Sparklint

a Tool for Identifying and Tuning Inefficient Spark Jobs Across Your Cluster

Simon Whitear

Principal Engineer @ Groupon

Why Sparklint?

- A successful Spark cluster grows rapidly
- Capacity and capability mismatches arise
- Leads to resource contention
- Tuning process is non-trivial
- Current UI operational in focus

We wanted to understand application efficiency

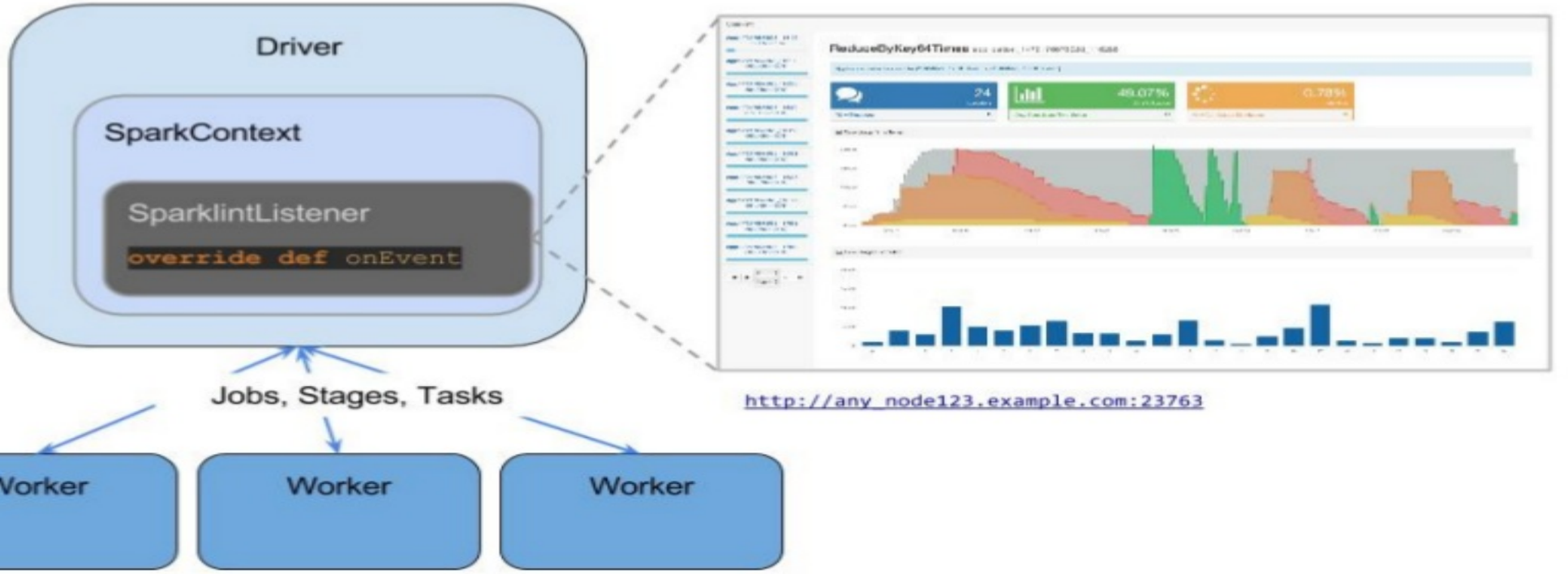
Sparklint provides:

