CS535 BIG DATA

PART 2. SCALABLE FRAMEWORKS FOR REAL-TIME BIG DATA ANALYTICS

1. SPEED LAYER: APACHE STORM

Sangmi Lee Pallickara
Computer Science, Colorado State University
http://www.cs.colostate.edu/~cs535

FAQs

- Term project proposal due on Friday
- Your presentation will be next week
- **IMPORTANT:** Please send me your slides at least 2 hours before your presentation

Today's topics

- SGD
- Speed Layer
- Apache Storm
  - Word count example
  - Parallelism

Using Gradient Descent Algorithm for Linear Regression Model

Gradient descent algorithm

```
Repeat until convergence {
```

$$\theta_0 - \alpha \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1)$$

```
        (for j=0 and j=1)
}
```

Linear Regression Model

$$h_\theta(x) = \theta_0 + \theta_1 x_1$$

$$J(\theta) = \frac{1}{2m} \sum_{i=0}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$$

$$\frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1) = \frac{\partial}{\partial \theta_j} \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - (y^{(i)}))^2$$

$$= \frac{\partial}{\partial \theta_j} \frac{1}{2m} \sum_{i=1}^{m} (\theta_0 + \theta_1 x^{(i)} - y^{(i)})^2$$

Case 1, θ_0 $(j = 0)$:

$$\frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1) = \frac{\partial}{\partial \theta_0} \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - (y^{(i)}))^2 = \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - (y^{(i)}))$$

Case 2, θ_1 $(j = 1)$:

$$\frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1) = \frac{\partial}{\partial \theta_0} \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - (y^{(i)}))^2 = \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - (y^{(i)})) x^{(i)}$$

Gradient descent for Linear Regression

```
Repeat until convergence {
```

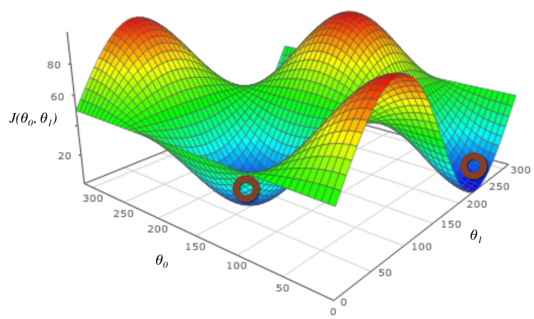
$$\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})$$

$$\theta_1 := \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x^{(i)}$$

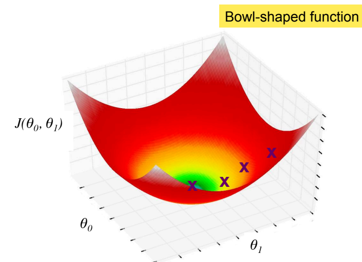
```
        (for j=0 and j=1)
}
```

Update θ_0 and θ_1 simultaneously

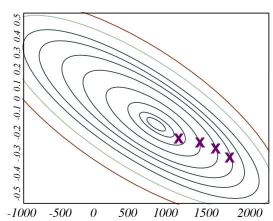
Multiple local optimal points



Convex function

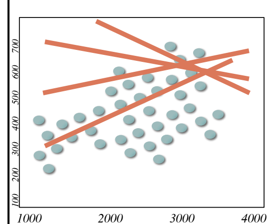


$J(\theta_0,\theta_1)$

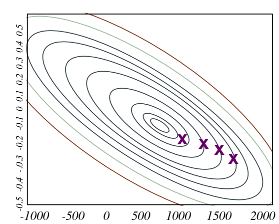


Fitting $h_\theta(x)$

$h_\theta(x)$



$J(\theta_0,\theta_1)$



"Batch" Gradient Descent

- Batch
  - Each step of gradient descent uses all of the training example

$$\theta_j := \theta_j + \alpha \frac{1}{m} \sum_{i=1}^{m} (y^{(i)} - h_\theta(x^{(i)}))x_j^{(i)}$$

