Scaling Stream Processing Out and Up

Tilmann Rabl

www.dima.tu-berlin.de | bbdc.berlin | rabl@tu-berlin.de

International Workshop on Web Data Processing & Reasoning
(WDPAR 2018)
Big Fast Data

• Data is growing and can be evaluated
  – Tweets, social networks (statuses, check-ins, shared content), blogs, click streams, various logs, …
  – *Facebook*: > 845M active users, > 8B messages/day
  – *Twitter*: > 140M active users, > 340M tweets/day

• Everyone is interested!

Image: Michael Carey
But there is so much more…

• Autonomous Driving
  – Requires rich navigation info
  – Rich data sensor readings
  – 1GB data per minute per car (all sensors)\(^1\)

• Traffic Monitoring
  – High event rates: millions events / sec
  – High query rates: thousands queries / sec
  – Queries: filtering, notifications, analytical

• Pre-processing of sensor data
  – CERN experiments generate ~1PB of measurements per second.
  – Unfeasible to store or process directly, fast preprocessing is a must.

\(^1\)Cobb: http://www.hybridcars.com/tech-experts-put-the-brakes-on-autonomous-cars/

Source: http://theroadtochangeindia.wordpress.com/2011/01/13/better-roads/
Stream Processing

Interesting streams
- Many different queries
- Continuous results
Why is this hard?

Tension between performance and algorithmic expressiveness
Agenda

Introduction to Streams
• Stream processing 101
• Efficient aggregation

Scale-Out Stream Processing Systems
• Ingredients of a stream processing system
• More details on Flink

Scale-Up Stream Processing
• New hardware

With slides from Data Artisans, Volker Markl, and Sebastian Bress
Stream Processing 101

Based on the Data Flow Model
What is a Stream?

• Unbounded data
  – Conceptually infinite, ever growing set of data items / events
  – Practically continuous stream of data, which needs to be processed / analyzed

• Push model
  – Data production and procession is controlled by the source
  – Publish / subscribe model

• Concept of time
  – Often need to reason about when data is produced and when processed data should be output
  – Time agnostic, processing time, ingestion time, event time

This part is largely based on Tyler Akidau’s great blog on streaming - https://www.oreilly.com/ideas/the-world-beyond-batch-streaming-101
Event Time

- Event time
  - Data item production time
- Ingestion time
  - System time when data item is received
- Processing time
  - System time when data item is processed

- Typically, these do not match!
- In practice, streams are unordered!
Windows

• Fixed
  – Also tumbling

• Sliding
  – Also hopping

• Session
  – Based on activity

• Triggered by
  – Event time, processing time, count, watermark

• Eviction policy
  – Window width / size
Processing Time Windows

- System waits for x time units
  - System decides on stream partitioning
  - Simple, easy to implement
  - Ignores any time information in the stream -> any aggregation can be arbitrary

- Similar: Counting Windows

Image: Tyler Akidau
Event Time Windows

- Windows based on the time information in stream
  - Adheres to stream semantic
  - Correct calculations
  - Buffering required, potentially unordered (more on this later)

Images: Tyler Akidau
Basic Stream Operators

• Windowed Aggregation
  – E.g., average speed
  – Sum of URL accesses
  – Daily highscore

• Windowed Join
  – Correlated observations in timeframe
  – E.g., temperature in time
Efficient Window Aggregation

Stream processing on overlapping windows
Aggregate computation is redundant
Partial aggregates can be shared
Challenge: session windows, user defined windows, out of order tuples
Session Window Observations

Windows with different gaps share partial aggregates
Session windows can share aggregates with sliding and tumbling windows
Slice on session and gap is equivalent to session slice
Slicing depends on session window with smallest gap

Stream Slicing Example:
Concurrent Session Windows with gaps 3, 5, 6, and 7
Generalized Stream Slicing*

Stream Slicer for non overlapping slices
Slice Manager for slice updates (out of order tuples) and window borders
Aggregate Store computes and stores partial aggregates (eager and lazy)
Window Manager combines aggregates and outputs windows

Out-of-Order Tuple Processing

- Slice Manager keeps minimum number of slices for out-of-order tuples
- Out-of-order tuple lead to updates
- Sufficient to store one partial aggregate per slice
- Reduced memory footprint
Stream Processing Systems

What makes a system a stream processing system?
8 Requirements of Big Streaming

- Keep the data moving
  - Streaming architecture
- Declarative access
  - E.g. StreamSQL, CQL
- Handle imperfections
  - Late, missing, unordered items
- Predictable outcomes
  - Consistency, event time
- Integrate stored and streaming data
  - Hybrid stream and batch
- Data safety and availability
  - Fault tolerance, durable state
- Automatic partitioning and scaling
  - Distributed processing
- Instantaneous processing and response

The 8 Requirements of Real-Time Stream Processing – Stonebraker et al. 2005
8 Requirements of Big Streaming

- **Keep the data moving**
  - Streaming architecture

- **Declarative access**
  - E.g. StreamSQL, CQL

- **Handle imperfections**
  - Late, missing, unordered items

- **Predictable outcomes**
  - Consistency, event time

- **Integrate stored and streaming data**
  - Hybrid stream and batch

- **Data safety and availability**
  - Fault tolerance, durable state

- **Automatic partitioning and scaling**
  - Distributed processing

- **Instantaneous processing and response**

The 8 Requirements of Real-Time Stream Processing – Stonebraker et al. 2005
Big Data Processing

• Databases can process very large data since forever (see VLDB)
  – Why not use those?

