# Apache Storm

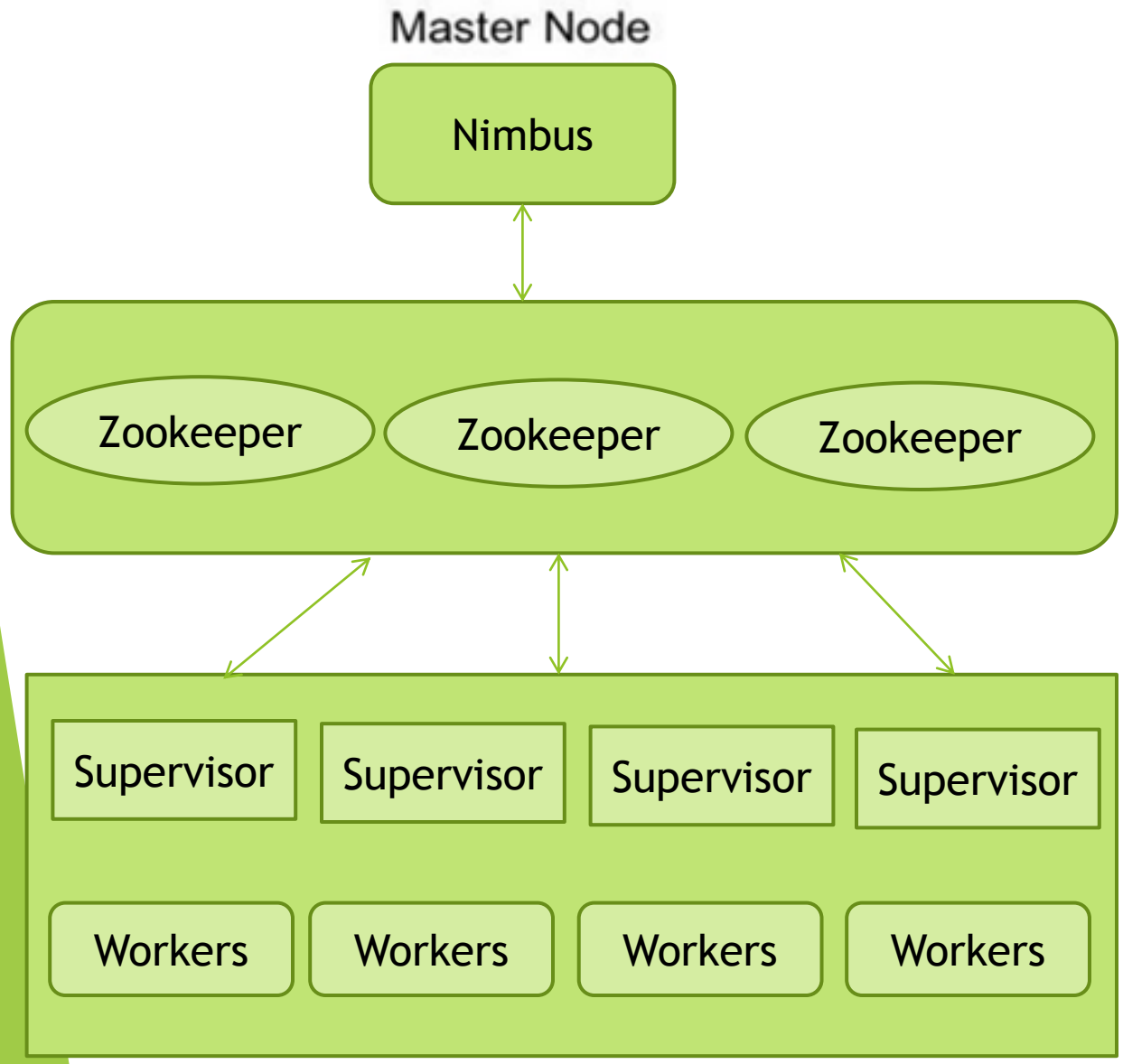
# Introduction

- Apache Storm is a real-time fault-tolerant and distributed Stream Processing Engine.
- Open Sourced September 19th 2011
- Main languages-Clojure and the JAVA.
- Some of the Characteristics of storm are Fast, Scalable, Fault-tolerant, Reliable and Easy to operate.
- Some of the organizations that currently use Storm are: Yahoo!, Groupon, The Weather Channel, Alibaba, Baidu, and Rocket Fuel.

# Why Storm?

- Dealing with Huge amount of data
- Streaming data processing
- Use cases:
  - real-time trading analytics
  - malfunction detection
  - social network
  - smart advertisement placement
  - log processing and metrics analytics.

