Processing over a trillion events a day

CASE STUDIES IN SCALING STREAM PROCESSING AT LINKEDIN
Processing over a trillion events a day

CASE STUDIES IN SCALING STREAM PROCESSING AT LINKEDIN

Jagadish Venkatraman
Sr. Software Engineer, LinkedIn
Apache Samza committer
jagadish@apache.org
<table>
<thead>
<tr>
<th></th>
<th>Today’s Agenda</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Stream processing scenarios</td>
</tr>
<tr>
<td>2</td>
<td>Hard problems in stream processing</td>
</tr>
<tr>
<td>3</td>
<td>Case Study 1: LinkedIn’s communications platform</td>
</tr>
<tr>
<td>4</td>
<td>Case Study 2: Activity tracking in the News feed</td>
</tr>
</tbody>
</table>
Data processing latencies

- Synchronous: 0 millis
- Batch Processing: Later. Possibly much later.

RPC/Databases vs. Batch Processing: Response latency
Data processing latencies

- **Synchronous**
  - 0 millis
  - Milliseconds to minutes

- **RPC/Databases**

- **Stream Processing**

- **Batch Processing**

**Response latency**

Later. Possibly much later.
Some stream processing scenarios at LinkedIn

Security

Real-time DDoS protection for members
Some stream processing scenarios at LinkedIn

Security
- Real-time DDoS protection for members

Notifications
- Notifications to members
Some stream processing scenarios at LinkedIn

- **Security**: Real-time DDoS protection for members
- **Notifications**: Notifications to members
- **News classification**: Real-time topic tagging of articles
Some stream processing scenarios at LinkedIn

- **Security**: Real-time DDoS protection for members
- **Notifications**: Notifications to members
- **News classification**: Real-time topic tagging of articles
- **Performance monitoring**: Real-time site-speed profiling by facets
Some stream processing scenarios at LinkedIn

- **Security**: Real-time DDoS protection for members
- **Notifications**: Notifications to members
- **News classification**: Real-time topic tagging of articles
- **Performance monitoring**: Real-time site-speed profiling by facets
- **Call-graph computation**: Analysis of service calls
Some stream processing scenarios at LinkedIn

Ad relevance

Tracking ads that were clicked
Some stream processing scenarios at LinkedIn

Ad relevance

Aggregates

Tracking ads that were clicked

Aggregated, real-time counts by dimensions
Some stream processing scenarios at LinkedIn

- **Ad relevance**: Tracking ads that were clicked
- **Aggregates**: Aggregated, real-time counts by dimensions
- **View-port tracking**: Tracking session duration
Some stream processing scenarios at LinkedIn

- **Ad relevance**: Tracking ads that were clicked
- **Aggregates**: Aggregated, real-time counts by dimensions
- **View-port tracking**: Tracking session duration
- **Profile standardization**: Standardizing titles, gender, education
AND MANY MORE SCENARIOS IN PRODUCTION...

200+ Applications
AND MANY MORE SCENARIOS IN PRODUCTION...

200+ Applications

Powered By

 Powered By
• A distributed stream processing framework

• 2012: Development at LinkedIn
• Used at Netflix, Uber, Tripadvisor and several large companies.

• 2013: Apache incubator
• 2015 (Jan): Apache Top Level Project
Today's agenda

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Stream processing scenarios</td>
</tr>
<tr>
<td>2</td>
<td>Hard problems in stream processing</td>
</tr>
<tr>
<td>3</td>
<td>Case Study 1: LinkedIn's communications platform</td>
</tr>
<tr>
<td>4</td>
<td>Case Study 2: Activity tracking in the news feed</td>
</tr>
</tbody>
</table>
Hard problems in stream processing

- **Scaling processing**
  - Partitioned streams
  - Distribute processing among all workers

- **State management**
  - Hardware failures are inevitable
  - Efficient check-pointing
  - Instant recovery

- **High performance remote I/O**
  - Need primitives for supporting remote I/O

- **Heterogeneous deployment models**
  - Running on a multi-tenant cluster
  - Running as an embedded library
  - Unified API for batch and real-time data
Partitioned processing model

- Hadoop
- Kafka
- DB change capture

Samza application

- Hadoop
- Kafka
- Elastic Search
- Serving stores
Partitioned processing model

- Each task processes one or more partitions
- The Samza master assigns partitions to tasks and monitors container liveness
- Scaling by increasing # of containers

Samza application

- Container-1
  - Task-1
  - Task-2
- Container-2
  - Task-3

Heartbeat

Samza master

Hadoop

Kafka

DB change capture
Hard problems in stream processing

**Partitioned processing**
- Partitioned streams
- Distribute processing among all workers

**State Management**
- Hardware failures are inevitable
- Instant recovery
- Efficient Check-pointing

**Efficient remote data access**
- Provides primitives for supporting efficient remote I/O.

**Heterogeneous deployment models**
- Samza supports running on a multi-tenant cluster
- Samza can also run as a light-weight embedded library
- Unified API for batch and streaming
Why local state matters?

