Sparklint

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

Simon Whitear

Principal Engineer @ Groupon

SPARK SUMMIT
EUROPE 2016

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

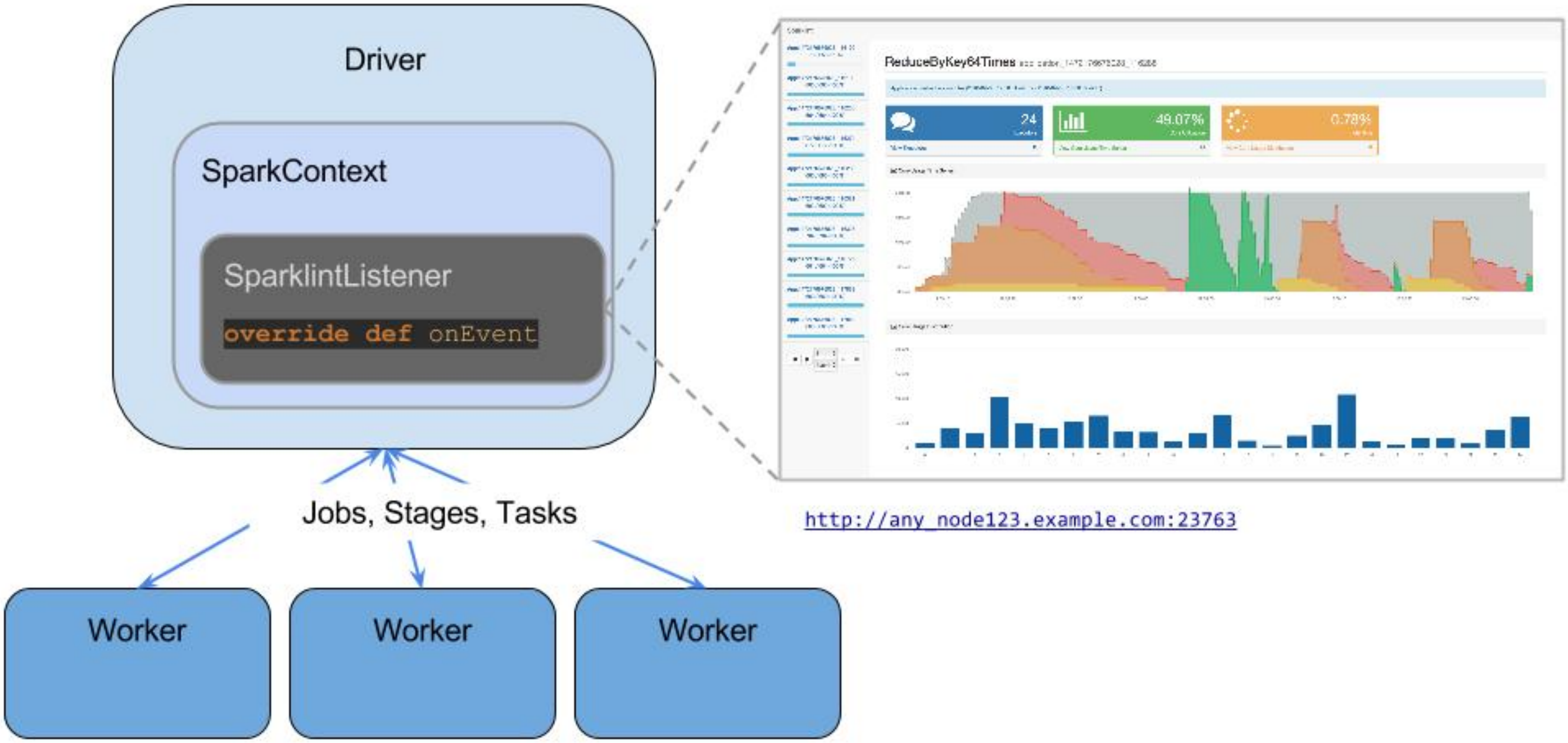
Sparklint provides:

