Information Flow Processing

Amir H. Payberah
amir@sics.se

Amirkabir University of Technology
(Tehran Polytechnic)

Motivation

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

Motivation

- 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
  - ...

Motivation

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

Motivation

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

- Traditional DBMSs:
  - Store and index data before processing it.
  - Process data only when explicitly asked by the users.

Motivation

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

- Traditional DBMSs:
  - Store and index data before processing it.
  - Process data only when explicitly asked by the users.
  - Both aspects contrast with our requirements.

One Name, Different Technologies

- Several research communities are contributing in this area:
  - Each brings its own expertise
  - Point of view
  - Vocabulary: event, data, stream, ...

One Name, Different Technologies

- Several research communities are contributing in this area:
  - Each brings its own expertise
  - Point of view
  - Vocabulary: event, data, stream, ...

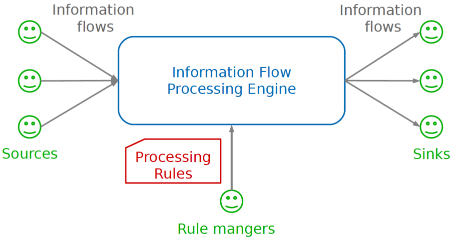
Tower of Babel Syndrome!

Come on! Let's go down and confuse them by making them speak different languages, then they won't be able to understand each other.

Genesis 11:7

Information Flow Processing (IFP)

- Source: produces the incoming information flows
- Sink: consumes the results of processing
- IFP engine: processes incoming flows
- Processing rules: how to process the incoming flows
- Rule manager: adds/removes processing rules



IFP Competing Models

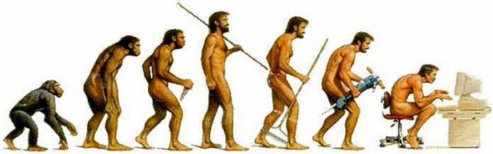
- Data Stream Management Systems (DSMS)
- Complex Event Processing (CEP)

IFP Competing Models

- Data Stream Management Systems (DSMS)
- Complex Event Processing (CEP)

Data Stream Management Systems (DSMS)

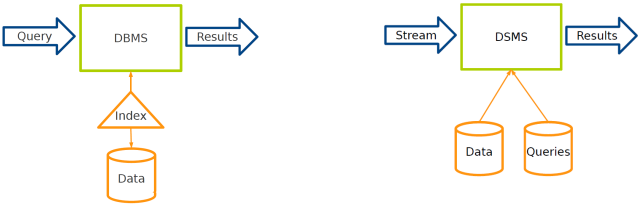
- An evolution of traditional data processing, as supported by DBMSs.



DBMS vs. DSMS (1/3)

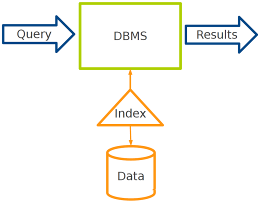
- DBMS: persistent data where updates are relatively infrequent.

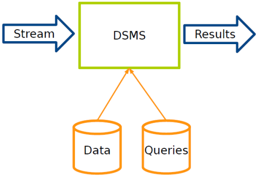
- DSMS: transient data that is continuously updated.



DBMS vs. DSMS (2/3)

- DBMS: runs queries just once to return a complete answer.
- DSMS: executes standing queries, which run continuously and provide updated answers as new data arrives.





DBMS vs. DSMS (3/3)

- Despite these differences, DSMSs resemble DBMSs: both process incoming data through a sequence of transformations based on SQL operators, e.g., selections, aggregates, joins.

Out of Scope of DSMS

- DSMSs focus on producing query answers.

- Detection and notification of complex patterns of elements are usually out of the scope of DSMSs:

Out of Scope of DSMS

- DSMSs focus on producing query answers.
- Detection and notification of complex patterns of elements are usually out of the scope of DSMSs: Complex Event Processing

IFP Competing Models

- Data Stream Management Systems (DSMS)
- Complex Event Processing (CEP)

Complex Event Processing (CEP)

- DSMSs limitation: detecting complex patterns of incoming items, involving sequencing and ordering relationships.