- Stateless versus Stateful operations

Use cases of Local state
- Temporary data storage
- Adjunct data lookups

Advantages
- Richer access patterns, faster than remote state
• Querying the remote DB from your app versus local embedded DB access
• Samza provides an embedded fault-tolerant, persistent database.
Architecture for Adjunct lookups

- Querying the remote DB from your app versus change capture from database.
- Turn remote I/O to local I/O
- Bootstrap streams
• Querying the remote DB from your app versus change capture from database.

• Bootstrap and partition the remote DB from the stream

100X Faster 1.1M TPS
Key optimizations for scaling local state

- Changelog
  1. Fault tolerant, replicated into Kafka.
  2. Ability to catch up from arbitrary point in the log.
  3. State restore from Kafka during host failures
Speed thrills but can also KILL.
Key optimizations for scaling local state

- Incremental state check-pointing

1. Re-use on-disk state snapshot (host affinity)
2. Write on-disk file on host at checkpoint
3. Catch-up on only delta from the Kafka change-log
Key optimizations for scaling local state

- Incremental state check-pointing
  1. Re-use on-disk state snapshot (host affinity)
  2. Write on-disk file on host at checkpoint
  3. Catch-up on only delta from the Kafka change-log

- Durability and host mapping
  - 1.2 TB State
  - 60x Faster than full checkpointing
  - 0 Downtime during restarts and recovery
Key optimizations for scaling local state

**Drawbacks of local state**

1. Size is bounded
2. Some data is not partitionable
3. Repartitioning the stream messes up local state
4. Not useful for serving results
<table>
<thead>
<tr>
<th>Hard problems in stream processing</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Scaling processing</strong></td>
</tr>
<tr>
<td>• Partitioned streams</td>
</tr>
<tr>
<td>• Distribute processing among all workers</td>
</tr>
<tr>
<td><strong>State management</strong></td>
</tr>
<tr>
<td>• Hardware failures are inevitable</td>
</tr>
<tr>
<td>• Efficient check-pointing</td>
</tr>
<tr>
<td>• Instant recovery</td>
</tr>
<tr>
<td><strong>High performance remote I/O</strong></td>
</tr>
<tr>
<td>• Need primitives for supporting remote I/O</td>
</tr>
<tr>
<td><strong>Heterogeneous deployment models</strong></td>
</tr>
<tr>
<td>• Running on a multi-tenant cluster</td>
</tr>
<tr>
<td>• Running as an embedded library</td>
</tr>
<tr>
<td>• Unified API for batch and real-time data</td>
</tr>
</tbody>
</table>
Why remote I/O matters?

1. Writing to a remote data store (eg: CouchDB) for serving
2. Some data is available only in the remote store or through REST
3. Invoking down-stream services from your processor
Scaling remote I/O

Hard problems

1. Parallelism need not be tied to number of partitions

2. Complexity of programming model

3. Application has to worry about synchronization and check-pointing
Scaling remote I/O

Hard problems

1. Parallelism need not be tied to number of partitions

2. Complexity of programming model

3. Application has to worry about synchronization and check-pointing

How we solved them?

1. Support out-of-order processing within a partition

2. Simple callback-based async API. Built-in support for both multi-threading and async processing

3. Samza handles checkpointing internally
Performance results

Experiment setup

- PageViewEvent topic - 10 partitions
- For each event, remote lookup of the member’s inbox information (latency: 1ms to 200ms)
- Single node Yarn cluster, 1 container (1 CPU core), 1GB memory.
Performance results

Experiment setup

- PageViewEvent topic - 10 partitions
- For each event, remote lookup of the member’s inbox information (latency: 1ms to 200ms)
- Single node Yarn cluster, 1 container (1 CPU core), 1GB memory.

