Summary of Realtime data processing in Facebook

Wen Jen Hsieh(wenjenh), Eric Hsin(erhsin), Kevin Chen(kevche)

Problem and Motivation

Realtime data processing systems are used widely to pro- vide insights about events as they happen. A good design should consider the following properties when designing the system. * Ease of use * Performance * Fault-tolerance * Scalability * Correctness

In this paper, the author presented five decisions regarding these points

Hypothesis

The author stated that to achieve the company's requirements, the design of the streaming system should rely on a set of decisions. All of them are crucial in terms of not only the performance but also how fast and easy the development can be. Furthermore, among all decisions, the author stated that the tolerance of second-latency is the most important, which allows the system to tradeoff with other aspects rather than pure performance.

Solution Overview

The system can be decomposed into several parts with respect to its nature as listed.

- Scribe: A persistent, distributed messaging system loading all streaming dataflow.
- Puma: A system provides precomputed results for small queries and scribe filtering.
- Swift: A system that provides checkpointing in Scribe.
- Stylus: A high performance streaming engine that can handle imperfect ordering.
- Laser: A key-value storage system that relays hive query to streaming tools.
- Scuba: A slice-and-dice analysis data store.
- Hive: An exaByte level storage. Updated and partitioned by day.

Design Choices

Various choices were made with respect to the intended usages. One can see how each design choices affects the quality of the system in this table.

Language Paradigm

Choices : Declarative, Functional, Procedural

- Declarative: easy to use, while being less flexible
- Functional: easy to use, while more controllable
- Procedural: hardest to use, while has the most flexibility

Use at Facebook:

The author stated that Facebook currently doesn't support any functional programming paradigm while they had started to explore Spark. * Puma : written in SQL, chosen for ease of deployment. * Swift : written in Python, suitable for low-throughput processing. * Stylus : written in C++ for high performance

Data Transfer

Choices: Direct message, Broker based message, Persistent storage based message

- Direct message transfer:
  - The advantage of the method is speed (tens of milliseconds end-to-end latency)
  - Handling failure and maintaining persistency is not easy

- Broker based message transfer
  - Provides intermediate process between nodes, allowing better management
  - Suffers from performance overhead of broker's workload

- Persistent storage based message transfer

    - Processor is connected by a persistent message bus so it can be recovered faster
    - Write and read could be at different speed

Use at Facebook: Scribe

Facebook use Scribe to connect processing nodes. Scribe is a persistent message bus which achieves the pros below,

- Fault tolerance: the recovery from failure is faster
- Ease of use: we can reproduce the problem easily by reading the same input. It is useful for debugging. Plus, by checking the consumer of the input, the system is able to monitor and alert for delays in processing streams
- Performance: when the speed of a node is slow, we can simply move the jobs a new machine, while the previous node would not be affected
- scalability: we can scale easily by changing the number of buckets per category

Processing Semantics

The processing semantic of a streaming system denotes how the system determines its correctness and fault tolerance. There are three types of activities in a streaming system,

- Input events: the incoming data flow of a node
- Generate outputs: the outgoing data flow from a node toward another
- Save checkpoints: saving current conditions of a node, where i-memory state of the processing node, current offset in the input stream, and output values are involved

Choices: At-least-once, At-most-once, Exactly-once

To implement all of the activities, the author further separates them into two relevant semantics and three choices of implementation as below,

- State semantics: determines how many times each input event should be counted
    - At-least-once: save the in-memory state, then input offset
    - At-most-once: save the input offset, then in-memory sate
    - Exactly-once: save the in-memory state and the offset atomically

- Output semantics: determines how many times each output value should be counted
    - At-least-once: emit outputs, then checkpoint the offset and in-memory state
    - At-most-once: checkpoint the offset and in-memory state, then emit output
    - Exactly-once: checkpoint the offset, in-memory state, and emit output atomically

Mostly, at-least-once is much preferable when a system requires low latency and handle small amounts of duplication. At-most-once is desirable when data loss is preferable to data duplication. Exactly-once is more deterministic while suffers from performance penalty and database limitations.

Used at Facebook

Different components of the streaming system have different semantics according to their expected behavior as listed below:

- Scuba: at-most-once output semantic is applied since the data ingestions are stateless (no state semantic), and query results from may be based on only a portion of the data.
- Stylus: offers all the options, as the system is meant to be the most flexible

The author further stated that a system like Stylus utilized side-effect-free work, which pipelines checkpointing and input deserialization

State-saving Mechanisms

State saving of streaming system can affect a lot in its fault tolerance and performance. One can see the multiple implementing choices as below,

Choices: Replication, Local database, Remote database, Upstream backup, Global consistence

- Replication: nodes are replicated, causing higher cost on hardware
- Local database: states are stored in a local database while mutations remote
- Remote database: both states and checkpoints are stored in a remote database
- Upstream backup: states are backuped in the upstream node so can one be replayed
- Global consistent snapshot: states are stored as a consistent, global snapshot

Used at Facebook

If the states are small enough, the system caches locally and copies asynchronously to HDFS. Otherwise, the node keeps a copy of states at the remote database in one of the following pattern depending on the system: * Read-modify-save * Read-merge-write * Append-only

Backfill Processing

The author stated that we often need to reprocess old data. For example, we sometimes need to test a new application against old data, re-run application when new metrics are added or bugs are discovered.

Choices: Stream only, Stream with batch, Stream on batch

There are a few approaches to reprocessing data,

- Stream only: requires retention that is long enough to replay streams
- Stream with batch: processes old data on batch, hard to maintain the consistency
- Stream on batch: streams that can be processed on batch, such as Spark Streaming

Used at Facebook

While Scribe stores short-term data, long-term data is stored in Hive. The replay of old data is read via MapReduce and run in the batch environment.

- Puma: an application can use the identical code on stream and batch
- Stylus: an application generates two binaries for both stream and batch
- Swift: an application can only be stateless

Limitations and Possible Improvements

- The real-time processing system has an assumption of latency of a few seconds in favor of other design requirment. Applications required faster process (such as a credit card transaction system) will be hard to achieve
- Though the framework provides various processing semantics for developers to choose, most of them are at-least-once and at-most-once semantics. Exactly-once semantics is costly due to the atomicity, but can still be vital in some case.
- Facebook has implemented two state-saving mechanisms:
    - Local database: if the system crashes before sending the data to HDFS, one can only recover the data after the checkpoint
    - Remote database: the timing of merging is critical for maintaining data persistence

- Possible Improvements
    - Automatic monitor: lag alerts on Puma and Stylus apps for the teams that use them
    - Scale system infrastructure: improvement of dynamic stream load balancing

Comparison between Systems

Here we will compare Facebook streaming systems with Twitter's framework, Heron. Please refer to the comparison table.

- Heron uses procedural language (JAVA) for higher performance. However, Facebook has several systems which provide more flexibility for the teams.

- Heron also chooses to apply broker based message data transfer instead of persistent storage message transfer. It uses a Stream Manager between the Heron instances to handle the stream process, and ended up spending much time discussing the details of their backpressure mechanism.

- Heron doesn't provide the exactly-once semantics.

Summary of Class Discussion

We discussed about different design paradigm regarding system design, and the several choices made based on different use-cases in Facebook. Furthermore, the speaker proposed several viable choices for audience if they were to build their own big data system. In the discussion about Heron, we examined the limitation of Storm and twitter's choice to build their own system for better performance.

When designing large-scale data-processing systems, ease-of-use should be valued as much as performance while it is usually neglected in the academia. These aspects should be taken into account since they are imperative for systems to be adopted by the community.