Running with Spark in parallel

- For the sample size 1,000 (m=1,000)
- Batch gradient descent:

$$\theta_j := \theta_j + \alpha \frac{1}{1{,}000} \sum_{i=1}^{1000} (y^{(i)} - h_\theta(x^{(i)}))x_j^{(i)}$$

- Using 4 machines

continued

- Step 1. 4 input splits
- $(x^{(1)},y^{(1)}),\ldots,((x^{(250)},y^{250)})$
- $(x^{(251)},y^{(251)}),\ldots,((x^{(500)},y^{500)})$
- $(x^{(501)},y^{(501)}),\ldots,((x^{(750)},y^{750)})$
- $(x^{(751)},y^{(751)}),\ldots,((x^{(1000)},y^{1000)})$
- Step 2. Calculate temp1 ~ 4
- Step 3. Calculate final results

$$temp1 = \sum_{i=1}^{250} (y^{(i)} - h_\theta(x^{(i)}))x_j^{(i)}$$

$$temp2 = \sum_{i=256}^{500} (y^{(i)} - h_\theta(x^{(i)}))x_j^{(i)}$$

$$temp3 = \sum_{i=501}^{750} (y^{(i)} - h_\theta(x^{(i)}))x_j^{(i)}$$

$$temp4 = \sum_{i=751}^{1000} (y^{(i)} - h_\theta(x^{(i)}))x_j^{(i)}$$

Implementation

- **`SVMWithSGD`**
- `LogisticRegressionWithBFGS`
- **`LogisticRegressionWithSGD`**
- **`LinearRegressionWithSGD`**
- **`RidgeRegressionWithSGD`**
- **`LassoWithSGD`**

- Cheng-Tao Chu, Sang Kyun Kim, Yi-An Lin, YuanYuan Yu, Gary Bradski, and Andrew Y. Ng, Map-Reduce for Machine Learning on Multicore, NIPS 2006: 281-288

Speed layer: Apache Storm

This material is built based on:

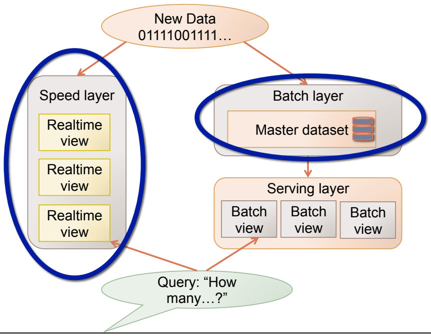
- Nathan Marz and James Warren, "Big Data, Principles and Best Practices of Scalable Real-Time Data System", 2015, Manning Publications, ISBN 9781617290343
- Toshniwal, Ankit and Taneja, Siddarth and Shukla, Amit and Ramasamy, Karthik and Patel, Jignesh M. and Kulkarni, Sanjeev and Jackson, Jason and Gade, Krishna and Fu, Maosong and Donham, Jake and Bhagat, Nikunj and Mittal, Sailesh and Ryaboy, Dmitriy, "Storm@twitter", Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data, SIGMOD June 22-27, 2014, Snowbird, Utah
- P. Taylor Goetz, and Brian O'Neill, "Storm Blueprints: Patterns for Distributed Real-time Computation" Packt Publishing (March 26, 2014)
- Apache's Storm
  - Open source project
  - https://**storm**.**apache**.org/

Speed layer: Apache Storm
Speed Layer

Where are we in the Lambda Architecture?