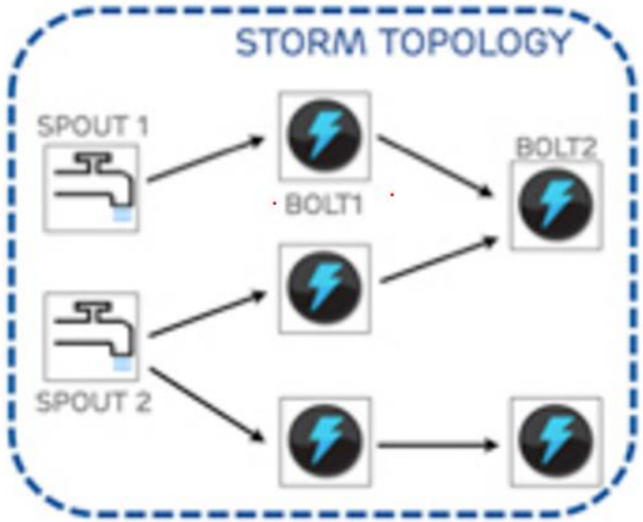
# Internal Architecture



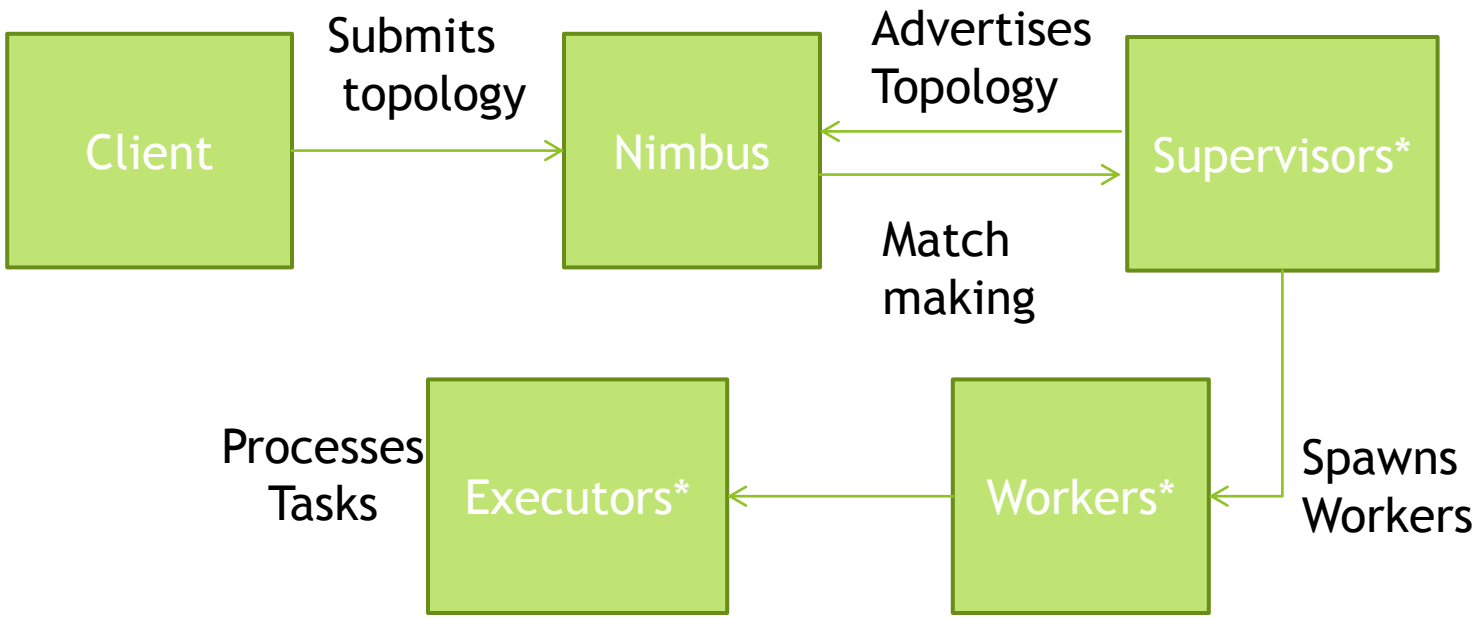

- **Nimbus-Master Node**
  - Assigns tasks
  - Monitors failures
- **Zookeeper**
  - Cluster state of Nimbus and Supervisor maintained in zookeeper
- **Supervisor**
  - Communicates with Nimbus through Zookeeper about topologies and available resources.
- **Workers**
  - Listens for assigned work and executes the application.