- Live view of batch & streaming application stats

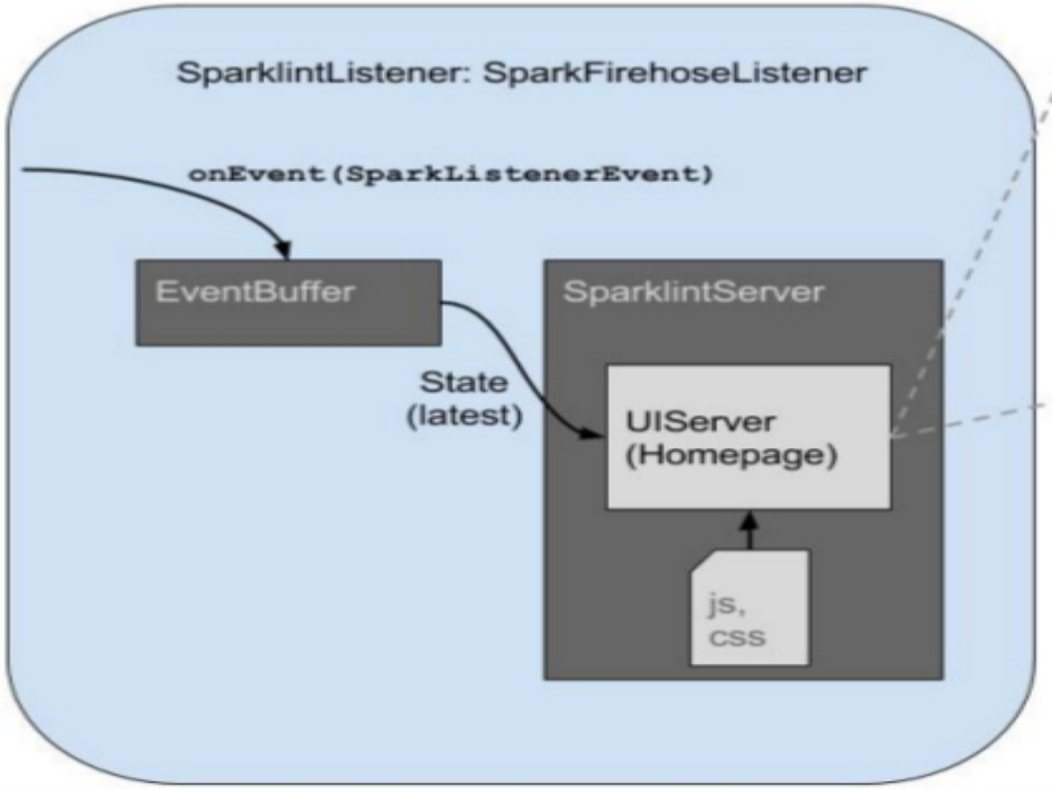
  or

- Event by event analysis of historical event logs
- Stats and graphs for:
  - Idle time
  - Core usage
  - Task locality

Sparklint Listener:

