4. Real-time Streaming Computing Models: Apache Storm and Twitter Heron

Reliability in Storm

FAQs
- Special GTA for PA1
  - Saptashwa Mitra
    - Saptashwa.Mitra@colostate.edu
- You can send an email to Saptashwa or me regarding PA1

Topics of Today's Class
- 4. Real-time Streaming Computing Models: Apache Storm and Twitter Heron
  - Reliability in Storm
  - Microbatch stream processing
  - Heron

This material is built based on
- Storm programming guide
  - http://storm.apache.org/releases/2.0.0-SNAPSHOT/index.html

Guaranteed processing
- Allows you to guarantee that a tuple emitted by a spout is fully processed
- Useful for failures

http://www.cs.colostate.edu/~cs535
Reliability in bolts

- Implementing a bolt that participates in guaranteed processing involves:
  - Anchoring to an incoming tuple when emitting a derived tuple
  - Acknowledging or failing tuples that have been processed successfully or unsuccessfully

- Anchoring to a tuple
  - Creating a link between an incoming tuple and derived tuples
  - Downstream bolts are supposed to acknowledge
  - Fail tuples
  - Time out

Reliability in spouts

- Keeps track of tuples it has emitted
  - Should be prepared to re-emit a tuple if downstream processing of that tuple or any child tuples fails

- Child tuple
  - Tuple emitted as a result of a tuple originating from a spout

- Tuple tree
  - Keeps track of tuples it has emitted
  - If processing fails, spout’s fail method will be called

ISpout interface

- Assign unique ID and pass that value to the emit() method of SpoutOutputCollector

```
public interface ISpout extends Serializable {
    void open(Map conf, TopologyContext context, SpoutOutputCollector collector);
    void close();
    void emit(Object value); // emit the tuple
    void ack(Object msgId);
    void fail(Object msgId);
    Constructor;
}
```

Anchoring to a tuple (or a list of tuples)

- `collector.emit(tuple, new Values(word));`
- Incoming tuple and emitting a new tuple that downstream should acknowledge or fail are anchored

- Only anchored tuple participates in the reliability of a stream

```
if (messageID != null) {
    this.pending.put(msgId, tuple);
}
```

```
collector.emit(new Values("value1", "value2"), msgId);
```

- With a message ID, this spout would like to receive notification when the tuple tree is completed
- Or if fails at any point
- If processing fails, spout’s fail method will be called

```
OutputCollector.fail();
```

Reliable word count

```
public class SentenceSpout extends BasicSpout {
    private ConcurrentHashMap<UUID, Values> pending;
    private SpoutOutputCollector collector;
    private String[] sentences = {"my dog has fleas", "I don't have a cow man", "the dog ate my homework", "my dog has fleas", "don't think like cold beverages"};
    private int index = 0;
    public SentenceSpout() {
        collector = new Collector();
        this.pending = new ConcurrentHashMap<UUID, Values>();
    }
    @Override
    public void declareOutputFields(OutputFieldsDeclarer declarer) {
        declarer.declare(new Fields("sentence"));
    }`
```

```
public void open(Map conf, TopologyContext context, SpoutOutputCollector collector) {
    this.collector = collector;
    this.pending = new ConcurrentHashMap<UUID, Values>();
}
```

```
public void nextTuple() {
    Values values = new Values(sentences[ index]);
    if (index == sentences.length) {
        index = 0;
    }
    this.pending.put(msgId, values);
    this.collector.emit(msgId, values);
}
```

```
public void ack(Object msgId) {
    this.pending.remove(msgId);
    if (index >= sentences.length)
        index = 0;
    this.collector.emit(msgId, sentences[ index++]);
}
```

```
public void fail(Object msgId) {
    this.pending.remove(msgId);
    this.collector.emit(msgId, UUID.randomUUID());
}
```

- If tuple processing fails as a result of a time out or an explicit call
- If it fails,
  - this collector.fail(tuple);
- The spout will be notified

Reliable word count-- Continued

```
public void nextTuple() {
    Values values = new Values(sentences[ index]);
    UUID msgId = UUID.randomUUID();
    this.pending.put(msgId, values);
    this.collector.emit(msgId, values);
    index ++;
    if (index == sentences.length) {
        index = 0;
    }
    this.collector.emit(msgId, values);
}
```

- Reliability stream should acknowledge the inbound tuple
- After successfully processing a tuple and emitting new or derived tuples (optional)
- `this.collector.ack(tupule);`
- If it fails,
  - this.collector.fail(tuple);

http://www.cs.colostate.edu/~cs535

Spring 2019 Colorado State University, page 2
Reliable word count--Continued

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(tuple, new Values(word));
        }
        this.collector.ack(tuple);
    }