- CEP models flowing information items as notifications of events happening in the external world.
  - They have to be filtered and combined to understand what is happening in terms of higher-level events.

CEP vs. Publish/Subscribe Systems

- CEP systems can be seen as an extension to traditional publish/subscribe systems.

CEP vs. Publish/Subscribe Systems

- CEP systems can be seen as an extension to traditional publish/subscribe systems.

- Traditional publish/subscribe systems consider each event separately from the others, and filter them based on their topic or content.

CEP vs. Publish/Subscribe Systems

- CEP systems can be seen as an extension to traditional publish/subscribe systems.

- Traditional publish/subscribe systems consider each event separately from the others, and filter them based on their topic or content.

- CEPs extend this functionality by increasing the expressive power of the subscription language to consider complex event patterns that involve the occurrence of multiple related events.

IFP Modeling

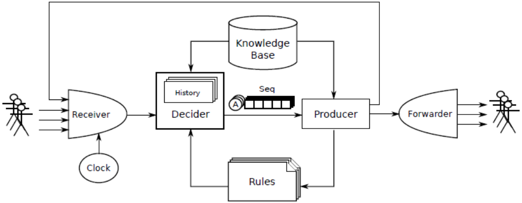
One Model, Several Models

- Different models to capture different viewpoints.
  - Functional model
  - Processing model
  - Time model
  - Data model
  - Rule model
  - Language model
  - Interaction model
  - Deployment model

Functional Model

Functional Model

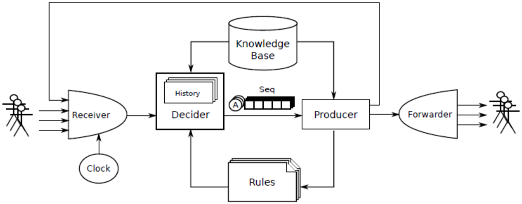
- An abstract architecture of the main functional components of an IFP engine.



[G. Cugolap et al., Processing Flows of Information: From Data Stream to Complex Event Processing, 2012]

Receiver and Clock

- Receiver manages the channels connecting the sources with the IFP engine.
- Clock models periodic processing of their inputs.



[G. Cugolap et al., Processing Flows of Information: From Data Stream to Complex Event Processing, 2012]

Rules, Decider and Producer

- We assume rules can be (logically) decomposed in two parts:
  - `C` $\rightarrow$ `A`
  - `C` is the condition
  - `A` is the action

- Example (in CQL):
  `Select IStream(Count(*))` (action)
  `From F1 [Range 1 Minute] Where F1.A > 0` (condition)

Rules, Decider and Producer

- We assume rules can be (logically) decomposed in two parts:
  - `C` $\rightarrow$ `A`
  - `C` is the condition
  - `A` is the action

- Example (in CQL):

```
Select IStream(Count(*)) (action)
From F1 [Range 1 Minute] Where F1.A > 0 (condition)
```

- This way we can split processing in two phases:
  - Decider: determines the items that trigger the rule.
  - Producer: use those items to produce the output of the rule.

Detection-Production Cycle (1/2)

- The Detector evaluates all the rules in the Rules store to find those whose condition part is true.

Detection-Production Cycle (1/2)

- The Detector evaluates all the rules in the Rules store to find those whose condition part is true.

- With the newly arrived information, the Detector may also use the information present in the Knowledge Base.

Detection-Production Cycle (1/2)

- The Detector evaluates all the rules in the Rules store to find those whose condition part is true.

- With the newly arrived information, the Detector may also use the information present in the Knowledge Base.

- At the end of this phase we have a set of rules that have to be executed.

Detection-Production Cycle (2/2)

- The Producer takes the information and executes each triggered rule (i.e., its action part).

Detection-Production Cycle (2/2)

- The Producer takes the information and executes each triggered rule (i.e., its action part).

- In executing rules, the Producer may combine the items that triggered the rule.
  - Received from the Decider together with the information present in the Knowledge Base.