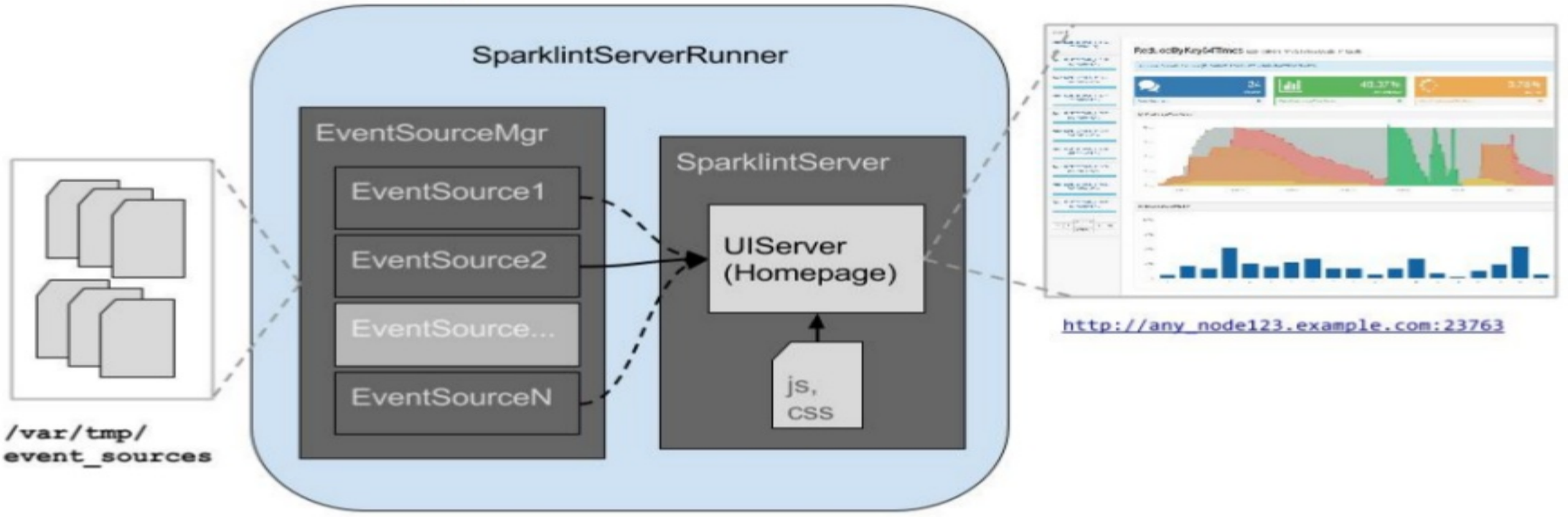


Sparklint Listener:



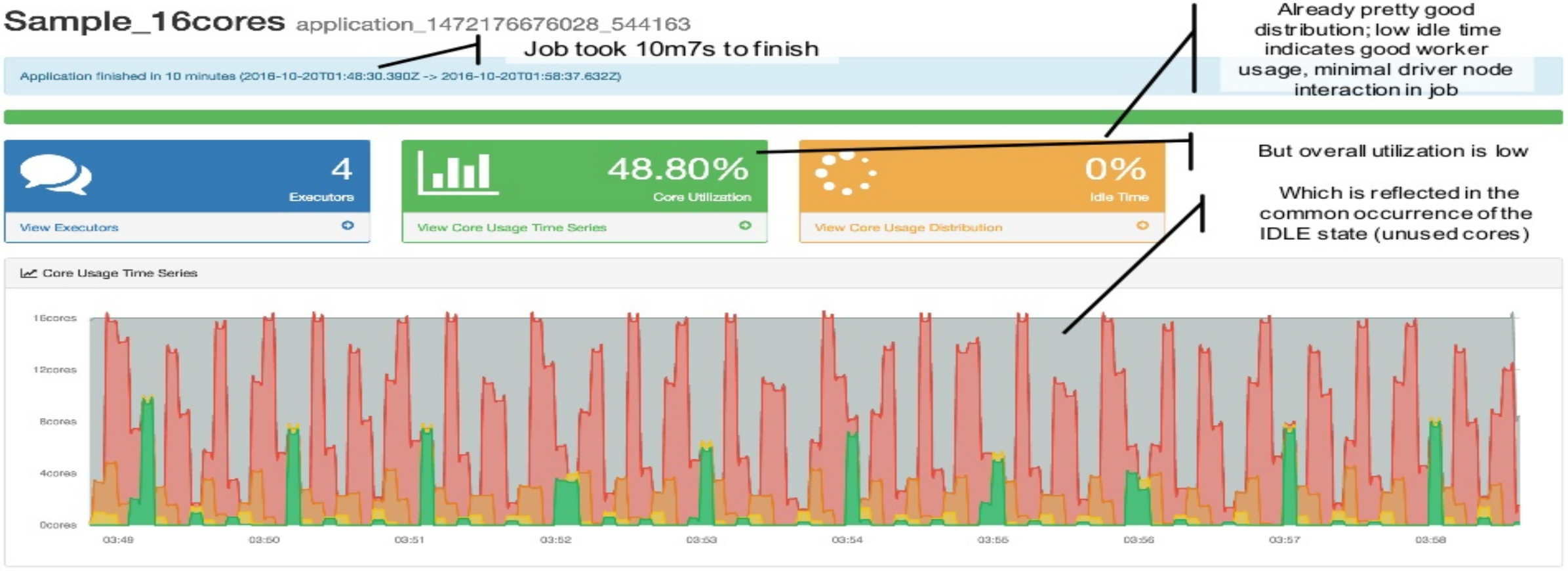
http://any_node123.example.com:23763

Sparklint Server:



Demo…

- Simulated workload analyzing site access logs:
  - read text file as JSON
  - convert to Record(ip, verb, status, time)
  - countByIp, countByStatus, countByVerb


