innovate achieve lead

BITS Pilani

Pilani | Dubai | Goa | Hyderabad

Real-time Processing Applications

Data Streams

BITS Pilani

- Data streams are an algorithmic abstraction to support real-time analytics
- They are sequences of items, possibly infinite, each item having a timestamp, and so a temporal order
  - Data items arrive one by one, and need to build and maintain models, such as patterns or predictors, of these items in real time
- There are two main algorithmic challenges when dealing with streaming data:
  - the stream is large and fast, and we need to extract information in real time from it. That means that usually need to accept approximate solutions in order to use less time and memory.
  - the data may be evolving, so our models have to adapt when there are changes in the data.

Time and Memory

- Three resource dimensions
  - Accuracy
  - Time
  - Memory
- Need methods that obtain the maximum accuracy with minimum time and low total memory
- Since data arrives at high speed, it cannot be buffered, so time to process one item is as relevant as the total time, which is the one usually considered in conventional data mining.

innovate achieve lead

BITS Pilani

Pilani | Dubai | Goa | Hyderabad

Parallel Processing with Apache Storm on Streams

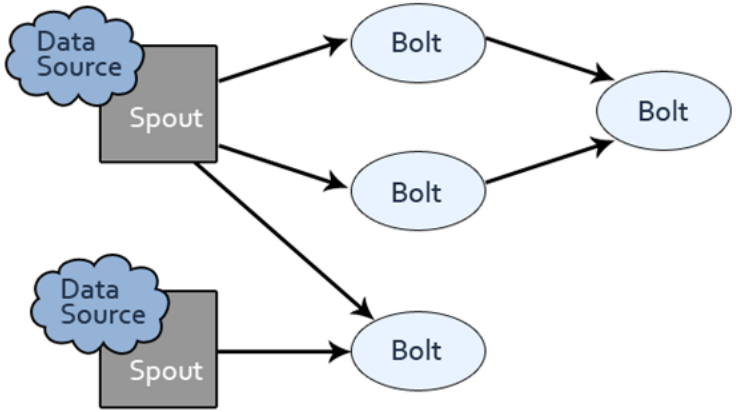
Storm

- Apache Storm is a free and open source distributed realtime computation system.
- Apache Storm makes it easy to reliably process unbounded streams of data, doing for realtime processing what Hadoop did for batch processing.

Strom Concepts

BITS Pilani

- 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.
  - Sink – bolt but doesn't generate data. Used for exporting data.

