This lecture is about the Storm stream processing framework.

Originally, Storm was developed by a startup which was acquired by Twitter.

Later the project was open sourced in the Apache foundation.

This lecture is a rapid presentation of Storm and a demonstration (without any labwork).

Apache Storm

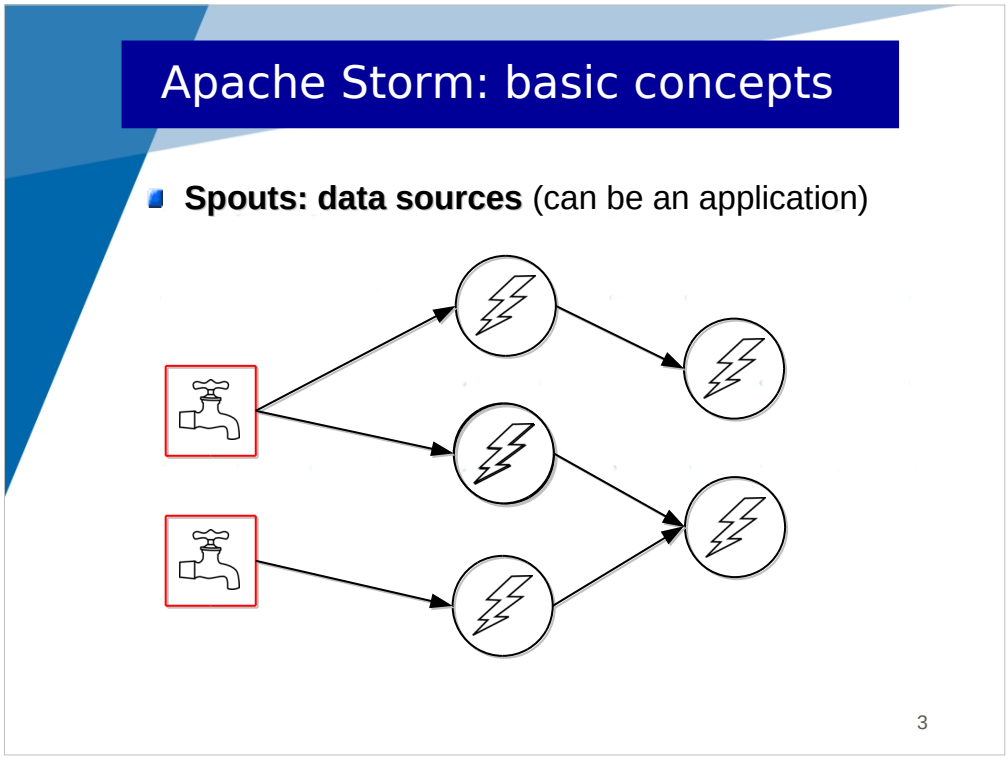
- Processing of data streams
  - Real-time (on the fly)
  - Pretty much like Spark streaming
- Developed in Clojure (a dialect of Lisp)
- As Hadoop, Spark or Spark-streaming
  - Distributed
  - Reliable
  - Open-source (Apache)
  - Used by large companies (e.g. Twitter)

2

Storm is a platform for the real time processing of data streams. It has many similarities with Spark Streaming regarding the goals, but differs regarding the implementation strategy.

