Streaming data analytics: Frameworks

Stream Processing Frameworks for (Big) Streaming Data Analytics

- Several frameworks have been proposed to process in real-time or in near real-time data streams
  - Apache Spark (Streaming component)
  - Apache Storm
  - Apache Flink
  - Apache Samza
  - Apache Apex
  - Apache Flume
  - Amazon Kinesis Streams
  - …
- All these frameworks use a cluster of servers to scale horizontally with respect to the (big) amount of data to be analyzed

2

Comparison among the state of the art streaming frameworks

3

Some of the most used Data Streaming Frameworks

- Apache Spark Streaming
  - Micro-batch applications
  - Processes each record **exactly once**

4

Some of the most used Data Streaming Frameworks

- Apache Storm
  - Continuous/real-time computation
    - Very low latency
  - Processes each record **at least once** in real-time
    - Each record could be processed multiple times
    - Hence, may update mutable state twice
  - Apache Storm Trident API
    - Micro-batch
    - A running "modality" of Apache Storm that processes each record **exactly once**
    - Slower than the Apache Storm version

5

Some of the most used Data Streaming Frameworks

- Apache Flink
  - Continuous/real-time stateful computations over data streams
    - Low latency
  - Processes each record **exactly once**

6

Introduction to Apache Storm

Apache Storm

- Apache Storm™ is a distributed framework that is used for real-time processing of data streams
  - E.g., Tweets analysis, Log processing, …
- Currently, it is an open source project of the Apache Software Foundation
  - http://storm.apache.org/
- It is implemented in Clojure and Java
  - 12 core committers, plus ~ 70 contributors

8

Apache Storm history

- Storm was first developed by Nathan Marz at BackType
  - BackType was a company that provided social search applications
- Later (2011), BackType was acquired by Twitter, and now it is a critical part of their infrastructure
- Currently, Storm is a project of the Apache Software Foundation (since 2013)

9

Apache Storm: data processing

- Continuous computation
  - Storm can do continuous computation on data streams in real time
    - It can process each message as it comes
    - An example of continuous computation is streaming trending topics detection on Twitter
- Real-time analytics
  - Storm can analyze and extract insights or complex knowledge from data that come from several real-time data streams

10

Features of Storm

- Storm is
  - Distributed
  - Horizontally scalable
  - Fast
  - Fault tolerant
  - Reliable - Guaranteed data processing
  - Easy to operate
  - Programming language agnostic

11

Features of Storm

- Distributed
  - Storm is a distributed system than can run on a cluster of commodity servers
- Horizontally scalable
  - Storm allows adding more servers (nodes) to your Storm cluster and increase the processing capacity of your application
  - It is linearly scalable with respect to the number of nodes, which means that you can double the processing capacity by doubling the nodes

12

Features of Storm

- Fast
  - Storm has been reported to process up to 1 million tuples per second per node
- Fault tolerant
  - Units of work are executed by worker processes in a Storm cluster. When a worker dies, Storm will restart that worker (on the same node or on to another node)

13

Features of Storm

- Reliable - Guaranteed data processing
  - Storm provides guarantees that each message (tuple) will be processed **at least once**
  - In case of failures, Storm will replay the lost tuples
  - It can be configured to process each tuple only once
- Easy to operate
  - Storm is simple to deploy and manage
  - Once the cluster is deployed, it requires little maintenance

14

Features of Storm

- Programming language agnostic
  - Even though the Storm platform runs on Java Virtual Machine, the applications that run over it can be written in any programming language that can read and write to standard input and output streams

15

Storm core concepts

16

Storm

- Storm can be considered a distributed Function Programming-like processing of data streams
- It applies a set of functions, in a specific order, on the elements of the input data streams and emits new data streams
  - However, each function can store its state by means of variables
  - Hence, it is not pure functional programming