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

4. Real-time Streaming Computing Models: Apache Storm and Twitter Heron

Apache Storm

Micro-batch stream processing

Achieving exactly-once semantics

- With one-at-a-time stream processing
  - Tuples are processed independently of each other

- Micro-batch stream processing
  - Small batches of tuples are processed at one time
  - If anything in a batch fails, the entire batch is replayed
  - Batches are processed in a strict order
  - Exactly-once semantics

Strongly ordered processing

- Suppose you want accuracy in your stream computing, regardless of how many failures there are:
  - Exactly once processing
  - Example processing: counting tuples

```java
process(tuple) {
    counter.increment();
}
```

- What if there is a failure?
  - Tuples will be replayed
  - For counter.increment();, you have no idea if that was processed or not

Exactly-once semantics

- How to handle failures?
  - Maintain Track ID
    - Store the ID of the latest tuple that was processed along with the count

- New tuple is delivered with the current tuple ID
  - If the stored ID is the same as that of the current tuple ID?
    - Do nothing
  - If the stored ID is different from the current tuple ID?
    - Increment the counter and update the stored ID

- You can use Ack/Nack to track tuples and maintain a queue for the tuples
- What is the problem of this approach?

Exactly-once semantics

- How to handle failures?
  - Maintain Track ID
    - Store the ID of the latest tuple that was processed along with the count

- New tuple is delivered with the current tuple ID
  - If the stored ID is the same as that of the current tuple ID?
    - Do nothing
  - If the stored ID is different from the current tuple ID?
    - Increment the counter and update the stored ID

- You can use Ack/Nack to track tuples and maintain a queue for the tuples
- What is the problem of this approach? ANSWER: Excessive memory consumption for failures

http://www.cs.colostate.edu/~cs535
Micro-batch stream processing

- Batches are processed in order
  - Each batch has a unique ID
  - Always the same on every replay
- Batches must be processed to completion before moving on to the next batch

Incoming stream of tuples

Batch 1
Batch 2
Batch 3
Batch 4
Batch 5

Micro-batch processing topologies

- Suppose that you are building a streaming application that computes the top-3 most frequently occurring words
  - Micro-batch can accomplish this task while being fully parallelized and being fault tolerant and accurate
  - Task 1
    - Keeps state on the frequency of each word
    - This can be done using key/value storage
  - Task 2
    - If any of the words has higher frequency than one of the current top-3 most frequent words, then the top-3 list must be updated

Each batch includes tuples from all partitions

Partition 1
Partition 2
Partition 3

Process

Parallelizing the global count example

Part 1: Counting and storing the state

- The words should be re-partitioned
  - Same word is always processed by the same task (bolt)
- Database update is done by only one thread per-word
  - No race condition
- Stores count and batch ID

For failures

- When a failed batch is replayed:
  - If the state has current batch ID?
    - No update
  - If the state has a non-current batch ID?
    - Update

Parallelizing the global count example

Part 2: Computing the top-3 most frequent words

- What if we direct any new counts for every word to a single task?
  - Not scalable!
    - The single task will be a bottleneck

- What if each word counting task computes the local top-3 words and sends them to the global top-3 task?
  - Better solution

Failure scenario

- If a node failed and one of the top-3 lists was not sent to the global top-3 task?
  - The batch is replayed it will be updated

- If a node failed after it updated the top-3 list
  - Update won't change the value
  - Idempotent operation

http://www.cs.colostate.edu/~cs535
4. Real-time Streaming Computing Models: Apache Storm and Twitter Heron

Apache Storm

Trident Topology

Trident Topologies

- Trident is a Java API that translates micro-batch processing topologies into the spouts and bolts of Storm
- Eliminates the details of transactional processing and state management
- Batching of tuples into a discrete set of transactions
- Abstracting operations on the data such as functions, filters and aggregations

Example case [1/2]

- Objective: Collecting medical reports to identify the outbreak of a disease
- The topology will process diagnosis events that contain:
  - Latitude
  - Longitude
  - Timestamp
  - Diagnosis Code (ICD9-CM)
- E.g.: 
  - (40.5392, -75.1642, "03/13/2013 at 3:30 PM", "320.0 (Hemophilus meningitis)

Example case [2/2]

- To detect an outbreak,
  1. The system will count the occurrence of specific disease codes within geographic location over a specified period of time
  2. The system will group the occurrences by hour and calculate a trend against the moving average
  3. The system will use a simple threshold to determine if there is an outbreak
  4. If the count of occurrences for the hour is greater than some threshold, the system will send an alert

