Storm@Twitter

Ankit Toshniwal, Siddarth Taneja, Amit Shukla, Karthik Ramasamy, Jignesh M. Patel*, Sanjeev Kulkarni, Jason Jackson, Krishna Gade, Maosong Fu, Jake Donham, Nikunj Bhagat, Sailesh Mittal, Dmitriy Ryaboy

Paper Presented by Harsha Yeddanapudy
Basic Storm data processing architecture consists of streams of tuples flowing through topologies.

vertices - computation
edges - data flow
**Spouts & Bolts**

Spouts produce tuples for the topology

Bolts process incoming tuples and pass them downstream to the next bolts
Partitioning Strategies

Shuffle grouping, which randomly partitions the tuples.

Fields grouping, which hashes on a subset of the tuple attributes/fields.

All grouping, which replicates the entire stream to all the consumer tasks.

Global grouping, which sends the entire stream to a single bolt.

Local grouping, which sends tuples to the consumer bolts in the same executor.
Storm Overview
Nimbus

responsible for distributing and coordinating the execution of the topology.
Nimbus cont.

- user sends topology as Apache Thrift object to Nimbus
- user code submitted as JAR file
- Nimbus stores topology on ZooKeeper and user code on local disk
supervisors advertise running topologies and vacancies to Nimbus every 15 sec

fail-fast and stateless

Nimbus w/ ZooKeeper & Supervisor

Nimbus

ZooKeeper

Supervisor

states
Supervisor

- runs on each storm node
- receives assignments from nimbus and starts workers
- also monitors health of workers
● responsible for managing changes in existing assignments
● downloads JAR files and libraries for the addition of new topologies
• reads worker heartbeats and classifies them as either valid, timed-out, not started or disallowed
Workers and Executors

- Executors are threads within the worker processes.
- An executor can run several tasks.
- A task is an instance of a spout or bolt.
- Tasks are strictly bound to their executors.
Workers

worker receive thread: listens on TCP/IP port for incoming tuples and puts them in the appropriate in-queue

worker send thread: examines each tuple in global transfer queue, sends it to next worker downstream based on its task destination identifier
Executors

User Logic Thread: takes incoming tuples from in-queue, runs actual task, and places outgoing tuples in out-queue

Executor Send Thread: takes tuples from out-queue and puts them in global transfer queue
message flow inside worker
Processing Semantics

Storm provides two semantics guarantees:

1. “at most once”
   - guarantees that a tuple is successfully processed or failed in each stage of the topology

2. “at least once”
   - no guarantee of tuple success or failure
Acker bolt is used to provide at least once semantics:

- Randomly generated 64-bit message id attached to each new tuple
- New tuples created by partitioning during tasks are assigned a new message id
- Backflow mechanism used to acknowledge tasks that contributed to output tuple
- Retires tuple once it reaches the spout that started tuple processing
XOR Implementation

- message ids are XORed and sent to the acker along with original tuple message id and timeout parameter
- when tuple processing is complete XORed message id and original id sent to acker bolt
- acker bolt locates original tuple and get its XOR checksum, then XORed again with acked tuple id
- if XOR checksum is zero acker knows tuple has been fully processed.
Possible Outputs

Acked - XOR checksum successfully goes to zero, hold dropped, tuple retired

Failed - ?

Neither - Timeout parameter alerts us, restart from last spout checkpoint
XOR Implementation cont.

Spout

Bolt
Experiment Setup

<table>
<thead>
<tr>
<th>Component</th>
<th># tasks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spout</td>
<td>200</td>
</tr>
<tr>
<td>DistributorBolt</td>
<td>200</td>
</tr>
<tr>
<td>UserCountBold</td>
<td>300</td>
</tr>
<tr>
<td>AggregatorBolt</td>
<td>20</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Time (relative to the start of the experiment)</th>
<th>#machines</th>
<th>#workers</th>
<th>Approximate #workers/machine</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 minutes</td>
<td>16</td>
<td>50</td>
<td>3</td>
</tr>
<tr>
<td>+15 minutes</td>
<td>13</td>
<td>50</td>
<td>4</td>
</tr>
<tr>
<td>+30 minutes</td>
<td>10</td>
<td>50</td>
<td>5</td>
</tr>
<tr>
<td>+45 minutes</td>
<td>7</td>
<td>50</td>
<td>7</td>
</tr>
<tr>
<td>+60 minutes</td>
<td>4</td>
<td>50</td>
<td>12</td>
</tr>
</tbody>
</table>
Results

# tuples processed by topology/minute
Operational Stories

*Overloaded Zookeeper* - less writes to zookeeper, tradeoff read consistency for high availability & write performance

*Storm Overheads* - Storm does not have more overhead than equivalent Java; add extra machines for business logic and tuple serialization costs

*Max Spout Tuning* - Number of tuples in flight value is set dynamically by algorithm for greatest throughput
Review

Storm@Twitter is...

● Scalable
● Resilient
● Extensible
● Efficient