- 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

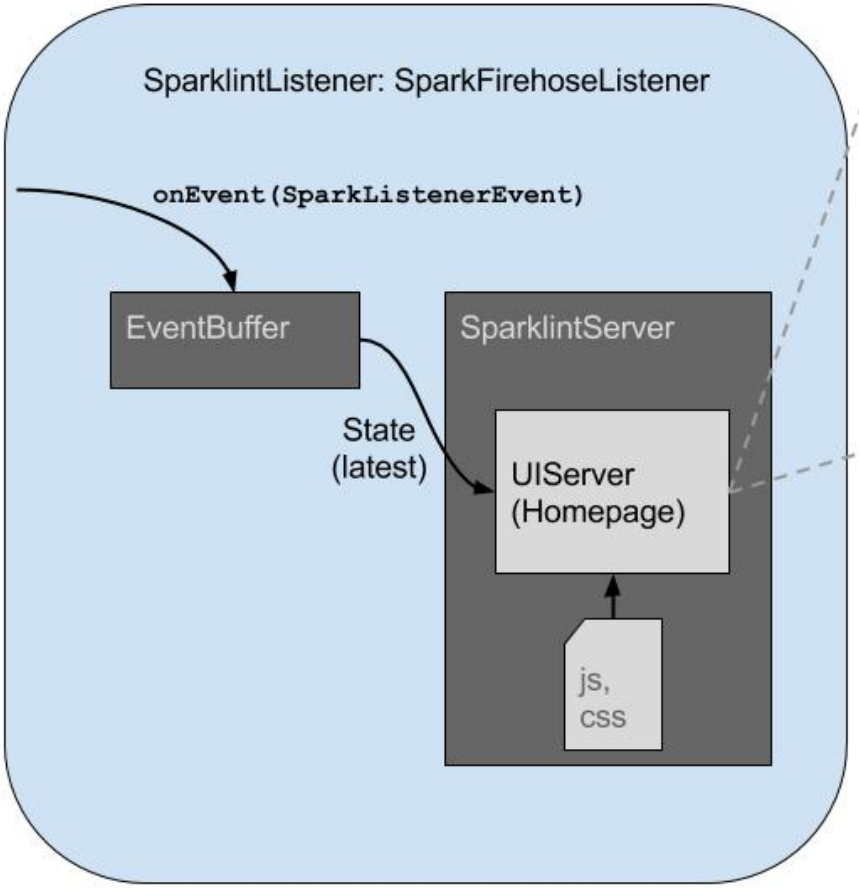
Sparklint Listener:



```
override def onEvent
```

http://any_node123.example.com:23763

Sparklint Listener:



SparklintListener: SparkFirehoseListener

`onEvent(SparkListenerEvent)`

EventBuffer

SparklintServer

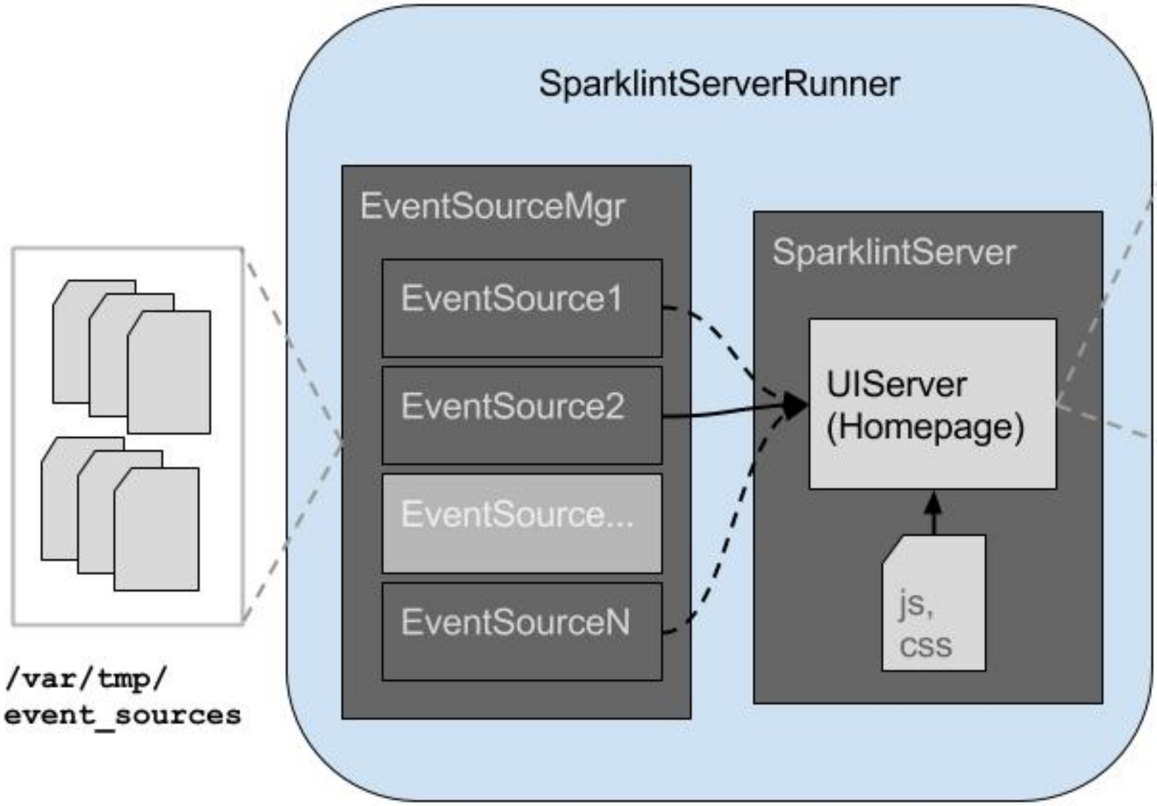
State (latest)