Workflow for our example case

```java
import static org.apache.storm.tuple.Fields.
import static org.apache.storm.tuple.Fields.
import static org.apache.storm.tuple.Fields.
import static org.apache.storm.tuple.Fields.

public class OutbreakDetectionTopology {
    public static StormTopology buildTopology() {
        TridentTopology topology = new TridentTopology();
        DiagnosisEventSpout spout = new DiagnosisEventSpout();
        Stream inputStream = topology.newStream("event", spout);
        inputStream
            .filter(new Fields("event"), new DiseaseFilter())
            .assignCity()
            .deriveHour()
            .groupBy(new Fields("cityDiseaseHour"))
            .outbreak()
            .trendState()
            .outbreakDetector();
        return topology;
    }
}
```

---

http://www.cs.colostate.edu/~cs535
Trident topology – continued

```java
// Count occurrences and persist the results.
persistentAggregate(new OutbreakTrendFactory(),
    new Count(),
    new Fields("count"));

// Detect an outbreak
each( new Fields("cityDiseaseHour", "count"),
    new OutbreakDetector(),
    new Fields("alert"));
```

Trident Spout interface

```java
public interface ITridentSpout<T> extends Serializable {
    BatchCoordinator<T> getCoordinator(String txStateId, Map conf, TopologyContext context);

    Emitter<T> getEmitter(String txStateId, Map conf, TopologyContext context);

    Map getComponentConfiguration();

    Fields getOutputFields();
}
```

DiagnosisEventSpout

```java
public class DiagnosisEventSpout implements ITridentSpout<Long> {
    private static final long serialVersionUID = 1L;

    private SpoutOutputCollector collector;

    private BatchCoordinator coordinator = new DefaultCoordinator();

    private Emitter<Long> emitter = new DiagnosisEventEmitter();

    @Override
    public BatchCoordinator getCoordinator(String txStateId, Map conf, TopologyContext context) {
        return coordinator;
    }

    @Override
    public Emitter<Long> getEmitter(String txStateId, Map conf, TopologyContext context) {
        return emitter;
    }

    @Override
    public Map getComponentConfiguration() {
        return null;
    }

    @Override
    public Fields getOutputFields() {
        return new Fields("event");
    }
}
```

4. Real-time Streaming Computing Models:
Apache Storm and Twitter Heron
Apache Storm System Architecture

http://www.cs.colostate.edu/~cs535
System architecture overview

- **Nimbus**
  - Master node
  - Distributes and coordinates the execution of the topology

- **Worker nodes**
  - Runs one or more worker processes
  - More than one worker process on the same machine may execute different parts of the same topology
  - Runs a JVM
  - Runs one or more executors
  - Executors are the actual work for a bolt or a spout

Supervisor

- Each worker node runs a supervisor
  - Communicates with Nimbus

Zookeeper

- Maintains the cluster state

Nimbus

- Schedules the topologies on the worker nodes
- Monitors the progress of the tuples flowing through the topology

Nimbus in depth

- Similar role as the "JobTracker" in Hadoop
  - ("Application master" in yarn)

- Contact point between the user and the Storm system

- Submitting a job to Storm
  - Topology described as a Thrift object should be sent to Nimbus
  - Any programming language can be used
  - User’s JAR file is uploaded to Nimbus

Maintaining state of the topology

- State about the topology is stored in the local disk and Zookeeper
  - User code
  - Nimbus and Zookeeper

Match-making topologies and nodes

- Nimbus match-makes between the pending topologies and the Supervisor
  - Supervisor contacts Nimbus
  - Heartbeat protocol
  - Advertising the current topologies
  - Any vacancies for future topologies

Coordination between Nimbus and Supervisors

- Using Zookeeper
- Nimbus and Supervisor daemons are stateless
- Their states are stored in Zookeeper or in the local disk

- If Nimbus fails,
  - Workers still continue to make forward progress
  - Users cannot submit new topologies
  - Reassigning of failed workers is not available

http://www.cs.colostate.edu/~cs535
Supervisor

- Receives assignments from Nimbus
- Spawns workers based on the assignments
- Monitors the status of the workers
- Re-spawns them if necessary

High level architecture of the Supervisor (1/2)

- Main thread
  - Reads the Storm configuration
  - Initializes the Supervisor’s global map
  - Creates a persistent local state in the file system
  - Schedules recurring timer events
- Event manager thread
  - Manages the changes in the existing assignments

High level architecture of the Supervisor (2/2)