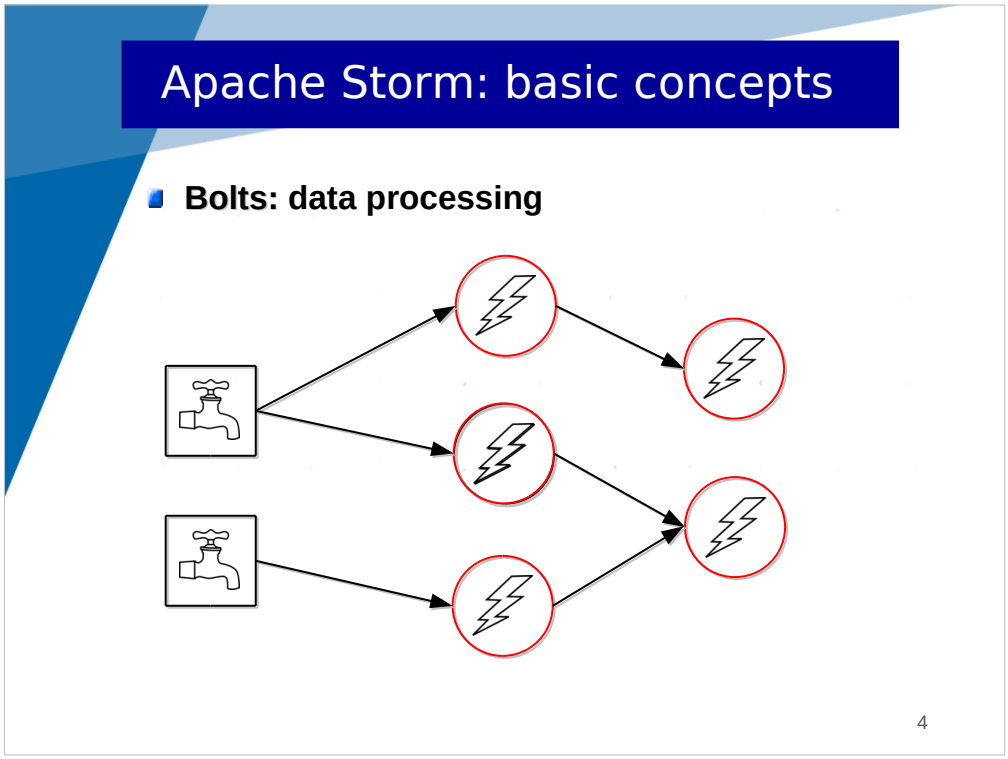
Storm was developed in Clojure (functional programming).

It shares many characteristics with big data platforms like Hadoop, Spark or Spark Streaming: distributed reliable, opensource and used by large companies.



Its architecture relies on 2 types of nodes (nodes here do not refer to machines, but to daemons which include processing) : Spouts and Bolts.

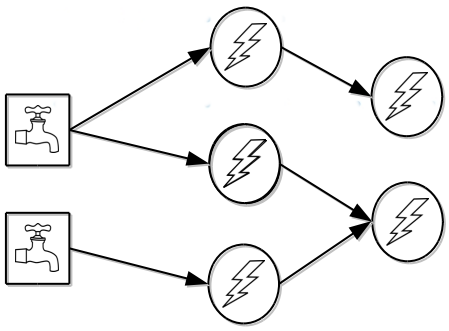
A Spout is a data source (in red in the figure), which may acquire data from a file or a network connection. A spout can emit data in direction of several bolts.



A Bolt is a data processing unit which includes a particular treatment on received data. It can emit data in direction of several other bolts.

Apache Storm: basic concepts

- **Topology:** interconnection of spouts and bolts
  - Represent an Apache Storm application
- Execute indefinitely (unlike MapReduce applications)



5

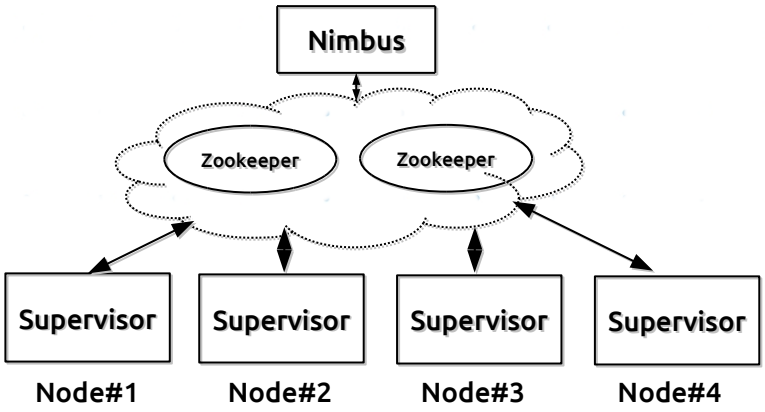
A topology is an interconnection of spouts and bolts.

It corresponds to an global application.

It executes indefinitely like Spark Streaming applications.