10x Faster
With In-order processing

40x Faster
Out-of-order processing

At our scale this is HUGE!
Hard problems in stream processing

**Scaling processing**
- Partitioned streams
- Distribute processing among all workers

**State management**
- Hardware failures are inevitable
- Efficient checkpointing
- Instant recovery

**High performance remote I/O**
- Need primitives for supporting remote I/O

**Heterogeneous deployment models**
- Running on a multi-tenant cluster
- Running as an embedded library
- Unified API for batch and real-time data
public interface StreamTask {
    void process(IncomingMessageEnvelope envelope, MessageCollector collector, TaskCoordinator coordinator) {
        // process message
    }
}

public class MyApp implements StreamApplication {
    void init(StreamGraph streamGraph, Config config) {
        MessageStream&lt;PageView&gt; pageViews = ..;
        pageViews.filter(myKeyFn)
            .map(pageView -> new ProjectedPageView())
            .window(Window.keyedTumblingWindow(keyFn, Duration.ofSeconds(30))
            .sink(outputTopic);
    }
}
Heterogeneity – Different deployment models

Samza on multi-tenant clusters

• Uses Yarn for coordination, liveness monitoring
• Better resource sharing
• Scale by increasing the number of containers

Samza as a light-weight library

• Purely embedded library: No Yarn dependency
• Use zoo-keeper for coordination, liveness and partition distribution
• Seamless auto-scale by increasing the number of instances

```java
StreamApplication app = new MyApp();
ApplicationRunner localRunner = ApplicationRunner.getLocalRunner(config);
localRunner.runApplication(app);
```
Heterogeneity – Support streaming and batch jobs

Samza on Hadoop

• Supports re-processing and experimentation
• Process data on hadoop instead of copying over to Kafka ($$)
• Job automatically shuts-down when end-of-stream is reached

Samza on streaming

• World-class support for streaming inputs

EXACT SAME CODE
Today’s agenda

1. Stream processing scenarios
2. Hard problems in stream processing
3. Case Study 1: LinkedIn’s communications platform
4. Case Study 2: Activity tracking in the news feed
5. Conclusion
ATC GOAL:

Craft a clear, consistent conversation between our members and their professional world
Features

Channel selection

- “Don’t blast me on all channels”
- Route through Inmail, email or notification in app
**Features**

**Channel selection**
- “Don’t blast me on all channels”
- Route through Inmail, email or notification in app

**Aggregation / Capping**
- “Don’t flood me, Consolidate if you have too much to say!”
- “Here’s a weekly summary of who invited you to connect”
Features

Channel selection

- “Don’t blast me on all channels”
- Route through Inmail, email or notification in app

Aggregation / Capping

- “Don’t flood me, Consolidate if you have too much to say!”
- “Here’s a weekly summary of who invited you to connect”

Delivery time optimization

- “Send me when I will engage and don’t buzz me at 2AM”
Features

Channel selection
- “Don’t blast me on all channels”
- Route through Inmail, email or notification in app

Aggregation / Capping
- “Don’t flood me, Consolidate if you have too much to say!”
- “Here’s a weekly summary of who invited you to connect”

Delivery time optimization
- “Send me when I will engage and don’t buzz me at 2AM”

Filter
- “Filter out spam, duplicates, stuff I have seen or know about”
Why Apache Samza?

### Requirements
1. Highly Scalable and distributed
2. Multiple sources of email
3. Fault tolerant state (notifications to be scheduled later)
4. Efficient remote calls to several services

### How Samza fits in?
1. Samza partitions streams and provides fault-tolerant processing
2. Pluggable connector API (Kafka, Hadoop, change capture)
3. Instant recovery and incremental checkpointing
4. Async I/O support
Why Apache Samza?

Requirements

1. Highly Scalable and distributed
2. Multiple stream joins
3. Fault tolerant state (notifications to be scheduled later)
4. Efficient remote calls to several services
The repeated blue in the color palette is intentionally designed to limit the use of colors outside the LinkedIn brand.
Inside each ATC Task
## Today’s agenda

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Stream Processing Scenarios</td>
</tr>
<tr>
<td>2</td>
<td>Hard Problems in Stream Processing</td>
</tr>
<tr>
<td>3</td>
<td>Case Study 1: LinkedIn’s communications platform</td>
</tr>
<tr>
<td>4</td>
<td>Case Study 2: Activity tracking in the news feed</td>
</tr>
</tbody>
</table>
Activity tracking in News feed

HOW WE USE SAMZA TO IMPROVE YOUR NEWS FEED..
Precise screen change for iPhone:
Select screen and go to inspector. Under the Arrange tab, note Size (14.07" x 7.91") and note Position (3.42", .39" — from top left corner)
Make sure your new screen matches those numbers.

ACTIVITY TRACKING: GOAL

Power relevant, fresh content for the LinkedIn Feed
Precise screen change for laptop:
Select screen and go to inspector. Under the Arrange tab, note Size (10.81 x 16.03) and note Position (-0.6, 2.23 - top left corner). Make sure your new screen matches those numbers.

Server-side tracking event
Feed Server
Precise screen change for laptop:
Select screen and go to inspector. Under the Arrange tab, note Size (10.81 x 16.03) and note Position (-0.6, 2.23 – top left corner). Make sure your new screen matches those numbers.

