Introduction to Stream Analytics

Chandan Mazumdar

Jadavpur University

July 04, 2015

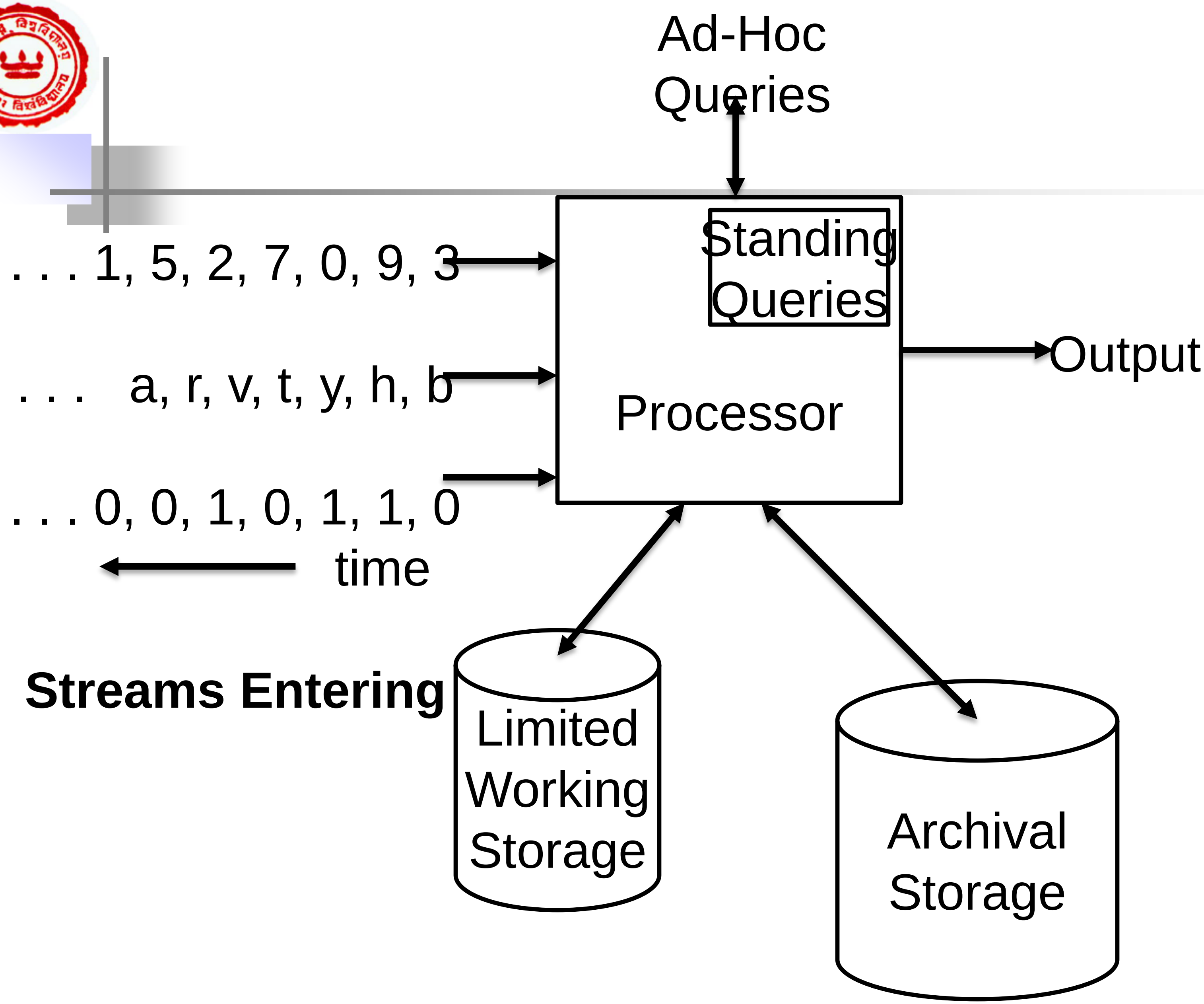
Data Streams

- In many data mining situations, we know the entire data set in advance

- Stream Management is important when the input rate is controlled **externally**:
  - Google queries
  - Twitter or Facebook status updates
- We can think of the data as infinite and non-stationary (the distribution changes over time)