Apache Storm: architecture

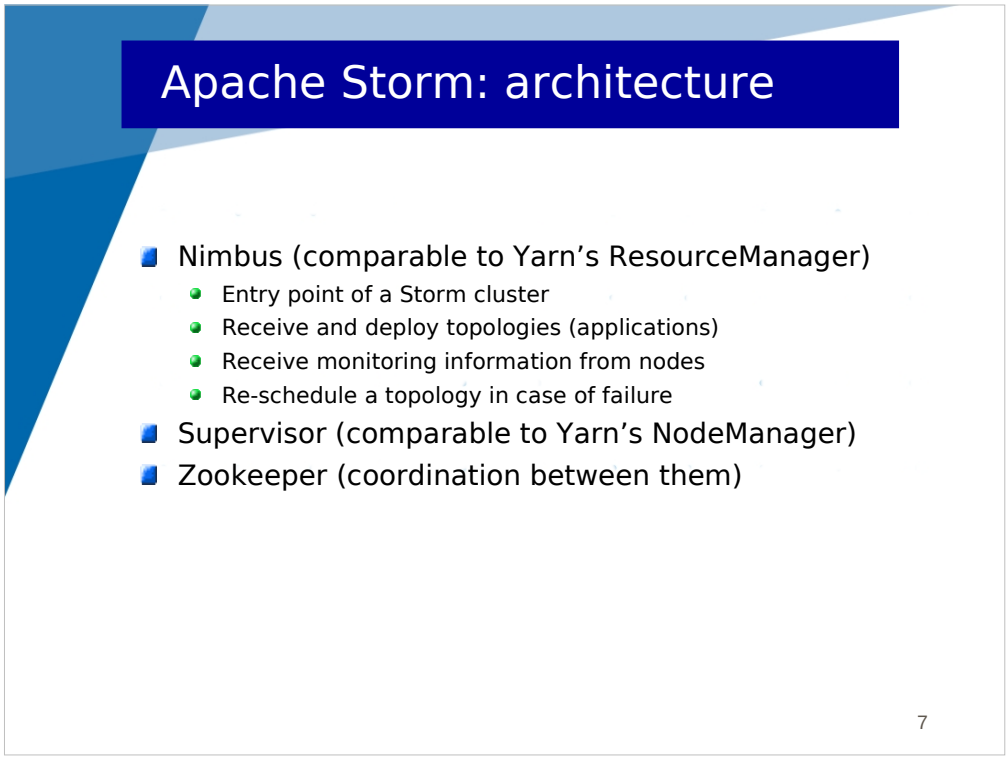
- A Storm cluster



6

A Storm cluster is composed of a global coordinator called Nimbus and a set of execution engines (called supervisors) which hosts spouts and bolts.

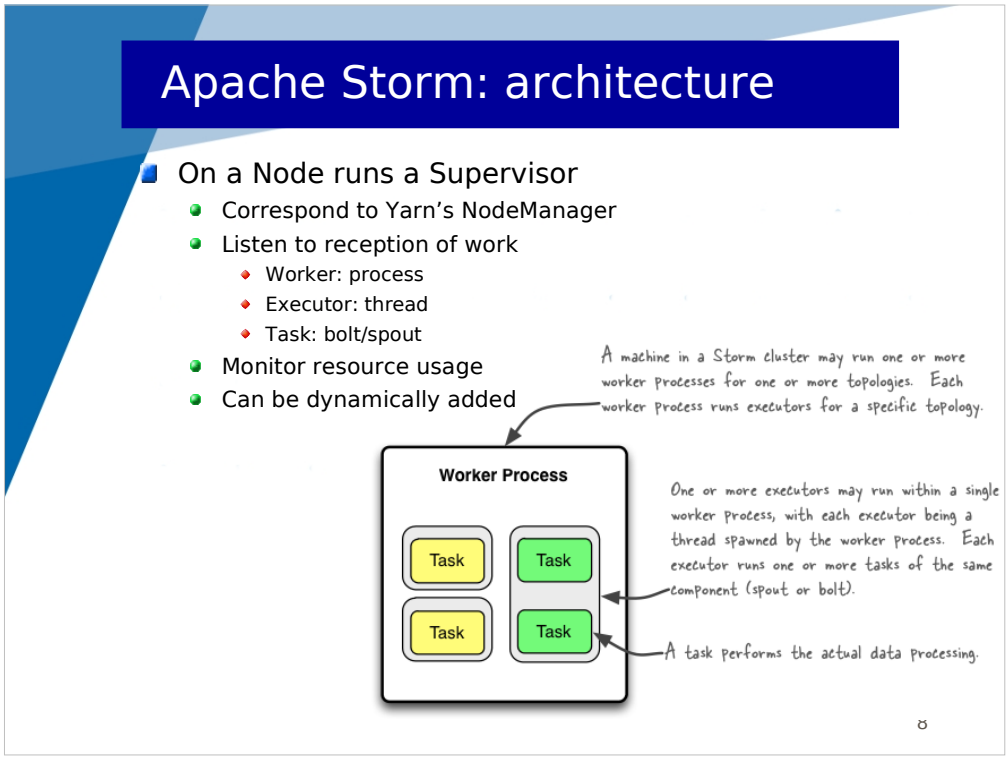
In the middle, zookeeper is a distributed middleware used for coordination between Nimbus and Supervisor nodes.