• Big data is not (fully) structured
  – No good for database 😞

• We want to learn more from data than just
  – Select, project, join

• First solution: MapReduce
How to keep data moving?

Discretized Streams (mini-batch)

while (true) {
    // get next few records
    // issue batch computation
}

Native streaming

while (true) {
    // process next record
}
Discussion of Mini-Batch

- Easy to implement
- Easy consistency and fault-tolerance
- Hard to do event time and sessions

Image: Tyler Akidau
True Streaming Architecture

- Program = DAG* of operators and intermediate streams
- Operator = computation + state
- Intermediate streams = logical stream of records

Stream transformations
- Basic transformations: Map, Reduce, Filter, Aggregations…
- Binary stream transformations: CoMap, CoReduce…
- Windowing semantics: Policy based flexible windowing (Time, Count, Delta…)
- Temporal binary stream operators: Joins, Crosses…
- Native support for iterations
Handle Imperfections – Watermarks

• Data items arrive early, on-time, or late
• Solution: Watermarks
  – Perfect or heuristic measure on when window is complete
Handle Imperfections – Watermarks

- Data items arrive early, on-time, or late
- Solution: Watermarks
  - Perfect or heuristic measure on when window is complete

Image: Tyler Akidau
Data Safety and Availability

• Ensure that operators see all events
  – “At least once”
  – Solved by replaying a stream from a checkpoint
  – No good for correct results

• Ensure that operators do not perform duplicate updates to their state
  – “Exactly once”
  – Several solutions

• Ensure the job can survive failure
Lessons Learned from Batch

- If a batch computation fails, simply repeat computation as a transaction
- Transaction rate is **constant**
- Can we apply these principles to a true streaming execution?
Taking Snapshots – the naïve way

Initial approach (e.g., Naiad)
- Pause execution on t1, t2, ...
- Collect state
- Restore execution
Asynchronous Snapshots in Flink

Automatic partitioning and scaling

• 3 Types of Parallelization

(a) Pipeline-parallel A || B.  (b) Task-parallel D || E.  (c) Data-parallel G || G.

• Big streaming systems should support all three
Apache Flink—
A Success Story
created in Berlin
Stratosphere: General Purpose Programming + Database Execution

Draws on Database Technology
- Relational Algebra
- Declarativity
- Query Optimization
- Robust Out-of-core

Adds
- Iterations
- Advanced Dataflows
- General APIs
- Native Streaming

Draws on MapReduce Technology
- Scalability
- User-defined Functions
- Complex Data Types
- Schema on Read

Draws on

MapReduce Technology
What is Apache Flink?

Apache Flink is an open source platform for scalable batch and stream data processing.

• The core of Flink is a distributed streaming dataflow engine.
  • Executing dataflows in parallel on clusters
  • Providing a reliable foundation for various workloads
• **DataSet** and **DataStream** programming abstractions are the foundation for user programs and higher layers

http://flink.apache.org
What can I do with it?

A big data processing system that can **natively** support all these workloads.

Stream processing

Batch processing

Machine Learning at scale

Graph Analysis

Flink
Big Data Analytics Ecosystem

Applications & Languages
- Hive
- Cascading
- Giraph
- Mahout
- Pig
- Crunch

Data processing engines
- MapReduce
- Flink
- Spark
- Storm
- Tez

App and resource management
- Yarn
- Mesos

Storage, streams
- HDFS
- HBase
- Kafka
- ...
Architecture

- Hybrid MapReduce and MPP database runtime

- Pipelined/Streaming engine
  - Complete DAG deployed
Sneak peak: Two of Flink’s APIs

```scala
case class Word (word: String, frequency: Int)

**DataSet API** (batch):

```scala
val lines: DataSet[String] = env.readTextFile(...) lines.flatMap {line => line.split(" ") .map(word => Word(word,1))} .groupBy("word").sum("frequency") .print()
```  

**DataStream API** (streaming):