Detection-Production Cycle (2/2)

- The Producer takes the information and executes each triggered rule (i.e., its action part).

- In executing rules, the Producer may combine the items that triggered the rule.
    - Received from the Decider together with the information present in the Knowledge Base.

- Usually, these new items are sent to sinks, through the Forwarder, or sent internally to be processed again.

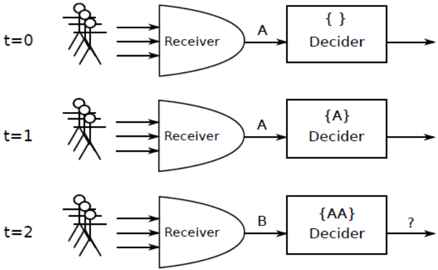
Processing Model

Processing Model

- Three policies affect the behavior of the system:
  - The selection policy
  - The consumption policy
  - The load shedding policy

Selection Policy

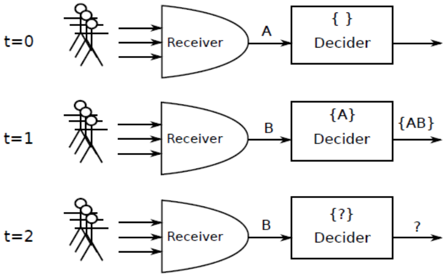
- Determines if a rule fires once or multiple times and the items actually selected from the History.



[G. Cugolap et al., Processing Flows of Information: From Data Stream to Complex Event Processing, 2012]

Consumption Policy

- Determines how the history changes after firing of a rule: what happens when new items enter the Decider



[G. Cugolap et al., Processing Flows of Information: From Data Stream to Complex Event Processing, 2012]

Load Shedding Policy

- A technique adopted by some IFP systems to deal with burst inputs.
- It can be described as an automatic drop of information items when the input rate becomes too high for the processing capabilities of the engine.

Time Model

Time Model

- The relationship between the information items flowing into the IFP engine and the passing of time.
- Ability of an IFP system to associate some kind of happened-before (ordering) relationship to information items.
- Four classes:
  1. Stream-only
  2. Causal
  3. Absolute
  4. Interval

Stream-Only Time Model

- Not any special meaning to time.

Stream-Only Time Model

- Not any special meaning to time.
- Timestamps are used mainly to order items at the frontier of the engine (i.e., within the receiver).

Stream-Only Time Model

- Not any special meaning to time.
- Timestamps are used mainly to order items at the frontier of the engine (i.e., within the receiver).
- They are lost during processing.
  - The ordering and timestamps of the output stream are conceptually separate from the ordering and timestamps of the input streams.

Stream-Only Time Model

- Not any special meaning to time.
- Timestamps are used mainly to order items at the frontier of the engine (i.e., within the receiver).
- They are lost during processing.
  - The ordering and timestamps of the output stream are conceptually separate from the ordering and timestamps of the input streams.
- Example: CQL/Stream

```
Select DStream(*)
From F1[Rows 5], F2[Range 1 Minute]
Where F1.A = F2.A
```

Causal Time Model

- Each item has a label reflecting some kind of causal relationship.

Causal Time Model

- Each item has a label reflecting some kind of causal relationship.
- Partial order

Causal Time Model

- Each item has a label reflecting some kind of causal relationship.
- Partial order
- Example: Gigascope

```
Select count(*)
From A, B
Where A.a-1 <= B.b and A.a+1 > B.b
```

`A.a` and `B.b` monotonically increase

Absolute Time Model

- Information items have an associated timestamp.

Absolute Time Model

- Information items have an associated timestamp.
- Defining a single point in time, wrt a (logically) unique clock.

