Introduction to Data Stream Processing

Corso di Sistemi e Architetture per Big Data
A.A. 2017/18

Valeria Cardellini

The reference Big Data stack

High-level Interfaces

Data Processing

Data Storage

Resource Management

Support / Integration
Why data stream processing?

• Applications such as:
  – Sentiment analysis on multiple tweet streams @Twitter
  – User profiling @Yahoo!
  – Tracking of query trend evolution @Google
  – Fraud detection
  – Bus routing management @city of Dublin

• Require:
  – Continuous processing of unbounded data streams generated by multiple, distributed sources
  – In (near) real-time fashion

Why data stream processing?

• In the past years data stream processing (DSP) was considered a solution for very specific problems (e.g., financial tickers)
• But now we have (and will have) more general settings
  – E.g., Internet of Things
Why data stream processing?

- Decrease the overall latency to obtain results
  - No data persistence on stable storage
    Recall “Latency numbers every programmer should know”!

<table>
<thead>
<tr>
<th>Activity</th>
<th>Latency (ns)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Main memory reference</td>
<td>100</td>
</tr>
<tr>
<td>L1 cache reference</td>
<td>0</td>
</tr>
<tr>
<td>Branch mispredict</td>
<td>3</td>
</tr>
<tr>
<td>L2 cache reference</td>
<td>4</td>
</tr>
<tr>
<td>Mutex lock/unlock</td>
<td>17</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Activity</th>
<th>Latency (ns)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Send 2,000 bytes</td>
<td>177</td>
</tr>
<tr>
<td>SSD random read</td>
<td>16,000</td>
</tr>
<tr>
<td>Read 1,000,000 bytes</td>
<td>7,000</td>
</tr>
<tr>
<td>Read 1,000,000 bytes</td>
<td>13</td>
</tr>
<tr>
<td>Disk seek</td>
<td>3,000</td>
</tr>
<tr>
<td>Round trip in same datacenter</td>
<td>500,000</td>
</tr>
<tr>
<td>Packet roundtrip CA to Netherlands</td>
<td>500,000,000</td>
</tr>
</tbody>
</table>

Traditional DSP challenges

- Stream data rates can be high and data arrive in large volumes
  - High resource requirements for processing (clusters, data centers, distributed Clouds)

- Processing stream data has real-time aspects
  - Stream processing applications have QoS requirements, e.g., end-to-end latency
  - Must be able to react to events as they occur
New challenge for large-scale DSP

- Goals: increase scalability and reduce latency
- How? Rely on distributed and near-edge computation

Data stream

- “A data stream is a real-time, continuous, ordered (implicitly by arrival time or explicitly by timestamp) sequence of items. It is impossible to control the order in which items arrive, nor is it feasible to locally store a stream in its entirety. Queries over streams run continuously over a period of time and incrementally return new results as new data arrive.”

DSP application model

• A DSP application is made of a network of operators (processing elements or PE) connected by streams, at least one data source and at least one data sink

• Represented by a directed graph
  – Graph vertices: operators
  – Graph edges: streams

• Graph can be cyclic
  – Some systems only support directed acyclic graph (DAG)

• Graph topology rarely changes

DSP programming model

• Data flow programming
  • Flow composition: techniques for creating the topology associated with the flow graph for an application
  • Flow manipulation: the use of processing elements (i.e., operators) to perform transformations on data
**Data flow manipulation**

- How the streaming data is manipulated by the different operators in the flow graph?
- Operator properties:
  - Operator type
  - Operator state
  - Windowing

**DSP operator**

- A self-contained *processing element* that:
  - transforms one or more input streams into another stream
  - can execute a generic user-defined code
    - Algebraic operation (filter, aggregate, join, ..)
    - User-defined (more complex) operation (POS-tagging, …)
  - can execute in parallel with other operators
Types of operators

• **Edge adaptation**: converting data from external sources into tuples that can be consumed by downstream operators
• **Aggregation**: collecting and summarizing a subset of tuples from one or more streams
• **Splitting**: partitioning a stream into multiple streams
• **Merging**: combining multiple input streams

Types of operators

• **Logical and mathematical operations**: applying different logical processing, relational processing, and mathematical functions to tuple attributes
• **Sequence manipulation**: reordering, delaying, or altering the temporal properties of a stream
• **Custom data manipulations**: applying data mining, machine learning, ...
DSP operator: state

• The operator can be stateless or stateful
• **Stateless:** know nothing about the state (e.g., filter, map) and thus process tuples independently of each other, independently of prior history, or even from the order of arrival of tuples
  – Easily parallelized
  – No synchronization in a multi-threaded context.
  – Restart upon failures without the need of any recovery procedure