Nimbus is the coordinator of the platform. It can be compared to Yarn's ResourceManager. It is responsible of the deployment (on nodes) and management of topologies.

Supervisors are comparable to Yarn's NodeManagers. They manage resources locally.

Zookeeper is a distributed middleware which provides a distributed naming scheme (like a filesystem) and the possibility to share (read and write) information and synchronize. It is used for distributed coordination of Nimbus and Supervisors.



Each Node runs a Supervisor. It listens to the reception of jobs.

Jobs may be scheduled on processors. More precisely, Supervisors manage processes which host threads, a thread executing for one topology. Several spouts and bolts may be scheduled on one thread.

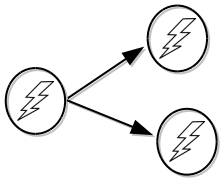
Supervisors monitor resource usage and report it to Nimbus.

Supervisors can be added dynamically in a cluster.

Apache Storm: data

- Spouts/bolts exchange Tuples
  - <v1, v2, v3 ...>
- A value in a Tuple can be of any serializable type
- Each spout/bolt defines the type of emitted tuples (1-tuple, 2-tuple ...) and field names

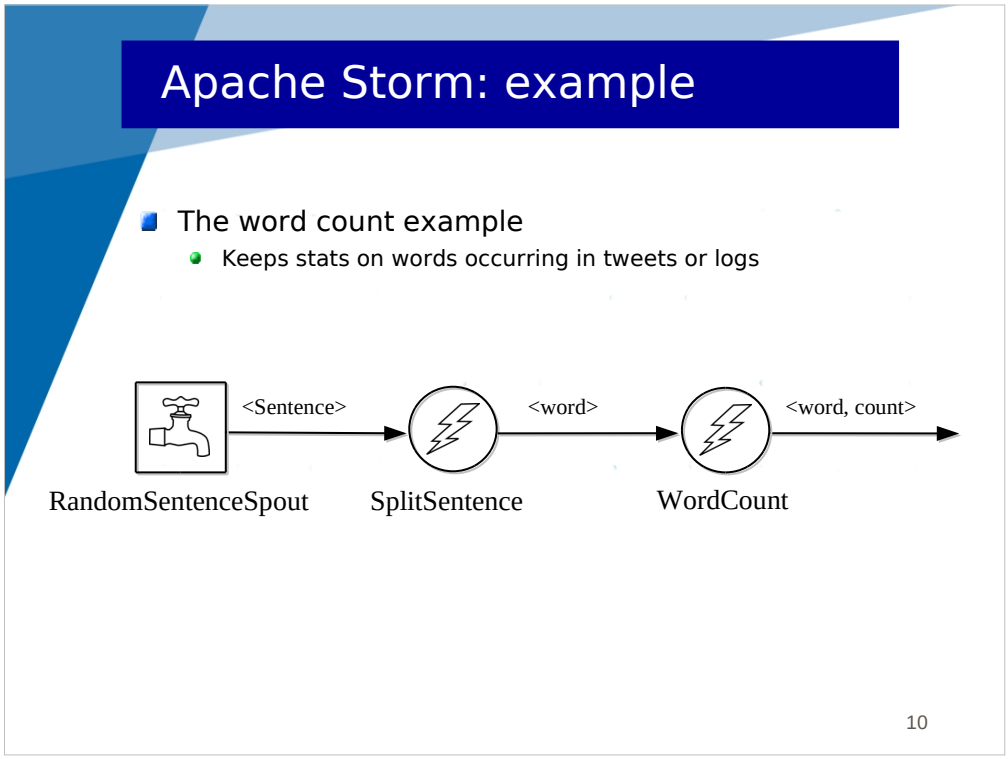
- Routing
  - A emitted tuple is sent on each of its outgoing connections



9

Spouts and bolts exchange tuples which include Java objects.

An emitted tuple (by a spout or a bolt) is sent to each outgoing connection.

