Lecture Notes to
Big Data Management and Analytics
Winter Term 2017/2018
Stream Processing

© Matthias Schubert, Matthias Renz, Felix Borutta, Evgeniy Faerman, Christian Frey, Klaus Arthur Schmid, Daniyal Kazempour, Julian Busch
© 2016-2018
Example application: Facebook

Example application: Twitter

Source: http://blog.kissmetrics.com/twitter-statistics/
Example application: CERN

- Experiments at CERN are generating an entire petabyte \((1\text{PB}=10^6 \text{ GB})\) of data every second as particles fired around the Large Hadron Collider (LHC) at velocities approaching the speed of light are smashed together.

- “We don’t store all the data as that would be impractical. Instead, from the collisions we run, we only keep the few pieces that are of interest, the rare events that occur, which our filters spot and send on over the network,” he said.

- This still means CERN is storing 25PB of data every year – the same as 1,000 years' worth of DVD quality video – which can then be analyzed and interrogated by scientists looking for clues to the structure and make-up of the universe.


Outline

• Data Streams & Data Stream Management System

• Data Stream Models
  • Insert-Only
  • Insert-Delete
  • Additive

• Streaming Methods
  • Sliding Windows & Ageing
  • Data Synopsis

• Stream Processing – Concepts & Tools
  • Micro-Batching with Apache Spark Streaming
  • Real-time Stream Processing with Apache Storm
Stream Processing

Data Streams

• Definition: A data stream can be seen as a continuous and potentially infinite stochastic process in which events occur independently from another

• Huge amount of data → Data objects cannot be stored

• Single scan
Stream Processing

Data Streams – Key Characteristics

• The data elements in the stream arrive on-line

• The system has no control over the order in which data elements arrive (either within a data stream or across multiple data streams)

• Data streams are potentially unbound in size

• Once an element has been processed it is discarded or archived
Stream Processing

Data Stream Management System

Data Streams

Stream Processor

Ad-hoc queries

Standing query

Output Streams

Limited working storage

Archival Storage

Big Data Management and Analytics
Stream Processing

Data Stream Models – Insert-Only Model

• Once an element $x_i$ is seen, it cannot be changed
Stream Processing

Data Stream Models – Insert-Delete Model

- Elements $x_i$ can be deleted or updated
Stream Processing

Data Stream Models – Additive Model

- Each element $x_i$ is an increment to the previous version of the given data object
Stream Processing

Streaming Methods

• Huge amount of data vs. limited resources in space → impractical to store all data

• Solutions:
  • Storing summaries of previously seen data
  • “Forgetting” stale data

• But: Trade-off between storage space and the ability to provide precise query answers
Stream Processing

Streaming Methods – Sliding Windows

• Idea: Keep most recent stream elements in main memory and discard older ones

• Timestamp-based:

![Diagram of Data Stream with Sliding Windows]

- Data Stream
- Sliding interval
- Window length
Stream Processing

Streaming Methods – Sliding Windows

• Idea: Keep most recent stream elements in main memory and discard older ones

• Sequence-based:

![Diagram of sliding window](image)
Streaming Methods – Ageing

- Idea: Keep only the summary in main memory and discard objects as soon as they are processed.

- Multiply the summary with a decay factor after each time epoch, resp. after a certain amount of occurring elements.
Stream Processing

Streaming Methods

- High velocity of incoming data vs. limited resources in time → impossible to process all data

- Solutions:
  - Data reduction
  - Data approximation

- But: Trade-off between processing speed and the ability to provide precise query answers
Stream Processing

Streaming Methods – Sampling

• Select a subset of the data
  → Reduce the amount of data to process

• Difficulty: Obtaining a representative sample

• Simplest form: random sampling
  – Reservoir Sampling
  – Min-Wise Sampling

• Load Shedding: Discard some fractions of data if the arrival rate of the stream might overload the system

Reservoir Sampling Algorithm

\begin{verbatim}
input: Stream \( S \), Size of reservoir \( k \)
begin
  Insert first \( k \) objects into reservoir;
  foreach \( v \in S \) do
    Let \( i \) be the position of \( v \);
    \( M := \) random integer in range \( 1..i \);
    if \( M \leq k \) then
      Insert \( v \) into reservoir;
      Delete an instance from the reservoir at random;
end
\end{verbatim}
Streaming Methods – Data Synopsis & Histograms