DSP operator: state

• **Stateful:** keep some sort of state and thus involve maintaining information across different tuples to detect complex patterns.
  – E.g., some aggregation or summary of processed elements, or state-machine for detecting patterns for fraudulent financial transaction
  – State might be shared between operators
  – A subset of recent tuples kept in a window buffer
Window-based Operator

- **Window**: a buffer associated with an input port to retain previously received tuples

- A window is characterized by:
  - **Size**: it determines the amount of data that should be buffered before triggering the operator execution;
    - Statically defined: time-based; count-based;
    - Dynamically defined: session-based
  - **Sliding interval**: it determines how the window moves forward
    - Usually: time-based or count-based

By combining the window size and sliding interval, different windowing patterns can be realized:

- **Sliding windows**: static window size and a sliding interval with value different from the window size
- **Tumbling windows**: the sliding period is equal to the window size (i.e., they do not overlap).
How to define a DSP application

- **Formal language**: more rigor and expressiveness
  - Declarative language: specify the result (SQL-like); e.g., IBM Streams Processing Language
  - Imperative language: specify the composition of basic operators, e.g., SQuAl (Stream Query Algebra) used in Aurora/Borealis

- **Topology description**: more flexibility
  - Explicitly define the operators (built-in or user-defined) and the links through a directed graph (often called topology)

**“Hello World”: a variant of WordCount**

- **Goal**: emit the top-k words in terms of occurrence when there is a rank update

- Where are the bottlenecks?
  - **How to scale the DSP application in order to sustain the traffic load?**
“Hello World”: a variant of WordCount

- The usual answer: replication!
- Use data parallelism

Example of DSP application: DEBS’14 GC

- Real-time analytics over high volume sensor data: analysis of energy consumption measurements for smart homes
  - Smart plugs deployed in households and equipped with sensors that measure values related to power consumption

- Input data stream:
  2967740693, 1379879533, 82.042, 0, 1, 0, 12
- Query 1: make load forecasts based on current load measurements and historical data
  - Output data stream:
    - ts, house_id, predicted_load
- Query 2: find the outliers concerning energy consumption
  - Output data stream:
    - ts_start, ts_stop, household_id, percentage
Example of DSP application: DEBS’15 GC

- Real-time analytics over high volume spatio-temporal data streams: analysis of taxi trips based on data streams originating from New York City taxis
- Input data streams: include starting point, drop-off point, corresponding timestamps, and information related to the payment

07290D3599E7A0D62097A346EFC1FB5,E7750A37CAB07D0DFF0AF7E3573AC141,2013-01-01 00:00:00,2013-01-01 00:02:00,120,0.44,-73.956528,40.716976,-73.962440,40.715008,CSH,3.50,0.50,0.50,0.00,0.00,4.50

Example of DSP application: DEBS’15 GC

- Query 1: identify the top 10 most frequent routes during the last 30 minutes
- Query 2: identify areas that are currently most profitable for taxi drivers
- Both queries rely on a sliding window operator
  - Continuously evaluate the query results
- Use geo-spatial grids to define the events of interest
Example of DSP application: DEBS’16 GC

• Real-time analytics for a dynamic (evolving) social-network graph
• Query 1: identify the posts that currently trigger the most activity in the social network
• Query 2: identify large communities that are currently involved in a topic
• Require continuous analysis of dynamic graph considering multiple streams that reflect graph updates

Distributed DSP system

• A distributed system that executes stream graphs
  – continuously calculates results for long-standing queries
  – over potentially infinite data streams
  – using operators
    • that can be stateless or stateful
• System nodes may be heterogeneous
• Must be highly optimized and with minimal overhead so to deliver real-time response for high-volume DSP applications
• Must manage a number of issues
  – Operator placement on computing nodes
  – Node failures
  – …
Distributed DSP system

- Usually run in locally distributed clusters within large data centers
- Assumptions:
  - Scale out and not scale up
    - Commodity servers
    - Data-parallelism is king
  - Software designed for failure
- Which software frameworks for distributed DSP systems?

DSP frameworks: processing model

- Two stream processing models:
  - One-at-a-time: each tuple is individually sent
  - Micro-batched: some tuples are grouped before being sent

<table>
<thead>
<tr>
<th></th>
<th>One-at-a-time (e.g., Apache Storm)</th>
<th>Micro-batched (e.g., Apache Spark Streaming)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lower latency</td>
<td>✔️</td>
<td></td>
</tr>
<tr>
<td>Higher throughput</td>
<td></td>
<td>✔️</td>
</tr>
<tr>
<td>At-least-once semantics</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>Exactly-once semantics</td>
<td></td>
<td>In some cases</td>
</tr>
<tr>
<td>Simpler programming model</td>
<td></td>
<td>✔️</td>
</tr>
</tbody>
</table>


The two approaches are complementary with distinct trade-offs and are suitable to different types of applications.