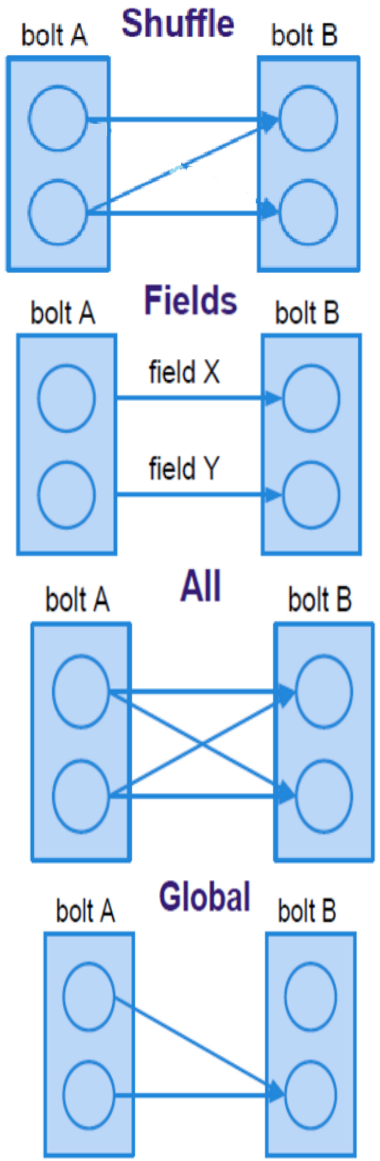


A Storm Topology

Image source:http://www.treselle.com/blog/twitter-analysis-with-apache-storm/

Streams Grouping

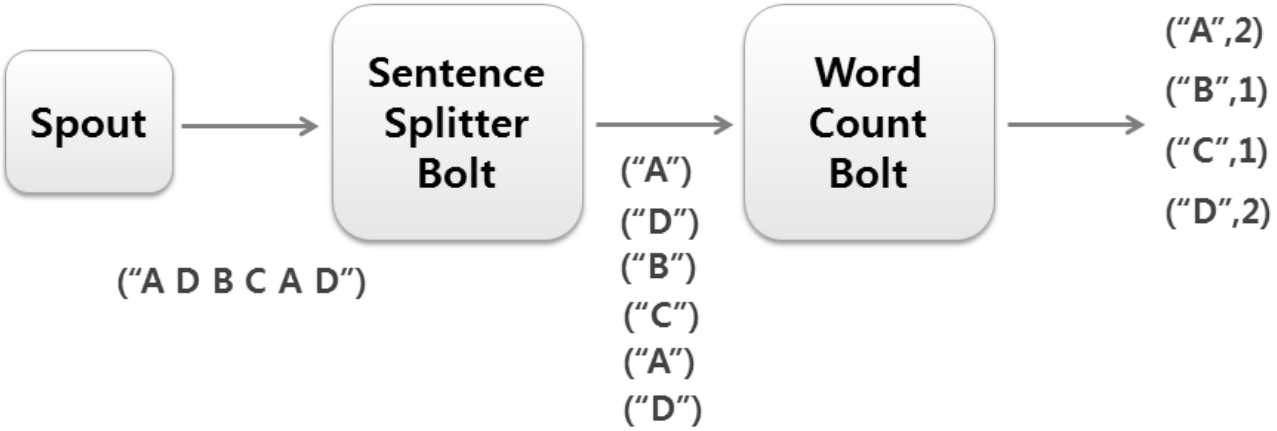
- Shuffle grouping
  - randomly partitions the tuples.
- Fields grouping
  - fields with same values in tuples are grouped together and are sent forward to the same worker executing the bolt instance.
- All grouping
  - replicates the entire stream to all the consumer tasks.
- Global grouping
  - sends the entire stream to a single bolt.



