Distributed Streaming SQL

For Fast Data Management

Milinda Pathirage & Beth Plale

School of Informatics and Computing, Indiana University
AGENDA

- Introduction
- Motivation
- Requirements
- SamzaSQL
- Future Work
INTRODUCTION

- SamzaSQL: Streaming SQL implementation on top of Apache Kafka and Apache Samza
- Utilizes Apache Calcite for query planning
- Extension of standard SQL
- Streams and Relations are first class citizens of both language and runtime
- Nearline applications
The sources of information over which real time processing can be done is significantly multiplied and varied.

- Lambda Architecture
- Kappa Architecture

Current distributed stream processing systems require developers to use programming APIs in high-level languages.

- Wide adoption of SQL based Big Data management solutions like Hive, Drill and Presto.

- Often real-time or near real-time processing applications are backed by computed summaries or modeled information generated by traditional batch-oriented processing systems.

LinkedIn’s stream analytics use cases
LAMBDA ARCHITECTURE

Jay Kreps; http://radar.oreilly.com/2014/07/questioning-the-lambda-architecture.html
KAPPA ARCHITECTURE

Jay Kreps; http://radar.oreilly.com/2014/07/questioning-the-lambda-architecture.html
LANGUAGE REQUIREMENTS

- Extension to **standard SQL**
- **Streams** and **relations** as first class entities in both language and runtime
- **Produce same results on a stream as if the same data were in a table**
- Rich set of window constructs for streaming aggregates and joins
  - SELECT STREAM START(rowtime), COUNT(*) FROM Orders GROUP BY TUMBLE(rowtime, INTERVAL '1' HOUR)
  - SELECT STREAM START(rowtime), COUNT(*) FROM Orders GROUP BY HOP(rowtime, INTERVAL '1' HOUR, INTERVAL '1' HOUR)
- Session windows
ARCHITECTURAL REQUIREMENTS

- Scaling across thousands of stream partitions
- Fault tolerance and ability to recover by replaying local storage change stream
- Out of order event handling
- Incremental processing and early results
- Support for multiple stream processing back-ends.
Uses Samza as stream processing back-end

Uses Apache Calcite for query planning

One or more partitions are mapped to a stream task

Local storage is checkpointed to a stream

In case of a failure tasks will be rescheduled in a different container and bootstrapped from local storage change stream
FUTURE WORK

- Performance evaluation
- Session windows in SQL
- How to handle stragglers
- Streaming specific cost model for enabling more optimizations
- SQL to Lambda Architecture style query plans
- Backend independent implementation
- Integrating with Big Data frameworks like HBase, Apache Phoenix
ACKNOWLEDGEMENTS

- Julian Hyde; Apache Calcite and Hortonworks
- Chris Riccomini; Apache Samza, WePay (Previously LinkedIn Data Infrastructure Team)
- Yi Pan; Apache Samza and LinkedIn Data Infrastructure Team
- Apache Samza Community
- Members of LinkedIn Data Infrastructure Team
REFERENCES

1. Apache Samza
2. Apache Calcite
3. Moving faster with data streams: The rise of Samza at LinkedIn
4. Real time insights into LinkedIn's performance using Apache Samza
5. Lambda Architecture
6. Kappa Architecture
7. Summingbird: A Framework for Integrating Batch and Online MapReduce Computations
Questions?