

Introduction to Data Stream Processing

Corso di Sistemi e Architetture per Big Data A.A. 2022/23 Valeria Cardellini

Laurea Magistrale in Ingegneria Informatica

The reference Big Data stack



Why data stream processing?

- Applications such as:
 - Sentiment analysis on tweets @Twitter
 - User profiling @Yahoo!
 - Tracking of query trend evolution @Google
 - Fraud detection in financial transactions
 - Real-time advertising
 - Healthcare analytics involving IoT medical sensors
- Require:
 - Continuous processing of unbounded data streams generated by multiple and distributed sources
 - In (near) real-time fashion

V. Cardellini - SABD 2022/23

Why data stream processing?

- In the early years data stream processing (DSP) was considered a solution for very specific problems (e.g., financial tickers)
- Now we have more general settings
 E.g., social media, Internet of Things



- Decrease latency to obtain results and improve data freshness
 - Events are processed close to the time they are generated
 - Applications respond to events as they occur
 - No delays involved with batch processing
 - No data persistence on stable storage
- Simplify data analytics pipelines and underlying infrastructure

V. Cardellini - SABD 2022/23

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." Golab and Özs, <u>Issues in data stream management</u>, ACM *SIGMOD Rec.*, 2003.
- A data stream refers to both velocity and variety of Big data
- A stream is an unbounded sequence of tuples, where a tuple is an ordered list of values

Data stream: example

Data stream related to maritime traffic in the Mediterranean 0x3b62baab6210a8e69d3e7f9df53d000c83d00fd0,2, 15.247220,37.287770,163,511,01-06-15 0:00,AUGUSTA,12 0x0fe9acdb3675a8a2942fafbd4af61bc37e44c0ec,146, tuples 23.694910,37.313620,13,15,01-06-15 0:00,SALERNO,88 0xb35dc6acdc29f2241296c44384fa2b0f7044d257,20, 15.669920,38.387740,339,339,01-06-15 0:00,MESSINA,66

Each tuple contains the fields:

SHIP ID, SPEED, LON2, LAT2, COURSE, HEADING, TIMESTAMP, departurePortName,Reported Draught

V. Cardellini - SABD 2022/23

Traditional DSP challenges

- Stream data can arrive at high velocity, with high volumes and highly variable arrival patterns
 - High resource requirements for processing
- 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
- Faults can happen during processing





Challenges for DSP in Cloud-Edge continuum

- Goals: increase scalability and reduce latency
- How? Rely not only on Cloud resources but also on distributed and near-edge computation (Fog/edge computing)



V. Cardellini - SABD 2022/23

DSP application model

- A DSP application is made of a network of operators (processing elements) connected by streams, at least one data source and at least one data sink
- Represented by a directed dataflow graph
 - Graph vertices: operators
 - Graph edges: streams
 - Graph is often referred to as topology



- Graph is typically acyclic: directed acyclic graph (DAG)
 - Most systems only support DAGs, few support also cyclic computations (e.g., Flink)
- Application topology does not usually change during processing



DSP programming model

• Dataflow programming

- Programming paradigm that models a program as a directed graph of data (dataflow) flowing between operations
- Pioneered by Jack Dennis and his students at MIT in the 1960s
- Examples
 - Apache NiFi: automates dataflow between systems
 - Apache Flink: stream and batch processing
 - Apache Beam: unifies batch and streaming data processing on top of several execution engines
 - TensorFlow: ML library based on dataflow programming

- Flow composition: how to create the topology associated with the directed graph for a DSP application
- Flow manipulation: use of processing elements (i.e., operators) to perform transformations on data

Data flow manipulation

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

- 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 and possibly complex operation (POS-tagging, machine learning algorithm, ...)
 - Can execute in parallel with other operators

14

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

- 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

- Operator can be stateless or stateful
- Stateless: processing depends only on current input
 - Operator knows nothing about state and thus processes tuples independently of each other, independently of prior history or even from tuple arrival order
 - E.g., filter, map
 - Easily parallelizable
 - No synchronization in a multi-threaded context
 - Easy restart upon failures (no need to recover state)

- Stateful: keeps some sort of state (i.e., information across multiple tuples) that operator can read and modify during execution, e.g.,
 - Aggregation or summary of tuples per minute/hour/day
 - When an application searches for certain patterns, the state will store the sequence of events encountered so far
 - When training a machine learning model over a stream of data points, the state holds the current version of the model parameters

V. Cardellini - SABD 2022/23

DSP operator: state

- State may be stored in different ways:
 - Entirely stored within inmemory data structures and replicated to disk only for fault tolerance



- Entirely stored on nonvolatile memory (e.g., disk)
- Hybrid solution: partially stored in memory for improved performance and flushed to disk to scale in size
- Stored on a storage service (e.g., Redis)
- State is mostly private to operator but in some system can be shared between operators

- Shared state makes execution more complex V. Cardellini - SABD 2022/23

- Window: buffer associated with an operator input port to retain incoming tuples over which we can apply computations so to process them as a whole
 - E.g., the most frequently purchased items over the last hour
- Window is characterized by:
 - Size: amount of data that should be buffered before triggering operator execution
 - Statically defined: time-based (e.g., 30 seconds) or countbased (e.g., the last 100 tuples)
 - Dynamically defined: session-based
 - Sliding interval: how the window moves forward
 - Time-based or count-based