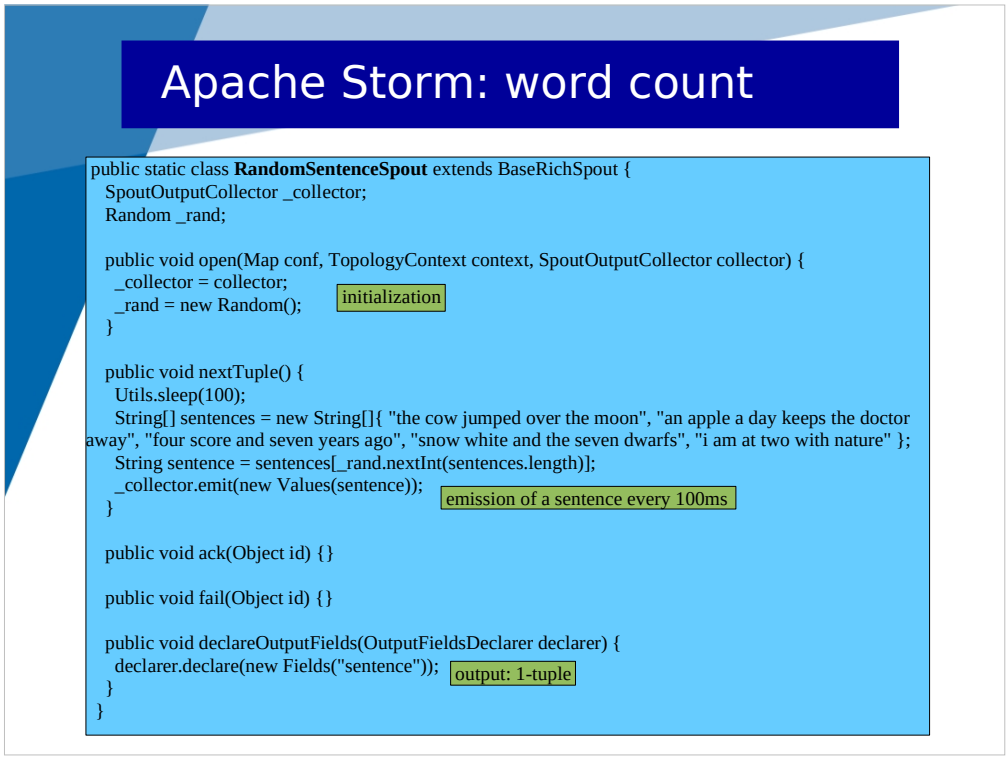


Here is a topology which implements the Wordcount application.

RandomSentenceSpout is a spout which emits a sentence (1-tuple <sentence>) randomly from a set of sentence (to simulate the arrival of sentences from the net).

SplitSentence is a bolt which receive sentences, splits them into words and emits words (1-tuple <word>).

WordCount is a bolt which receives words, manages a wordcount table and emits each wordcount which is modified (2-tuple <word, count>).



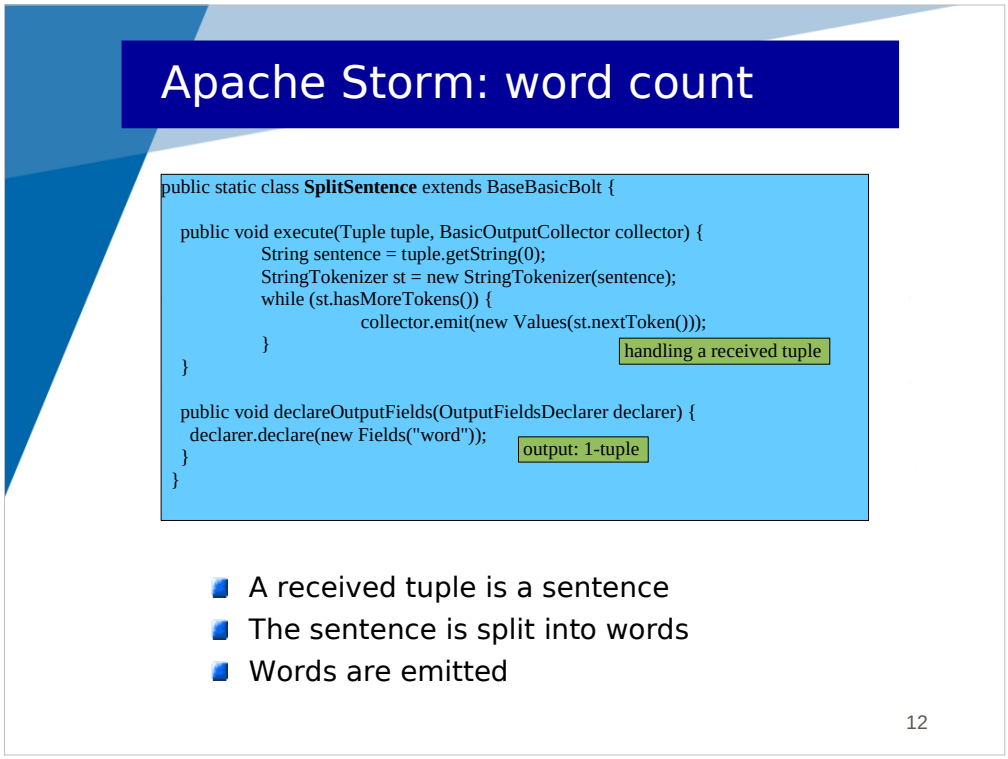
Here is the implementation of RandomSentenceSpout.

