Introduction to Data Stream Processing

Amir H. Payberah
amir@sics.se

KTH Royal Institute of Technology
Many applications must process large streams of live data and provide results in real-time.

- Wireless sensor networks
- Traffic management applications
- Stock marketing
- Environmental monitoring applications
- Fraud detection tools
- ...
Stream Processing Systems

- Database Management Systems (DBMS): data-at-rest analytics
  - Store and index data before processing it.
  - Process data only when explicitly asked by the users.
Stream Processing Systems

- Database Management Systems (DBMS): data-at-rest analytics
  - Store and index data before processing it.
  - Process data only when explicitly asked by the users.

  - Processing information as it flows, without storing them persistently.
DBMS vs. SPS

▶ DBMS

- **Persistent** data where updates are relatively **infrequent**.
- Runs queries just **once** to return a complete answer.

▶ SPS

- **Transient** data that is **continuously** updated.
- Executes **standing queries**, which run **continuously** and provide updated answers as new data arrives.
SPS Data Model

- Data stream is unbound and broken into a sequence of individual data items, called tuples.

- A data tuple is the atomic data item in a data stream.
  - Similar to a database row.

![Diagram of SPS Data Model]
SPS Data Model

- Data stream is **unbound** and broken into a sequence of individual data items, called **tuples**.

- A data **tuple** is the **atomic** data item in a data stream.
  - Similar to a **database row**.

- Three classes:
  - **Structured**: known schema
  - **Semi-structured**: self-describing tags, e.g., HTML or XML
  - **Unstructured**: custom or proprietary formats, e.g., video, audio
The tuples are processed by the application’s operators or processing element (PE).

A PE is the basic functional unit in an application.
- A PE processes input tuples, applies a function, and outputs tuples.
- A set of PEs and stream connections, organized into a data flow graph.
SPS Programming Model
SPS Programming Model

- SPS data flow programming

- Flow composition: techniques for creating the topology associated with the flow graph for an application.

- Flow manipulation: the use of PEs to perform transformations on data flows.
Data Flow Composition
Data Flow Composition

- Static composition
- Dynamic composition
- Nested composition

Source
Multi-port source
Sink
Multi-port sink
Simple
Splitter
Merger
Complex

Direct
Fan-out
Fan-in
Data Flow Composition

- Flow composition patterns:
  - **Static** composition
  - **Dynamic** composition
  - **Nested** composition
Creating the parts of the application topology that are known at development time.
Static Flow Composition

Creating the parts of the application topology that are known at development time.

Example:

- The input stream from Twitter feed.
- The analysis PE probes the messages for positive or negative tone.
- The connection between the source and the analysis PE is known at development time.
- Explicitly connecte the output port of the source PE to the input port of the analysis PE.
Creating the segments of an application topology that are not fully known at development time.
Dynamic Flow Composition

▶ Creating the segments of an application topology that are not fully known at development time.

▶ Example:

- An application with an analysis PE that can consume multiple input streams.
- The input sources are dynamic (appear and disappear).
- The connection between the analysis PE and sources can be specified implicitly at development time.
Nested Flow Composition

- Addresses the modularity problem in large scale flow graphs.
- **Group** a subset of the flow graph as a regular PE.
- Producing smaller and more manageable views of the overall data flow graph.
Data Flow Manipulation
Data Flow Manipulation

- How the streaming data is manipulated by the different PE instances in the flow graph?

- PEs properties:
  - PEs tasks
  - PEs states
  - Windowing
  - Selectivity and arity
PEs Tasks (1/2)

- **Edge adaptation**: converting data from external sources into tuples that can be consumed by downstream PEs.
PEs Tasks (1/2)

- **Edge adaptation**: converting data from external sources into tuples that can be consumed by downstream PEs.

- **Aggregation**: collecting and summarizing a subset of tuples from one or more streams.
PEs Tasks (1/2)

- **Edge adaptation**: converting data from external sources into tuples that can be consumed by downstream PEs.

- **Aggregation**: collecting and **summarizing** a subset of tuples from one or more streams.

- **Splitting**: **partitioning** a stream into multiple streams.
PEs Tasks (1/2)

- **Edge adaptation**: converting data from external sources into tuples that can be consumed by downstream PEs.