Sample_16cores application_1472176676028_544163
Job took 10m7s to finish
Application finished in 10 minutes (2016-10-20T01:48:30.390Z -> 2016-10-20T01:58:37.632Z)
Already pretty good distribution; low idle time indicates good worker usage, minimal driver node interaction in job
4
Executors
View Executors
48.80%
Core Utilization
View Core Usage Time Series
0%
Idle Time
View Core Usage Distribution
But overall utilization is low
Which is reflected in the common occurrence of the IDLE state (unused cores)
Core Usage Time Series
16cores
12cores
8cores
4cores
0cores
03:49
03:50
03:51
03:52
03:53
03:54
03:55
03:56
03:57
03:58

Sample_8cores application_1472176676028_544172

Job took 15m 14s to finish

Core usage increased, job is more efficient, execution time increased, but the app is not cpu bound

Application finished in 15 minutes (2016-10-20T01:49:17.941Z -> 2016-10-20T02:04:31.009Z)

2 Executors

View Executors

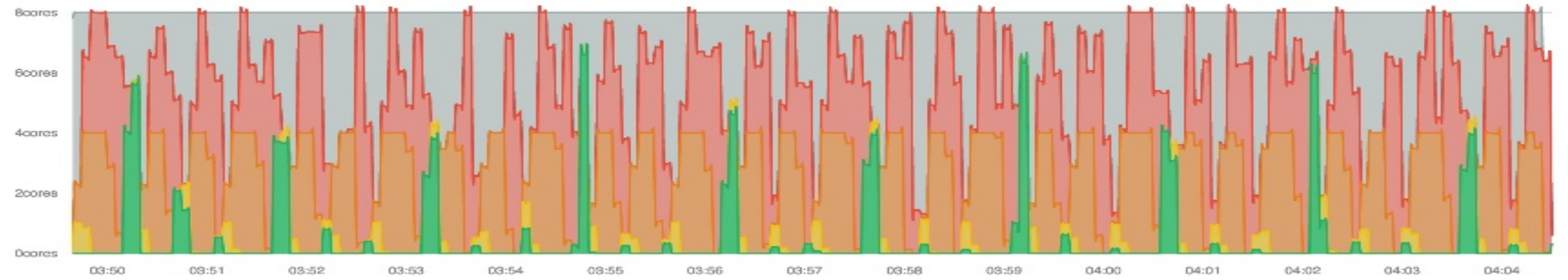