The spount receives with the open() method the collector reference which allows emitting data (the variable is save in a _collector variable).

Storm invokes the nextTuple() method all the time (iteratively). If there's nothing to send, it is supposed to wait a little time (not to waste too muche CPU) and return.

ack() and fail() allow managing a queue of messages and to re-emit them if thir processing failed.

Notice that the declareOutputFields() method allows declaring the fields of emitted tuples (here a 1-tuple with a "sentence" field).



Here is the implementation of SplitSentence.

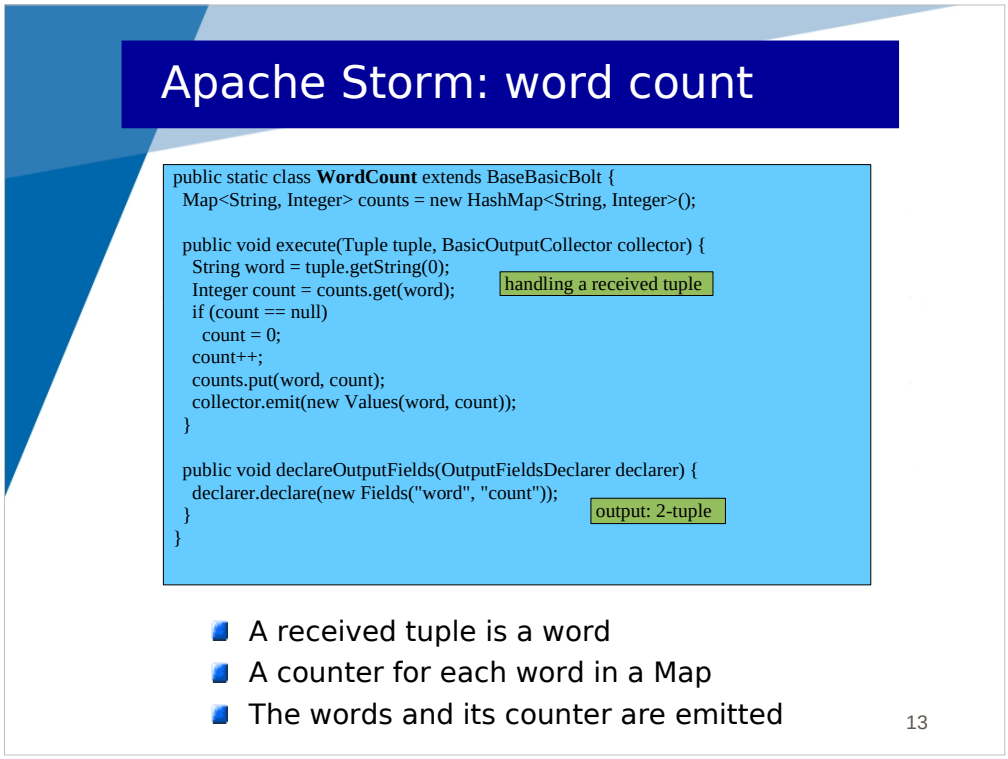
It defines the execute() method which includes the treatment to apply on received tuples :

- tuple is the received tuple

- collector is the variable for emitting tuples

The implementation of the method splits the received sentence into words and emits one tuple for each word.

As in the previous class, the declareOutputFields() method allows declaring the fields of emitted tuples (here a 1-tuple with a "word" field).