Absolute Time Model

- Information items have an associated timestamp.
- Defining a single point in time, wrt a (logically) unique clock.
- Information items can be timestamped at source or entering the engine

Absolute Time Model

- Information items have an associated timestamp.
- Defining a single point in time, wrt a (logically) unique clock.
- Information items can be timestamped at source or entering the engine
- Total order

Absolute Time Model

- Information items have an associated timestamp.
- Defining a single point in time, wrt a (logically) unique clock.
- Information items can be timestamped at source or entering the engine
- Total order
- Example: TESLA/T-Rex

```
Define Fire(area: string, measuredTemp: double)
From Smoke(area=$a) and last Temp(area=$a and
value>45) within 5 min. from Smoke
Where area=Smoke.area and measuredTemp=Temp.value
```

Interval Time Model

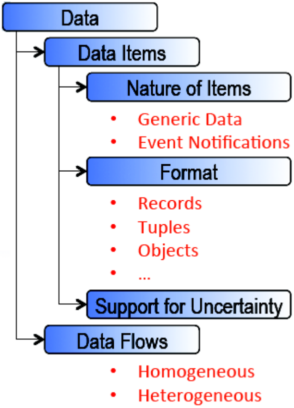
- Associate items with an interval, i.e., two timestamps taken from a global time.

- Usually representing: the time when the related event started, the time when it ended.

Data Model

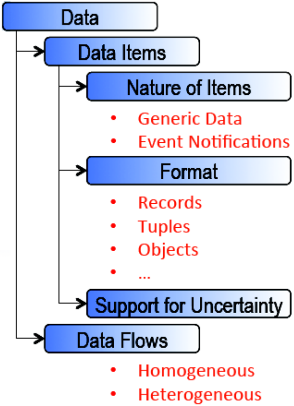
Data Model

- Studies how the different systems
  - Represent single data items
  - Organize them into data flows



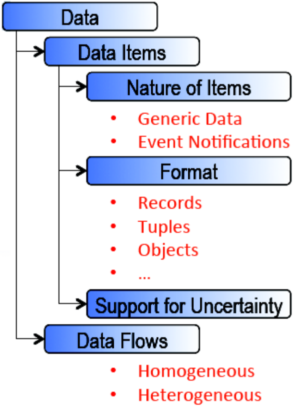
Nature of Items

- The meaning we associate to information items
  - Generic data
  - Event notifications
- Influences other aspects of an IFP system
  - Time model
  - Rule language
  - Semantics of processing



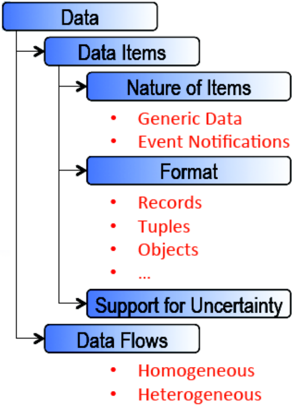
Format of Items

- How information is represented
- Influences the way items are processed, e.g., relational model requires tuples



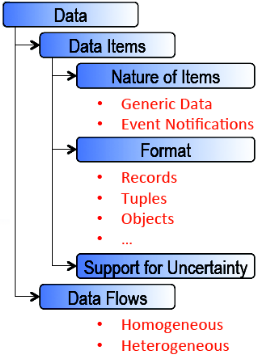
Support for Uncertainty

- Ability to associate a degree of uncertainty to information items.
- When present, probabilistic information is usually exploited in rules during processing.



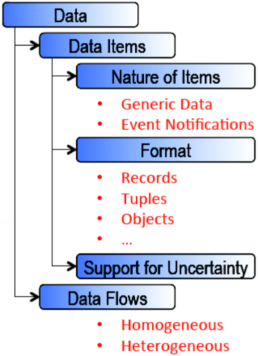
Data Flows

- Homogeneous
  - Each flow contains data with the same format and kind.
  - E.g., a sequence of unbounded tuples generated continuously in time: $\cdots(a_1, a_2, \cdots, a_n, t-1)(a_1, a_2, \cdots, a_n, t)(a_1, a_2, \cdots, a_n, t+1)\cdots$, where a_i denotes an attribute.



Data Flows

- Homogeneous
  - Each flow contains data with the same format and kind.
  - E.g., a sequence of unbounded tuples generated continuously in time: $\cdots(a_1, a_2, \cdots, a_n, t-1)(a_1, a_2, \cdots, a_n, t)(a_1, a_2, \cdots, a_n, t+1)\cdots$, where a_i denotes an attribute.
- Heterogeneous
  - Information flows are seen as channels connecting sources, processors, and sinks
  - Each channel may transport items with different kind and format.