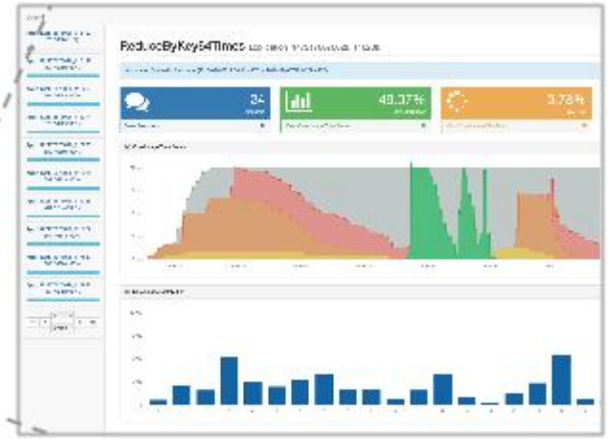
UIServer (Homepage)

js, css

http://any_node123.example.com:23763

Sparklint Server:

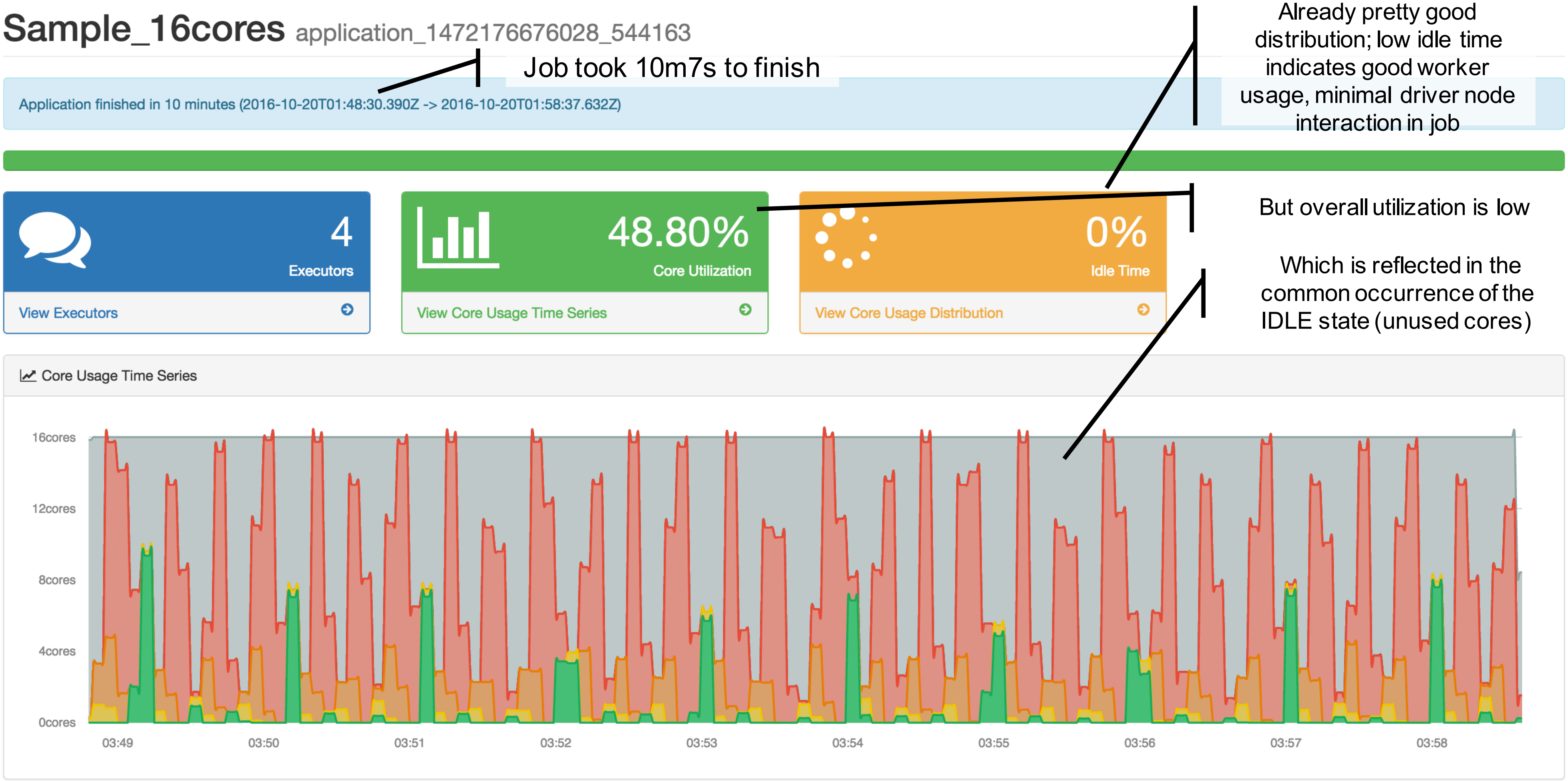




http://any_node123.example.com:23763

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

Application finished in 10 minutes (2016-10-20T01:48:30.390Z -> 2016-10-20T01:58:37.632Z)

