Engineering
Back

Infrastructure

Processing billions of events in real time at Twitter

By
Lu Zhang
and
Chukwudiuto Malife
Friday, 22 October 2021

At Twitter (https://www.twitter.com), we process approximately 400 billion events in real time and generate petabyte (PB) scale data every day. There are various event sources we consume data from, and they are produced in different platforms and storage systems, such as Hadoop, Vertica, Manhattan distributed databases, Kafka, Twitter Eventbus, GCS, BigQuery, and PubSub.

To process those types of data in those sources and platforms, the Twitter Data Platform team has built internal tools like Scalding for batch processing, Heron for streaming, an integrated framework called TimeSeries AggregatoR (TSAR) for both batch and real-time processing, and Data Access Layer for data discovery and consumption. However, with data growing rapidly, the high scale is still challenging the data infrastructure that engineers use to run pipelines. For example, we have an interaction and engagement pipeline which processes high-scale data in batch and real time. As our data scale is growing fast, we face high demands to reduce streaming latency and provide higher accuracy on data processing, as well as real-time data serving.

For the interaction and engagement pipeline, we collect and process data from various real-time streams and server and client logs, to extract Tweet and user interaction data with various levels of aggregations, time granularities, and other metrics dimensions. That aggregated interaction data is particularly important and is the source of truth for Twitter's ads revenue services and data product services to retrieve information on impression and engagement metrics. In addition, we need to guarantee fast queries on the interaction data in the storage

systems with low latency and high accuracy across different data centers. To build such a system, we split the entire workflow into several components, including pre-processing, event aggregation, and data serving.

Old architecture

The old architecture is shown below. We have a lambda architecture with both batch and real-time processing pipelines, built within the Summingbird Platform and integrated with TSAR. For more information on lambda architecture see What is Lambda Architecture? (https://databricks.com/glossary/lambda-architecture) The batch component sources are Hadoop logs, such as client events, timeline events, and Tweet events, stored on Hadoop Distributed File System (HDFS). We built several Scalding pipelines to preprocess the raw logs and ingest them into Summingbird Platform as offline sources. The real-time component sources are Kafka topics.

The real-time data is stored in the Twitter Nighthawk distributed cache, and batch data is stored in Manhattan distributed storage systems. We have a query service to access the real-time data from both stores, used by customer services. Currently, we have real-time pipelines and query services in 3 different data centers. To reduce batch computing cost, we run batch pipelines in one data center and replicate the data to the other 2 data centers.

Existing challenges

Because of the high scale and high throughput of data we process in real time, there can be data loss and inaccuracy for real-time pipelines. For Heron topology, in cases where there are more events to process and the Heron bolts cannot process in time, there is back pressure within the topology. Also, the Heron bolts can become slow because of high garbage collection cost.

When the system is under back pressure for a long time, the Heron bolts can accumulate spout lag which indicates high system latency. Usually when that happens, it takes a very long time for the topology lag to go down. More often, as seen in our Heron pipeline, there are also many Heron Stream Manager deaths (Stream Manager manages the routing of tuples between topology components), and the lag keeps going up.

The current operational solution is to restart Heron containers to bring up the stream managers, so that bolts can restart processing streams. This can cause event loss during the operation, which could lead to inaccuracy in the aggregated counts in Nighthawk storage.

For the batch component, we built several heavy computation pipelines processing PB scale of data and running hourly to sink data into Manhattan. The centralized TSAR query service consolidates both Manhattan and Nighthawk data to provide data serving for customer services. Because of potential loss of real-time data, the TSAR service might provide less aggregated metrics to our customers.

To overcome this data loss issue, reduce system latency, and optimize the architecture, we propose to build pipelines in kappa architecture to process the events in streaming-only mode. For more information on kappa architecture, see What is the Kappa Architecture? (https://hazelcast.com/glossary/kappa-architecture/) In that solution, we remove batch components and rely on real-time components to provide low latency and high accuracy of data, which simplifies the architecture and removes computation cost in the batch pipelines.

New architecture on Kafka and dataflow

The new architecture is built on both Twitter Data Center services and Google Cloud Platform. On-premise, we built preprocessing and relay event processing which converts Kafka topic events to pubsub topic events with at-least-once semantics. On Google Cloud, we used streaming Dataflow jobs to apply deduping and then perform real-time aggregation and sink data into BigTable.

For the first step, we built several event migrators as preprocessing pipelines, which do transformation and re-mapping fields, then send events to a Kafka topic. We created those streaming pipelines using our internal customized streaming framework based on Kafka for at-least-once semantics. As the second step, we build Event Processors to stream events with at-least-once semantics. The Event Processors handle transformation to Pubsub event representation and generate event context consisting of UUID and other meta information related to processing context. The UUID is used for deduping by the Dataflow workers in the downstream. We apply an almost-infinite-retry setting for the internal Pubsub Publisher to achieve at-least-once for sending messages from Twitter Data Centers to Google Cloud. After the new Pubsub representation event is created, the Event Processors send the events to the Google Pubsub topic.

