StreamMapReduce
When Stream Processing crosses MapReduce

André Martin*, Andrey Brito# and Christof Fetzer*

andre.martin@tu-dresden.de, andrey@dsc.ufcg.edu.br, christof.fetzer@tu-dresden.de

*Technische Universität Dresden - Dresden, Germany
#Universidade Federal de Campina Grande - Campina Grande, Brazil

STREAM 2015 @ October, 28th 2015, Indianapolis, IN
Outline

• Why StreamMapReduce?
  ▪ Programming Model
  ▪ Application Examples
• Fault Tolerance & Elasticity in SMR
• Conclusion & Summary
• Ongoing & Future Work
Why StreamMapReduce?

• Massive data growth
  ▪ RFDI, click streams, net flow data, credit card transactions
  ▪ user tracking, recommendations, intrusion and fraud det.

• Large scale data processing got very popular
  ▪ Google’s MapReduce is very simple
  ▪ Mature open source implementation hadoop

• Batch processing ⇒ Real time data processing
StreamMapReduce

PROGRAMMING MODEL
Programming Model

- Inspired by MapReduce (UDFs)
- **Stateful** event processing

```java
public class SomeOperator extends OperatorImpl {
    public void process(int routingKey, byte[] event, Object state, Collector collector)
    {
        /* some computation */
        collector.emitEvent(someKey, timestamp, event.getBytes());
    }
    ...
}
```
We consider everything as a **black box**

- State (simple ptr to some arbitrary data structure)
- Operator Code = User Defined Function
- Event (except timestamp & routing key)
StreamMapReduce

APPLICATION EXAMPLES
Application Examples

- ACM DEBS’15 Challenge: Geo-spatial data context / taxi rides NYC area:
  - **Query #1**: Provide top-10 most frequently driven routes using a 30mins sliding window
  - **Query #2**: Provide top-10 most profitable areas using a 15 & 30 mins sliding window
Application Examples - Properties

• Access to specific **historic data**
• Complex computation, e.g. **profitable areas**:
  – accumulated fares / # of empty taxis
  – # of empty taxis = # taxis w/o follow-up pickup
• “Conditional” output
  – Only generate output if top-k changes

*Very complicated to express in CQL!*
StreamMapReduce

FAULT TOLERANCE & ELASTICITY
Fault Tolerance & Elasticity

- **State persistence:**
  - User provides serialize & de-serialize methods

- **Event logging** (in memory):
  - Timestamps associated w/ events

- **Repeatability:**
  - Deterministic merge of multiple up-streams

  ⇒ Provide **rollback recovery** in a precise manner

  ⇒ **Active replication** w/o costly atomic broadcast (state machine replication)

- Re-use FT mechanisms for **elasticity** [1]

---

Black box vs. Performance

• Use flat data structures for state [1]

• User annotations for operators, e.g.
  – stateful vs. stateless
  – commutativity [2] or
  – window lengths [3]

Summary & Conclusions

• SMR allows a **quick and easy transition** from batch to online processing of MapReduce applications

• Simplifies the development of complex queries and operators

• Good foundation for higher-level abstractions such as CQL/SQL [1] or K3

• Fault tolerance through check-pointing and logging

• Re-use of fault tolerance mechanisms to achieve elasticity

On-Going & Future Work

• Privacy preserving ESP using Intel SGX
  ▪ Putting the whole system vs. operators into the enclave

• Integration of StreamMine3G into the cloud stack using OpenStack Sahara
  ▪ Apache Storm integration already existing

Thank you for your attention – Q&A
andre.martin@se.inf.tu-dresden.de