4 Executors

View Executors

48.80% Core Utilization

View Core Usage Time Series

0% Idle Time

View Core Usage Distribution

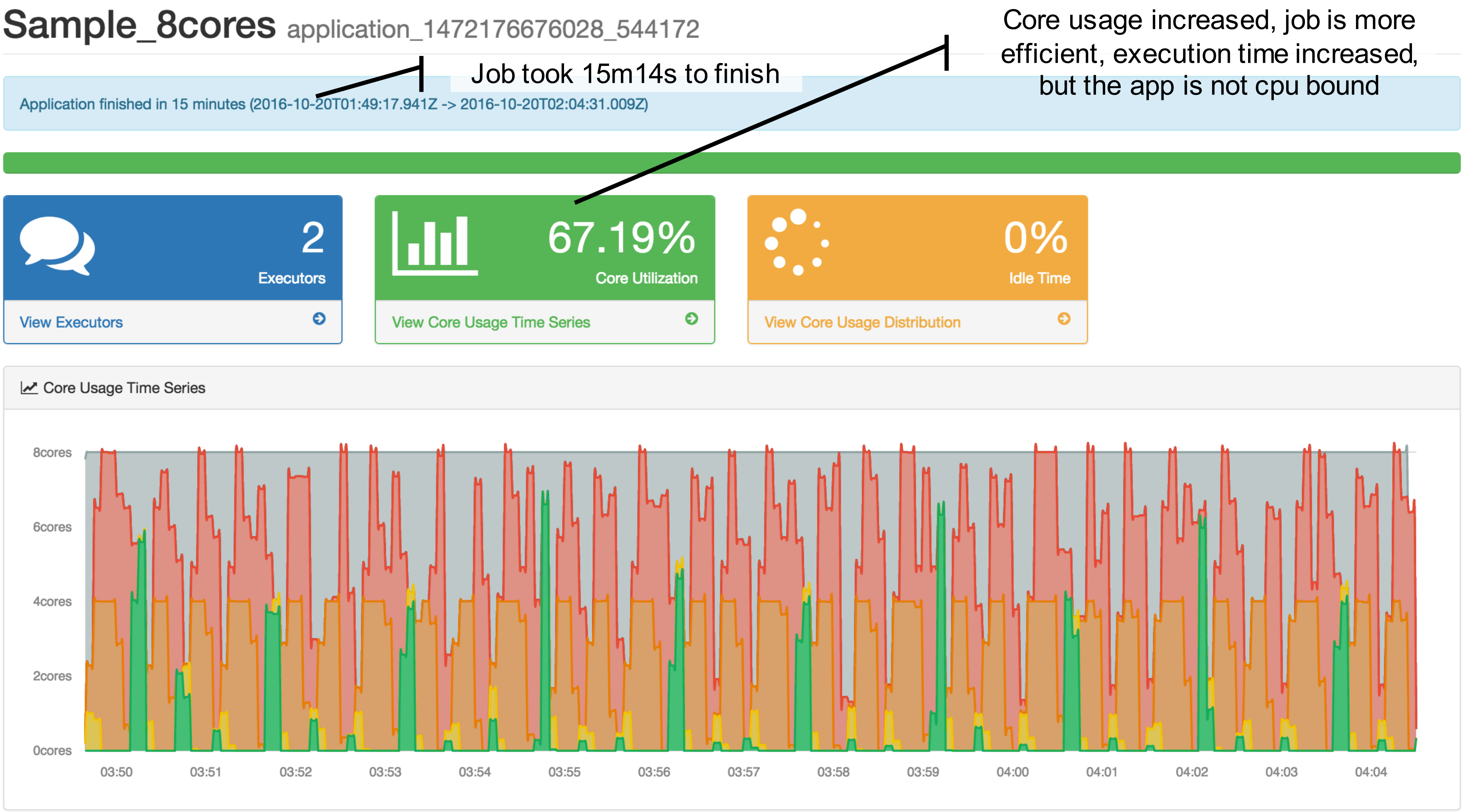
Core Usage Time Series

Job took 10m7s to finish

Already pretty good distribution; low idle time indicates good worker usage, minimal driver node interaction in job

But overall utilization is low

Which is reflected in the common occurrence of the IDLE state (unused cores)