On Google Cloud, we use a Twitter internal framework built on Google Dataflow for real time aggregation. The Dataflow workers handle deduping and aggregation in real time. The deduping procedure accuracy is dependent on the timed window. We tuned our system to achieve best effort deduping in the deduping window. We proved that high deduping accuracy by simultaneously

writing data into BigQuery and querying continuously on the percentage of duplicates, explained below. Lastly, the aggregated counts with query keys are written to Bigtable.

For the serving layer, we use the Twitter internal LDC query service with front-end in Twitter Data Centers and different back-ends such as Bigtable and BigQuery. The whole system can stream millions of events per second with a low latency of up to ~10s and can scale up with high traffic in both our on-prem and cloud streaming systems. We use Cloud Pubsub as a message buffer while guaranteeing no data loss throughout our on-prem streaming system. This is followed by deduping to achieve near exactly-once processing.

This new architecture saves the cost to build the batch pipelines, and for real-time pipelines, we are able to achieve higher aggregation accuracy and stable low latency. Also, we do not need to maintain different real-time event aggregations in multiple data centers.

Evaluation

System Performance Evaluation

Below is the table for metrics comparison between the two architectures. The new architecture provides lower latency compared with Heron topology in the old architecture and offers higher throughput. Also, the new architecture handles late events counting and does not lose events when doing real-time aggregation. Moreover, there is no batch component in the new architecture, so it simplifies the design and reduces compute cost which existed in the old architecture.

Table 1. System performance comparison for old and new architectures.

Aggregation counts validation

We split the counts validation process into two steps. First, we evaluated the duplicate event percentages inside Dataflow before and after deduping. Second, we directly compared counts from both original TSAR batch pipelines and the counts from Dataflow after deduping, for all keys.

For the first step, we created a separate Dataflow pipeline to export raw events before deduping directly from Pubsub into BigQuery. Then, we created scheduled queries to query counts continuously with time. At the same time, we created another Dataflow pipeline to export deduped event counts to BigQuery. In this way, we are able to see the duplicate event percentages and the percentage change after deduping.

For the second step, we built a validation workflow in which we exported the deduped and aggregated data into BigQuery and loaded data generated from original TSAR batch pipelines from Twitter Data Center to BigQuery on Google Cloud. This is so that we could run scheduled queries to compare the counts for all keys.

We are able to have >95% exact match with the batch processing data for our Tweet interaction stream. We investigated the <5% discrepancy and found out it was mainly because the original TSAR batch pipelines discard late events, which were captured by our new stream pipelines. That further proves our current system yields higher accuracy.

Conclusion

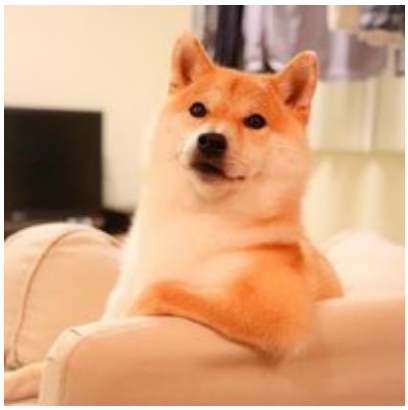
With migrating the old architecture built on TSAR to the hybrid architecture on both Twitter Data Center and Google Cloud Platform, we are able to process billions of events in real time and achieve low latency, high accuracy, stability, architecture simplicity, and reduced operation cost for engineers.

Next steps

For the next steps, we will make Bigtable datasets resilient to region failures and migrate our customers to onboard the new LDC query server.

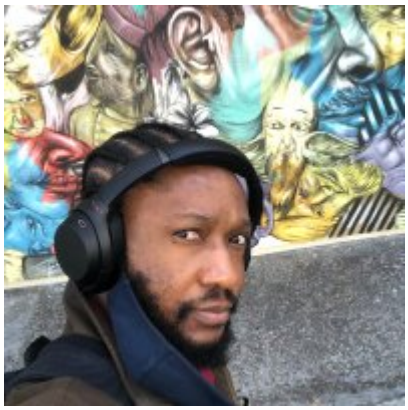
Acknowledgement

We would like to thank the following contributors and reviewers for their work on this blog: Naveen Tirupattur (https://twitter.com/ntirupattur), Kanishk Karanawat (https://twitter.com/kkdoon), Prasad Wagle (https://twitter.com/prasadwagle), Huijun Wu (https://twitter.com/fatj2020), Yao Li, Weipu Zhao (https://twitter.com/WeipuZ), Shailendra Toutam, Wilmin Martono, David Jeong (https://twitter.com/davidjjeong), Anisha Nagarajan (https://twitter.com/NagarajanAnisha), Maggie Shi (https://twitter.com/maggieeshi), Cary Wun (https://twitter.com/cwun), Srikanth Thiagarajan (https://twitter.com/Srikantht).



(https://www.twitter.com/zhlu2006)

Lu Zhang



(https://www.twitter.com/Chuk_Almighty)

Chukwudiuto Malife