# Storm- Data Processing

- Streams of tuples flowing through topologies
- Vertices represent computation and edges represent the data flow
- Vertices divided into
  - **Spouts -read tuples from external sources.**
  - **Bolts - encapsulate the application logic.**

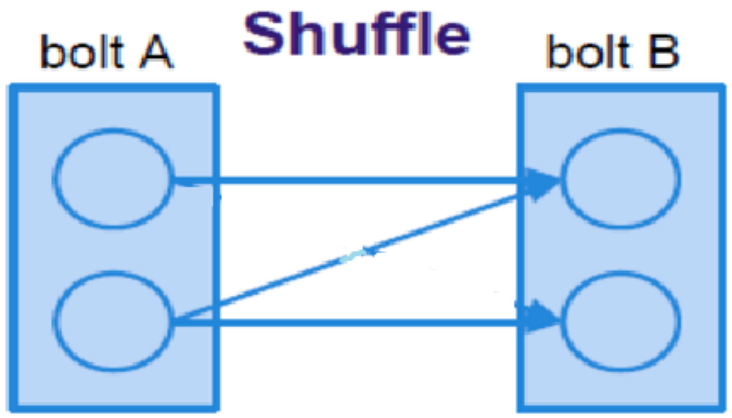


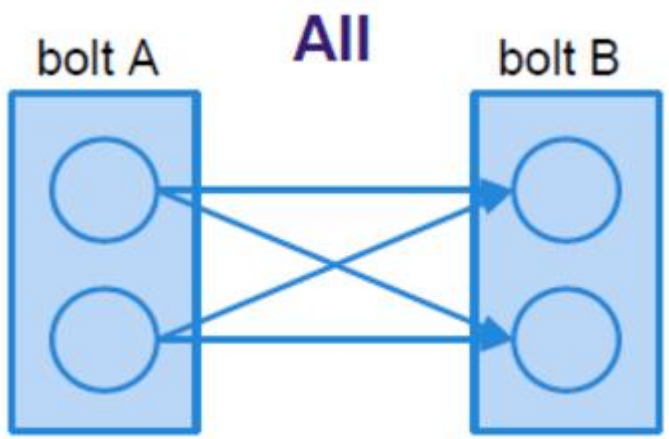
# Interaction between Storm Internals Components

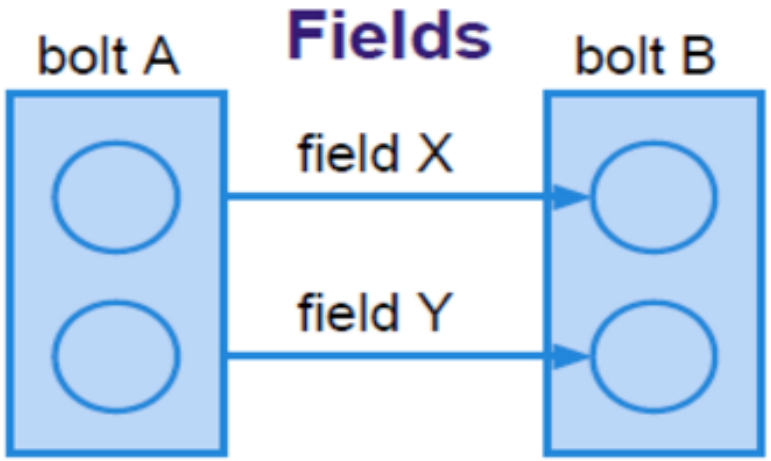


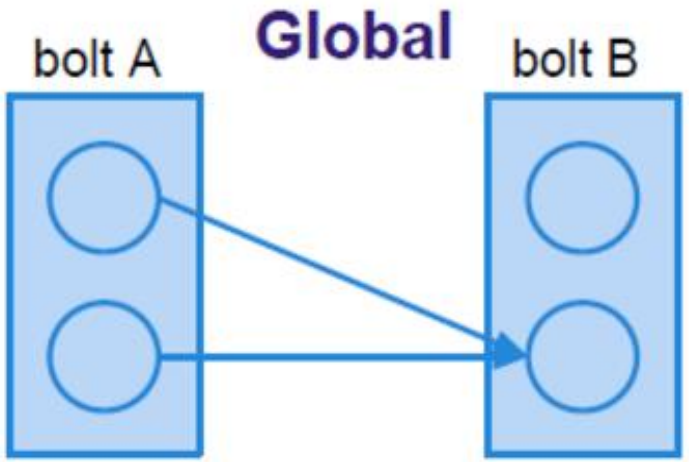

Events - Heartbeat protocol (every 15 seconds), synchronize supervisor event(every 10 seconds) and synchronize process event(every 3 seconds).

