Re-introducing the Stream Processor

A Universal Tool for Continuous Data Analysis

Paris Carbone
Committer @ Apache Flink
PhD Candidate @ KTH
Data Stream Processors

val windowCounts = text.flatMap { w => w.split("\s") }
  .map { w => WordWithCount(w, 1) }
  .keyBy("word")
  .timeWindow(Time.seconds(5))
  .sum("count")

can set up any data pipeline for you
Data Stream Processors

A growing open-source ecosystem:

e.g. flink, kafka, beam, apex

General Idea of the tech:

- Processes pipeline computation in a cluster
- Computation is continuous and parallel (like data)
- Event-processing logic <-> Application state
- It’s production-ready and aims to simplify analytics

Is this really a step forward in data processing?
4 Aspects of Data Processing

- Event logs
- Complex event processing
  - Fast
  - Approximate
  - Streaming
- Production database
- "Microservices"
  - Application state
  - Failover
- Complex analytics
- Large-scale processing systems
- Data warehouses + Historical data
- Interactive queries
- Data science reports
- Dev
- User
- Analyst
- Data engineer
4 Aspects of Data Processing

- **Event Logs**
  - Complex event processing
  - Fast approximate streaming
  - ETL

- **Production Database**
  - Application state + failover
  - "Microservices"

- **Interactive Queries**
  - Complex analytics
  - Large-scale processing systems

- **Data Science Reports**
  - Data warehouses + historical data

- **User**
  - Data engineer

- **Developer**
  - Analyst
4 Aspects of Data Processing

1. Speed

- Event logs
- Complex event processing
- Fast approximate streaming
- ETL

- Rules
- "Microservices"
- Application state + failover

- Production database
- "Microservices"
- User

- Stream processor

- Complex analytics
- Large-scale processing systems
- Data engineer

- Interactive queries
- Analyst

- Data science reports
- Historical data

- Dev
- User

- Data engineer
- Analyst
1. Speed
Low-Latency Data Processing

Traditionally the **sole** reason stream processing was used

How do stream processors achieve **low latency**?

- **No** intermediate **scheduling** (you let it run)
- **No** physical **blocking** (pre-compute on the go)
- **Copy-on-write** for state and output

**CEP semantics etc.** are nowadays provided as additional libraries for stream processors

*But Is this is only relevant for live data?*
4 Aspects of Data Processing

1. Speed
   - complex event proc
   - fast approximate streaming
   - ETL
   - rules
   - event logs

2. History
   - complex analytics
   - large-scale processing systems
   - data engineer
   - data warehouses + historical data

- stream processor
- interactive queries
- data science reports
- analyst
- user
- application state + failover
- "microservices"
2. History

Offline Data Processing

It is possible and better over bulk historical data analysis.

What can stream processors do for historical data?

- Ability to define custom state to build up models
- Large-scale support is a given (inherits cluster computing benefits)
- Separation of notions of time and out-of-order processing

E.g.,

But isn’t streaming hard to deal with failures?
4 Aspects of Data Processing

1. Speed
   - complex event processing
   - fast approximate streaming
   - ETL
   - event logs

2. History
   - data warehouses + historical data
   - large-scale processing systems
   - complex analytics

3. Durability
   - production database
   - "microservices"
   - application state + failover

4. Interactive
   - queries
   - data science reports
   - analyst
   - data engineer

1. Speed
2. History
3. Durability
3. Durability
Exactly-Once Data Processing

Traditionally streaming ~ lossy, approximate processing
This is no longer true. Forget the ‘lambda architecture’.

- Input records are durably stored and indexed in logs (e.g., Kafka)
- Systems handle state snapshotting & transactions with external stores transparently.
- Idempotent and transactional writes to external stores

E.g.

on Flink each stream computation either completes or repeats

part 1 part 2 part 3 part 4
3. Durability

Exactly-Once Data Processing
4 Aspects of Data Processing

1. Speed
- complex event proc
- fast approximate streaming
- ETL
- event logs
- rules

2. History
- complex analytics
- large-scale processing systems
- data engineer

3. Durability
- production database
- "microservices"
- application state + failover

4. Interactivity
- interactive queries
- data science reports
- analyst

stream processor
4. Interactivity

Querying Data Processing State

Stream Processor ~ Inverse DBMS

Application state holds fresh knowledge we want to query:

- In some systems (e.g. Kafka-Streams) we can use the changelog
- In other systems (i.e., Flink) we can query the state externally…or stream queries on custom query processor on-top of them*

*https://techblog.king.com/rbea-scalable-real-time-analytics-king/
4 Aspects of **Data Processing**

1. **Speed**
   - no physical blocking/staging
   - no rescheduling
   - efficient pipelining
   - copy-on-write data structures

2. **History**
   - different notions of time
   - flexible stateful processing
   - high throughput

3. **Durability**
   - durable input logging is a standard
   - automated state management
   - exactly-once processing
   - output commit & Idempotency

4. **Interactivity**
   - external access to state/changelogs
   - ability to ‘stream queries’ over state
Try out Stream Processing

https://flink.apache.org/
https://beam.apache.org/
https://kafka.apache.org/

@SenorCarbone