17

Main concepts

- Tuple
- Data Stream
- Spout
- Bolt
- Topology

18

Data model

- The basic unit of data that can be processed by a Storm application is called a **tuple**
- Each tuple is a predefined list of fields
  - The data type of each field can be common data types, e.g., byte, char, string, integer, ..
  - Or your own data types, which can be serialized as fields in a tuple
- Each field of a tuple has a name

19

Data model

- A tuple is dynamically typed, that is, you just need to define the names of the fields in a tuple and not their data type

20

Data model

- Storm processes streams of tuples
  - Each stream is an unbounded sequence of tuples
- Each stream
  - has a name
  - is composed of homogenous tuples (i.e., tuples with the same structure)
- However, each applications can process multiple, heterogonous, data streams

21

Data model: Example

- Tuple
  - (1.1.1.1, "foo.com")
  IP address   Domain
- Stream of tuples

      …
  (1.1.1.1, "foo.com")
  (2.2.2.2, "bar.net")
  (3.3.3.3, "foo.com")
        …

22

Spout

- Spout
  - It is the component "generating/handling" the input data stream
- Spouts read or listen to data from external sources and publish them (emit in Storm terminology) into streams

23

Spout

- Examples
  - A spout can be used to connect to the Twitter API and emit a stream of tweets
  - A spout can be used to read a log file and emit a stream of composed of the its lines
  - …

24

Spout

- Each spout can emit multiple streams, with different schemas
  - For example, we can implement a spout that reads 10-field records from a log file and emits them as two different streams of 7-tuples and 4-tuples, respectively
- Spouts can be
  - "unreliable" (fire-and-forget)
  - or "reliable" (can replay failed tuples)

25

Bolt

- Bolt
  - It is the component that is used to apply a function over each tuple of a stream
- Bolts consume one or more streams, emitted by spouts or other bolts, and potentially produce new streams

26

Bolt

- Bolts can be used to
  - Filter or transform the content of the input streams and emit new data streams that will be processed by other bolts
  - Or process the data streams and store/persist the result of the computation in some of "storage" (files, Databases, ..)
- Each bolt can emit multiple streams, with different schemas

27

Bolt

- Examples
  - A bolt can be used to extract one field from each tuple of its input stream
  - A bolt can be used to join two streams, based on a common field
  - A bolt can be used to count the occurrences of a set of URLs
  - ...

28

Spouts and Bolts

- The input streams of a Storm cluster are handled by spouts
- Each spout passes the data streams to bolts, which transform them in some way
- Each bolt either persists the data in some sort of storage or passes it to some other bolts
- A Storm program is a chain of bolts making some computations/transformations on the data exposed by spouts and bolts
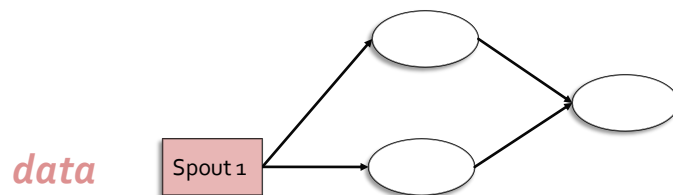
29