# Stream Grouping



## Processing Semantics

- Atleast once:

  Each tuple that is input to the topology will be processed atleast once.

- Atmost Once:

  Each tuple is processed  once or dropped in case of failure.

## States of workers

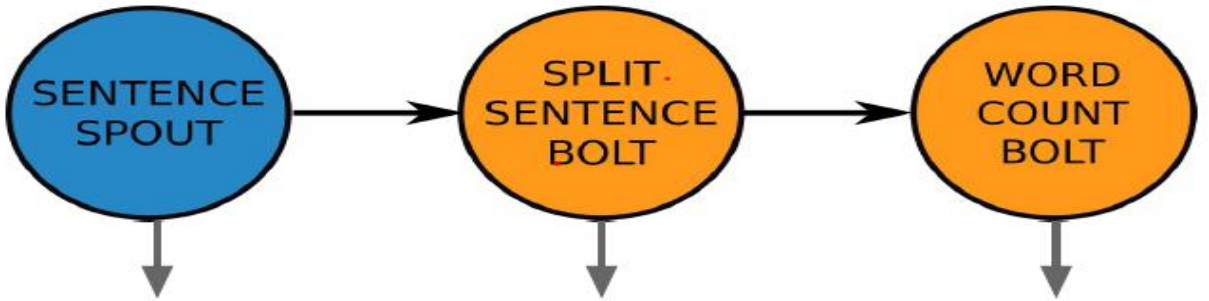
Supervisor periodically checks the state of workers for managing the worker processes.

- Timed out
- Not started
- Disallowed
- Valid

# Topology Example

```
public class WordCountTopology {
    public static void main(String[] args) throws Exception {

        //...Topology construction...
        Config config = new Config();

        LocalCluster cluster = new LocalCluster();

        cluster.submitTopology(TOPOLOGY_NAME, config, builder.createTopology());

        waitForSeconds(10);
        cluster.killTopology(TOPOLOGY_NAME);
        cluster.shutdown();
    }
}
```

```
TopologyBuilder builder = new TopologyBuilder();
builder.setSpout("sentencs-spout", new SentenceSpout());
builder.setBolt("split-bolt", new SplitSentenceBolt())
    .shuffleGrouping("sentences-spout");
builder.setBolt("count-bolt", new WordCountBolt())
    .fieldsGrouping("split-bolt", new Fields("word"));
```

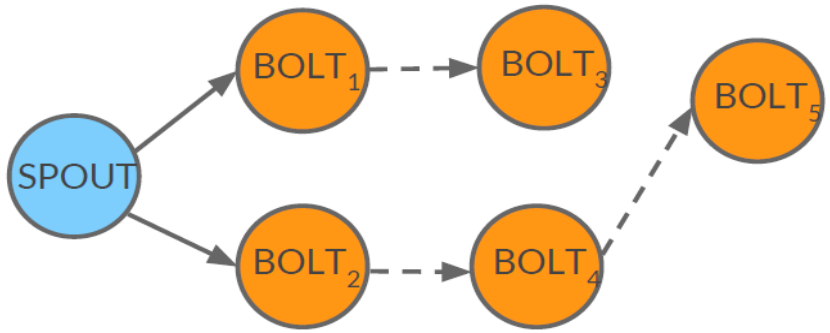


# Topology Example-contd

```
public class SplitSentenceBolt extends BaseRichBolt{
    private OutputCollector collector;

    public void declareOutputFields(OutputFieldsDeclarer declarer) {
        declarer.declare(new Fields("word"));
    }

    public void prepare(Map config, TopologyContext context,OutputCollector collector) {
        this.collector = collector;
    }

    public void execute(Tuple tuple) {
        String sentence = tuple.getStringByField("sentence");
        String[] words = sentence.split(" ");
        for(String word : words){
            this.collector.emit(new Values(word));
        }
    }
```

```
public class WordCountBolt extends BaseRichBolt{
    private OutputCollector collector;
    private HashMap<String,Long> counts=null;

    public void declareOutputFields(OutputFieldsDeclarer declarer) {
        //this bolt does not emit anything
    }

    public void prepare(Map config, TopologyContext context,OutputCollector collector) {
        this.collector = collector;
        this.counts = new HashMap<String,Long>();
    }

    public void execute(Tuple tuple) {
        String word = tuple.getStringByField("word");
        //...increments count..
    }
}
```