67.19% Core Utilization

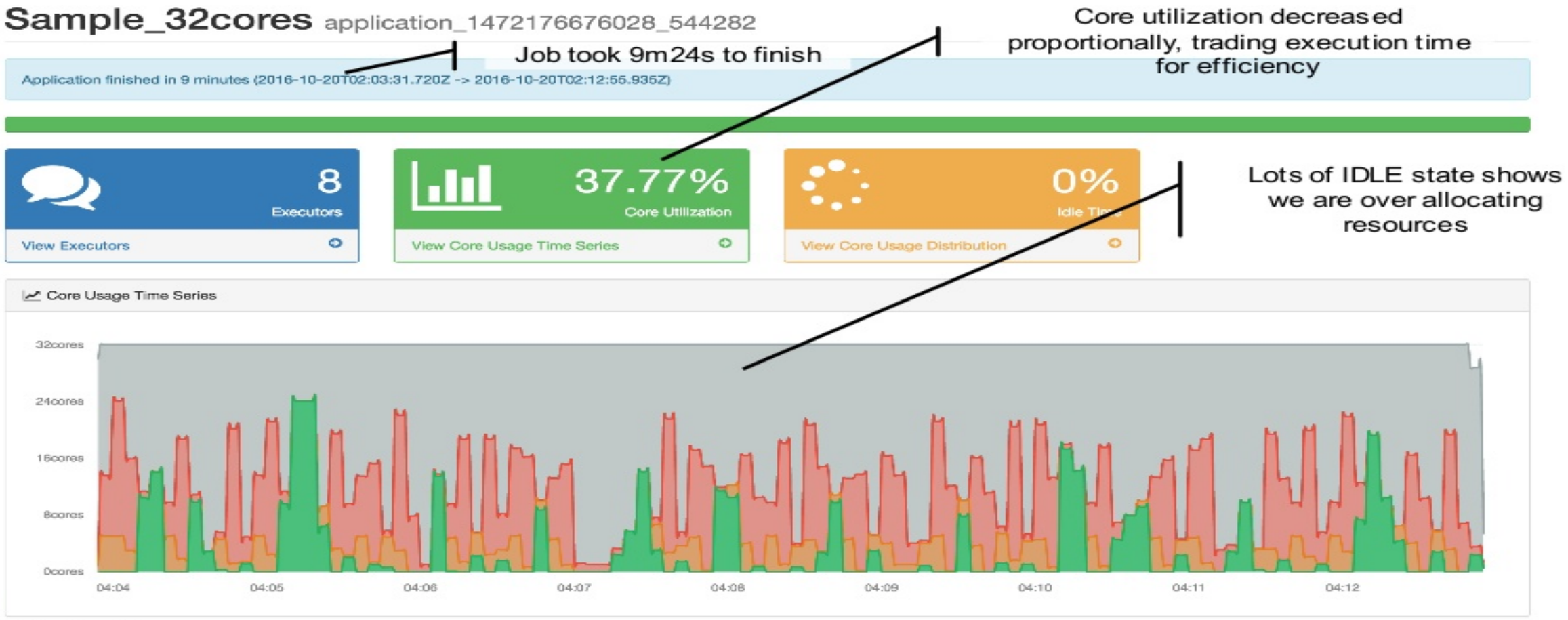
View Core Usage Time Series

0% Idle Time

View Core Usage Distribution

Core Usage Time Series




Sample_32cores application_1472176676028_544282
Job took 9m24s to finish
Core utilization decreased proportionally, trading execution time for efficiency
Application finished in 9 minutes (2016-10-20T02:03:31.720Z -> 2016-10-20T02:12:55.935Z)
8
Executors
View Executors
37.77%
Core Utilization
View Core Usage Time Series
0%
Idle Time
View Core Usage Distribution
Lots of IDLE state shows we are over allocating resources
Core Usage Time Series
32cores
24cores
16cores
8cores
0cores
04:04
04:05
04:06
04:07
04:08
04:09
04:10
04:11
04:12

Sample_DynamicAllocation application_1472176676028_544292

Job took 11m34s to finish

Core utilization remains low, the config settings are not right for this workload.

Application finished in 12 minutes (2016-10-20T02:05:58.773Z -> 2016-10-20T02:17:32.291Z)

