The tables in the TPC-DI database were created in Postgres as the data warehouse and populated with historic data to run analytical queries.

8 Summarize the main results

There are 2 cases for the main results:

1. when data is pushed from the data ingestion engine to the data warehouse:
    - Staleness: data staleness for analytical queries is directly proportional to the time between migrations. staleness is seen to be approximately half the duration between migrations when query is repeated multiple times
    - Run time: The run time in this case is almost negligible
    - Max Ingestion Latency (Data migration): The type of method used has no significance and only depends on interval between migrations
2. data warehouse automatically pulls data from the engine:
    - Staleness: Here since most recent data is pulled post issuing of the OLAP query the staleness is 0

- Run time: performance is affected by the frequency of the push of data
- Max Ingestion Latency (Data migration): The type of method used has no significance and only depends on interval between migrations

9 When doesn't it work?

1. S-Store is designed to handle ordered batch data in a streaming context but its query support is ingrained in relational databases specifically. S-Store also has to be carefully designed and is expensive for distributed database systems.
2. For SciDB time-series can be stored as a one-dimensional array, but an array database lacks support for complicated time-series queries.
3. Both S-Store and SciDB center around strong support for mutable state rather than append-heavy time-series workloads. Further optimization is possible

10 What assumptions does the paper make and when are they valid?

The paper is strictly aimed at solving a set of problems in the IoT domain or in the TPC-DI domain. The paper assumes that data ingestion and analytics takes place in different locations. Kafka, S-Store and Big Dawg might not be as easy to scale as projected

11 Related work: List the main competitors and describe how they differ

1. AsterixDB highlights the need for fault-tolerant streaming and persistence for ingestion, and embeds data feed management into its big data stack to achieve higher performance than consolidating separate systems for stream processing (Storm) and persistent storage (MongoDB)

   In the suggested method, this requirement is fulfilled by using a single streaming ETL system for streaming and storage with multiple guarantees that include fault tolerance.
2. Conventional approach is to use filter based tools to periodically ingest large batches of new data from operational systems into backend data warehouses. There are also approaches using micro-batch ETL to maintain a more up-todate data warehouse. However fine-granular ETL comes with consistency challenges. In this approach, ETL system is the main source

of updates to the warehouse and the OLAP system takes care of the query requests.

In the second alternative approach, it is difficult to enable a true “real-time” analytics capability. However, the architecture proposed allows queries to have access to the most recent data in the ETL pipeline in addition to the warehouse data, with more comprehensive consistency guarantees since the ETL pipeline is placed on top of an in-memory transactional stream processing system.

3. Shen et al. propose a stream-based distributed data management architecture for IoT applications that is a three layer (edge-fog-cloud) architecture with main emphasis on embedding lightweight stream processing on network devices located at the edge layer with support for various types of window joins that can address the disorder and time alignment issues common in IoT streams.

   The proposed architecture emphasizes on the data collection layer instead and does not focus on lightweight stream processing on network devices on the edge layer.

4. KDB+ is a commercial, column-oriented database based on the Q vector programming language that is proprietary and highly specialized to the financial domain. The proposed architecture is suitable for IoT applications that the KDB+ does not cover.

   InfluxDB, Gorilla and OpenTSDB provide valuable insight into time-series databases but do not meet ingestion requirements and are not appropriate for the specific problem in the IoT domain that the given paper considers.

12 Propose 2 questions about the paper to discuss in class

Is it sufficient to prove successful results of one experiment to denote that the proposed methodology is successful?

The paper mentions in several places that a certain part is left for future work. Are these not considered limitations of the paper? For example: ”We leave cross-system replication to future work.”

13 Follow-up research ideas

For the IoT proof of concept using the MIMIC II dataset, the result of the experiment strongly relies on the validity of the MIMIC II dataset and is therefore privy to its limitations. The experiment is also run particularly on one use case and does not guarantee results or prove that same results can be expected for all other use cases in the IoT domain.