# Guaranteed Processing




- Storm provides an API to guarantee that a tuple emitted by a spout is fully processed by the topology (at-least-once semantic).
- With guaranteed processing, each bolt in the tree can either *acknowledge* or *fail* a tuple

## Spout Side

```
public void nextTuple() {
    //prepare the next sentence S to emit
    this.collector.emit(new Values(S), msgID);
    //...
}

public void ack(Object msgID) {
    //handle success
    //...
}

public void fail(Object msgID) {
    //handle failure
    //...
}
```

Assign a unique ID to any emitted tuple

Implement the ack and fail methods for handling successes and failures

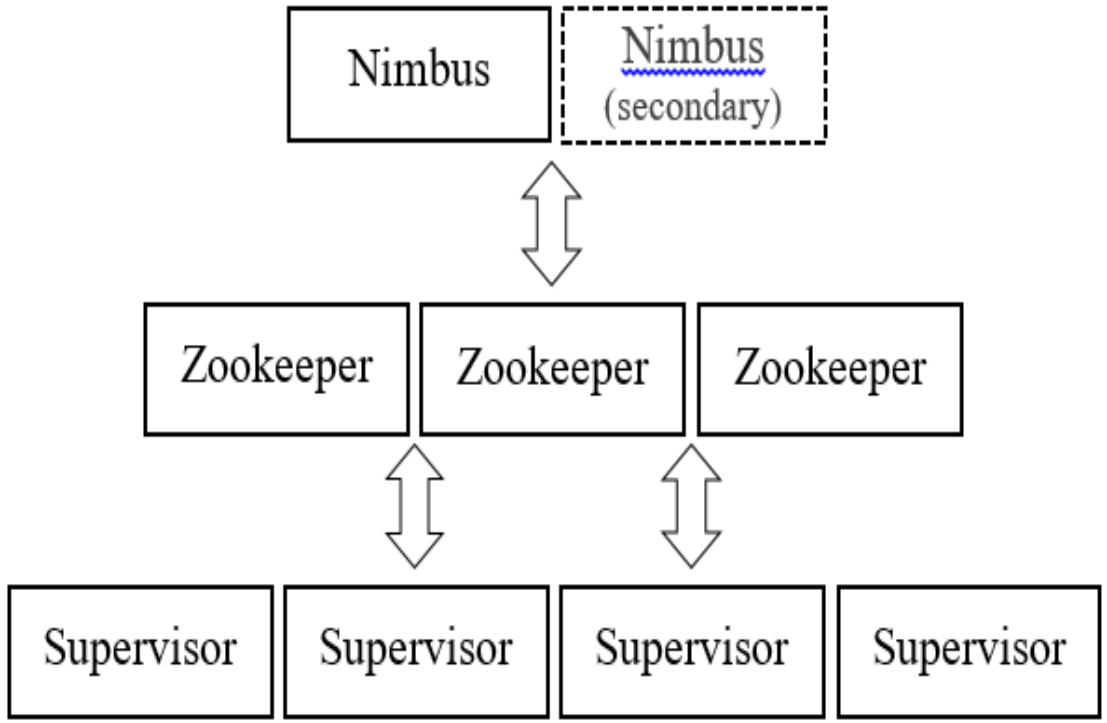
## Bolts Side

```
public void execute(Tuple tuple) {
    //... processing...
    this.collector.emit(tuple, new Values(word));

    //acknowledgment
    this.collector.ack(tuple);

    //or, if something goes wrong, fail
    this.collector.fail(tuple);
}
```

Anchoring (through overloaded `emit` method)

Ack or fail the tuple

# Deployment model of the cluster




- The secondary Nimbus instance starts working when the primary one temporary fails.
- Each spout deals with a specific data stream which allows to produce tuples from streams with different protocols and data formats.
- The bolts from current layer are involved on the filtering, aggregating and analysis stages

# Storm Use Cases

- **Twitter's** infrastructure, including database systems (Cassandra, Memcached, etc), the messaging infrastructure, Mesos, and the monitoring/alerting systems
- **Yahoo!** is developing a next generation platform that enables the convergence of big-data and low-latency processing.
- **Groupon** Storm helps us analyze, clean, normalize, and resolve large amounts of non-unique data points with low latency and high throughput.
- **Alibaba** uses storm to process the application log and the data change in database to supply real time stats for data apps.