The Stream Model

- Input tuples enter at a rapid rate, at one or more input ports (i.e., streams)
- The system cannot store the entire stream accessibly


Ad-Hoc Queries
Standing Queries
. . . 1, 5, 2, 7, 0, 9, 3
. . . a, r, v, t, y, h, b
. . . 0, 0, 1, 0, 1, 1, 0
time
Processor
Output
Streams Entering
Limited Working Storage
Archival Storage

Problems on Data Streams

- **Types of queries one wants answer on a stream:**
  - **Sampling data from a stream**
    - Construct a random sample
  - **Queries over sliding windows**
    - Number of items of type x in the last k elements of the stream

Problems on Data Streams

- **Types of queries one wants on answer on a stream:**
  - **Filtering a data stream**
    - Select elements with property *x* from the stream
  - **Counting distinct elements**
    - Number of distinct elements in the last *k* elements of the stream
  - **Estimating moments**
    - Estimate avg./std. dev. of last *k* elements
  - **Finding frequent elements**

Applications – (1)

- **Mining query streams**
  - Google wants to know what queries are more frequent today than yesterday
- **Mining click streams**
  - Yahoo wants to know which of its pages are getting an unusual number of hits in the past hour
- **Mining social network news feeds**
  - E.g., look for trending topics on Twitter, Facebook

Applications – (2)

- **Sensor Networks**
  - Many sensors feeding into a central controller
- **Telephone call records**
  - Data feeds into customer bills as well as settlements between telephone companies
- **IP packets monitored at a switch**
  - Gather information for optimal routing
  - Detect denial-of-service attacks

Examples of Streaming Data

- Ocean behavior at a point
  - Temperature (once every half an hour)
  - Surface height (once or more / second)
  - Several places in the ocean: one per 100 km^2
  - Overall 1.5 million sensors
  - A few terabytes of data everyday
- Satellite image data
  - Terabytes of images sent to the earth everyday
  - Convert to low resolution, but many satellites, a lot of data
- Web stream data
  - More than hundred million search queries per day
  - Clicks

Mining Streaming Data

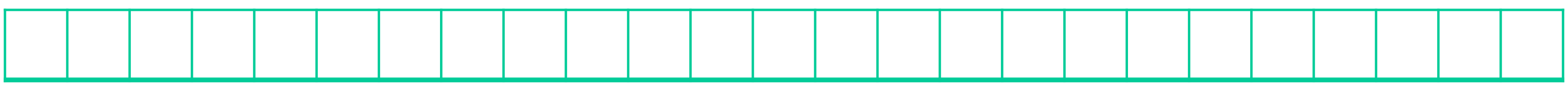
- Standard (non-stream) setting: data available when we need it
- Streaming data: data comes in one or more streams
- If you can, process, store results
  - Size of results much smaller than the stream size
- Then the data is lost forever
- Queries
  - Temperature alert if > some degree (standing query)
  - Maximum temperature in this month
  - Number of distinct users in the last month

Filtering Streaming Data

- Filter part of the stream based on a criteria
- If the criteria can be calculated, then easy
  - Example: Filter all words starting with *ab*
- Challenge: The criteria involves a membership lookup
  - Simplified example: Emails <email address, email> stream
  - Task: Filter emails based on email addresses
  - Have *S* = Set of 1 billion email address which are not spam
  - Keep emails from addresses in *S*, discard others
- Each email ~ 20 bytes or more. Total > 20GB
  - Not to keep in main memory
  - Option 1: make disk access for each stream element and check
  - Option 2: Bloom filter, use 1GB main memory

Filtering with One Hash Function

- Available memory: n bits (e.g. 1GB ~ 8 billion bits)
- Use a bit array of n bits (in main memory), initialize to all 0s
- A hash function h: maps an email address → one of the n bits
- Pre-compute hash values of S
- Set the hashed bits to 1, leave the rest to 0



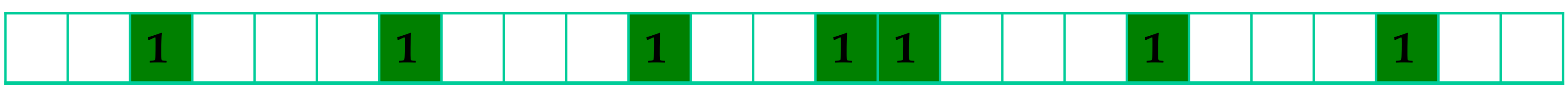
Filtering with One Hash Function

- Available memory: *n* bits (e.g. 1GB ~ 8 billion bits)
- Use a bit array of *n* bits (in main memory), initialize to all 0s
- A hash function *h*: maps an email address → one of the *n* bits
- Pre-compute hash values of *S*
- Set the hashed bits to 1, leave the rest to 0