Sample_8cores application_1472176676028_544172

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

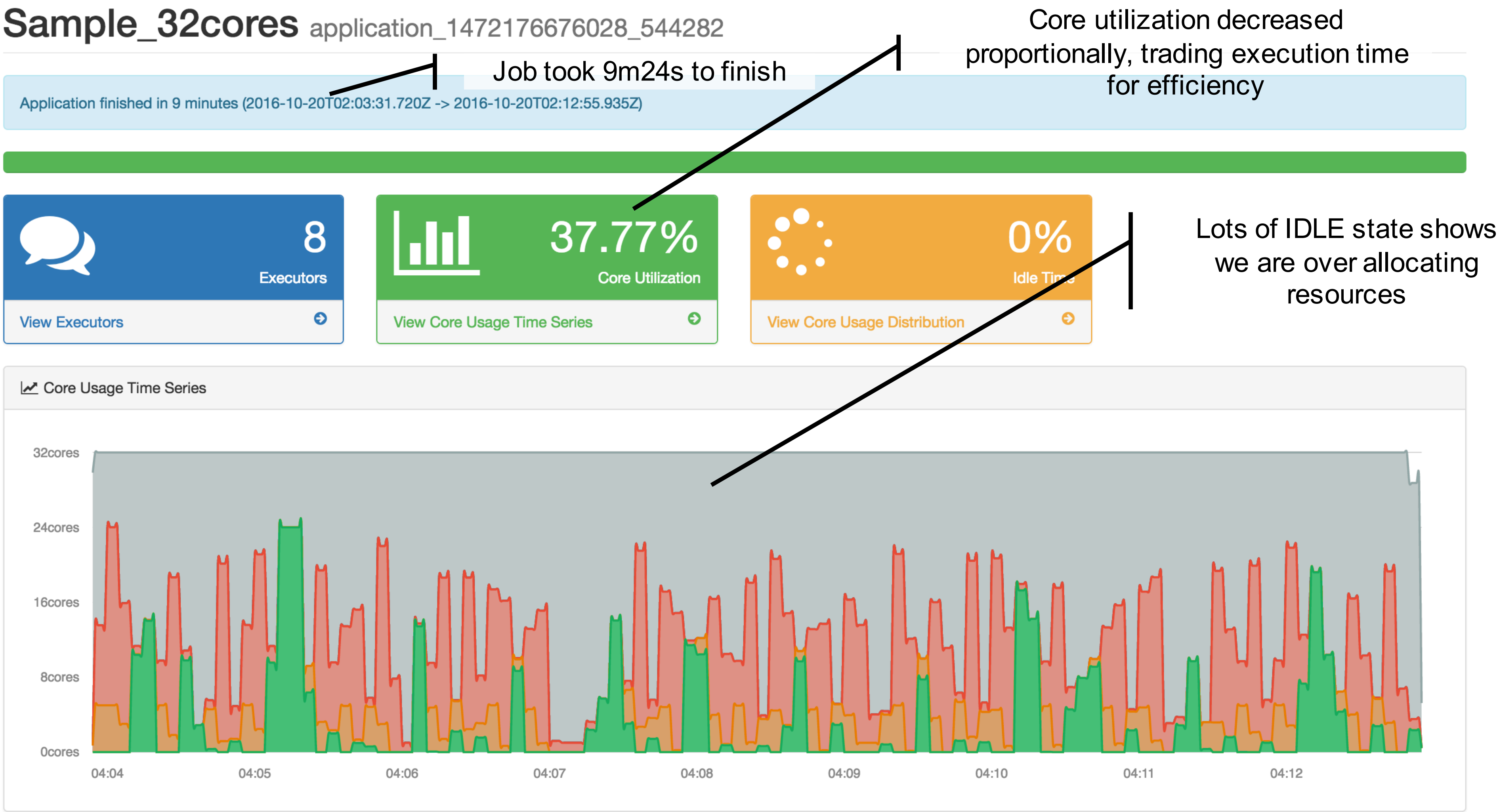
0% Idle Time

View Core Usage Distribution

Core Usage Time Series

Job took 15m14s to finish