# Storm Use Cases-contd

| | "Prevent" Use Cases | "Optimize" Use Cases |
|---|---|---|
| Financial Services | ✓ Securities fraud<br>✓ Operational risks & compliance violations | ✓ Order routing<br>✓ Pricing |
| Telecom | ✓ Security breaches<br>✓ Network outages | ✓ Bandwidth allocation<br>✓ Customer service |
| Retail | ✓ Shrinkage<br>✓ Stock outs | ✓ Offers<br>✓ Pricing |
| Manufacturing | ✓ Preventative maintenance<br>✓ Quality assurance | ✓ Supply chain optimization<br>✓ Reduced plant downtime |
| Transportation | ✓ Driver monitoring<br>✓ Predictive maintenance | ✓ Routes<br>✓ Pricing |
| Web | ✓ Application failures<br>✓ Operational issues | ✓ Personalized content |

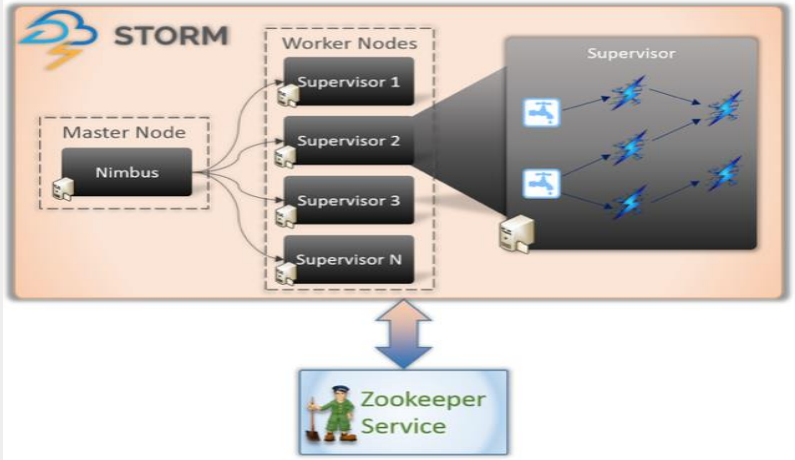
# Comparison big data open source tools

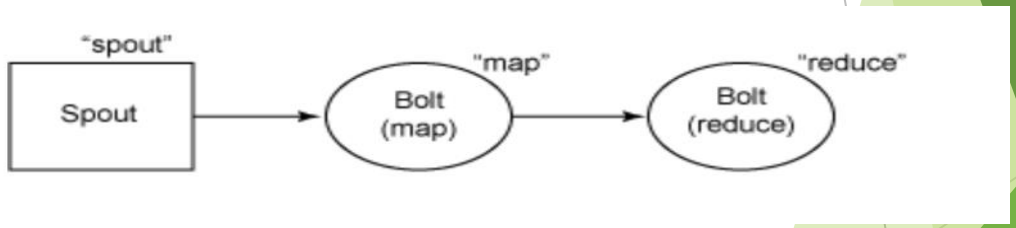
- A Storm cluster is superficially similar to a Hadoop cluster. Whereas on Hadoop you run “MapReduce jobs”, on Storm you run “topologies”. “Jobs” and “topologies” themselves are very different — one key difference is that a MapReduce job eventually finishes, whereas a topology processes messages forever (or until you kill it).Storm can do real time processing of streams of tuple’s (incoming data) while Hadoop do batch processing with MapReduce job.
- Storm behave like true streaming processing systems with lower latencies, While Spark is able to handle higher throughput while having somewhat higher latencies.
- Storm is better choice for real time data processing

- REFERENCE -Benchmarking Streaming Computation Engines: Storm, Flink and Spark Streaming, By Sanket Chintapalli, Derek Dagit, Bobby Evans, Reza Farivar, Thomas Graves, Mark Holderbaugh Zhuo Liu, Kyle Nusbaum, Kishorkumar Patil, Boyang Jerry Peng and Paul Poulosky Yahoo Inc., Presented at 2016 IEEE International Parallel and Distributed Processing Symposium Workshops

# Storm vs Hadoop

| Storm | Hadoop |
|---|---|
| Real-time stream processing | Batch processing |
| Stateless | Stateful |
| Master/Slave architecture with ZooKeeper based coordination. The master node is called as **nimbus** and slaves are **supervisors**. | Master-slave architecture with/without ZooKeeper based coordination. Master node is **job tracker** and slave node is **task tracker**. |
| A Storm streaming process can access tens of thousands messages per second on cluster. | Hadoop Distributed File System (HDFS) uses MapReduce framework to process vast amount of data that takes minutes or hours. |
| Storm topology runs until shutdown by the user or an unexpected unrecoverable failure. | MapReduce jobs are executed in a sequential order and completed eventually. |
| **Both are distributed and fault-tolerant** | |
| If nimbus / supervisor dies, restarting makes it continue from where it stopped, hence nothing gets affected. | If the JobTracker dies, all the running jobs are lost. |