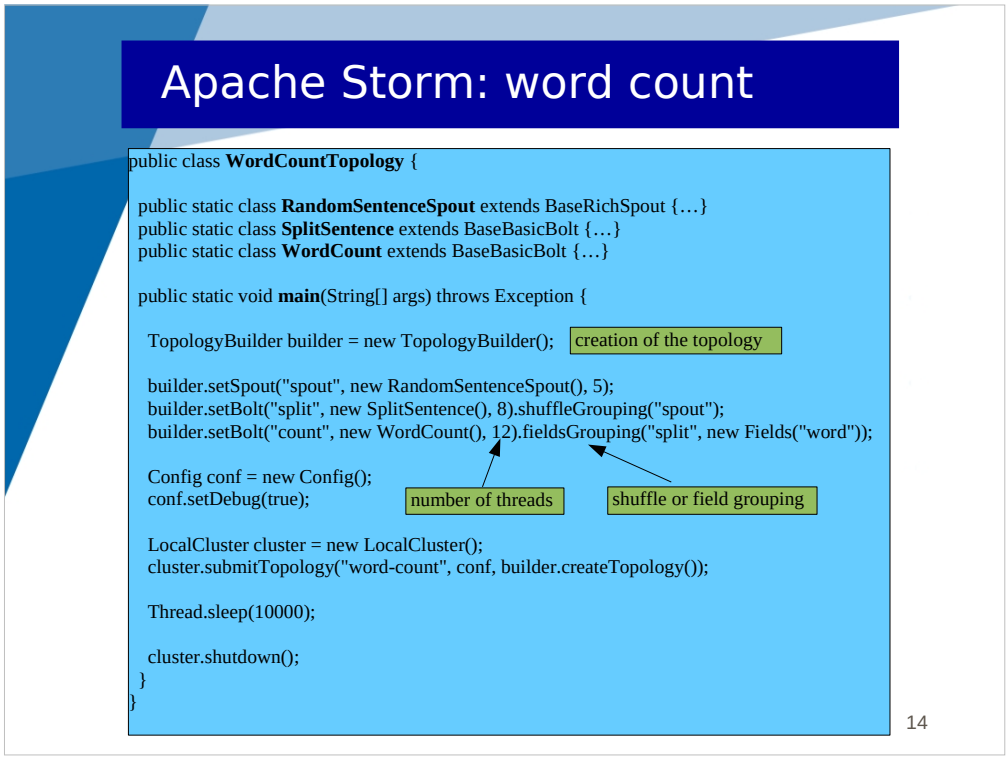


Here is the implementation of WordCount.

Like the previous bolt, it implements the execute() and declareOutputFields() methods.

declareOutputFields: it emits 2-tuples including "word" and "count" fields.

execute: it counts the word occurrences in a Hashtable and emits the new count for each received word.



This is the main program. It defines the Storm topology.

It is composed of 5 instances of RandomSentenceSpout. Each instance is executed by a different thread.

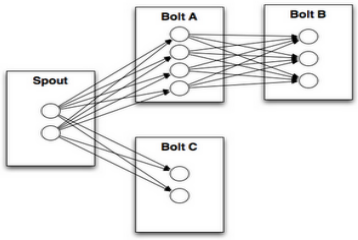
It is composed of 8 instances of SplitSentence bolt. The routing of messages fromRandomSentenceSpout instances to SplitSentence instances is defined with the shuffleGrouping policy.

It is composed of 12 instances of WordCount bolt. The routing of messages from SplitSentence instances to WordCount instances is defined with the fieldsGrouping policy (with field "word").

Grouping policies are explained in the next slide.

Apache Storm: routing

- Spouts/bolts execute in parallel (threads/tasks)



- When a tuple is sent from Bolt A to Bolt B, who receive it ?
  - Many modes (grouping)
  - shuffleGrouping: one tasks randomly chosen
  - fieldsGrouping: same field value=> same task
  - allGrouping, directGrouping, customGrouping ...
- For Wordcount: the same word goes to the same task

15

Spouts and bolts are instantiated several times and executed in parallel. Each instance is associated with a different thread.

The binding between a bolt A and a bolt B actually means a binding between a pool of instances of bolt A and a pool of instances of bolt B. The behavior of this binding is defined with a mode of grouping, defining which instance receives an emitted message.

- shuffleGrouping means that the target instance is randomly chosen

- fieldsGrouping with a given field means that tuples with the same value for that field should always be transmitted to the same target instance.

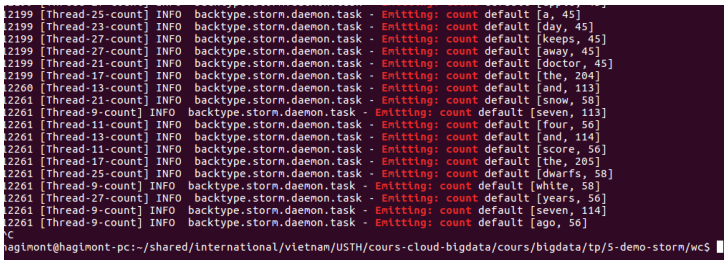
- allGrouping means messages are replicated and sent to all instances

There are many other grouping modes.

In the WordCount application, for the binding between SplitSentence instances and WordCount instances, the fieldsGrouping allows to always send the same word to the same instances (task), so that the count will be consistent.