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



Sample_32cores application_1472176676028_544282

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

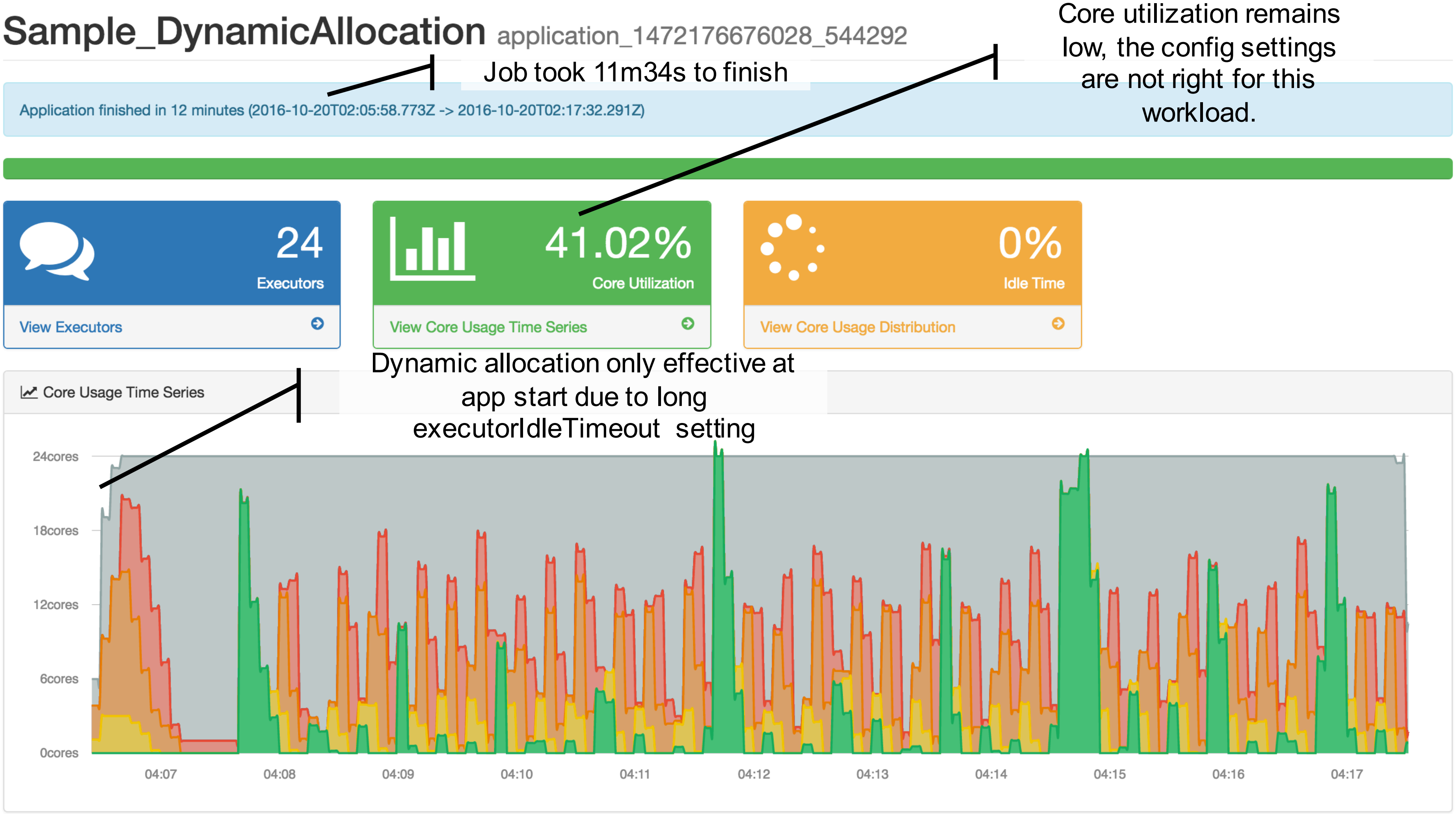
0%
Idle Time
View Core Usage Distribution

