Sparrow
Distributed Low-Latency Spark Scheduling

Kay Ousterhout, Patrick Wendell, Matei Zaharia, Ion Stoica

amplab
Outline

The Spark scheduling bottleneck

Sparrow's fully distributed, fault-tolerant technique

Sparrow's near-optimal performance
Spark Today

User 1

User 2

User 3

Spark Context

Query Compilation

Storage

Scheduling

Worker

Worker

Worker

Worker

Worker
Spark Today

User 1

User 2

User 3

Spark Context

Query Compilation

Storage

Scheduling

Worker

Worker

Worker

Worker

Worker
Job Latencies Rapidly Decreasing

2004: MapReduce batch job
2009: Hive query
2010: Dremel Query
2012: Impala query
2010: In-memory Spark query
2012: Impala query
2013: Spark streaming

10 min. 10 sec. 100 ms 1 ms

Job Latencies Rapidly Decreasing
Job latencies rapidly decreasing
Job latencies rapidly decreasing
+
Spark deployments growing in size

Scheduling bottleneck!
Spark scheduler throughput:

1500 tasks / second

Task Duration

- 10 second
- 1 second
- 100 ms

Cluster size (# 16-core machines)

- 1000
- 100
- 10
Optimizing the Spark Scheduler

0.8: Monitoring code moved off critical path

0.8.1: Result deserialization moved off critical path

Future improvements may yield 2-3x higher throughput
Is the scheduler the bottleneck in my cluster?

tinyurl.com/sparkdemo
Cluster Scheduler

Scheduler delay

Task launch

Worker
Worker
Worker
Worker
Worker

Task completion

tinyurl.com/sparkdemo
### Details for Stage 28

Total duration across all tasks: 16.7 m

### Summary Metrics for 1000 Completed Tasks

<table>
<thead>
<tr>
<th>Metric</th>
<th>Min</th>
<th>25th percentile</th>
<th>Median</th>
<th>75th percentile</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>Duration</td>
<td>1.0 s</td>
<td>1.0 s</td>
<td>1.0 s</td>
<td>1.0 s</td>
<td>1.0 s</td>
</tr>
<tr>
<td>Time spent fetching task results</td>
<td>0 ms</td>
<td>0 ms</td>
<td>0 ms</td>
<td>0 ms</td>
<td>0 ms</td>
</tr>
<tr>
<td>Scheduler delay</td>
<td>1 ms</td>
<td>3 ms</td>
<td>6 ms</td>
<td>43 ms</td>
<td>107 ms</td>
</tr>
</tbody>
</table>

### Tasks

<table>
<thead>
<tr>
<th>Task Index</th>
<th>Task ID</th>
<th>Status</th>
<th>Locality Level</th>
<th>Executor</th>
<th>Launch Time</th>
<th>Duration</th>
<th>GC Time</th>
<th>Errors</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>218003</td>
<td>SUCCESS</td>
<td>PROCESS_LOCAL</td>
<td>ip-10-181-131-16.ec2.internal</td>
<td>2013/12/02 21:54:59</td>
<td>1.0 s</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>218005</td>
<td>SUCCESS</td>
<td>PROCESS_LOCAL</td>
<td>ip-10-183-137-216.ec2.internal</td>
<td>2013/12/02 21:54:59</td>
<td>1.0 s</td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>218020</td>
<td>SUCCESS</td>
<td>PROCESS_LOCAL</td>
<td>ip-10-171-6-82.ec2.internal</td>
<td>2013/12/02 21:54:59</td>
<td>1.0 s</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>218006</td>
<td>SUCCESS</td>
<td>PROCESS_LOCAL</td>
<td>ip-10-29-137-86.ec2.internal</td>
<td>2013/12/02 21:54:59</td>
<td>1.0 s</td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>218019</td>
<td>SUCCESS</td>
<td>PROCESS_LOCAL</td>
<td>ip-10-232-26-201.ec2.internal</td>
<td>2013/12/02 21:54:59</td>
<td>1.0 s</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>218013</td>
<td>SUCCESS</td>
<td>PROCESS_LOCAL</td>
<td>ip-10-45-169-219.ec2.internal</td>
<td>2013/12/02 21:54:59</td>
<td>1.0 s</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>218009</td>
<td>SUCCESS</td>
<td>PROCESS_LOCAL</td>
<td>ip-10-45-171-84.ec2.internal</td>
<td>2013/12/02 21:54:59</td>
<td>1.0 s</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**Details for Stage 29**

Total duration across all tasks: **5.1 s**

### Summary Metrics for 1000 Completed Tasks

<table>
<thead>
<tr>
<th>Metric</th>
<th>Min</th>
<th>25th percentile</th>
<th>Median</th>
<th>75th percentile</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>Duration</td>
<td>5 ms</td>
<td>5 ms</td>
<td>5 ms</td>
<td>5 ms</td>
<td>6 ms</td>
</tr>
<tr>
<td>Time spent fetching task results</td>
<td>0 ms</td>
<td>0 ms</td>
<td>0 ms</td>
<td>0 ms</td>
<td>0 ms</td>
</tr>
<tr>
<td>Scheduler delay</td>
<td>19 ms</td>
<td>37 ms</td>
<td>49 ms</td>
<td>55 ms</td>
<td>98 ms</td>
</tr>
</tbody>
</table>