24 Executors

View Executors

41.02% Core Utilization

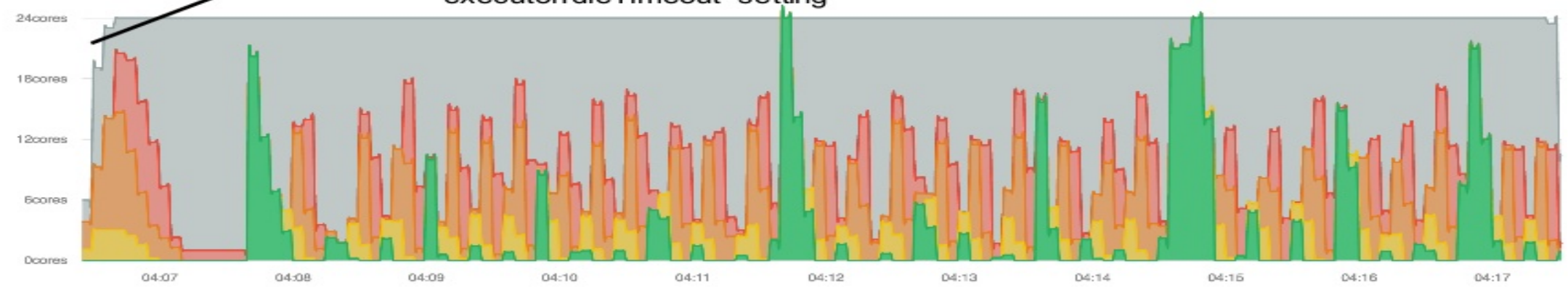
View Core Usage Time Series

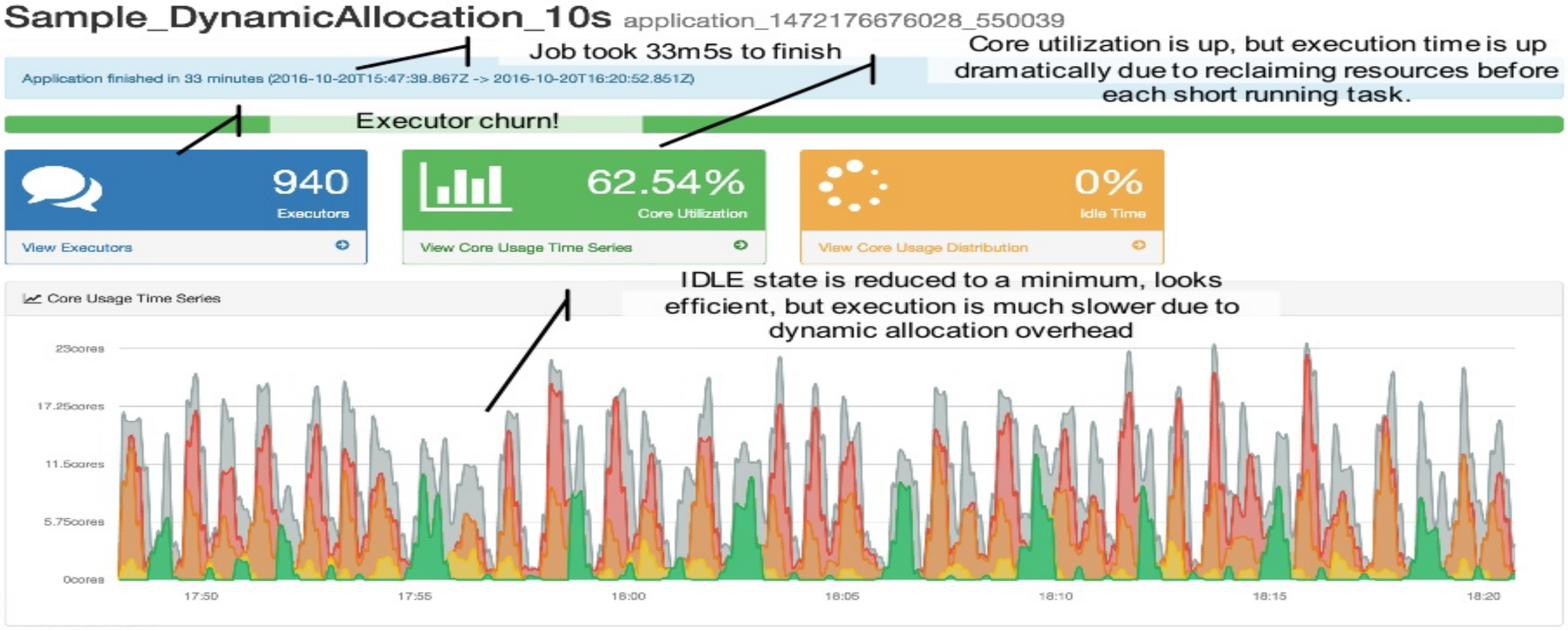
0% Idle Time

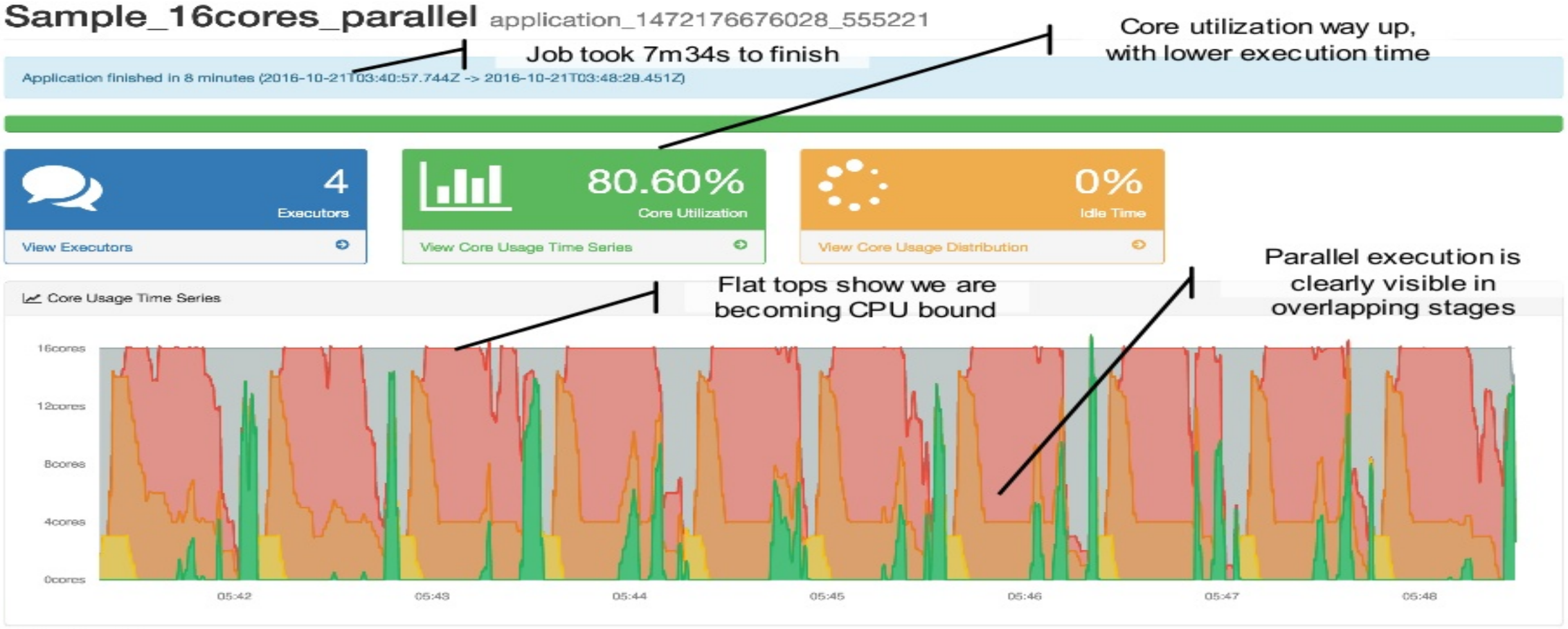
View Core Usage Distribution

Dynamic allocation only effective at app start due to long executorIdleTimeout setting

Core Usage Time Series




Sample_DynamicAllocation_10s application_1472176676028_550039
Job took 33m5s to finish
Application finished in 33 minutes (2016-10-20T15:47:39.867Z -> 2016-10-20T16:20:52.851Z)
Core utilization is up, but execution time is up dramatically due to reclaiming resources before each short running task.
Executor churn!
940
Executors
View Executors
62.54%
Core Utilization
View Core Usage Time Series
0%
Idle Time
View Core Usage Distribution
IDLE state is reduced to a minimum, looks efficient, but execution is much slower due to dynamic allocation overhead
Core Usage Time Series
23cores
17.25cores
11.5cores
5.75cores
0cores
17:50
17:55
18:00
18:05
18:10
18:15
18:20


Sample_16cores_parallel application_1472176676028_555221
Job took 7m34s to finish
Core utilization way up, with lower execution time
Application finished in 8 minutes (2016-10-21T03:40:57.744Z -> 2016-10-21T03:48:29.451Z)
4
Executors
View Executors
80.60%
Core Utilization
View Core Usage Time Series
0%
Idle Time
View Core Usage Distribution
Parallel execution is clearly visible in overlapping stages
Flat tops show we are becoming CPU bound
Core Usage Time Series
16cores
12cores
8cores
4cores
0cores
05:42
05:43
05:44
05:45
05:46
05:47
05:48

Sample_32cores_parallel application_1472176676028_555209

Job took 5m6s to finish

Core utilization decreases, trading execution time for efficiency again here

Application finished in 5 minutes (2016-10-21T03:39:10.199Z -> 2016-10-21T03:44:16.894Z)