Rule Model

Rule Model

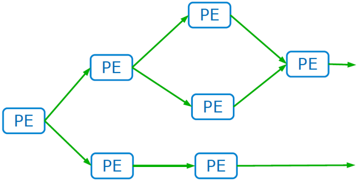
- Rules are classified into two macro classes:
  - Transforming rules
  - Detecting rules

Transforming Rules (1/2)

- No explicit distinction between detection and production.

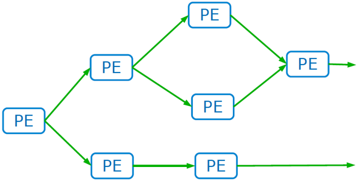
Transforming Rules (1/2)

- No explicit distinction between detection and production.
- Execution plan of primitive operators (processing elements (PE)).
  - A logical network of PEs connected in a DAG.



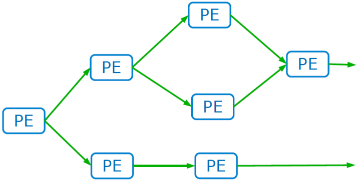
Transforming Rules (1/2)

- No explicit distinction between detection and production.
- Execution plan of primitive operators (processing elements (PE)).
  - A logical network of PEs connected in a DAG.
- Each PE transforms one or more input flows into one or more output flows.



Transforming Rules (2/2)

- PEs execute independently and in parallel
- Not synchronized
- Communicate through messaging
- Upstream node vs. downstream node



Detecting Rules

- An explicit distinction between detection and production.

- Usually, the detection is based on a logical predicate that captures patterns of interest in the history of received items.

Language Model

Language Model

- Following the rule model, we define two classes of languages:
  - Transforming languages: declarative languages and imperative languages
  - Detecting languages: patternbased

Language Model

- Following the rule model, we define two classes of languages:
  - Transforming languages: declarative languages and imperative languages
  - Detecting languages: patternbased
- Specify operations to filter, join, aggregate, ...
- Input flows
- To produce one or more output flows

Declarative Languages

- Specify the expected result rather than the desired execution flow.
- Usually derive from relational languages, e.g., SQL

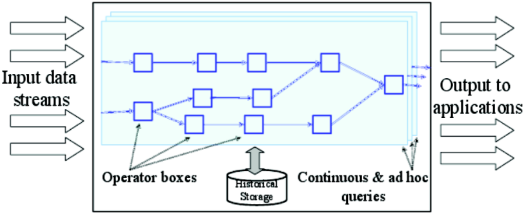
Declarative Languages

- Specify the expected result rather than the desired execution flow.
- Usually derive from relational languages, e.g., SQL
- Example CQL/Stream:

```
Select IStream(*)
From F1[Rows 5], F2[Rows 10]
Where F1.A = F2.A
```

Imperative Languages

- Specify the desired execution flow
- Starting from primitive operators, i.e., PEs
- Usually adopt a graphical notation



Pattern-Based Languages

- Specify a firing condition as a pattern
- Select a portion of incoming flows
- The action uses selected items to produce new knowledge

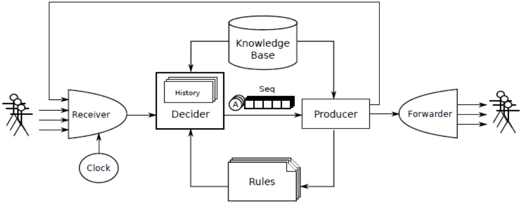
Pattern-Based Languages

- Specify a firing condition as a pattern
- Select a portion of incoming flows
- The action uses selected items to produce new knowledge
- Example, TESLA / T-Rex