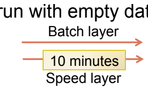
- We have focused on batch computing in the Lambda Architecture
  - Computing framework
  - Scalable algorithms
- Low-latency update
  - Jobs of the speed layer
  - Incremental computation

Lambda architecture

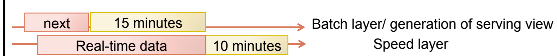


Communication between speed layer and batch layer

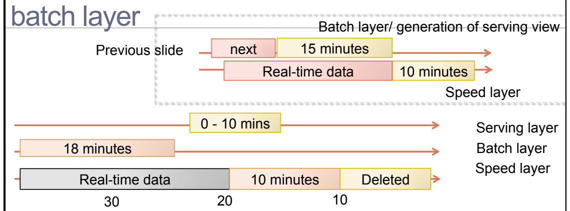
- Assume that the first batch layer will run with empty dataset (or, Wait)
  - First 10 minutes
  - Data is processed in the speed layer



- Second run of the batch layer immediately commences to process 10 minutes of data that accumulated during the first run
  - Assume that the second run takes 15 minutes
  - After this run, the serving layer will represent the first 10 minutes of data
- The first 10 minutes can now be removed from the speed layer



Communication between speed layer and batch layer



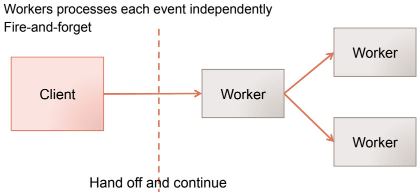
- The third run of the batch layer takes 18 minutes for data that arrived between 10 ~ 20 minutes

Speed layer: Apache Storm

Queuing and Stream processing

Queuing

- A system without persistent queuing
  - Events would be handed directly to workers
    - Workers processes each event independently
    - Fire-and-forget



- **Can this scheme guarantee that all the data is successfully processed?**
  - **No**
  - **Why?**

Interface

- If a worker dies before completing its assigned task
  - Is there any mechanism to detect or correct the error?
    - No
- If there is bursty traffic that exceeds the capacity of the processing cluster
  - Is there any mechanism to process all of the events?
    - No

```
Interface Queue{
    void add(Object item);
    Object poll();
    Object peek();
}
```

Adds new item to the queue

Inspects the item at the head of the queue without removing it

Remove the item from the head of the queue

Single consumer queue servers

```
struct Item{
    long id;
    byte[] item;
}
Interface Queue {
    Item get();
    void ack(long id);
    void fail (long id);
}
```

A generic Item consists of an identifier and a binary payload

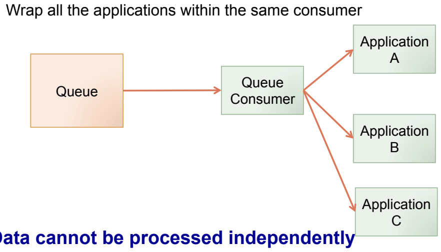
Acknowledges successful processing

Reports a failure

- When you read an event from the queue
  - The event is not immediately removed
  - The item is returned by the get() function
  - Only when an event is acked, will it be removed from the queue
  - For failed retrieval, another client can retrieve via separate get() function
  - The data is processed **at least once**

Multiple application with a single queue (1/2)

- What if multiple applications want to consume the same stream?
- **Approach 1.**
  - Wrap all the applications within the same consumer



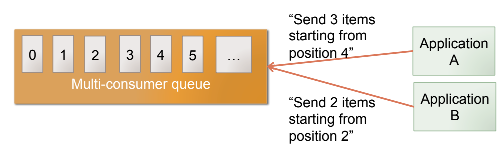
- **Data cannot be processed independently**

Multiple application with a single queue (2/2)

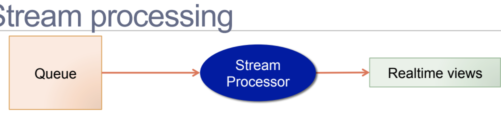
- **Approach 2:**
  - Maintaining a separate queue for each consumer application
  - If you have three applications
    - There are three separate copies of the queue on the queue server
- This increases the load on the queue server

Multi-consumer queues

- The applications track the consumed/unconsumed status of events from the queue
- Servers
  - Guarantee that a certain amount of the stream is available
    - E.g. all events from the past 12 hours or the last 50GB of events