Demonstration

- Install Apache Storm
  - tar xzf apache-storm-0.9.5.tar.gz
- Compile the WordCount application
  - javac -cp "../apache-storm-0.9.5/lib/*" WordCountTopology.java
  - Or in Eclipse
    - Add storm-core-0.9.5.jar in the buildpath
- Observe traces in debug mode
  - java -cp "../apache-storm-0.9.5/lib/*":. WordCountTopology | grep "Emitting: count"

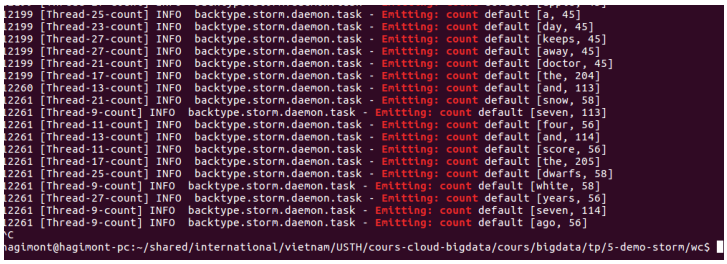


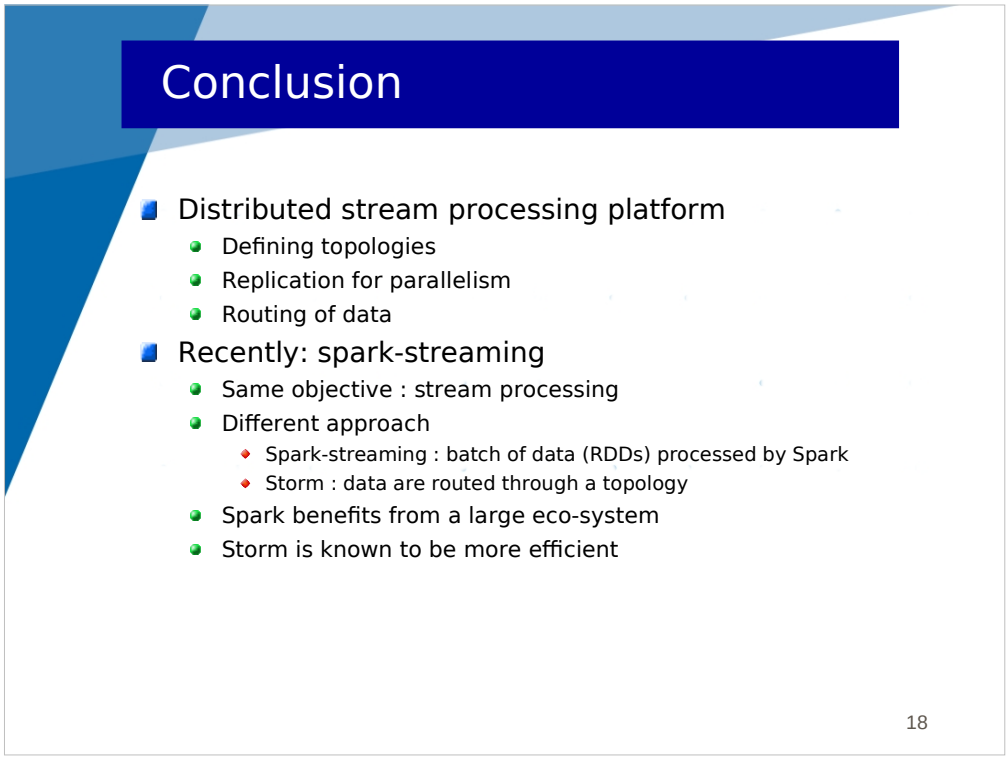
Here is the procedure to execute a Storm topology locally.

Installation in a cluster is well documented.

Demonstration

- Install Apache Storm
  - tar xzf apache-storm-0.9.5.tar.gz
- Compile the WordCount application
  - javac -cp "../apache-storm-0.9.5/lib/*" WordCountTopology.java
  - Or in Eclipse
    - Add storm-core-0.9.5.jar in the buildpath
- Observe traces in debug mode
  - java -cp "../apache-storm-0.9.5/lib/*":. WordCountTopology | grep "Emitting: count"





In conclusion,Spark-streaming and Storm share the same objective (stream processing) but they have different approaches.

Spark-streaming slices incoming data into batches (RDDs) which are processed with the Spark engine.

Storm deploys computing components (instances of spout or bolt) in the cluster and routes incoming data in the topology.

Spark is known to be part of a very rich ecosystem

Storm is known to be more efficient