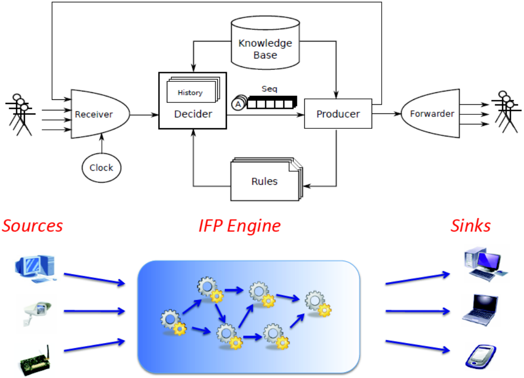
```
Define Fire(area:  string, measuredTemp:  double)
From Smoke(area=$a) and last
Temp(area=$a and value>45)
within 5 min.  from Smoke
Where area=Smoke.area and measuredTemp=Temp.value
```

Interaction Model

Interaction Model

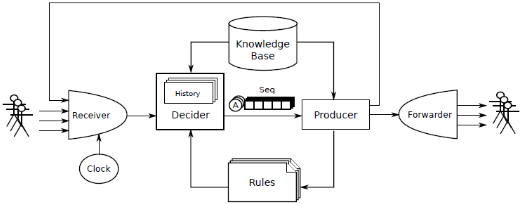


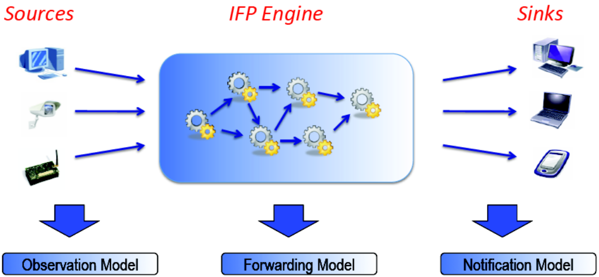
Interaction Model



- Processing Element (PE): a processing unit in a IFP system.
- How do PEs interact?

Interaction Model





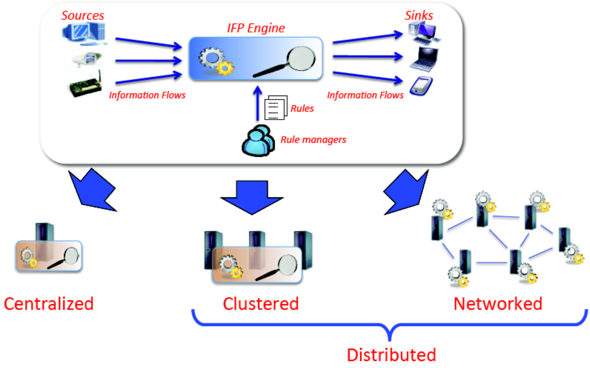
Observation Model	Forwarding Model	Notification Model
• Push • Pull	• Push • Pull	• Push • Pull

Deployment Model

Deployment Model

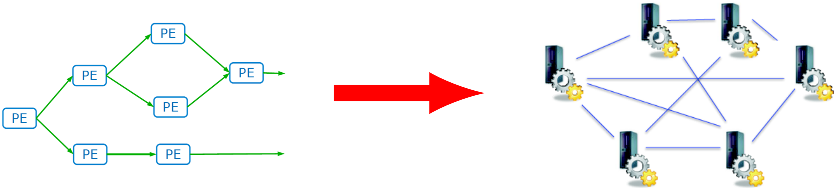
- IFP applications may include a large number of sources, sinks, and PEs.
- Possibly dispersed over a wide geographical area.
- How the components of the functional model can be distributed to achieve scalability.

Deployment Model



Deployment Model

- Given a network of PEs.
- How to map it onto the physical network of nodes



Scaling Mechanism

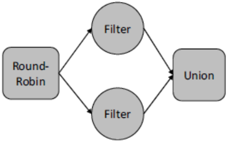
- How to scale with increasing the number queries and the rate of incoming events?

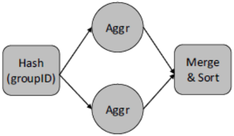
Scaling Mechanism

- How to scale with increasing the number queries and the rate of incoming events?
- Two main solutions:
  - Data partitioning: a reasonable data partitioning and merging scheme as well as mechanisms to detect points for parallelization.
  - Query Partitioning: to distribute the load across available hosts and to achieve a load balance between these machines.