- **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.
PEs Tasks (2/2)

- **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.
▶ **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*, ...

Amir H. Payberah (KTH)
A PE can either maintain internal state across tuples while processing them, or process tuples independently of each other.

Stateful vs. stateless tasks
Stateless tasks: do not maintain state and process each tuple independently of prior history, or even from the order of arrival of tuples.
Stateless tasks: do not maintain state and process each tuple 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.
- **Stateful** tasks: involves maintaining information across different tuples to detect complex patterns.
Stateful tasks: involves maintaining information across different tuples to detect complex patterns.

A PE is usually a synopsis of the tuples received so far.

A subset of recent tuples kept in a window buffer.
- **Window**: a buffer associated with an input port to retain previously received tuples.
Windowing (1/3)

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

▶ **Window policies** define the operational semantics of a window: eviction policy and trigger policy.
Windowing (1/3)

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

- **Window policies** define the **operational semantics** of a window: eviction policy and trigger policy.

- **Eviction policy**: determines the **properties** of the tuples that can be held in the buffer.
  - For example by a **property of the window**, e.g., buffer’s capacity.
Window: a buffer associated with an input port to retain previously received tuples.

Window policies define the operational semantics of a window: eviction policy and trigger policy.

Eviction policy: determines the properties of the tuples that can be held in the buffer.
  - For example by a property of the window, e.g., buffer’s capacity.

Trigger policy: determines how often the data buffered in the window gets processed by the operator internal logic.
Windowing (2/3)

- **Four different windowing management policies.**

- **Count-based policy:** characterized by the **maximum number** of tuples a window buffer can hold.

- **Delta-based policy:** specified using a **delta threshold** value and a tuple attribute.

- **Time-based policy:** specified using a **wall-clock time** period.

- **Punctuation-based policy:** a window buffer becomes ready for processing every time a **punctuation** is received.
Two types of windows: tumbling and sliding
- Both store tuples in the order they arrive.
- They differ in the eviction and trigger policies.

**Tumbling window**: supports batch operations.
- When the buffer fills up, all the tuples are evicted.

```
   [ ]  1  2  1  3  2  1  4  3  2  1  [ ]  5  6  5
```

**Sliding window**: supports incremental operations.
- When the buffer fills up, older tuples are evicted.

```
   [ ]  1  2  1  3  2  1  4  3  2  1  5  4  3  2  6  5  4  3
```
Selectivity and Arity

- **Selectivity**: the relationship between the number of tuples produced and the number of tuples it ingested.
  - Fixed and variable
Selectivity and Arity

- **Selectivity**: the relationship between the number of tuples produced and the number of tuples it ingested.
  - Fixed and variable

- **Arity**: the number of ports an operator has.
  - One-to-one (1:1)
  - One-to-at-most-one (1:[0, 1])
  - One-to-many (1:N)
  - Many-to-one (M:1)
  - Many-to-many (M:N)
SPS Runtime System
At runtime, an application is represented by one or more jobs.

Jobs are deployed as a collection of PEs.

Job management component must identify and track individual PEs, the jobs they belong to, and associate them with the user that instantiated them.
**Logical plan**: a data flow graph, where the vertices correspond to PEs, and the edges to stream connections.

**Physical plan**: a data flow graph, where the vertices correspond to OS processes, and the edges to transport connections.
Logical Plan vs. Physical Plan (2/3)

Logical plan

Different physical plans
How to map a network of PEs onto the physical network of nodes?

- Parallelization
- Fault tolerance
- Optimization
Parallelization
Parallelization

▶ How to scale with increasing the number queries and the rate of incoming events?

▶ Three forms of parallelisms.
  • Pipelined parallelism
  • Task parallelism
  • Data parallelism
Pipelined Parallelism

- **Sequential stages** of a computation execute **concurrently** for different data items.
- **Independent processing** stages of a larger computation are executed **concurrently** on the same or distinct data items.
The same computation takes place **concurrently** on different data items.
How to allocate data items to each computation instance?