8 Executors

View Executors

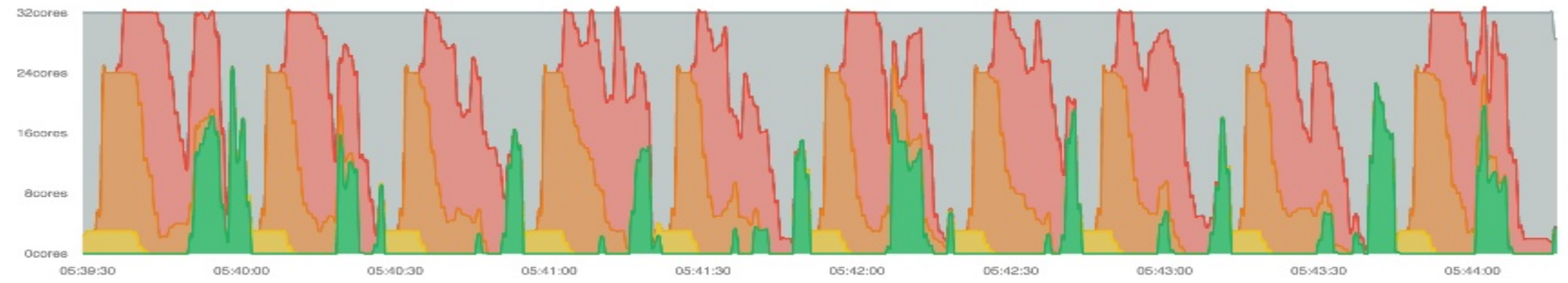
60.74% Core Utilization

View Core Usage Time Series

0% Idle Time

View Core Usage Distribution

Core Usage Time Series



junoHourly-2016-10-05:21 application_1472176676028_407508

Application finished in 43 minutes (2016-10-05T22:11:39.283Z -> 2016-10-05T22:54:09.709Z)

Thanks to dynamic allocation the utilization is high despite being a bi-modal application

58 Executors

View Executors

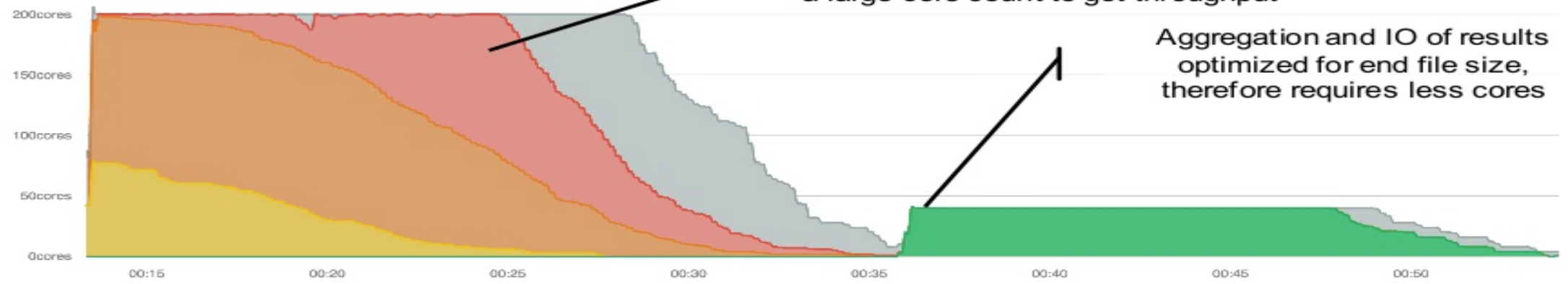
82.02% Core Utilization

View Core Usage Time Series

0.39% Idle Time

View Core Usage Distribution

Core Usage Time Series



Future Features:

- Increased job & stage detail in UI
- History Server event sources
- Inline recommendations
- Auto-tuning
- Streaming stage parameter delegation

The Credit:

- Lead developer is Robert Xue
- https://github.com/roboxue
- SDE @ Groupon

Contribute!

Sparklint is OSS:

https://github.com/groupon/sparklint

THANK YOU.

swhitear@groupon.com