Unified Streaming And Batch Pipelines At LinkedIn Using Beam

Shangjin Zhang
Yuhong Cheng

July, 2022
Hi, I am Shangjin

- CE Master @ Columbia University
- 4 Years @ Bloomberg
- 3 Years working in AI Infra @ LinkedIn
Hi, I am Yuhong

- CS Master @ Rice University
- Intern @ IBM Austin
- 3 Years working in Streaming Infra @ LinkedIn
Agenda

1. Background: Standardization Pipelines & Backfilling
2. Problem: Backfilling Issues
3. Solution: Unified Pipelines
4. Outcome: Performance Gains
Standardization

Convert user input information into a set of pre-defined IDs

Widely used for search / model training and etc

A heavy process with NLP and deep learning models

Raw company: "LinkedIn"

LinkedIn (0.96) urn:li:company:1234

Industry (name:category)

Internet:Tech (1) urn:li:industry:1
Original Architecture

Kappa

- **Real-Time**
  - 100+ parallel streaming pipelines
  - 200/sec throughput
  - Apache Beam & Apache Samza

- **Backfilling**
  - Exactly same streaming pipelines
  - Deployed temporarily
  - 830 million member profiles
  - 40,000/sec throughput
Agenda

1. Background: Standardization Pipelines & Backfilling
2. Problem: Backfilling Issues
3. Solution: Unified Pipelines
4. Outcome: Performance Gains
Backfilling Issues

Heavy load leads to long backfilling time

- Hours to days turnaround time
- Complex model can't finish within reasonable time

Hard to scale

- Model iterates weekly instead of quarterly now
- Streaming cluster is not optimized for spiky resource footprint
- Can only host 3 concurrent backfillings
(More) Backfilling Issues

Impact on other systems
- Flood lookup tables
- Noisy neighbor to co-located streaming pipelines

Operational overhead
- Need to monitor and stop the backfilling manually

Motivation
- Run backfilling as batch job (lambda architecture)
1. Background: Standardization Pipelines & Backfilling
2. Problem: Backfilling Issues
3. Solution: Unified Pipelines
4. Outcome: Performance Gains
Dropped Solution: Two Codebases

Real-Time/streaming Codebase

Backfilling/batch Codebase

deploy

Samza Cluster

deploy

Spark Cluster

image: by Yuhong Cheng

I JUST CAN'T HANDLE LIFE TODAY
Unified Architecture

- Samza Cluster
- Computing Engine
- Unified Pipeline

Single Beam Codebase

Target: streaming
Deploy

Target: batch
PipelineOptions pipelineOpts = PipelineOptionsFactory.fromArgs(args).create();
Pipeline pipeline = Pipeline.create(pipelineOpts);

pipeline.apply(ProfileData.read())
  .apply(Joins.tableJoin(sideTable))
  .apply(Standardizer())
  .apply(Result.write());
Unified IO

pipeline.apply(ProfileData.read())
  .apply(Joins.tableJoin(sideTable))
  .apply(Standardizer())
  .apply(Result.write());
Unified PTransform

A special PTransform that provides a unified interface to users but allows different implementations according to pipeline type.

```java
public static class Read extends UnifiedPTransform<PBegin, PCollection<String>> {
    @Override
    protected PCollection<String> expandStreaming(PBegin pBegin) {
        return pBegin.getPipeline()
            .apply(KafkaIO.<String>read()
                .withTopic(getStreamingInput()))
            .apply(...);
    }

    @Override
    protected PCollection<String> expandBatch(PBegin pBegin) {
        return pBegin.getPipeline()
            .apply(FileIO.match().filepattern(getBatchInput()))
            .apply(...);
    }
}
```
Unified Table Join

pipeline.apply(ProfileData.read())
  .apply(Joins.tableJoin(sideTable))
  .apply(Standardizer())
  .apply(Result.write());

Provide options to do join based on the table type to avoid unnecessary data shuffling.

Typically
- streaming => key lookup
- batch => coGroupByKey
1. **Background:** Standardization Pipelines & Backfilling

2. **Problem:** Backfilling Issues

3. **Solution:** Unified Pipelines

4. **Outcome:** Performance Gains
Benchmarks

Backfilling Computation Resources

- **Total Memory Allocated (GB-Hours)**
  - Streaming: 5760
  - Batch: 2043

- **Total CPU Time (Hours)**
  - Streaming: 3840
  - Batch: 1761
(More) Benchmarks

Backfilling Duration

- Streaming
- Batch

Duration (Mins)

- 25
- 450
Wins

Dev productivity
Write code **ONCE**
run everywhere

Faster
Saved 94%
processing time

Resources
Used ~50% less
cpu time and memory

Cost to Serve
Reduced ~11X cost
Future Works

Python Support

More Use Cases

Other Runners
Thank you