Server-side tracking event
Feed Server
Precise screen change for laptop:
Select screen and go to inspector. Under the Arrange tab, note Size (10.81 x 16.03) and note Position (-0.6, 2.23 - top left corner).

Make sure your new screen matches those numbers.

Server-side tracking event

```
{
  "paginationId":
  "feedUpdates": [{
    "updateUrn": "update1"
    "trackingId":
    "position":
    "creationTime":
    "numLikes":
    "numComments":
    "comments": [
      {"commentId": }
    ]
  }, {
    "updateUrn": "update2"
    "trackingId":
    ..
  }
}
```

Rich payload
Precise screen change for laptop:

Select screen and go to inspector. Under the Arrange tab, note Size (10.81 x 16.03) and note Position (-0.6, 2.23 – top left corner). Make sure your new screen matches those numbers.
Precise screen change for laptop:
Select screen and go to inspector. Under the Arrange tab, note Size (10.81 x 16.03) and note Position (-0.6, 2.23 – top left corner)
Make sure your new screen matches those numbers.

Client-side tracking event
• Track user activity in the app
• Fire timer during a scroll or a viewport change
Precise screen change for laptop:
Select screen and go to inspector. Under the Arrange tab, note Size (10.81 x 16.03) and note Position (-0.6, 2.23 – top left corner)
Make sure your new screen matches those numbers.

Client-side tracking event

```
"feedImpression": {
  "urn":
  "trackingId":
  "duration":
  "height":
}, {
  "urn":
  "trackingId":
}
```

- Light pay load
- Bandwidth friendly
- Battery friendly
Join client-side event with server-side event

```
"FEED_IMPRESSION": {
  "trackingId": "abc"
  "duration": "100"
  "height": "50"
}, {
  "trackingId":
},
"FEED_SERVED": {
  "paginationId":
  "feedUpdates": [{
    "updateUrn": "update1"
    "trackingId": "abc"
    "position":
    "creationTime":
    "numLikes":
    "numComments":
    "comments": [
      {"commentId": }
    ]
  }, {
    "updateUrn": "update2"
    "trackingId": "def"
    "creationTime":
  }
}
```
Samza Joins client-side event with server-side event

```
"FEED_IMPRESSION": {
    "trackingId": "abc"
    "duration": "100"
    "height": "50"
    },
    "trackingId":
    },

"FEED_SERVED": {
    "paginationId":
    "feedUpdates": [{
        "updateUrn": "update1"
        "trackingId": "abc"
        "position":
        "creationTime":
        "numLikes":
        "numComments":
        "comments": [
            {"commentId": }
            ],
        }, {
        "updateUrn": "update2"
        "trackingId": "def"
        "creationTime":
    }
    ]
```
Precise screen change for laptop:
Select screen and go to inspector. Under the Arrange tab, note Size (10.81 x 16.03) and note Position (-0.6, 2.23 – top left corner). Make sure your new screen matches those numbers.

Architecture

The repeated blue in the color palette is intentionally designed to limit the use of colors outside the LinkedIn brand.

To Downstream Ranking Systems

1.2 Billion Processed per-day
90 Containers Distributed, Partitioned

CLIENT EVENT

SERVER EVENT

FEED JOINER

Container-1
Task-1

Container-2
Task-2

Feed Server

To Downstream Ranking Systems
Recap

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Stream processing scenarios</td>
</tr>
<tr>
<td>2</td>
<td>Hard problems in stream processing</td>
</tr>
<tr>
<td>3</td>
<td>Case Study 1: LinkedIn’s communications platform</td>
</tr>
<tr>
<td>4</td>
<td>Case Study 2: Activity tracking in the news feed</td>
</tr>
</tbody>
</table>
Recap

SUMMARY OF WHAT WE LEARNT IN THIS TALK
Key differentiators for Apache Samza

- Samza
- GearPump
- Beam
- AWS Kinesis Analytics
- Kafka Streams
- Spark Streaming
- Azure Stream Analytics
- Flink
- Dataflow

Yes It is CROWDED !!

Not meant to be an accurate timeline..
Key differentiators for Apache Samza

- Stream Processing both as a multi-tenant service and a light-weight embedded library
- No micro batching (first class streaming)
- Unified processing of batch and streaming data
- Efficient remote calls I/O using built-in async mode
- World-class support for scalable local state
  - Incremental check-pointing
  - Instant restore with zero down-time
- Low level API and Stream based high level API (DSL)
Coming soon

- Event time, and out of order arrival handling
- Beam API integration
- SQL on streams