- Process event manager thread
  - Manages worker processes on the same node as the supervisor
  - Reads worker heartbeats from the local state
  - Classifies those workers as valid, timed out, not started, or disabled
  - "valid
    - The worker did not provide a heartbeat in the specified time frame
  - "timed out"
    - The worker did not provide a heartbeat in the specified time frame
  - "not started"
    - Newly submitted topology or recently moved worker
  - "disabled"
    - The worker should not be running either because its topology has been killed or the worker has been moved to another node

Routing incoming and outgoing tuples

1. Worker receive thread
   - Listens on all TCP/IP ports
   - De-multiplexing point for all incoming tuples
   - Checks the tuple destination task identifier and queues
2. User logic thread
   - Takes incoming tuples from the in-queue
   - Checks the destination task identifier
   - Runs actual task (a spout or bolt instance)
   - Generates output tuples
   - These tuples are placed in an out-queue for this executor
3. Executor send thread
   - Takes tuples from the out-queue
   - Puts them in a global transfer queue
   - Contains all the outgoing tuples from several executors
4. Worker send thread
   - Checks tuples in the global transfer queue
   - Sends it to the next worker downstream

Message flow inside worker

4. Real-time Streaming Computing Models:
   Apache Storm and Twitter Heron

Apache Storm
Apache Heron

http://www.cs.colostate.edu/~cs535
Limitation of the Storm worker architecture

- Multi-level scheduling and complex interaction
  - Tasks are scheduled using JVM’s preemptive and priority-based scheduling algorithm
  - Each thread runs several tasks
  - Executor implements another scheduling algorithm
- Hard to isolate its resource usage
  - Tasks with different characteristics are scheduled in the same executor (e.g. Kafka spout, a bolt writing output to a key-value store, and a bolt joining data can be in a single executor)
- Logs from multiple tasks are written into a single file
  - Hard to debug and track the topology

You mentioned that the hard-to-isolate resource usage is a limitation. Can you elaborate on how this affects performance and debugging in Storm applications?

Resource allocation

- Resource allocation
  - Waste resources
    - e.g. 3 spouts: each requires 5GB
    - 1 bolt: requires 10GB
  - On 2 workers
  - Resource allocation policy
    - Divides tasks to 2:2 (2 spouts and 1 spout + 1 bolt)
    - The current storm’s policy will allocate maximum possible resource per worker
    - 10+5 GB per worker
    - Total 30 GB with 2 workers
    - The topology requires 25GB
  - 5GB will be wasted

Can you explain how the current resource allocation policy impacts the efficiency and scalability of Storm applications?

Limitation of the Storm Nimbus

- Scheduling, monitoring, and distributing JARs
  - Topologies are untraceable
  - Nimbus does not support resource reservation and isolation
  - Storm workers that belong to different topologies running on the same machine
    - Interfere with each other
  - Zookeeper manages heartbeats from workers and the supervisors
    - Becomes a bottleneck
  - The Nimbus component is a single point of failure

How does the limited resource allocation and the single point of failure in the Nimbus component affect the reliability and performance of Storm applications?

Apache Heron

- Maintains compatibility with the Storm API
- Data processing semantics
  - At most once – No tuple is processed more than once, although some tuples may be dropped, and thus may miss being analyzed by the topology
  - At least once – Each tuple is guaranteed to be processed at least once, although some tuples may be processed more than once, and may contribute to the result of the topology multiple times

Can you compare Apache Heron’s data processing semantics with Storm’s to highlight their differences and implications for developers?

Aurora Scheduler

- Aurora
  - Generic service scheduler runs on Mesos

What are the key features of the Aurora Scheduler that make it suitable for Storm applications, and how does it compare to other Storm schedulers in terms of resource management and scalability?
Aurora Scheduler
- Each topology runs as an Aurora job
  - Consisting several containers
  - Topology master
  - Stream manager
  - Heron instances
  - Generic service scheduler runs on Mesos

Topology Backpressure
- Dynamically adjust the rate at which data flows through the topology
  - Skewed data flows

• Strategy 1: TCP Backpressure
  - Using TCP windowing
  - TCP connection between HI and SM
  - E.g. for the slow HI, SM will notice that its send buffer is filling up
  - SM will propagate it to other SMs

• Strategy 2: Spout Backpressure
  - SMs clamp down their local spouts to reduce the new data that is injected into the topology
  - Step 1: Identifies local spouts reading data to the straggler HIs
  - Step 2: Sends special message (start backpressure) to other SMs
  - Step 3: Other SMs clamp down their local spouts
  - Step 4: Once the straggler HI catches up, send a stop backpressure message to other SMs
  - Step 5: Other SMs start consuming data

• Strategy 3: Stage-by-stage backpressure
  - Gradually propagates the backpressure stage-by-stage until it reaches the spouts
  - which represent the 1st stage in any topology

Questions?