# Storm- Pros and Cons

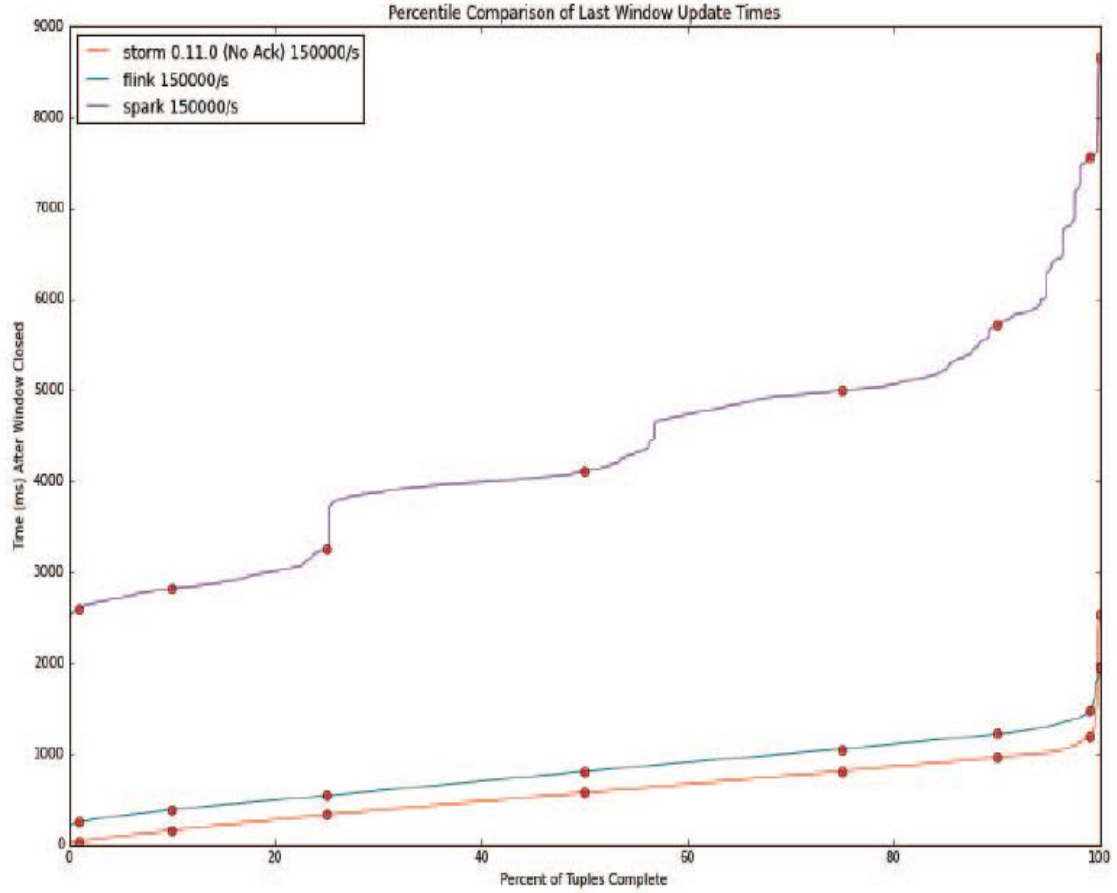
Pros

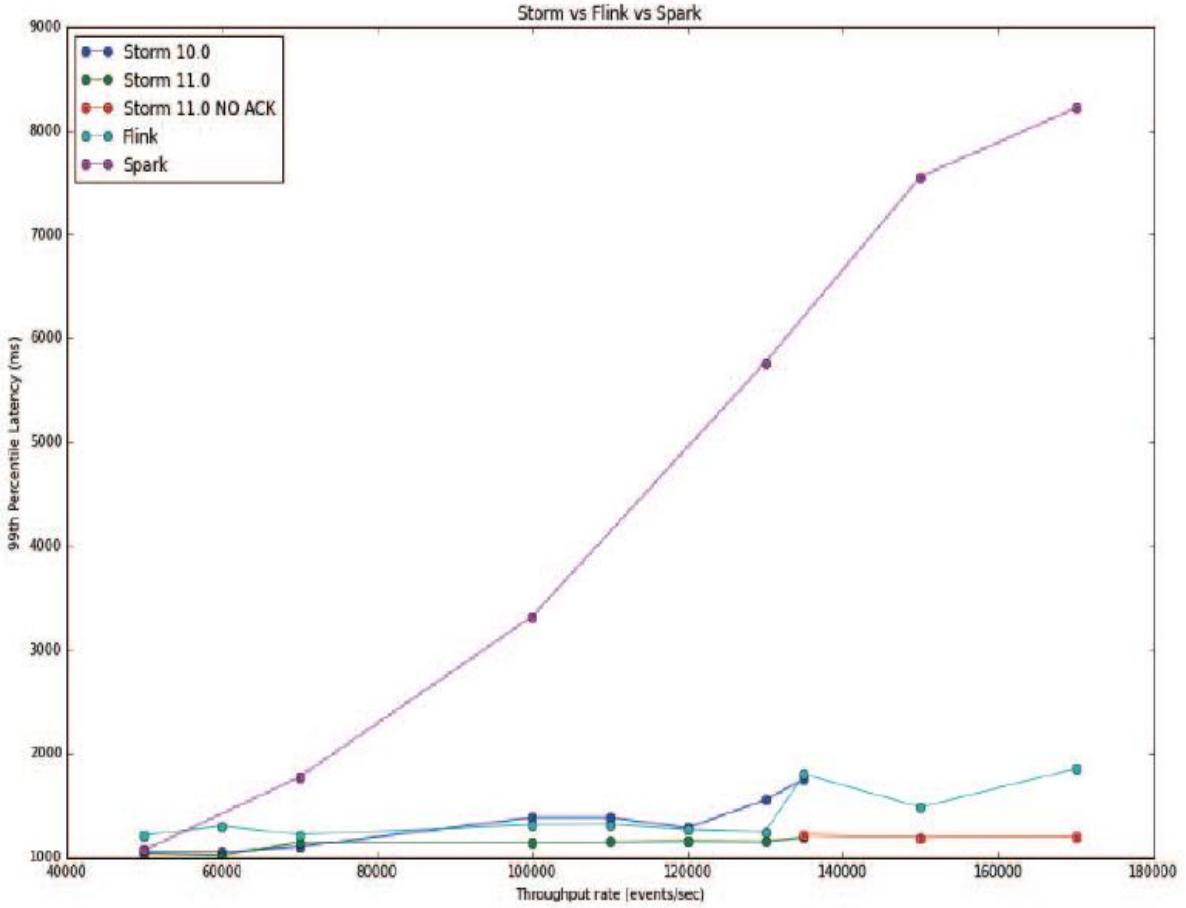
- Fault tolerance: High fault tolerance
- Latency: very less
- Processing Model: Real-time stream processing model
- Programming language dependency: any programming language
- Reliable: each tuple of data should be processed at least once
- Scalability: high scalability

Cons

- Use of native scheduler and resource management feature (Nimbus) in particular, become bottlenecks.
- Difficulties with debugging given the way the threads and data flows are split.

# Benchmarking Streaming Computation Engines: Storm, Flink and Spark Streaming



# References


- Apache Storm Based on Topology for Real-Time Processing of Streaming Data from Social Networks, By Anatoliy Batyuk, Volodymyr Voityshyn, Presented at IEEE First International Conference on Data Stream Mining & Processing http://ieeexplore.ieee.org/document/7583573/?reload=true