Topology

- A Storm topology is an abstraction that defines the graph of the computation
  - It specifies which spouts and bolts are used and how they are connected
- A topology can be represented by a direct acyclic graph (DAG), where each node does some kind of processing and eventually forwards it to the next node(s) in the flow
  - i.e., a topology in Storm wires **data** and **functions** via a **DAG**
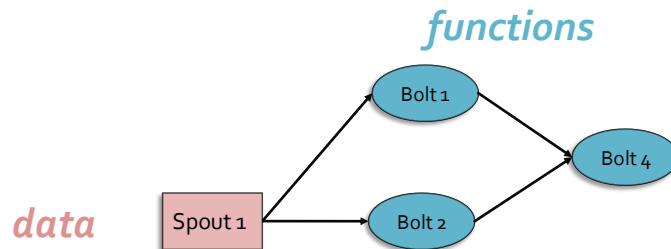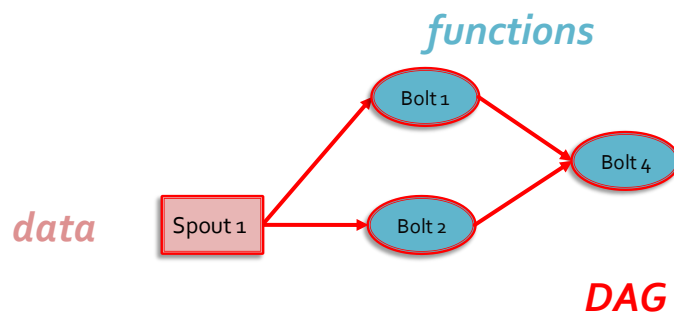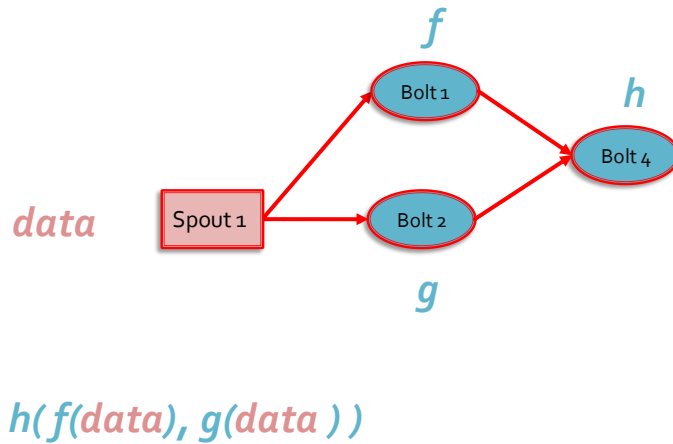
30
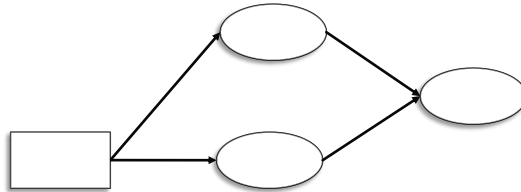
Topology: Example

Topology: Example

data   Spout 1

Topology: Example

functions

data

Topology: Example

functions

data

DAG

Relation Topology – Functional programming

f

h

$data$

g

$h(\,f(data),\,g(data\,)\,)$

Topology: Example #2

$functions$

$data\ 1$

$data\ 2$

28/05/2020

Topology: Example #2

functions
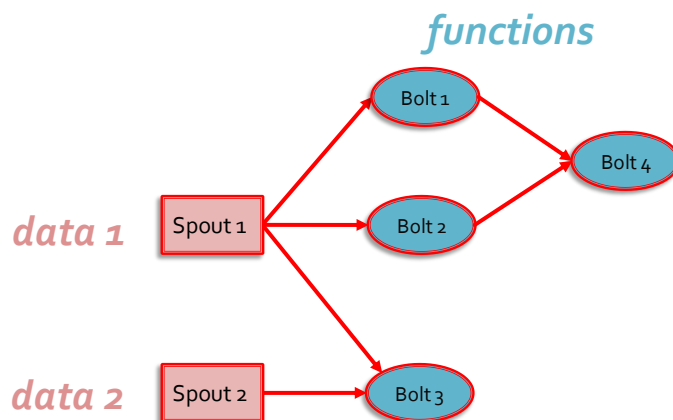
There are two input data
streams in this topology