```scala
val lines: DataStream[String] = env.fromSocketStream(...) lines.flatMap {line => line.split(" ") .map(word => Word(word,1))} .keyBy("word") .window(Time.of(5,SECONDS)).every(Time.of(1,SECONDS)) .sum("frequency") .print()
```
Yahoo! Benchmark Results

Performed by Yahoo! Engineering, Dec 16, 2015

[..]Storm 0.10.0, 0.11.0-SNAPSHOT and Flink 0.10.1 show sub-second latencies at relatively high throughputs[..]. Spark streaming 1.5.1 supports high throughputs, but at a relatively higher latency.

Flink achieves highest throughput with competitive low latency!
Our benchmarks*

**Streaming**

<table>
<thead>
<tr>
<th></th>
<th>2 Node</th>
<th>4 Node</th>
<th>8 Node</th>
</tr>
</thead>
<tbody>
<tr>
<td>Storm</td>
<td>408K</td>
<td>696K</td>
<td>992K</td>
</tr>
<tr>
<td>Spark</td>
<td>379K</td>
<td>642K</td>
<td>912K</td>
</tr>
<tr>
<td>Flink</td>
<td>1230K</td>
<td>1260K</td>
<td>1260K</td>
</tr>
</tbody>
</table>

Windowed Aggregations

*Benchmarking Distributed Stream Data Processing Systems.* Jeyhun Karimov, Tilmann Rabl, Asterios Katsifodimos, Roman Samarev, Henri Heiskanen, and Volker Markl. ICDE 2018
Stream Processing on Modern Hardware
Modern Hardware

- Multi-Core CPUs
- Fast Networks
- Non-Volatile Memory
Scale Out vs. Scale Up Stream Processing

Scale-Up: Operate a small cluster of nodes, keep all data in distributed main memory
Modern Multi-Core CPUs

• High Parallelism:
  – Multiple cores (task parallelism): Multiple threads can perform different tasks at the same time
  – Vector units (data parallelism): The same instruction is performed on multiple data items at once

• High Memory Bandwidth:
  – Aggregated memory bandwidth of 51.2GB/s per CPU (DDR3-1600 memory with four channels, 12.8GB/s per channel)
  – Multiple processors are organized in NUMA (Non-Uniform Memory Access) architecture
  – Cache coherent memory across all CPUs
Modern Multi-Core CPUs

Two principle resource limitations:

• Computation Bound:
  – Executing many instructions per input tuple
  – Performing many function calls
  – Encountering many branch mispredictions

• Memory Bound:
  – Bound by Memory Latency:
    • Random Memory Accesses (e.g., hash table operations)
  – Bound by Memory Bandwidth:
    • Executing few instructions per input tuple
    • Reading input tuples sequentially with maximal memory speed
Fast Networks

• Infiniband:
  – A new generation network protocol, native support for RDMA
  – Very high bandwidth (currently ~100Gbit per port)
  – Very small access latency to memory of remote machine
    (~1 microsecond for InfiniBand FDR 4x)

• RDMA (Remote Direct Memory Access):
  – Network adapter can directly read or write to application memory of remote machine
    → Avoids the overhead of copying data into OS buffers
    → Can access remote memory without consuming any CPU time in the remote machine
Bandwidth of Different Network Technologies

New network technologies have similar bandwidth as main memory!

Source: Following Binning et al. The End of Slow Networks: It’s Time for a Redesign. VLDB 2016.
Infiniband Future

Bandwidth of networks is going to be even larger than memory bandwidth

New streaming systems need to process streams with memory bandwidth to keep up
Scale Up vs. Scale Out Stream Processing

Current streaming systems cannot saturate memory bandwidth, but hand optimized implementations can!
Non-Volatile Memory

- Also called Storage Class Memory (SCM)

- Blurs the distinction between
  - Memory (= fast, expensive, volatile)
  - Storage (= slow, cheap, non-volatile)

- Byte-addressable; accessing NVRAM is similar to accessing DRAM

- Latencies are within the same order of magnitude as DRAM

- 10x higher density than DRAM, allows to keep more data (state) in-memory
Non-Volatile Memory: Use Cases

• Accelerate Checkpointing
  – Use NVRAM to store checkpoints
  – Reduces checkpointing overhead during run-time
  – Accelerates starting time when a node comes up again

• New system architectures:
  – Keep all data in NVRAM, no redo recovery needed!
  – Very fast startup times compared to checkpointing-based systems
  – Cache frequently accessed data in RAM for fast access
Non-Volatile Memory: Challenges

• **Any point crash recovery**: byte-addressable persistency makes any write to memory persistent
  → System may crash at any time and writes (log file) may be incomplete
  → Classic recovery techniques assume block-wise atomic writes for blocks on disk

• **Hole detection**: when a transaction just allocates chunks in NVRAM but has not written anything yet, there can be empty log records (holes) in the NVRAM log space

• **Partial write detection**: detect during recovery that transaction has not fully finished writing log data to NVRAM
Towards Scale Up Streaming Systems

Modern hardware allows us to built even faster streaming systems:

- **Scale-Up architecture**: operate a small cluster of nodes, which can keep all data and state in main memory

- **Fast Networks**: offer low latency and high bandwidth communication between nodes

- **Reduced Logging Overhead**: checkpoint application data in NVRAM
Conclusion

Introduction to Streams
• How to do real streaming

Stream Processing Systems
• Ingredients of a stream processing system
• Flink

Streaming on Modern Hardware
• How to optimize

Future Work
• Edge and fog
• Geodistribution
Thank You

Contact:
Tilmann Rabl
rabl@tu-berlin.de

We are hiring!