Data Partitioning (1/3)

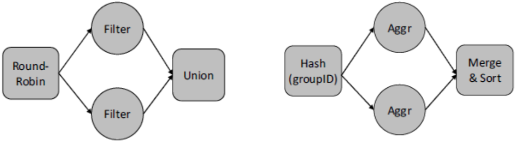
- Early approaches parallelize operators by introducing a splitter and merge operator.





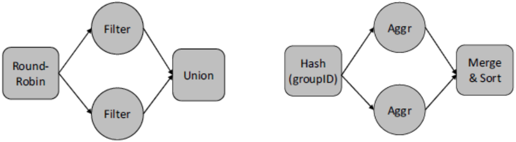
Data Partitioning (1/3)

- Early approaches parallelize operators by introducing a splitter and merge operator.
- The merge operator: union or sort



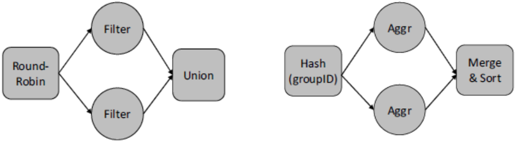
Data Partitioning (1/3)

- Early approaches parallelize operators by introducing a splitter and merge operator.
- The merge operator: union or sort
- E.g., parallelize a filter operation using a round robin scheme.



Data Partitioning (1/3)

- Early approaches parallelize operators by introducing a splitter and merge operator.
- The merge operator: union or sort
- E.g., parallelize a filter operation using a round robin scheme.
- E.g., parallelize an aggregation using a hash scheme.



Data Partitioning (2/3)

- In more recent systems:
- E.g., in Storm a user can express data parallelism by defining the number of parallel tasks per operator.
- E.g., S4 creates a PE for each new key in the data stream.

Data Partitioning (2/3)

- In more recent systems:
- E.g., in Storm a user can express data parallelism by defining the number of parallel tasks per operator.
- E.g., S4 creates a PE for each new key in the data stream.
- In both approaches the user needs to understand the data parallelism and explicitly enforce in its code the sequential ordering.

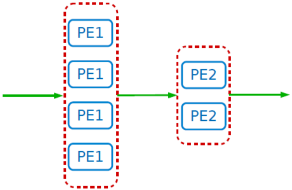
Data Partitioning (3/3)

- An auto-parallelization approach has recently be proposed.

- A combination of compiler and runtime:
  - The compiler detects regions for parallelization.
  - The system runtime guarantees that output tuples follow the same order as for a sequential execution.

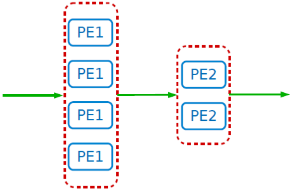
Query Partitioning

- Operator placement problem: the problem of assigning a set of operators to a set of available hosts.
- A single PE can be running in parallel on different nodes.



Query Partitioning

- Operator placement problem: the problem of assigning a set of operators to a set of available hosts.
- A single PE can be running in parallel on different nodes.
- E.g., SEEP can dynamically vary the number of processing nodes within the system based on the workload.



Fault Tolerance Mechanism

Recovery Methods

- The recovery methods of streaming frameworks must take:
  - Correctness, e.g., data loss and duplicates
  - Performance, e.g., low latency

Basic Idea

- Each processing node has an associated backup node.

Basic Idea

- Each processing node has an associated backup node.
- The backup node's stream processing engine is identical to the primary one.

Basic Idea

- Each processing node has an associated backup node.
- The backup node's stream processing engine is identical to the primary one.
- But the state of the backup node is not necessarily the same as that of the primary.

Basic Idea

- Each processing node has an associated backup node.
- The backup node's stream processing engine is identical to the primary one.
- But the state of the backup node is not necessarily the same as that of the primary.
- If a primary node fails, its backup node takes over the operation of the failed node.