• Summaries of data objects often used to reduce the amount of data
  – e.g. Microclusters that describe groups of similar objects

• Histograms are used to approximate the frequency distribution of element values
  – Commonly used for query optimizers (e.g. range queries)
Stream Processing

- Overview of techniques to build a summary (reduced representation) of a sequence of numeric attributes:
Discrete Wavelet Transformation (DWT)

Idea:

- Sequence represented as linear combination of basic wavelet functions
- Wavelet transformation decomposes a signal into several groups of coefficients at different scales
- Small coefficients can be eliminated \(\rightarrow\) Small errors when reconstructing the signal
  \(\rightarrow\) Take only the first function coefficients
- Often: Haar-wavelets used (easy to implement)
Stream Processing

Example:

Step-wise transformation of sequence (stream) $X=\{8,4,1,3\}$ into Haar-wavelet representation $H=[4,2,2,-1]$

(Lossless) Reconstruction of original sequence (stream) from Haar-wavelet representation:

$h_1 = 4 = \text{mean}(8, 4, 1, 3)$

$h_2 = 2 = \text{mean}(8, 4) - h_1$

$h_3 = 2 = (8-4)/2$

$h_4 = -1 = (1-3)/2$

$X = \{8, 4, 1, 3\}$
Stream Processing

Haar Wavelet Transformation

Input sequence:
\[ S = (2, 5, 8, 9, 7, 4, -1, 1) \]

**Step 1:**
\[ s_1 = \frac{(2 + 5, 8 + 9, 7 + 4, -1 + 1)}{2}, \quad d_1 = \frac{(2 - 5, 8 - 9, 7 - 4, -1 - 1)}{2} \]
\[ s_1 = (3.5, 8.5, 5.5, 0), \quad d_1 = \{-1.5, -0.5, 1.5, -1\} \]

**Step 2:**
\[ s_2 = \frac{(3.5 + 8.5, 5.5 + 0)}{2}, \quad d_2 = \frac{(3.5 - 8.5, 5.5 - 0)}{2} \]
\[ s_2 = (6, 2.75), \quad d_2 = \{-2.5, 2.75\} \]

**Step 3:**
\[ s_3 = \frac{(6 + 2.75)}{2}, \quad d_3 = \frac{(6 - 2.75)}{2} \]
\[ s_3 = 4.375, \quad d_3 = \{1.625\} \]

→ Wavelet coefficients \{4.375, 1.625, -2.5, 2.75, -1.5, -0.5, 1.5, -1\}

---

Haar Wavelet Transform Algorithm

**input:** Sequence \( S = (x_0, x_1, \ldots, x_{2n}, x_{2n+1}) \) of even length

**output:** Sequence of wavelet coefficients

**begin**
Transform \( S \) into a sequence of two-component-vectors \(((s_0, d_0), \ldots, (s_n, d_n))\) where \(\begin{pmatrix} s_i \\ d_i \end{pmatrix} = \frac{1}{2} \cdot \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \cdot \begin{pmatrix} x_i \\ x_{i+1} \end{pmatrix} \);
Separate the sequences \( s \) and \( d \);
Recursively transform sequence \( s \);
Stream Processing

Spark Streaming

- Spark’s Streaming Framework build on top of Spark’s Core API

- Data ingestion from several different data sources

- Stream processing might be combined with other Spark libraries (e.g. Spark Mllib)
Stream Processing

Spark Streaming

• Spark’s Streaming Workflow:

  Streaming engine receives data from input streams
  Data stream is divided into several microbatches, i.e. sequences of RDDs
  Microbatches are processed by Spark engine
  The result is a data stream of batches of processed data
Stream Processing

Spark Streaming

- DStreams (Discretized Streams) as basic abstraction

- Any operation applied on a DStream translates to operations on the underlying RDDs (computed by Spark Engine)