- **Broadcast**
- **Shuffle**
- **Key-based**
Fault Tolerance
The recovery methods of streaming frameworks must take:

- **Correctness**, e.g., data loss and duplicates
- **Performance**, e.g., low latency
Recovery Methods (2/2)

- GAP recovery
- Rollback recovery
- Precise recovery
- The weakest recovery guarantee
- A new task takes over the operations of the failed task.
- The new task starts from an empty state.
- Tuples can be lost during the recovery phase.
Rollback Recovery

- The information loss is avoided, but the output may contain duplicate tuples.

- Three types of rollback recovery:
  - Active backup
  - Passive backup
  - Upstream backup
Each processing node has an associated **backup node**.

Both primary and backup nodes are given the **same** input.

The output tuples of the **backup node** are **logged at the output queues** and they are **not sent downstream**.

If the **primary** fails, the **backup** takes over by **sending the logged tuples** to all downstream neighbors and then continuing its processing.
Rollback Recovery - Passive Backup

- Periodically check-points processing state to a shared storage.

- The backup node takes over from the latest checkpoint when the primary fails.

- The backup node is always equal or behind the primary.
Rollback Recovery - Upstream Backup

- **Upstream nodes** store the tuples until the downstream nodes acknowledge them.

- If a node fails, an empty node rebuilds the latest state of the failed primary from the logs kept at the upstream server.

- There is no backup node in this model.
Precise Recovery

- Post-failure output is exactly the same as the output without failure.

- Can be achieved by modifying the algorithms for rollback recovery.
  - For example, in passive backup, after a failure occurs the backup node can ask the downstream nodes for the latest tuples they received and trim the output queues accordingly to prevent the duplicates.
Optimization
Performance Optimization

- Data sources continuously producing the data.

- Applications must keep up with the rate of the input data they process.

- Optimization techniques:
  - Early data volume reduction
  - Redundancy elimination
  - Operator fusion
  - Tuple batching
  - Load balancing
  - Load shedding
Early Data Volume Reduction

- Reducing the data volume as early as possible.
  - Sampling, filtering, quantization, projection, and aggregation.

- Operator reordering
  - Executing the computationally cheaper operator and/or the more selective operator earlier reduces the overall cost.
- **Removing** the *redundant* segments from a data flow graph.
Operator Fusion

- It changes only the physical layout.

- If two operators of the two ends of a stream connection are placed on different hosts: non-negligible network cost

- But, if these two operators are fused inside a single PE in the same host: the direct call is used

- Operator fusion can be effective if the per-tuple processing cost of the operators being fused is low compared to the cost of transferring the tuples across the stream connection.
Tuple Batching

- Processing a group of tuples in every iteration of an operator’s internal algorithm.

- Can increase the throughput at the expense of higher latency.
Load Balancing

- Flow partitioning to **distribute the workload**, e.g., data or task parallelism.

- Distributing the **load evenly** across the different subflows.
Load Shedding

- Used by an operator to reduce the amount of computational resources it uses.
  - Sidestepping sustained increases in memory utilization.
  - Limiting the amount of work an operator performs per unit of time: decrease the operator latency, and improve the throughput.

- Different techniques: dropping incoming tuples, data reduction techniques (e.g., sampling), ...
Distributed Messaging System
What is Messaging? (1/2)

- Suppose you have a website, and every time someone loads a page, you send a user viewed page event to a messaging system.
What is Messaging? (1/2)

- Suppose you have a website, and every time someone loads a page, you send a user viewed page event to a messaging system.

- The consumers may do any of the following:
  - Store the message in HDFS for future analysis
  - Count page views and update a dashboard
  - Trigger an alert if a page view fails
  - Send an email notification to another user
What is Messaging? (1/2)

- Suppose you have a website, and every time someone loads a page, you send a user viewed page event to a messaging system.

- The consumers may do any of the following:
  - Store the message in HDFS for future analysis
  - Count page views and update a dashboard
  - Trigger an alert if a page view fails
  - Send an email notification to another user

- A messaging system lets you decouple all of this work from the actual web page serving.
▶ Messaging system is a way of implementing near-realtime asynchronous computation.

▶ Messages can be added to the messaging systems when something happens.
Messaging system is a way of implementing near-realtime asynchronous computation.

Messages can be added to the messaging systems when something happens.