Strom- Word Count Example

BITS Pilani

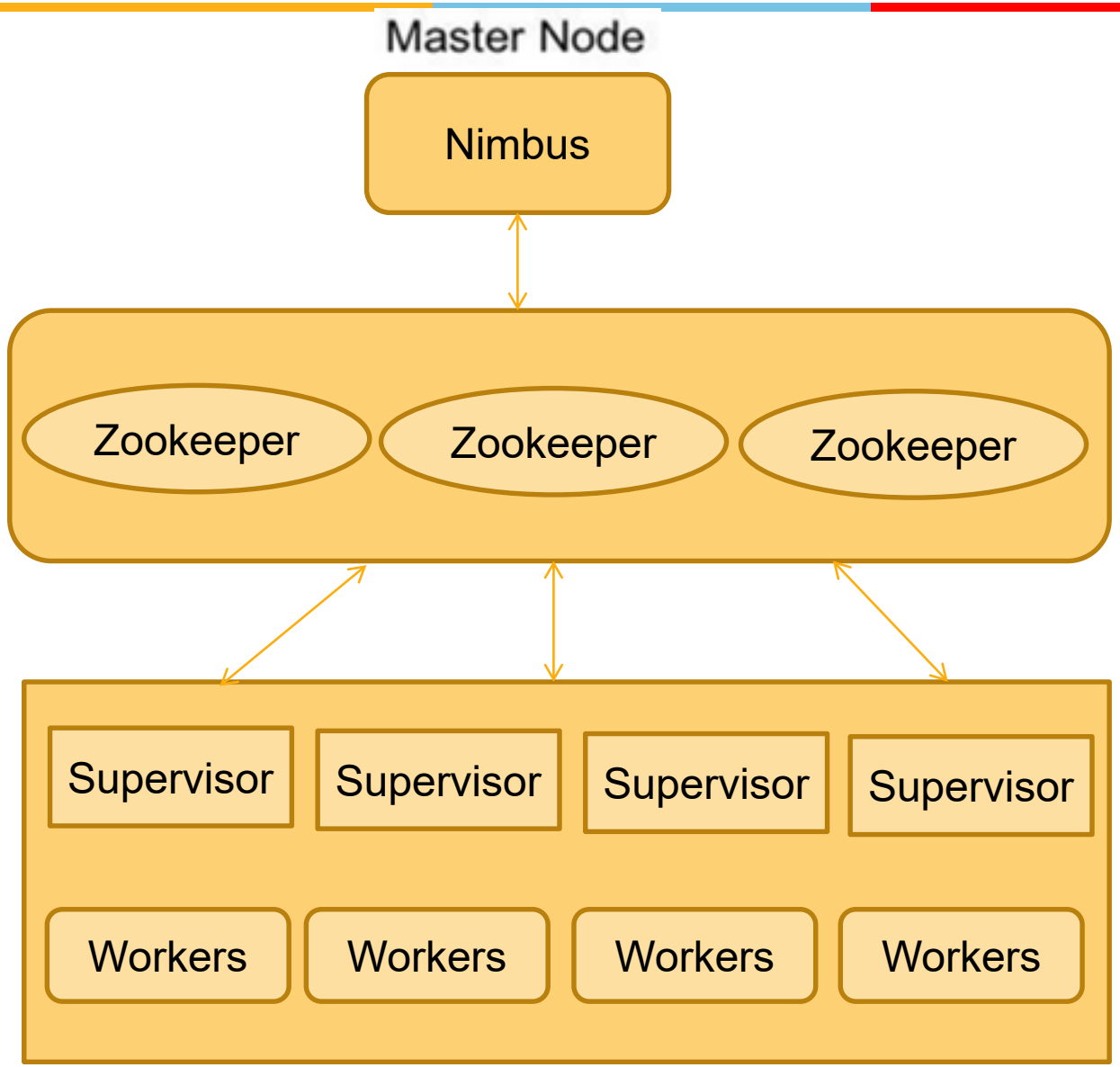
- Topology



  - ShuffleGrouping between spout and splitter bolt
  - FieldsGrouping between splitter and wordcount bolt

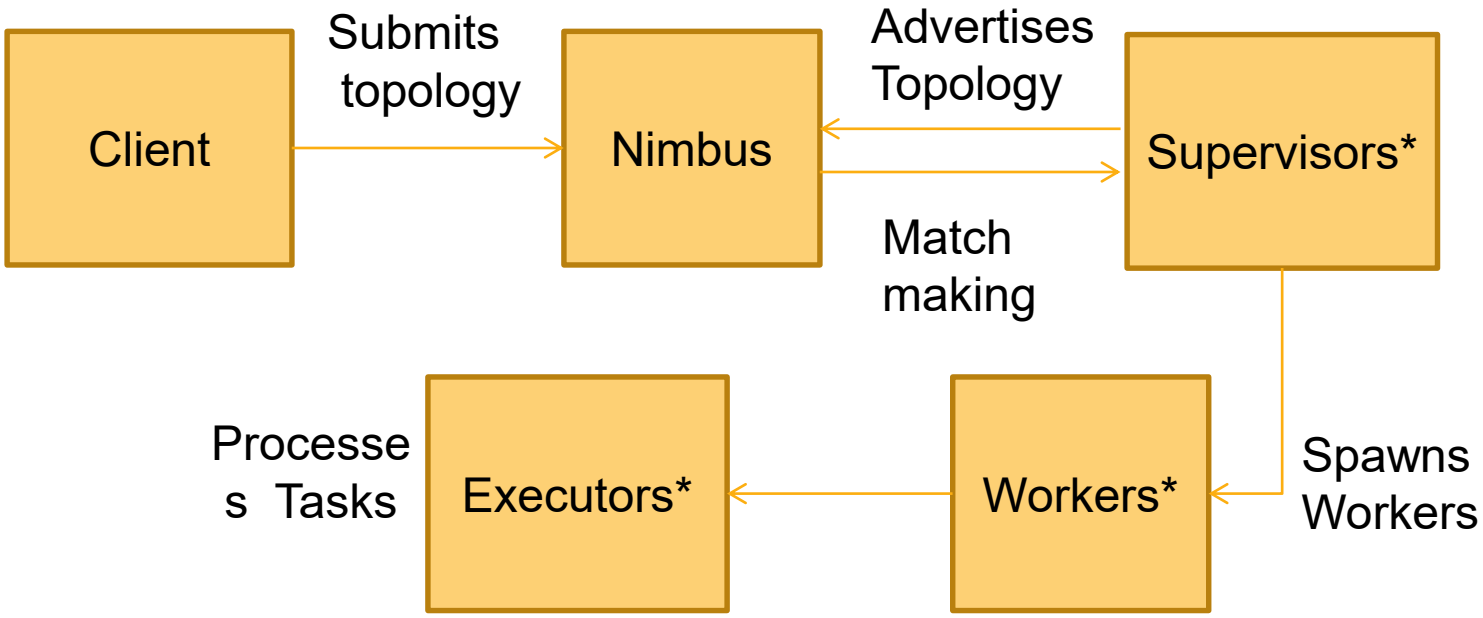
```
1   TopologyBuilder builder = new TopologyBuilder();
2   builder.setSpout("spout", new RandomSentenceSpout(), 5);
3   builder.setBolt("split", new SplitSentence(), 8).shuffleGrouping("spout");
4   builder.setBolt("count", new WordCount(), 12)
5           .fieldsGrouping("split", new Fields("word"));
```