		1				1				1			1	1				1				1		

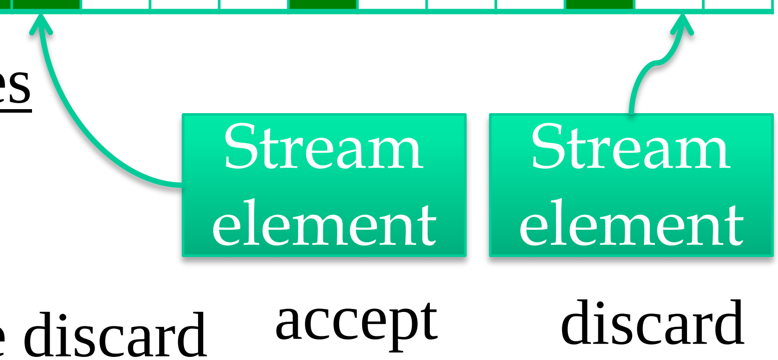
Filtering with One Hash Function

- Available memory: *n* bits (e.g. 1GB ~ 8 billion bits)
- Use a bit array of *n* bits (in main memory), initialize to all 0s
- A hash function *h*: maps an email address → one of the *n* bits
- Pre-compute hash values of *S*
- Set the hashed bits to 1, leave the rest to 0



<u>Online process: streaming data comes</u>

- Hash an element (email address)
- Check if the hashed bit was 1
- If yes, accept the email, otherwise discard
- Note: $x = y$ implies $h(x) = h(y)$, but not vice versa
- So, there would be false positives



The Bloom Filter

- Available memory: *n* bits
- Use a bit array of *n* bits (in main memory), initialize to all 0s
- Want to minimize probability of false positives
- Use *k* hash functions h_1, h_1, …, h_k
- Each h_i maps an element → one of the *n* bits
- Pre-compute hash values of *S* for all h_i
- Set a bit to 1 if any element is hashed to that bit for any h_i
- Leave the rest of the bits to 0

Online process: streaming data comes

- Hash an element with all hash functions
- Check if the hashed bit was 1 for all hash functions
- If yes, accept the element, otherwise discard

The Bloom Filter: Analysis

Let $|S| = m$, bit array is of n bits, k hash functions $h_1, h_1, \ldots, h_k$

- Assumption: the hash functions are independent and they map one element to each bit with equal probability
- P[a particular h_i maps a particular element to a particular bit] = $1/n$
- P[a particular h_i does not map a particular element to a particular bit] = $1 - 1/n$
- P[No h_i maps a particular element to a particular bit] = $(1 - 1/n)^k$
- P[After hashing m elements of S, one particular bit is still 0] = $(1 - 1/n)^{km}$
- P[A particular bit is 1 after hashing all of S] = $1 - (1 - 1/n)^{km}$

False positive analysis

- Now, let a new element x not be in S. Should be discarded.
- Each $h_i(x) = 1$ with probability $1 - (1 - 1/n)^{km}$
- $P[h_i(x) = 1 \text{ for all } i] = (1 - (1 - 1/n)^{km})^k$
- This probability is $\approx (1 - e^{-km/n})^k$
- Optimal number k of hash functions: $\log_e 2 \times n/m$

$(1-\varepsilon)^{1/\varepsilon} \approx 1/e$
for small ε

Available Frameworks for Stream Analytics

- Apache Storm – created by Twitter, open-source
- Apache Spark - Cloudera and MapR partner with Databricks.
- IBM Infosphere Streams – Eclipse based IBM flagship tool
- TIBCO StreamBase - offers a live data mart for streaming data
- Apache Samza -A distributed stream processing framework processor, recently open-sourced by LinkedIn.
- AWS Kinesis - A managed cloud service from Amazon. Commercial.
- Etc…

Apache Storm

- Storm is a distributed real time computation system.
- Since 2010.
- Created by Backtype (acquired by Twitter).
- Composed of many open source components, especially ZooKeeper for cluster management, ZeroMQ for multicast messaging, and Kafka for queued messaging.
- Early stages of development.
- Non commercial.
- Most starred project on Github.