Core Usage Time Series

Job took 9m24s to finish

Core utilization decreased proportionally, trading execution time for efficiency

Lots of IDLE state shows we are over allocating resources



Sample_DynamicAllocation application_1472176676028_544292

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

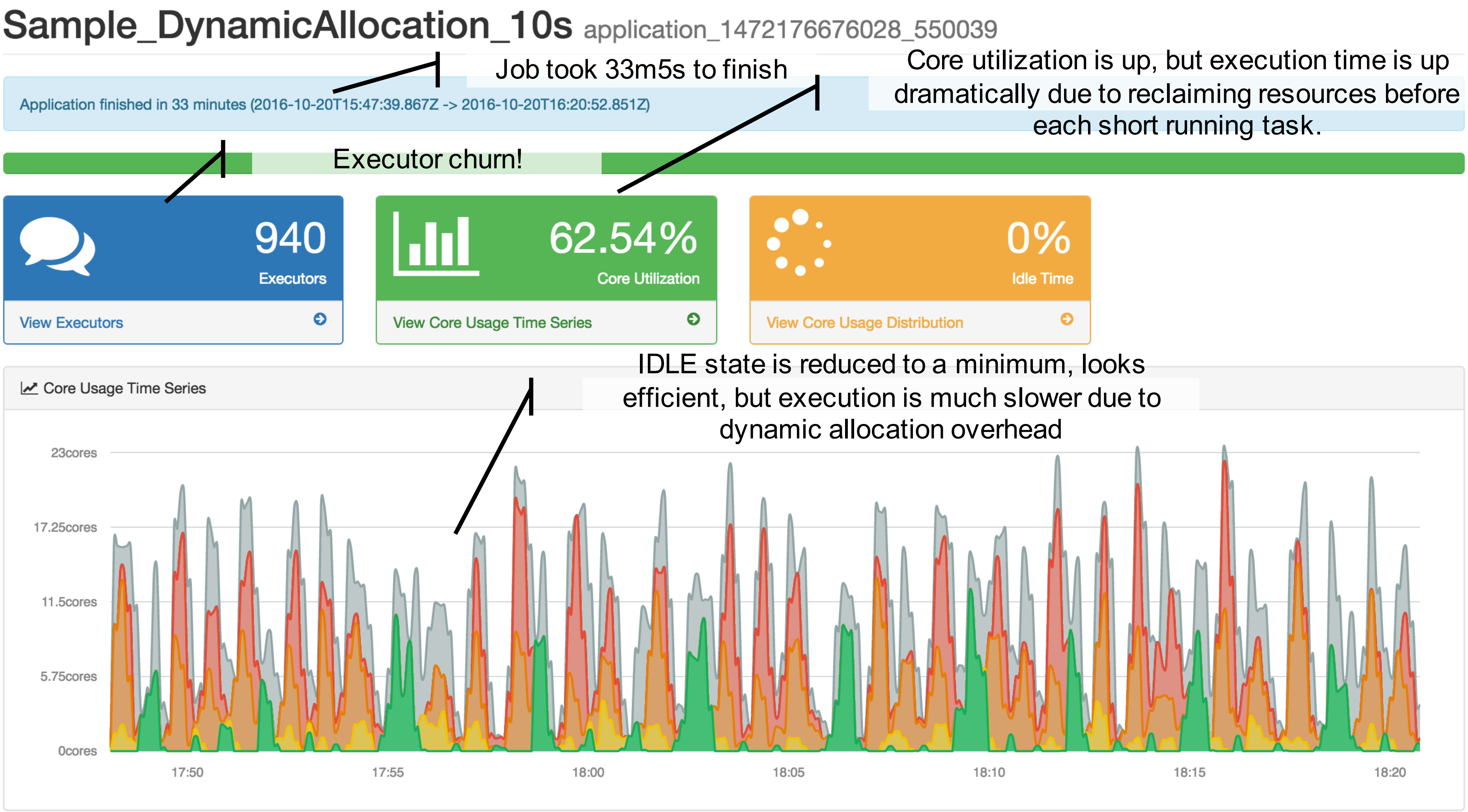
View Core Usage Distribution

Core Usage Time Series

Job took 11m34s to finish

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

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



Sample_DynamicAllocation_10s application_1472176676028_550039

Application finished in 33 minutes (2016-10-20T15:47:39.867Z -> 2016-10-20T16:20:52.851Z)