Stream processing

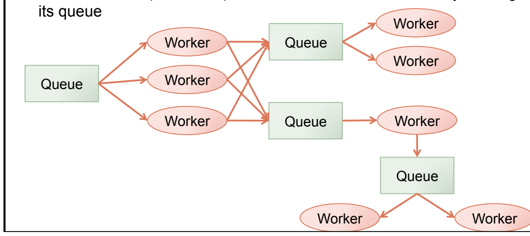


- One-at-a-time
  - Processes streams with lower latency than micro-batched
  - Queues-and-workers model
- Micro-batched
  - Small batches of tuples are processed at one time

	One-at-a-time	Micro-batched
Lower latency	Yes	No
Higher throughout	No	Yes
At-lease-once semantics	Yes	Yes
Exactly-once semantics	In some cases	Yes
Simpler programming model	Yes	No

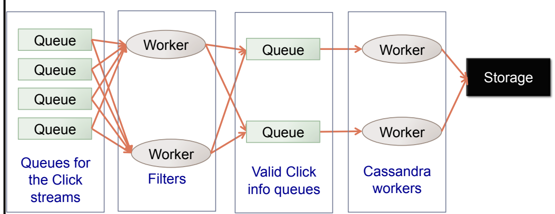
Queues-and-workers model (1/2)

- Common approach to achieve **one-at-a-time** stream processing
  - Divides processing pipeline into worker processes
  - Places queues between them
  - If a worker fails (or restarts) it can continue where it left off by reading from its queue



Queues-and-workers model (2/2)

- If you count number of clicks for a set of Web pages

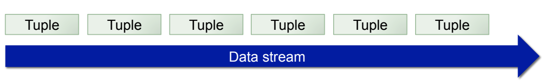


Queues-and-workers pitfalls

- Race conditions
  - Multiple workers should not attempt to update the count of page click of the same page at the same time
  - To avoid this, partitioning over the entire set of URLs should be spread among the queues but any event(click) for the same URL should be delivered to the same queue
- Operational burden
  - If you need to change the topology of your processing, intermediate queues should be cleared before you redeploy
  - Queues add latency
  - Queues decrease the throughput
  - Managing overhead
- Complex to implement

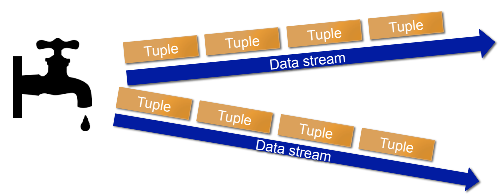
Storm Model

- One-at-a-time stream processing
- Represents the entire stream processing pipeline as a graph of computation called a **topology**
- A single program is deployed across a cluster
- A stream is represented an infinite sequence of **tuples**
  - **A tuple: a named list of values**



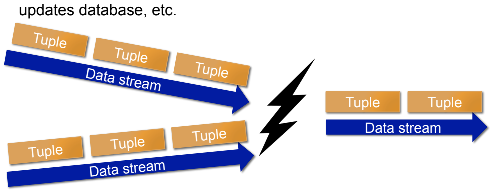
Spout in the Storm model

- Spout
  - A source of streams in a topology
  - A spout can read from a Kestrel or Kafka queue
  - Turns the data into a tuple stream
  - Timer spout could emit a tuple into its output stream every 10 seconds



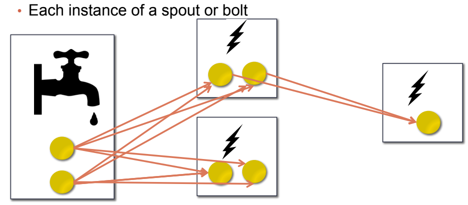
Bolt in the Storm model

- Bolt
  - Performs **actions** on streams
  - Takes any number of streams as input and produces any number of streams as output
  - Runs functions, filters data, computes aggregations, does streaming joins, updates database, etc.



