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

1. Speed Layer: Apache Storm

This material is built based on:
- Toshniwal, Ankit and Taneja, Siddarth and Shukla, Amit and Ramasamy, Kartik and Patel, Jignesh M. and Kulkarni, Sanjeev and Jackson, Jason and Oade, Krishna and Fu, MaoSang and Donham, Jake and Bhagat, Nikunj and Mital, Saleesh and Rubay, Dmitriy, "Storm@twitter", Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data, SIGMOD June 22-27, 2014, Snowbird, Utah
- Apache’s Storm
- Open source project
- https://storm.apache.org/
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

Queueing

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

1. Speed layer: Apache Storm

Queuing and Stream processing
Simple Queue with `poll()` and `peek()`

- 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

```java
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

- 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 acknowledged, 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

```java
struct Item {
    long id;
    byte[] item;
}
```

- A generic Item consists of an identifier and a binary payload
- Acknowledges successful processing
- Reports a failure

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
- Micro-batched
  - Small batches of tuples are processed at one time

<table>
<thead>
<tr>
<th></th>
<th>One-at-a-time</th>
<th>Micro-batched</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lower latency</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Higher throughput</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>At-least-once semantics</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Exactly-once semantics</td>
<td>In some cases Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Simpler programming model</td>
<td>Yes</td>
<td>No</td>
</tr>
</tbody>
</table>
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
  
  ![Diagram of queues and workers model](image)

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 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 as 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
  - Sends 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 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}"

SentenceSpout.java

```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"));
  }

  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

```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

```java
public class WordCountBolt extends BaseRichBolt {
  private OutputCollector collector;

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

  public void execute(Tuple tuple) {
    String word = tuple.getStringByField("word");
    long count = this.collector.count(word);
    if(count != 0) {
      this.collector.emit(new Values(word, count));
    }
  }

  public void declareOutputFields(OutputFieldsDeclarer declarer) {
    declarer.declare(new Fields("word", "count"));
  }
}
```
Components of the Storm cluster

- **Nodes** (machines)
  - Executes portions of a topology

- **Workers** (JVMs)
  - Independent JVM processes running on a node
  - Each node is configured to run one or more workers

- **Executors** (threads)
  - Java threads running within a worker JVM process
  - Multiple tasks can be assigned to a single executor
  - Unless explicitly overridden, Storm will assign one task to each executor

- **Tasks** (bolt/spout instances)
  - Instances of spouts and bolts whose nextTuple() and execute() methods are called by executor threads

---

1. Speed layer: Apache Storm
   Parallelism in Storm
Parallelism in the WordCount topology

- In our example, we have NOT used any of Storm’s parallelism
- Default setting is a factor of one
- Topology execution flow

Adding workers to a topology

- Through configuration
- Through APIs
- Passing Config object to the submitTopology() method

Adding executors and tasks

- Specify the number of executors when defining a stream grouping

Adding workers to a topology

- Through configuration
- Through APIs
- Passing Config object to the submitTopology() method

In SplitSentenceBolt and WordCountBolt

- Set up the split sentence bolt to execute as 4 tasks and 2 executors
  - Each executor thread will be assigned two tasks to execute
  builder.setBolt(SPLIT_BOLT_ID, splitBolt, 2)
  .setNumTasks(2)
  .shuffleGrouping(SENTENCE_SPOUT_ID);

- Set up the Word count bolt to execute as 4 tasks each with its own executor thread
  builder.setBolt(COUNT_BOLT_ID, countBolt, 4)
  .fieldsGrouping(SPLIT_BOLT_ID, new Fields("word"));
What will be the results with given parallelism?

--- FINAL COUNTS ---

a : 1426
ate : 1426
beverages : 1426
cold : 1426
cow : 1426
dog : 1426
don't : 2852
dog's : 2851
fleas : 2851
has : 1426
have : 1426
homework : 1426
i : 4276
like : 2851
man : 1426
my : 2852
the : 1426
think : 1425

--------------

--- FINAL COUNTS ---

a : 2726
ate : 2722
beverages : 2723
cold : 2723
cow : 2726
dog : 5445
don't : 5444
dog's : 5451
fleas : 5451
has : 2723
have : 2722
homework : 2722
i : 8175
like : 5449
man : 2722
my : 5445
the : 2727
think : 2722

--------------

Increased counts

- Spout emits data indefinitely
- Stops when the topology is killed
- Only task and executor parallelism settings have effect
- A topology running in local mode always runs within a single JVM process
- Use your application in a cluster for true parallelism