Storm: 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: Internal Architecture

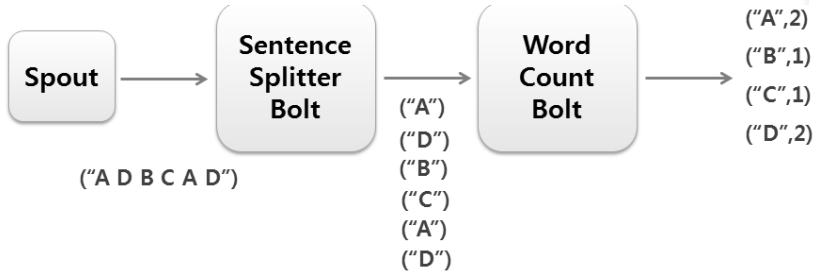


- Events - Heartbeat protocol (every 15 seconds), synchronize supervisor event(every 10 seconds) and synchronize process event(every 3 seconds).

Strom- Word Count Example

BITS Pilani

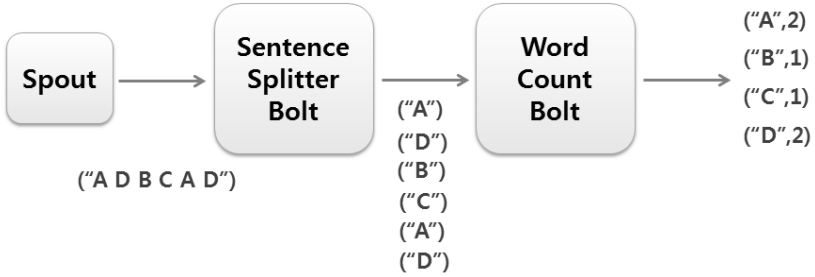
```java
public class RandomSentenceSpout extends BaseRichSpout {
  SpoutOutputCollector _collector;
  Random _rand;
  @Override
  public void open(Map conf, TopologyContext context, SpoutOutputCollector collector) {
    _collector = collector;
    _rand = new Random();
  }
  @Override
  public void nextTuple() {
    Utils.sleep(100);
    String[] sentences = new String[]{ "the cow jumped over the moon", "an apple a day keeps the doctor away",
        "four score and seven years ago", "snow white and the seven dwarfs", "i am at two with nature" };
    String sentence = sentences[_rand.nextInt(sentences.length)];
    _collector.emit(new Values(sentence));
  }
  @Override
  public void ack(Object id) {
  }
  @Override
  public void fail(Object id) {
  }
  @Override
  public void declareOutputFields(OutputFieldsDeclarer declarer) {
    declarer.declare(new Fields("word"));
  }
}
```



Strom- Word Count Example

BITS Pilani

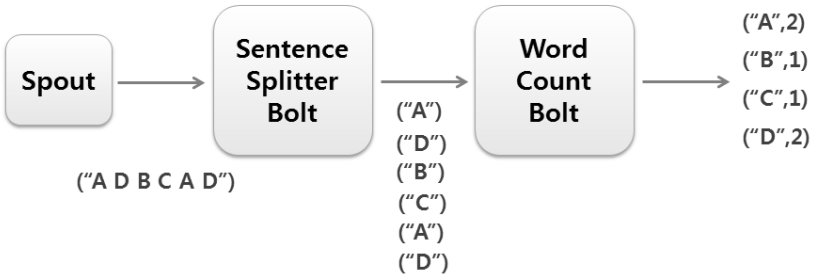
```
1  public static class SplitSentence extends BaseBasicBolt {
2      @Override
3      public void declareOutputFields(OutputFieldsDeclarer declarer) {
4        declarer.declare(new Fields("word"));
5      }
6      @Override
7      public Map<String, Object> getComponentConfiguration() {
8        return null;
9      }
10     public void execute(Tuple tuple, BasicOutputCollector basicOutputCollector) {
11       String sentence = tuple.getStringByField("sentence");
12       String words[] = sentence.split(" ");
13       for (String w : words) {
14         basicOutputCollector.emit(new Values(w));
15       }
16     }
17 }
```