data 1 — Spout 1 → Bolt 1, Bolt 2, Bolt 3

Bolt 1 → Bolt 4
Bolt 2 → Bolt 4

data 2 — Spout 2 → Bolt 3

Topology: Example #2

functions

data 1 — Spout 1 → Bolt 1, Bolt 2, Bolt 3

Bolt 1 → Bolt 4
Bolt 2 → Bolt 4

data 2 — Spout 2 → Bolt 3

There are also two output
data streams

19
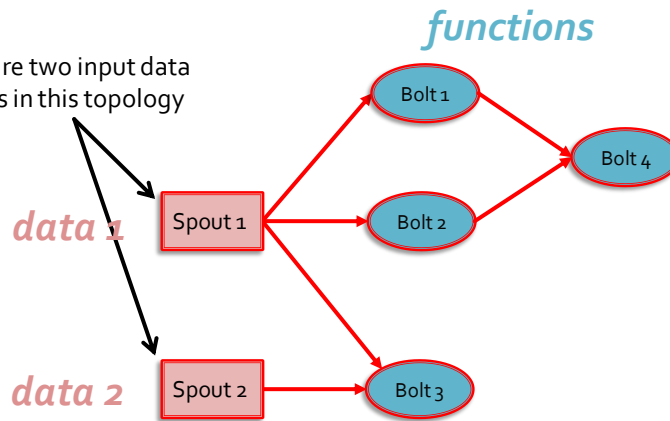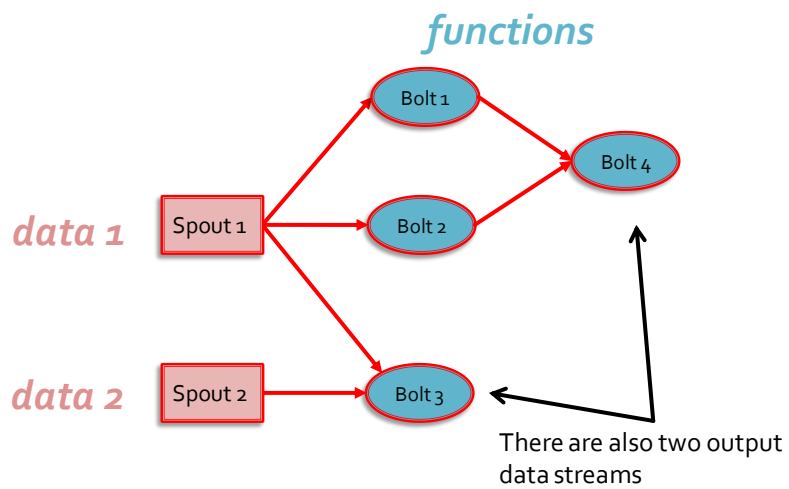
"Execution" of a Topology

- The topology is executed on the servers of the cluster running Storm
  - The system automatically decides which parts of the topology are executed by each server of the cluster
- Each topology runs until its is explicitly killed
- Each cluster can runs multiple topologies at the same time

39

"Execution" of a Topology

- Worker processes
  - Each node in the cluster can run one or more JVMs called **worker processes** that are responsible for processing a part of the topology.
  - Each topology executes across one or more worker processes
  - Each worker process is bound to one of the topologies and can execute multiple components (spouts and/or bolts) of that topology
    - Hence, even if multiple topologies are run at the same time, none of them will share any of the workers

40

"Execution" of a Topology

- Executor
  - Within each worker process, there can be multiple threads that execute parts of the topology. Each of these threads is called an **executor**
  - An executor can execute only one of the components of the topology, that is, any one spout or bolt in the topology
    - But it may run one or more tasks for the same component
  - Each spout or bolt can be associated with many executors and hence executed in parallel

41

"Execution" of a Topology

- Tasks
  - A task is the most granular unit of task execution in Storm
  - Each task is an instance of a spout or bolt
  - A task performs the actual data processing
  - Each spout or bolt that you implement in your code executes as many tasks across the cluster
  - Each task can be executed alone or with another task of the same type (in the same executor)

42

"Execution" of a Topology

- The number of tasks for a component is always the same throughout the lifetime of a topology
  - You set it when you submit the topology
- But the number of executors (threads) for a component can change over time
  - You can add/remove executors for each component

43

Parallelism of the topology

- The parallelism of the topology is given by the number of executors = number of threads
- For each spout/bolt the application can specify
  - The number of executors
    - This value can be changed at runtime
  - The number of tasks
    - This value is set before submitting the topology and cannot be change at runtime

44