Typical Use Cases

- Processing streams
- Continuous computation
  - Send data to clients continuously so they can update and show results in real time, such as site metrics.
- Distributed remote procedure call
  - Easily parallelize CPU-intensive operations.
- Online Machine Learning

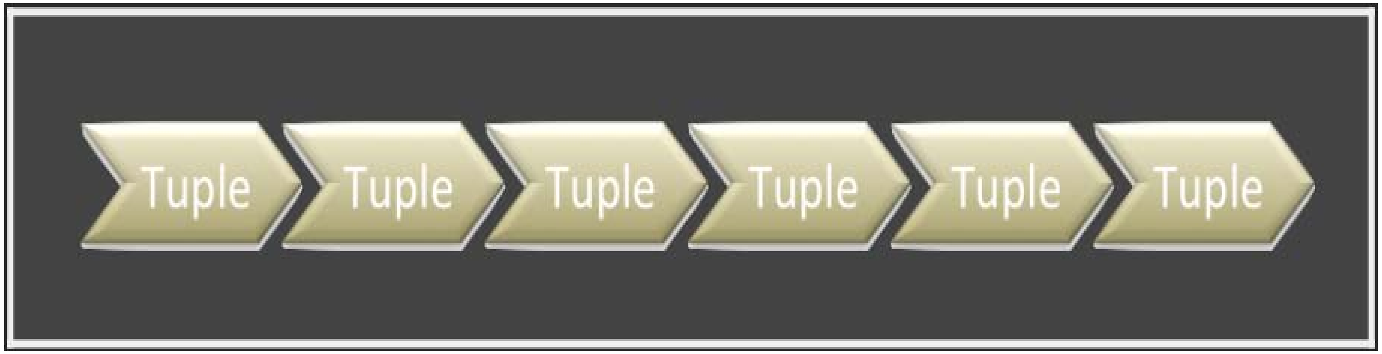
Key Concepts : Tuple

- A tuple is a heterogeneous ordered collection of named data objects.
- Example:

```
[ 198735697, "foobar", { "ip" : "10.0.0.1" } ]
```

Key Concepts : Streams

- An unbounded sequence of Tuples.



Key Concepts : Spouts

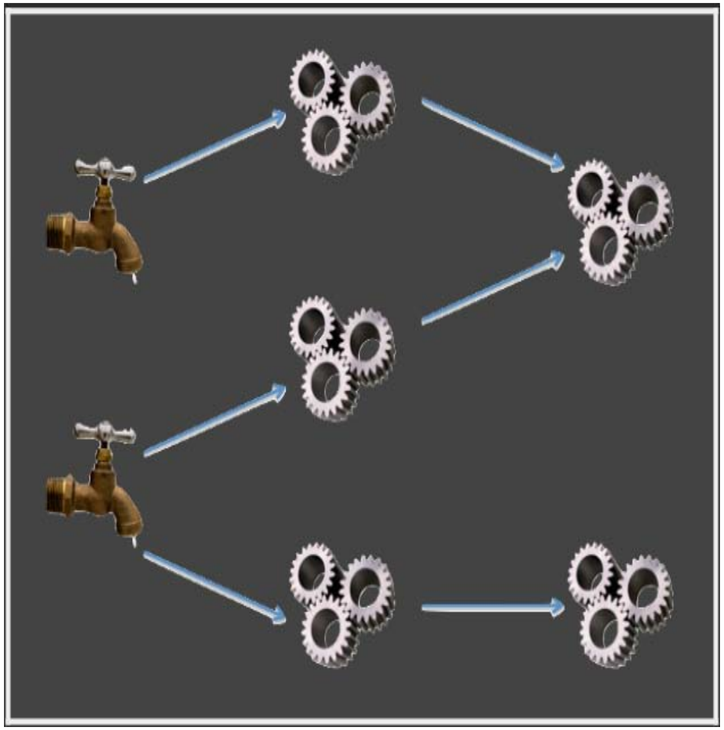
- Generates tuples from other sources i.e. produces data streams.
  - Event Data
  - Log Files
  - Queues
  - Sensor Data from a port
  - Internet

Key Concepts : Bolts

- Processes each tuple.
- Can be any functionality.
- Filter, computation, aggregation, join, talking to databases
- Can create a new stream
- Obtains input from one/more spouts and bolts.

