Storm Applied

brief contents

1 ▪ Introducing Storm 1
2 ▪ Core Storm concepts 12
3 ▪ Topology design 33
4 ▪ Creating robust topologies 76
5 ▪ Moving from local to remote topologies 102
6 ▪ Tuning in Storm 130
7 ▪ Resource contention 161
8 ▪ Storm internals 187
9 ▪ Trident 207

1 *Introducing Storm 1*

1.1 What is big data? 2
The four Vs of big data 2 ▪ Big data tools 3
1.2 How Storm fits into the big data picture 6
Storm vs. the usual suspects 8
1.3 Why you'd want to use Storm 10
1.4 Summary 11

2 *Core Storm concepts* **12**

2.1 Problem definition: GitHub commit count dashboard 12
Data: starting and ending points 13
Breaking down the problem 14
2.2 Basic Storm concepts 14
Topology 15 ▪ Tuple 15 ▪ Stream 18 ▪ Spout 19
Bolt 20 ▪ Stream grouping 22
2.3 Implementing a GitHub commit count dashboard in Storm 24
Setting up a Storm project 25 ▪ Implementing the spout 25
Implementing the bolts 28 ▪ Wiring everything together to form the topology 31
2.4 Summary 32

3 *Topology design* **33**

3.1 Approaching topology design 34
3.2 Problem definition: a social heat map 34
Formation of a conceptual solution 35
3.3 Precepts for mapping the solution to Storm 35
Consider the requirements imposed by the data stream 36

Represent data points as tuples 37 **.** *Steps for determining the topology composition* *38*
3.4 Initial implementation of the design 40
Spout: read data from a source 41 **.** *Bolt: connect to an external service 42* **.** *Bolt: collect data in-memory* *44*
Bolt: persisting to a data store 48 **.** *Defining stream groupings between the components 51* **.** *Building a topology for running in local cluster mode* *51*
3.5 Scaling the topology 52
Understanding parallelism in Storm 54 **.** *Adjusting the topology to address bottlenecks inherent within design 58* **.** *Adjusting the topology to address bottlenecks inherent within a data stream* *64*
3.6 Topology design paradigms 69
Design by breakdown into functional components *70*
Design by breakdown into components at points of repartition *71*
Simplest functional components vs. lowest number of repartitions *74*
3.7 Summary 74

4 *Creating robust topologies* 76

4.1 Requirements for reliability 76
Pieces of the puzzle for supporting reliability *77*
4.2 Problem definition: a credit card authorization system 77
A conceptual solution with retry characteristics *78*
Defining the data points 79 **.** *Mapping the solution to Storm with retry characteristics* *80*

4.3 Basic implementation of the bolts 81
The AuthorizeCreditCard implementation *82*
The ProcessedOrderNotification implementation *83*
4.4 Guaranteed message processing 84
Tuple states: fully processed vs. failed 84 **.** *Anchoring, acking, and failing tuples in our bolts 86* **.** *A spout's role in guaranteed message processing* *90*
4.5 Replay semantics 94
Degrees of reliability in Storm 94 **.** *Examining exactly once processing in a Storm topology 95* **.** *Examining the reliability guarantees in our topology* *95*
4.6 Summary 101

5 *Moving from local to remote topologies* 102

5.1 The Storm cluster 103
The anatomy of a worker node 104 **.** *Presenting a worker node within the context of the credit card authorization topology* *106*
5.2 Fail-fast philosophy for fault tolerance within a Storm cluster 108
5.3 Installing a Storm cluster 109
Setting up a Zookeeper cluster 109 **.** *Installing the required Storm dependencies to master and worker nodes 110* **.** *Installing Storm to master and worker nodes 110* **.** *Configuring the master and worker nodes via storm.yaml 110* **.** *Launching Nimbus and*

Supervisors under supervision *111*
5.4 Getting your topology to run on a Storm cluster 112
Revisiting how to put together the topology components *112*
Running topologies in local mode 113 ▪ *Running topologies on a remote Storm cluster 114* ▪ *Deploying a topology to a remote Storm cluster* *114*
5.5 The Storm UI and its role in the Storm cluster 116
Storm UI: the Storm cluster summary 116 ▪ *Storm UI: individual Topology summary 120* ▪ *Storm UI: individual spout/bolt summary* *124*
5.6 Summary 129