### Tasks

<table>
<thead>
<tr>
<th>Task Index</th>
<th>Task ID</th>
<th>Status</th>
<th>Locality Level</th>
<th>Executor</th>
<th>Launch Time</th>
<th>Duration</th>
<th>GC Time</th>
<th>Errors</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>219007</td>
<td>SUCCESS</td>
<td>PROCESS_LOCAL</td>
<td>ip-10-232-20-44.ec2.internal</td>
<td>2013/12/02 21:55:57</td>
<td>5 ms</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>219005</td>
<td>SUCCESS</td>
<td>PROCESS_LOCAL</td>
<td>ip-10-183-137-216.ec2.internal</td>
<td>2013/12/02 21:55:57</td>
<td>5 ms</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>219003</td>
<td>SUCCESS</td>
<td>PROCESS_LOCAL</td>
<td>ip-10-181-131-16.ec2.internal</td>
<td>2013/12/02 21:55:57</td>
<td>6 ms</td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>219016</td>
<td>SUCCESS</td>
<td>PROCESS_LOCAL</td>
<td>ip-10-45-166-166.ec2.internal</td>
<td>2013/12/02 21:55:57</td>
<td>5 ms</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>219004</td>
<td>SUCCESS</td>
<td>PROCESS_LOCAL</td>
<td>ip-10-181-216-68.ec2.internal</td>
<td>2013/12/02 21:55:57</td>
<td>5 ms</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>219013</td>
<td>SUCCESS</td>
<td>PROCESS_LOCAL</td>
<td>ip-10-45-166-219.ec2.internal</td>
<td>2013/12/02 21:55:57</td>
<td>5 ms</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>219012</td>
<td>SUCCESS</td>
<td>PROCESS_LOCAL</td>
<td>ip-10-232-28-213.ec2.internal</td>
<td>2013/12/02 21:55:57</td>
<td>6 ms</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>219006</td>
<td>SUCCESS</td>
<td>PROCESS_LOCAL</td>
<td>ip-10-20-137-88.ec2.internal</td>
<td>2013/12/02 21:55:57</td>
<td>5 ms</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Spark Today

User 1
User 2
User 3

Spark Context
Query Compilation
Storage
Scheduling

Worker
Worker
Worker
Worker
...
Worker
Future Spark

Benefits:
High throughput
Fault tolerance

User 1
Scheduler
Query compilation
Worker
Worker
Worker

User 2
Scheduler
Query compilation
Worker
Worker
Worker

User 3
Scheduler
Query compilation
Worker
Worker
...
Future Spark

User 1
Scheduler
Query compilation
Worker
Worker
Worker
Worker
Worker

User 2
Scheduler
Query compilation
Worker
Worker
Worker
Worker
Worker

User 3
Scheduler
Query compilation
Worker
Worker
Worker
Worker

Storage: Tachyon
Scheduling with Sparrow

Stage

Scheduler

Worker

Worker

Worker

Worker

Worker

Worker

Worker
Batch Sampling

Place $m$ tasks on the least loaded of $2m$ workers
Queue length poor predictor of wait time

Poor performance on heterogeneous workloads
Late Binding

Place $m$ tasks on the least loaded of $d \cdot m$ workers
Place $m$ tasks on the least loaded of $d \cdot m$ workers
Late Binding

Place $m$ tasks on the least loaded of $d \cdot m$ workers
What about constraints?
Per-Task Constraints

Probe separately for each task
Technique Recap

Batch sampling
+ Late binding
+ Constraints
How well does Sparrow perform?
How does Sparrow compare to Spark's native scheduler?

100 16-core EC2 nodes, 10 tasks/job, 10 schedulers, 80% load
TPC-H Queries: Background

TPC-H: Common benchmark for analytics workloads

- **Shark**: SQL execution engine
- **Spark**
- **Sparrow**
TPC-H Queries

Within 12% of ideal
Median queuing delay of 9ms

100 16-core EC2 nodes, 10 schedulers, 80% load
Policy Enforcement

Priorities
Serve queues based on strict priorities

High Priority
Low Priority

Fair Shares
Serve queues using weighted fair queuing

User A (75%)
User B (25%)
Weighted Fair Sharing

The graph shows the running tasks over time for two users, User 0 and User 1. The x-axis represents time in seconds, ranging from 0 to 50. The y-axis represents the number of running tasks, ranging from 0 to 400. The blue line represents User 0, and the red dotted line represents User 1. The graph demonstrates how tasks are distributed fairly and proportionally between the two users.
Fault Tolerance

Timeout: 100ms
Failover: 5ms
Re-launch queries: 15ms
Making Sparrow feature-complete

Interfacing with UI

Delay scheduling

Speculation
(1) Diagnosing a Spark scheduling bottleneck

(2) Distributed, fault-tolerant scheduling with Sparrow

www.github.com/radlab/sparrow