Key Concepts : Topology

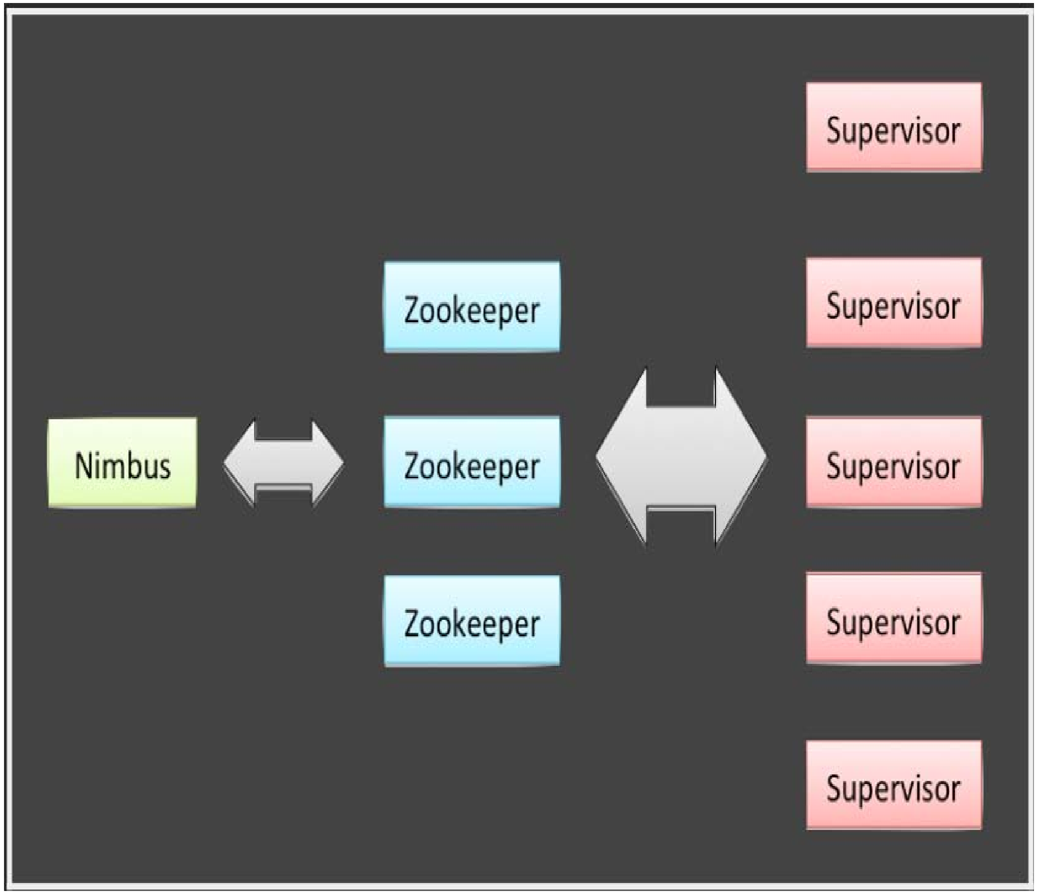
- Configured Graph of Spouts and Bolts.



Key Concepts : Cluster

- Nimbus Node (master node) :
  - Distributes code across the cluster
  - Launches workers across the cluster
  - Monitors computation and reallocates workers as needed
- Supervisor Nodes (worker nodes) :
  - Each worker node is controlled by a Supervisor, which executes a portion of a topology
  - Starts and Stops worker nodes based on signals from Nimbus.
- Zookeeper :
  - Coordinates the storm cluster.
  - Maintains state of worker nodes.