6 ***Tuning in Storm*** ***130***

6.1 Problem definition: Daily Deals! reborn 131
Formation of a conceptual solution 132 ▪ *Mapping the solution to Storm concepts* *132*
6.2 Initial implementation 133
Spout: read from a data source 134 ▪ *Bolt: find recommended sales 135* ▪ *Bolt: look up details for each sale* *136*
Bolt: save recommended sales *138*
6.3 Tuning: I wanna go fast 139
The Storm UI: your go-to tool for tuning *139*
Establishing a baseline set of performance numbers *140*
Identifying bottlenecks 142 ▪ *Spouts: controlling the rate data flows into a topology* *145*
6.4 Latency: when external systems take their time 148
Simulating latency in your topology 148 ▪ *Extrinsic and intrinsic reasons for latency* *150*
6.5 Storm’s metrics-collecting API 154
Using Storm’s built-in CountMetric 154 ▪ *Setting up a metrics consumer 155* ▪ *Creating a custom SuccessRateMetric* *156*
Creating a custom MultiSuccessRateMetric *158*
6.6 Summary 160

7 ***Resource contention*** ***161***

7.1 Changing the number of worker processes running on a worker node 163
Problem 163 ▪ *Solution 164* ▪ *Discussion* *165*
7.2 Changing the amount of memory allocated to worker processes (JVMs) 165
Problem 165 ▪ *Solution 165* ▪ *Discussion* *166*
7.3 Figuring out which worker nodes/processes a topology is executing on 166
Problem 166 ▪ *Solution 166* ▪ *Discussion* *167*
7.4 Contention for worker processes in a Storm cluster 168
Problem 169 ▪ *Solution 170* ▪ *Discussion* *171*
7.5 Memory contention within a worker process (JVM) 171
Problem 174 ▪ *Solution 174* ▪ *Discussion* *175*

7.6 Memory contention on a worker node 175

Problem 178 ▪ Solution 178 ▪ Discussion 178

7.7 Worker node CPU contention 178

Problem 179 ▪ Solution 179 ▪ Discussion 181

7.8 Worker node I/O contention 181

Network/socket I/O contention 182 ▪ Disk I/O contention 184

7.9 Summary 186

8 ***Storm internals* 187**

8.1 The commit count topology revisited 188

Reviewing the topology design 188 ▪ Thinking of the topology as running on a remote Storm cluster 189 ▪ How data flows between the spout and bolts in the cluster 189

8.2 Diving into the details of an executor 191

Executor details for the commit feed listener spout 191
Transferring tuples between two executors on the same JVM 192
Executor details for the email extractor bolt 194 ▪ Transferring tuples between two executors on different JVMs 195 ▪ Executor details for the email counter bolt 197

8.3 Routing and tasks 198

8.4 Knowing when Storm’s internal queues overflow 200

The various types of internal queues and how they might overflow 200 ▪ Using Storm’s debug logs to diagnose buffer overflowing 201

8.5 Addressing internal Storm buffers overflowing 203

Adjust the production-to-consumption ratio 203 ▪ Increase the size of the buffer for all topologies 203 ▪ Increase the size of the buffer for a given topology 204 ▪ Max spout pending 205

8.6 Tweaking buffer sizes for performance gain 205

8.7 Summary 206

9 ***Trident* 207**

9.1 What is Trident? 208

The different types of Trident operations 210 ▪ Trident streams as a series of batches 211

9.2 Kafka and its role with Trident 212

Breaking down Kafka’s design 212 ▪ Kafka’s alignment with Trident 215

9.3 Problem definition: Internet radio 216

Defining the data points 217 ▪ Breaking down the problem into a series of steps 217

9.4 Implementing the internet radio design as a Trident topology 217

Implementing the spout with a Trident Kafka spout 219
Deserializing the play log and creating separate streams for each of the fields 220 ▪ Calculating and persisting the counts for artist, title, and tag 224

9.5 Accessing the persisted counts through DRPC 229

Creating a DRPC stream 230 ▪ Applying a DRPC state query to a stream 231 ▪ Making DRPC calls with a DRPC client 232

9.6 Mapping Trident operations to Storm primitives 233

9.7 Scaling a Trident topology 239
Partitions for parallelism 239 ▪ *Partitions in Trident streams 240*
9.8 Summary 243
afterword 244
index 247