Recovery Methods

- GAP recovery
- Rollback recovery
- Precise recovery

GAP 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.

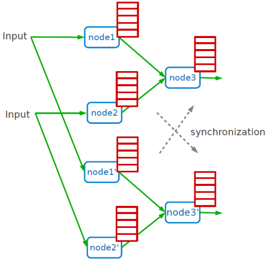
Rollback Recovery

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

- Three types of rollback recovery:
  - Active backup
  - Passive backup
  - Upstream backup

Rollback Recovery - Active Backup

- 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

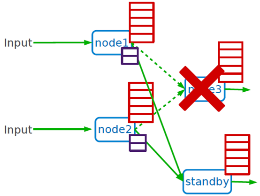
- Periodically check-points processing state to a shared storage.
- The backup node takes over from the latest checkpoint when the primary fails.
- The backup node is always equal or behind the primary.

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.

Brief History of IFP Systems

First Generation

- A stand-alone prototypes or as extensions of existing database engines.
- They were developed with a specific use case in mind and are very limited regarding the supported operator types as well as available functionalities.
- Niagara, Telegraph, Aurora, ...

First Generation Example - Aurora

- A single site stream-processing engine (centralized).
- DAG based processing model for streams.
- Push-based strategy.
- The first Aurora did not support fault tolerance.
- Stream Query Algebra (SQuAl), i.e., SQL with additional features, e.g., windowed queries.

Second Generation

- Systems extended the ideas of data stream processing with advanced features such as fault tolerance, adaptive query processing, as well as an enhanced operator expressiveness.

- Borealis, CEDR, System S, CAPE, ...

Second Generation Example - Borealis

- Distributed version of Aurora.
- Advanced functionalities on top of Aurora:
  - Dynamic revision of query results: correct errors in previously reported data.
  - Dynamic query modifications: change certain attributes of the query at runtime.
- Pull-based strategy.
- Rollback recovery with active backup.

Third Generation

- Driven by the trend towards cloud computing: highly scalable and robust towards faults.

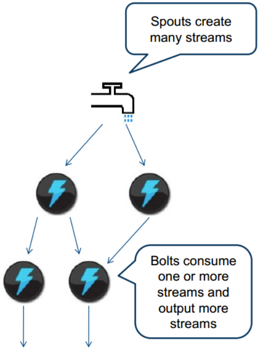
- Storm, Apache S4, D-Streams, SEEP, StreamCloud, ...

Third Generation Example - Storm (1/2)

- Stream processing is guaranteed: a message cannot be lost due to node failures.

- DAG based processing:
  - the DAG is called Topology
  - the PEs are called Bolts
  - the stream sources are called Spouts

- It does not have an explicit programming paradigm.



Third Generation Example - Storm (2/2)

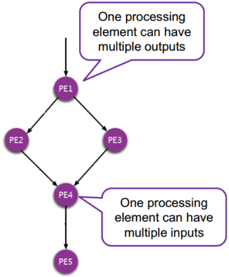
- Pull-based strategy.

- Rollback recovery with upstream backup.

- Three sets of nodes:
  - Nimbus: distributes the code among the worker nodes, and keeps track of the progress of the worker nodes
  - Supervisor: the set of worker nodes
  - Zookeeper: coordination between supervisor nodes and the Nimbus

- Built by twitter

Third Generation Example - S4 (1/2)



- S4: Simple Scalable Streaming System.
- Constructing a DAG structure of PEs at runtime.
  - A PE is instantiated for each value of the key attribute.
- The processing model is inspired by MapReduce.
- Events are dispatched to nodes according to their key.

Third Generation Example - S4 (2/2)

- Push-based strategy
- GAP recovery
- Communication layer: coordination between the processing nodes and the messaging between nodes.
  - Uses Zookeeper
- Built by yahoo

Summary

- IFP: DSMS and CEP
- IFP modeling: functional, processing, time, data, rule, language, interaction, deployment
- Recovering models: GAP, Rollback, and Precise

Questions?

Acknowledgements

Some slides and pictures were derived from G. Cugola slides (Politecnico di Milano).