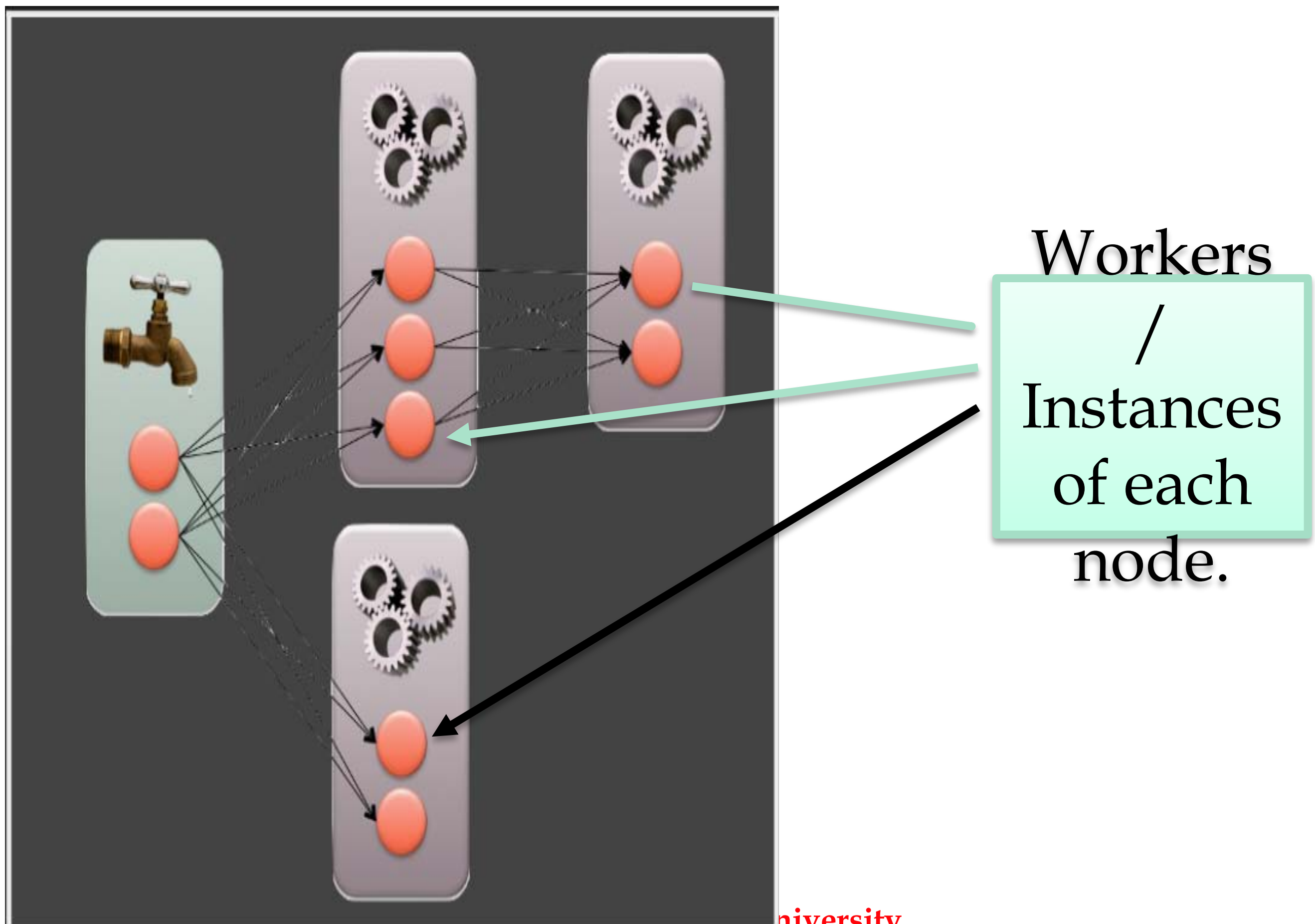
Sample Cluster



Key Concepts : Cluster

- A cluster can be in two modes:
  - **Local** : Storm topologies run on the local machine in a single JVM. This mode is used for development, testing, and debugging because it's the easiest way to see all topology components working together.
  - **Remote/Production Mode**: In this case, we submit our topology to the Storm cluster, which is composed of many processes, usually running on different machines. Remote Mode doesn't show debugging information, which is why it's considered Production Mode.
- A worker is a thread spawned by a supervisor to do work.

Workers



Stream Grouping

When a tuple is emitted by a spout or bolt, which task/instance does it go to?

- Shuffle grouping : does random task distribution.
- Field grouping : groups by specific field.
- All grouping replicates to all tasks (Careful!!)
- Global grouping sends an entire stream to one task.
- Custom Grouping : design your own!

Example

Problem At Hand :

- Given a real time data stream, we intend to summarize the data in the form of windows and answer window-based user queries.