- Benchmarking Streaming Computation Engines: Storm, Flink and Spark Streaming, By Sanket Chintapalli, Derek Dagit, Bobby Evans, Reza Farivar, Thomas Graves, Mark Holderbaugh Zhuo Liu, Kyle Nusbaum, Kishorkumar Patil, Boyang Jerry Peng and Paul Poulosky Yahoo Inc., Presented at 2016 IEEE International Parallel and Distributed Processing Symposium Workshops https://yahooeng.tumblr.com/post/135321837876/benchmarking-streaming-computation-engines-at
- INTRODUCTION TO APACHE STORM, by Tiziano De Matteis http://www.slideshare.net/tizianodem/introduction-to-apache-storm-55467258
- Using apache storm for big data, S Surshanov*, IITU, Kazakhstan http://www.cmnt.lv/upload-files/ns_24brt003_CMNT1903-802.pdf
- Storm @Twitter, Ankit Toshniwal, Siddarth Taneja, Amit Shukla, Karthik Ramasamy, Jignesh M. Patel*, Sanjeev Kulkarni, Jason Jackson, Krishna Gade, Maosong Fu, Jake Donham, Nikunj Bhagat, Sailesh Mittal, Dmitriy Ryaboy, Twitter, Inc., *University of Wisconsin – Madison https://cs.brown.edu/courses/csci2270/archives/2015/papers/ss-storm.pdf