Consumers read messages from these systems, and process them or take actions based on the message contents.
Existing Messaging Systems

- **Message queues**: ActiveMQ and RabbitMQ
- **Pub/Sub systems**: Kafka and Kestrel
- **Log aggregation systems**: Flume and Scribe
Kafka
Kafka is a distributed, topic oriented, partitioned, replicated commit log service.
Kafka is a distributed, topic oriented, partitioned, replicated commit log service.
Kafka is a distributed, topic oriented, partitioned, replicated commit log service.
Kafka is a distributed, topic oriented, partitioned, replicated commit log service.
Kafka is a distributed, topic oriented, partitioned, replicated commit log service.
Kafka is also a pub-sub messaging system.
Kafka Basic Messaging Terminology

- Kafka maintains feeds of messages in categories called topics.
Kafka Basic Messaging Terminology

- Kafka maintains feeds of messages in categories called topics.
- Processes that publish messages to a Kafka topic called producers.
Kafka Basic Messaging Terminology

- Kafka maintains feeds of messages in categories called topics.
- Processes that publish messages to a Kafka topic called producers.
- Processes that subscribe to topics and process the feed of published messages called consumers.
Kafka Basic Messaging Terminology

- Kafka maintains feeds of messages in categories called topics.
- Processes that publish messages to a Kafka topic called producers.
- Processes that subscribe to topics and process the feed of published messages called consumers.
- Kafka is run as a cluster comprised of one or more servers each of which is called a broker.
Kafka is about logs.

Topics are queues: a stream of messages of a particular type.
Each message is assigned a sequential id called an offset.
Topics are logical collections of partitions (the physical files).

- Ordered
- Append only
- Immutable
Ordering is only guaranteed within a partition for a topic.

Messages sent by a producer to a particular topic partition will be appended in the order they are sent.

A consumer instance sees messages in the order they are stored in the log.
- **Partitions** of a topic are replicated: fault-tolerance.

- A **broker** contains some of the **partitions** for a topic.

- One broker is the **leader** of a partition: all **writes** and **reads** must go to the leader.
Producers

- Producers publish data to the *topics* of their choice.

- Producers are responsible for choosing which *message* to assign to which *partition* within the *topic*.
  - Round-robin
  - Key-based
Consumers **pull** a range of messages from brokers.

- **Multiple** consumers can read from the same topic on their **own pace**.
- Consumers maintain the message **offset**.
Consumers and Consumer Groups (2/3)

- Consumers can be organized into consumer groups.
- Each message is delivered to only one of the consumers within the group.
- All messages from one partition are consumed only by a single consumer within each consumer group.
  - A partition is in a topic the smallest unit of parallelism.
If all consumers instances are in one group: a traditional queue with load balancing

If all consumers instances are in different groups: all messages are broadcast to all consumer instances

If many consumers are instances in a group: each consumer instance reads from one or more partitions for a topic
Brokers

- The published messages are stored at a set of servers called brokers.
- Brokers are stateless.
- Messages are kept on log for predefined period of time.
Kafka uses Zookeeper for the following tasks:

- Detecting the addition and the removal of brokers and consumers.
- Triggering a rebalance process in each consumer when the above events happen.
- Maintaining the consumption relationship and keeping track of the consumed offset of each partition.
Delivery Guarantees

- Kafka guarantees that messages from a single partition are delivered to a consumer in order.

- There is no guarantee on the ordering of messages coming from different partitions.

- Kafka only guarantees at-least-once delivery.

- No exactly-once delivery: two-phase commits
Summary
Summary

- SPS vs. DBMS
- Data stream, unbounded data, tuples
- PEs and dataflow
- SPS programming languages: declarative, imperative, pattern-based, visualized
- SPS data flow: composition and manipulation
- SPS runtime: parallelization, fault-tolerance, optimization
Summary

- Messaging system: decoupling
- Kafka: distributed, topic oriented, partitioned, replicated log service
- Logs, topics, partition
- Kafka architecture: producer, consumer (groups), broker, coordinator
Questions?
H. Andrade et al., Fundametal of Stream Processing
  • Sections 1, 2, 3, 4, 5, 7, and 9