Assumptions :

- In this case, we considered mean and standard deviation as two typical user queries.
- We considered fixed window size, although that can be adjusted based on application.

Spout : DataReader

- `Extends BaseRichSpout`

- A spout in general needs to define the following functions:
  - `public void open(Map conf, TopologyContext ctx, SpoutOutputCollector soc)`
    - Called once when a Spout starts.
    - conf contains the variables shared by the topology components.
    - Ctx contains all information regarding the structure of the topology.
    - A SpoutOutputCollector object allows us the emit tuples to be processed by bolts
  - `public void declareOutputFields(OutputFieldsDeclarer arg0)`
  - `public void nextTuple()`
  - `public void ack(Object msgId)`
  - `public void fail(Object msgId)`

Open function definition

```
public void open(Map arg0, TopologyContext arg1, SpoutOutputCollector arg2) {
    this.collector=arg2;
    try{
        fr=new FileReader(arg0.get("dataFile").toString());
        windowSize=Integer.parseInt(arg0.get("windowSize").toString());
        sc=new Scanner(fr);
        window=new ArrayList<Integer>();

    }
    catch(Exception e){
        throw new RuntimeException("Error reading file ["+arg0.get("dataFile")+"]");
    }

}
```

A window size is specified at command line.

We use a file as a data source in this case. The file contains integers.

Initialize a default empty window

Handle Expception

Declare Output Fields

```java
public void declareOutputFields(OutputFieldsDeclarer arg0) {
    arg0.declare(new Fields("window","timestamp"));

}
```

Define the name of fields to be emitted per tuple

```java
public void nextTuple() {

    if(!sc.hasNext()){
        try {
            Thread.sleep(1000);
        } catch (InterruptedException e) {
            //Do nothing
        }
        return;
    }
    try{
    while(sc.hasNext()){
        window.add(sc.nextInt());
        if(window.size()==windowSize)
            {
            this.collector.emit(new Values(window,new Timestamp(System.currentTimeMillis())),window.size() );

            for(Integer i:window){

            //System.out.println("Data Reader : "+i);
            }
            window=new ArrayList<Integer>();
            }
    }
```

Called continuously while topology is running.

Check if more numbers are available. If no, then sleep.

Add numbers to window.

If window is full, emit the current timestamp and the window.

Reinitialize the window

Ack and Fail

Acknowledgement for each tuple

```
public void ack(Object msgId) {
    System.out.println("OK:"+msgId);
}
public void fail(Object msgId) {
    System.out.println("FAIL: "+msgId);
}
```

Failure msg for each tuple

Bolt

- A bolt needs to define the following:
  - `extends BaseBasicBolt`
  - `public void prepare(Map stormConf, TopologyContext context)`
  - `public void execute(Tuple input, BasicOutputCollector arg1)`
  - `public void declareOutputFields(OutputFieldsDeclarer arg0)`
  - `public void cleanup()`

Bolt : CalcStandardDeviation

```
public void prepare(Map arg0, TopologyContext arg1) {

    this.componentID = arg1.getThisComponentId();
    this.taskID = arg1.getThisTaskId();
    this.logger=new Logger((String) arg0.get("timestamp"));
    data=new HashMap<Timestamp, Double[]>();

}
```

Called once when the Bolt starts

Internal Record stores SD and mean for each timestamped window.

Get the component ID(bolt name) and taskID (instance ID)

Used to emit new tuples if required

Run each time a tuple is received

Get window and timestamp from tuple

Store in internal data structure

```
public void execute(Tuple input, BasicOutputCollector arg1) {
    ArrayList<Integer> window = (ArrayList<Integer>) input.getValueByField("window");
    Timestamp ts = (Timestamp) input.getValueByField("timestamp");
    Double[] measure = calcSDnMean(window);
    data.put(ts, measure);
    logger.write("TaskID: "+taskID+" ComponentID: "+componentID+" Put "+measure[0]+" "+measure[1]);

}
```

SD and Mean calculation routine

```
private Double[] calcSDnMean(ArrayList<Integer> arr) {
    Double measures[] = new Double[2];
    measures[0] = 0.0;
    measures[1] = 0.0;
    for (Integer i : arr) {
        measures[0] += i;
        measures[1] += i * i;
    }
    measures[0] = measures[0] / arr.size();
    measures[1] = Math.sqrt(measures[1] / arr.size() - measures[0]);
    return measures;
}
```

```java
public void cleanup() {

    for(Timestamp t:data.keySet()){

        logger.write(t.toString()+" : "+data.get(t)[0]+ ","+data.get(t)[1]);
    }
    logger.close();

}
```