V. Cardellini - SABD 2022/23

20

Windowing: patterns

- Different windowing patterns by combining window size and sliding interval:
 - **Sliding window**: static window size and sliding interval with value different from window size, single tuples may be included in multiple consecutive windows
 - **Tumbling window:** sliding interval equal to window size, no overlapping of windows

Sliding window (size:2; slide:1)



Tumbling window (size:2; slide:2)



Windowing: patterns



Windowing: patterns

- Window can be also dynamically defined: session window
 - Dynamic size of window length, depending on inputs
 - Starts with an input and expands itself if the following input has been received within the gap duration
 - Closes when there's no input received within the gap duration after receiving the latest input
 - Enables to group events until there are no new events for specified time duration (inactivity)



- Once a trigger determines that a window is ready for processing, it fires, i.e., emits the result of the current window
- Example: tumbling time window of 1 minute that sums the values



How to define a DSP application

Topology description

- Explicitly defines operators (built-in or user-defined) and links through a DAG
- Used in Flink, Storm, Spark Streaming, ...

Formal language

- Declarative language that specifies result (SQL-like)
 - e.g., Streams Processing Language (SPL) in IBM Streams
- Imperative language that specifies composition of basic operators
 - e.g., SQuAI (Stream Query Algebra) used in Aurora/Borealis
- The first offers more flexibility, the latter more rigor and expressiveness

"Hello World": a variant of WordCount

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



- Which operators can be a bottleneck?
- How to scale the application in order to sustain the traffic load?

V. Cardellini - SABD 2022/23

"Hello World": a variant of WordCount

- Replication is the usual answer: let's replicate the operators when possible
- We use data parallelism (aka operator fission) and redesign the application by dividing sorter into two operators
- We need to partition the output stream: how?



Example of DSP application: DEBS'14 GC

debs.org/grand-challenges/2014

- 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 outliers concerning energy consumption

– Output data stream:

ts_start, ts_stop, household_id, percentage

V. Cardellini - SABD 2022/23

Example of DSP application: DEBS'15 GC

debs.org/grand-challenges/2015

- Real-time analytics over high volume spatio-temporal data streams: analysis of taxi trips based on data streams originating from New York City taxis
- Data stream composed of tuples
- Each tuple includes: pickup and drop-off points (longitude and latitude), corresponding timestamps plus information related to payment

07290D3599E7A0D62097A346EFCC1FB5,E7750A37CAB07D0DFF0AF 7E3573AC141,2013-01-01 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

debs.org/grand-challenges/2015

- Query 1: identify 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 sliding window operators
 - Continuously evaluate query results



V. Cardellini - SABD 2022/23

30

Example of DSP application: DEBS'16 GC

debs.org/grand-challenges/2016

- Real-time analytics for a dynamic (evolving) socialnetwork 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

- Distributed system that executes DSP applications
 - Continuously calculates results for long-standing queries
 - Over potentially infinite data streams
 - Using stateless or stateful operators
- System nodes may be heterogeneous
 - Computing capacity, network bandwidth, ...
- Must be highly optimized and with minimal overhead so to deliver real-time response
- Must manage a number of issues
 - Operator placement on computing nodes
 - Node and operator failures

- ...

V. Cardellini - SABD 2022/23

Distributed DSP system

- Traditionally runs in a locally distributed cluster within a data center (also Cloud-based)
- Assumptions:
 - Scale out
 - Commodity servers
 - Data-parallelism (operator parallelism) is king
 - Designed to handle failures
- Newer environments: edge computing and Cloudedge continuum







Data-intensive system: a common view

• Distributed data-intensive systems for batch and stream processing share some common characteristics in terms of architecture



Margara et al. A Model and Survey of Distributed Data-Intensive Systems, 2023

```
V. Cardellini - SABD 2022/23
```

34

Data-intensive system: a common view

- · Applications (i.e., jobs) and their lifecycle
 - Job lifecycle includes: definition using API, compilation into an *execution plan*, deployment, and execution
 - Jobs are compiled into elementary units of execution (i.e., tasks) and run on slots offered by worker nodes
 - Each task can be replicated (data parallelism)
 - Tasks must be deployed onto the slots of the underlying infrastructure through a placement algorithm



Main DSP frameworks

- Apache Storm
- Apache Flink
- <u>Apache Samza</u>
- Apache Spark Streaming
- Kafka Streaming
- Cloud-based services
 - Google Cloud Dataflow
 - Amazon Kinesis

V. Cardellini - SABD 2022/23

DSP frameworks: processing model

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

| | One-at-a-time (e.g., Apache Storm) | Micro-batched (e.g., Apache Spark | Streaming) |
|---------------------------|---------------------------------------|--------------------------------------|------------|
| Lower latency | | | |
| Higher throughput | | | |
| At-least-once semantics | | | |
| Exactly-once semantics | In some cases | | |
| Simpler programming model | | | |

Source: N. Marz, J. Warren, Big Data, Manning Pub., 2015

DSP frameworks: evolution



- Early systems were designed as extensions of relational execution engines with the addition of windows
- Modern systems have evolved considering completeness and ordering (e.g., out-of-order computation) and have witnessed architectural paradigm shifts (e.g., processing guarantees, reconfiguration and state management)
- Recent shift towards general event-driven architectures, actor-like programming models and microservices, and growing use of hw accelerators

Fragkoulis et al., <u>A Survey on the Evolution of Stream Processing Systems</u>, 2023 38