Sparrow
Distributed Low-Latency Scheduling

Kay Ousterhout, Patrick Wendell, Matei Zaharia, Ion Stoica
Sparrow schedules tasks in clusters using a decentralized, randomized approach. It supports constraints and fair sharing, and provides response times within 12% of ideal.
Scheduling Setting

Map Reduce/Spark/Dryad

Job

Task  Task  Task

...  ...

Map Reduce/Spark/Dryad

Job

Task  Task

...  ...

...
Job Latencies Rapidly Decreasing

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

10 min. 10 sec. 100 ms 1 ms
Scheduling challenges:

- Millisecond Latency
- Quality Placement
- Fault Tolerant
- High Throughput
2004: MapReduce batch job

2009: Hive query

2010: Dremel query

2010: Impala query

2012: Impala query

2013: Spark streaming

Scheduler throughput: 26 decisions/second

1.6K decisions/second

160K decisions/second

16M decisions/second

1000 16-core machines
Today: Completely Centralized

Sparrow: Completely Decentralized

Less centralization

✗ Millisecond Latency ✓
✓ Quality Placement ?
✗ Fault Tolerant ✓
✗ High Throughput ✓
Millisecond Latency
Quality Placement
Fault Tolerant
High Throughput

Today: Completely Centralized
Sparrow: Completely Decentralized

Less centralization

✗ Millisecond Latency ✓
✓ Quality Placement ✓
✗ Fault Tolerant ✓
✗ High Throughput ✓
Sparrow

Decentralized approach
Existing randomized approaches
Batch Sampling
Late Binding
Analytical performance evaluation

Handling constraints

Fairness and policy enforcement

Within 12% of ideal on 100 machines
Scheduling with Sparrow

Job ➔ Scheduler ➔ Scheduler ➔ Scheduler ➔ ...

Worker ➔ Worker ➔ Worker ➔ Worker ➔ ...

Diagram showing the flow of a job being scheduled across multiple schedulers and workers.
Random

Job -> Scheduler -> Worker

...
Simulated Results

100-task jobs in 10,000-node cluster, exp. task durations

Omniscient: infinitely fast centralized scheduler

Response Time (ms)

Load
Per-task sampling

Power of Two Choices
Per-task sampling

Job

Scheduler

:::

Scheduler

:::

Scheduler

Worker

Worker

Worker

Worker

Worker

Power of Two Choices
Per-task sampling

Scheduler

Scheduler

Scheduler

Scheduler

Worker

Worker

Worker

Worker

Power of Two Choices
Per-task sampling

Scheduler

Worker

Worker

Worker

Worker

Worker

Scheduler

Scheduler

Scheduler

Scheduler

Scheduler

Scheduler

Power of Two Choices
Simulated Results

100-task jobs in 10,000-node cluster, exp. task durations
Response Time Grows with Tasks/Job!

70% cluster load
Per-Task Sampling

Scheduler → Task 1 → Worker
Scheduler → Task 2 → Worker

Job → ... → Scheduler → Task 1 → Worker
Job → ... → Scheduler → Task 2 → Worker
Place $m$ tasks on the least loaded of $d \cdot m$ slaves
Per-task Sampling
Batch

Job

Scheduler

Scheduler

Scheduler

Scheduler

Worker

Worker

Worker

Worker

Worker

Worker

Worker

Worker

Place $m$ tasks on the least loaded of $d \cdot m$ slaves
Per-task versus Batch Sampling

Response Time (ms)

Tasks/Job

100

70% cluster load
Simulated Results

100-task jobs in 10,000-node cluster, exp. task durations
Queue length poor predictor of wait time

Poor performance on heterogeneous workloads
Place $m$ tasks on the least loaded of $d \cdot m$ slaves
Late Binding

Place \( m \) tasks on the least loaded of \( d \cdot m \) slaves
Late Binding

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

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

Place $m$ tasks on the least loaded of $d \cdot m$ slaves
Simulated Results

100-task jobs in 10,000-node cluster, exp. task durations
What about constraints?
Job Constraints

Restrict probed machines to those that satisfy the constraint
Per-Task Constraints

Probe separately for each task
Technique Recap

Batch sampling + Late binding + Constraint handling
How does Sparrow perform on a real cluster?
Spark on Sparrow

Query: DAG of Stages

Sparrow Scheduler

Worker
Worker
Worker
Worker
Worker
Worker
Spark on Sparrow

Query: DAG of Stages

Sparrow Scheduler

Worker

Worker

Worker

Worker

Worker
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**: Distributed in-memory analytics framework

**Sparrow**
TPC-H Queries

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

Random
Per-task sampling
Batch sampling
Batch + late binding

Response Time (ms)

q3 q4 q6 q12

Percentiles

95 75 50 25 5

4217 (med.) 5396 (med.) 7881 (med.)
TPC-H Queries

Within 12% of ideal
Median queuing delay of 9ms

100 16-core EC2 nodes, 10 schedulers, 80% load
Fault Tolerance

Timeout: 100ms
Failover: 5ms
Re-launch queries: 15ms
When does Sparrow not work as well?

High cluster load

Response Time (ms)

Load

Sparrow
Omniscient
Related Work

Centralized task schedulers: e.g., Quincy

Two level schedulers: e.g., YARN, Mesos

Coarse-grained cluster schedulers: e.g., Omega

Load balancing: single task
Sparrows provides near-ideal job response times without global visibility

www.github.com/radlab/sparrow
Backup Slides
Can we do better without losing simplicity?