Called when the bolt instance is terminated

We print all records at the end of execution

```java
public void declareOutputFields(OutputFieldsDeclarer arg0) {
    // TODO Auto-generated method stub

}
```

Output fields must be mentioned here just like spouts if a tuple is emitted

CalcPolyFit Bolt

```
public void execute(Tuple input, BasicOutputCollector arg1) {
    ArrayList<Integer> window = (ArrayList<Integer>) input.getValueByField("window");
    Timestamp ts = (Timestamp) input.getValueByField("timestamp");
    double[] measure = calcPolyParams(window);
    data.put(ts, measure);
    logger.write("TaskID: "+taskID+" ComponentID: "+componentID+" Put 10");

}
```

Same as SD calculating bolt. We just calculate the coefficients of a polynomial fitted to the window

All other methods are same as SDBolt

Polynomial Fitting

```java
import org.apache.commons.math3.fitting.PolynomialCurveFitter;
import org.apache.commons.math3.fitting.WeightedObservedPoints;

 private double[] calcPolyParams(ArrayList<Integer> window) {
     PolynomialCurveFitter pcf=PolynomialCurveFitter.create(10);
     WeightedObservedPoints obs=new WeightedObservedPoints();

     for(int i=0;i<window.size();i++){
        obs.add(i,window.get(i) );
     }


     double[] coeff=pcf.fit(obs.toList());

     return coeff;
 }
```

Import the apache math commons library

Specify the degree of the polynomial

Add all points of the window to the list

Fit the values and create a polynomial

Define the topology

```
public static void main(String args[]) throws InterruptedException{

    TopologyBuilder builder=new TopologyBuilder();
    builder.setSpout("value-reader",new DataReader());
    builder.setBolt("sd-calc", new CalcStandardDeviation(),10).shuffleGrouping("value-reader");
    builder.setBolt("polyfit-calc", new CalcPolyFit(),10).shuffleGrouping("value-reader");

    //Logger logger=new Logger("");
    Config conf=new Config();
    //conf.put("logger", logger);
    conf.put("dataFile",args[0]);
    conf.put("windowSize", args[1]);
    conf.put("timestamp", String.valueOf(System.currentTimeMillis()));
    //conf.put("logger",logger);

    conf.setDebug(false);

    conf.put(Config.TOPOLOGY_MAX_SPOUT_PENDING, 1);
    LocalCluster cluster=new LocalCluster();
    cluster.submitTopology("SD-Topology", conf, builder.createTopology());
    Thread.sleep(20000);
    //logger.close();
    cluster.shutdown();
}
```

How to run?

- Initialize storm
  - zkServer.sh start => starts zookeeper
  - storm nimbus => starts nimbus node
  - storm supervisor => starts supervisor node
  - storm ui => starts the web interface to check the progress while running a topology
- Package the java files into an executable jar
- Place the dependent jar files into Storm's lib directory
- Run:

  storm jar SDStorm.jar proj.dev.SDTopologyMain <filename> <windowsize>

Output is stored in log files.

Conclusion : Relation to Hadoop / Big Data

- Big data is not just about Volume, but also about Velocity and Variety.
- A big data architecture contains several parts. Often, masses of structured and semi-structured historical data are stored in Hadoop (Volume + Variety). On the other side, stream processing is used for fast data requirements (Velocity + Variety).
- Hadoop initially started with MapReduce, which offers batch processing where queries take hours, minutes or at best seconds. This is and will be great for complex transformations and computations of big data volumes. However, it is not so good for ad hoc data exploration and real-time analytics.

- Stream processing is required when data has to be processed fast and / or continuously, i.e. reactions have to be computed and initiated in real time. This requirement is coming more and more into every vertical.
- Apache Storm is a good, open source framework; however custom coding is required due to a lack of development tools and there's no commercial support right now.
- Storm and Spark were not invented to run on Hadoop, but now they are integrated and supported by the most popular Hadoop distributions (Cloudera, Hortonworks, MapR), and can be used for implementing stream processing on top of Hadoop.
- Big Data and Internet of Things are huge drivers of change!

THANK YOU !