Strom- Word Count Example

BITS Pilani

```
1  public static class WordCount extends BaseBasicBolt {
2      Map<String, Integer> counts = new HashMap<String, Integer>();
3      @Override
4      public void execute(Tuple tuple, BasicOutputCollector collector) {
5        String word = tuple.getString(0);
6        Integer count = counts.get(word);
7        if (count == null)
8          count = 0;
9        count++;
10       counts.put(word, count);
11       collector.emit(new Values(word, count));
12     }
13     @Override
14     public void declareOutputFields(OutputFieldsDeclarer declarer) {
15       declarer.declare(new Fields("word", "count"));
16     }
17 }
```



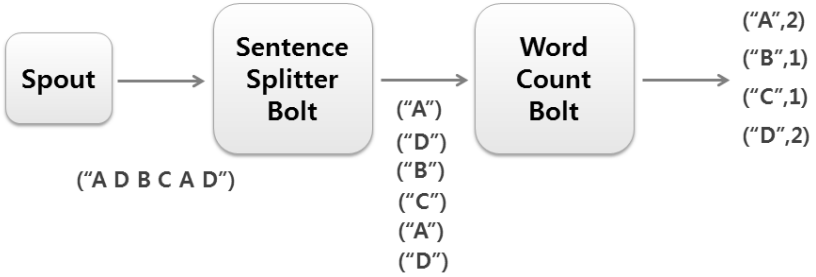
Strom- Word Count Example

BITS Pilani

```java
public static void main(String[] args) throws Exception {
    TopologyBuilder builder = new TopologyBuilder();
    builder.setSpout("spout", new RandomSentenceSpout(), 5);
    builder.setBolt("split", new SplitSentence(), 8).shuffleGrouping("spout");
    builder.setBolt("count", new WordCount(), 12).fieldsGrouping("split", new Fields("word"));
    /*config setup*/
    Config conf = new Config();
    conf.setDebug(true);

    if (args != null && args.length > 0) {
      conf.setNumWorkers(3);

      StormSubmitter.submitTopologyWithProgressBar(args[0], conf, builder.createTopology());
    } else {
      conf.setMaxTaskParallelism(3);
      LocalCluster cluster = new LocalCluster();
      cluster.submitTopology("word-count", conf, builder.createTopology());
      Thread.sleep(10000);
      cluster.shutdown();
    }
}
```



Windowing Support in Storm

- Storm core has support for processing a group of tuples that falls within a window. Windows are specified with the following two parameters,
  - Window length - the length or duration of the window
  - Sliding interval - the interval at which the windowing slides
- Sliding Window
  - Tuples are grouped in windows and window slides every sliding interval. A tuple can belong to more than one window.
- Tumbling Window
  - Tuples are grouped in a single window based on time or count. Any tuple belongs to only one of the windows.
- Storm supports specifying the window length and sliding intervals as a count of the number of tuples or as a time duration.

Windowing Support in Storm


```
public static void main(String[] args) {
    TopologyBuilder builder = new TopologyBuilder();
     builder.setSpout("spout", new RandomSentenceSpout(), 1);
     builder.setBolt("slidingwindowbolt",
                     new SlidingWindowBolt().withWindow(new Count(30), new Count(10)),
                     1).shuffleGrouping("spout");
    Config conf = new Config();
    conf.setDebug(true);
    conf.setNumWorkers(1);

    StormSubmitter.submitTopologyWithProgressBar(args[0], conf, builder.createTopology());

}
```

```
public interface IWindowedBolt extends IComponent {
    void prepare(Map stormConf, TopologyContext context, OutputCollector collector)
    /**
     * Process tuples falling within the window and optionally emit
     * new tuples based on the tuples in the input window.
     */
    void execute(TupleWindow inputWindow);
    void cleanup();
}
```

Every time the window activates, the execute method is invoked. The TupleWindow parameter gives access to the current tuples in the window, the tuples that expired and the new tuples that are added since last window was computed which will be useful for efficient windowing computations.

Q&A

- Understanding the Parallelism of a Storm Topology (apache.org)
- Windowing Support in Core Storm (apache.org)

innovate achieve lead

BITS Pilani
Pilani Campus

Thank You