- StreamingContext objects as starting points

```python
sc = SparkContext(master, appName)
ssc = StreamingContext(sc, 1)  #params: SparkContext, time interval
```
Stream Processing

Spark Streaming

General schedule for a Spark Streaming application:

1. Define the StreamingContext `ssc`
2. Define the input sources by creating input DStreams
3. Define the streaming computations by applying transformations and output operations to Dstreams
4. Start receiving data and processing it using `ssc.start()`
5. Wait for the processing to be stopped (manually or due to any error) using `ssc.awaitTermination()`
6. The processing can be manually stopped using `ssc.stop()`
# Create a local StreamingContext with two working threads and batch interval of 1 sec

```python
sc = SparkContext("local[2]", "NetworkWordCount")
ssc = StreamingContext(sc, 1)
```

# Create a DStream that will connect to localhost:9999

```python
lines = ssc.socketTextStream("localhost", 9999)
```

# Split each line into words

```python
words = lines.flatMap(lambda line: line.split(" "))
```

# Count each word in each batch

```python
pairs = words.map(lambda word: (word, 1))
wordCounts = pairs.reduceByKey(lambda x, y: x + y)
```

# Print the first ten elements of each RDD of this DStream to the console

```python
wordCounts.pprint()
```

# Start the computation and wait for it to terminate

```python
ssc.start()
ssc.awaitTermination()
```
Stream Processing

Spark Streaming

- Support of window operations

- Two basic parameters:
  - `windowLength`
  - `slideInterval`

- Support of many transformations for windowed DStreams

```python
# Reduce last 30 sec of data, every 10 sec
winWordCounts = pairs
    .reduceByKeyAndWindow(lambda x, y: x+y, 30, 10)
```
Stream Processing

Apache Storm

- Alternative to Spark Streaming
- Support of Real-time Processing
- Three abstractions:
  - Spouts
  - Bolts
  - Topologies
Stream Processing

Apache Storm

- **Spouts:**
  - Source of streams
  - Typically reads from queuing brokers (e.g. Kafka, RabbitMQ)
  - Can also generate its own data or read from external sources (e.g. Twitter)

- **Bolts:**
  - Processes any number of input streams
  - Produces any number of output streams
  - Holds most of the logic of the computations (functions, filters,...)
Stream Processing

Apache Storm

- Topologies:
  - Network of spouts and bolts
  - Each edge represents a bolt subscribing to the output stream of some other spout or bolt
  - A topology is an arbitrarily complex multi-stage stream computation
Stream Processing

Apache Storm

- Streams:
  - Core abstraction in Storm
  - A stream is an unbounded sequence of tuples that is processed and created in parallel in a distributed fashion
  - Tuples can contain standard types like integer, float, short, Boolean, string and so on
  - Custom types can be used if a own serializer is defined
  - A stream grouping defines how that stream should be partitioned among the bolt's tasks
Stream Processing

Apache Storm

```
Config conf = new Config();
conf.setNumWorkers(2); // use two worker processes

topologyBuilder.setSpout("blue-spout", new BlueSpout(), 2); // set parallelism hint to 2

topologyBuilder.setBolt("green-bolt", new GreenBolt(), 2)
    .setNumTasks(4)
    .shuffleGrouping("blue-spout");
// 4 Tasks spread across 2 Executors and the
// tuples shall be randomly distributed across
// the bolt's tasks, each bolt shall get an
// equal number of tuples

topologyBuilder.setBolt("yellow-bolt", new YellowBolt(), 6)
    .shuffleGrouping("green-bolt");

StormSubmitter.submitTopology("mytopology", conf, topologyBuilder.createTopology());
```
Stream Processing

Further Reading

• Joao Gama: *Knowledge Discovery from Data Streams* (http://www.liaad.up.pt/area/jgama/DataStreamsCRC.pdf)
• Jure Leskovec, Anand Rajaraman, Jeff Ullman: *Mining of Massive Datasets*
• Holden Karau, Andy Konwinski, Patrick Wendell, Matei Zaharia: *Learning Spark - Lightning-Fast Big Data Analysis*
• http://spark.apache.org/docs/latest/streaming-programming-guide.html
• http://storm.apache.org/documentation/Concepts.html