Topology in the Storm model

- **Topology**
  - A network of spouts and bolts with each edge representing a bolt that processes the output stream of another spout or bolt
- **Task**
  - Each instance of a spout or bolt

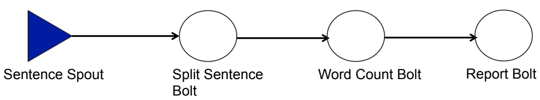


Storm

- Scalability
  - Nodes should be added or removed from the Storm cluster without disrupting existing data flows (standing query)
- Resiliency
  - During hardware failures, existing topologies must continue processing with minimal performance impact
- Extensibility
  - External functions should be compatible
- Efficiency
  - Good performance characteristics must be provided for realtime applications
- Easy to Administer
  - Failure or performance issues should be addressed immediately

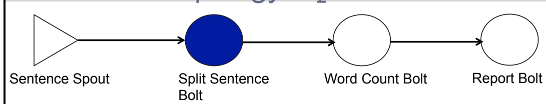
Word Count Example

Word count topology: `Sentence Spout`



- Sentence spout
  - **Emits a stream of single-value tuples continuously** with the key name "`sentence`" and a string value
  - `{"sentence":"my dog has fleas"}`

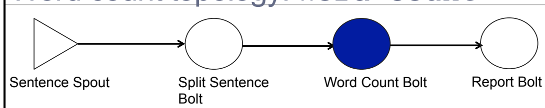
Word count topology: `Split Sentence`



- Split Sentence Bolt
  - Subscribes to the sentence spout's tuple stream

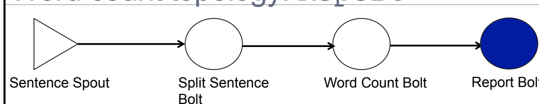
```
{"word":"my"}
{"word":"dog"}
{"word":"has"}
{"word":"fleas"}
```


Word count topology: `Word Count`



- `Word count bolt`
  - Subscribes to the output of the `SplitSentenceBolt` class
  - Keeps a count of how many times it has seen a particular word
  - Whenever it receives a tuple, it will increment the counter and emit
  - `{"word":"dog", "count":5}`

Word count topology: `Report`



- `Report bolt`
  - Subscribes to the output of the `WordCountBolt` class
  - Keeps a count of how many times it has seen a particular word
  - Whenever it receives a tuple, it will update the table and print the contents to the console

SentenceSpout.java

```
public class SentenceSpout extends BaseRichSpout {
        private SpoutOutputCollector collector;
        private String[] sentences = {
                "my dog has fleas",
                "i like cold beverages",
                "the dog ate my homework",
                "don't have a truck",
                "i don't think i like fleas"
        };

        private int index = 0;
        public void declareOutputFields(OutputFieldsDeclarer declarer) {
                declarer.declare(new Fields("sentence"));
        }
```


SentenceSpout.java: Continued

```
        public void open(Map config, TopologyContext context,
                        SpoutOutputCollector collector) {
                this.collector = collector;
        }

        public void nextTuple() {
                this.collector.emit(new Values(sentences[index]));
                index + +;
                if (index > = sentences.length) {
                        index = 0;
                }
                Utils.waitForMillis(1);
        }
}
```


SplitSentenceBolt.java

```
public class SplitSentenceBolt extends BaseRichBolt{
        private OutputCollector collector;
        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 void declareOutputFields(OutputFieldsDeclarer declarer) {
                declarer.declare(new Fields("word"));
        }
}
```


WordCountBolt.java

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

                Long count = this.counts.get(word);
                if(count = = null){ count = 0; }
                count + +;
                this.counts.put( word, count);
                this.collector.emit(new Values(word, count));
        }
        public void declareOutputFields( OutputFieldsDeclarer declarer) {
                declarer.declare(new Fields("word", "count"));
        }
}
```