940 Executors

View Executors

62.54% Core Utilization

View Core Usage Time Series

0% Idle Time

View Core Usage Distribution

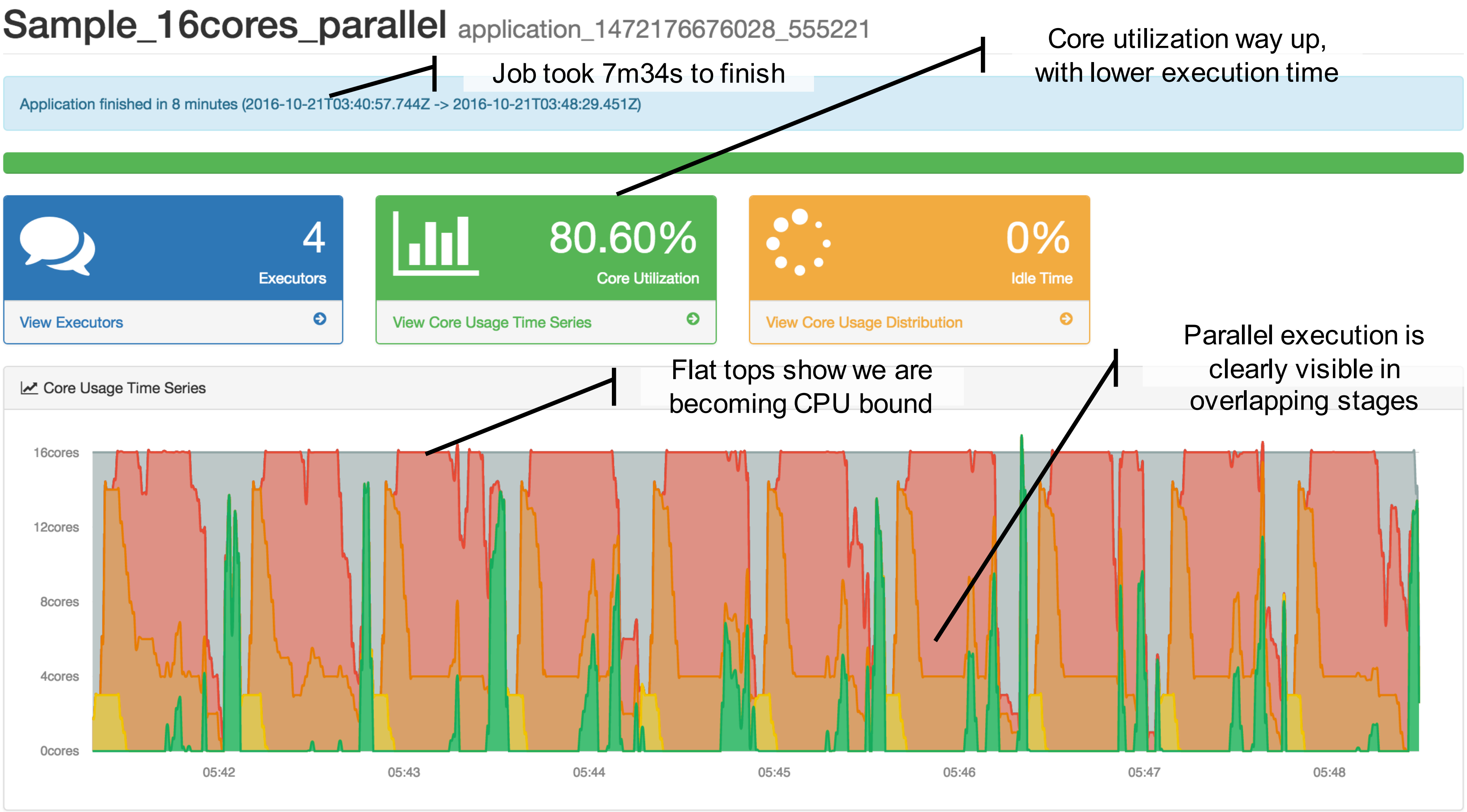
Core Usage Time Series

Job took 33m5s to finish

Core utilization is up, but execution time is up dramatically due to reclaiming resources before each short running task.

Executor churn!

IDLE state is reduced to a minimum, looks efficient, but execution is much slower due to dynamic allocation overhead



Sample_16cores_parallel application_1472176676028_555221

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

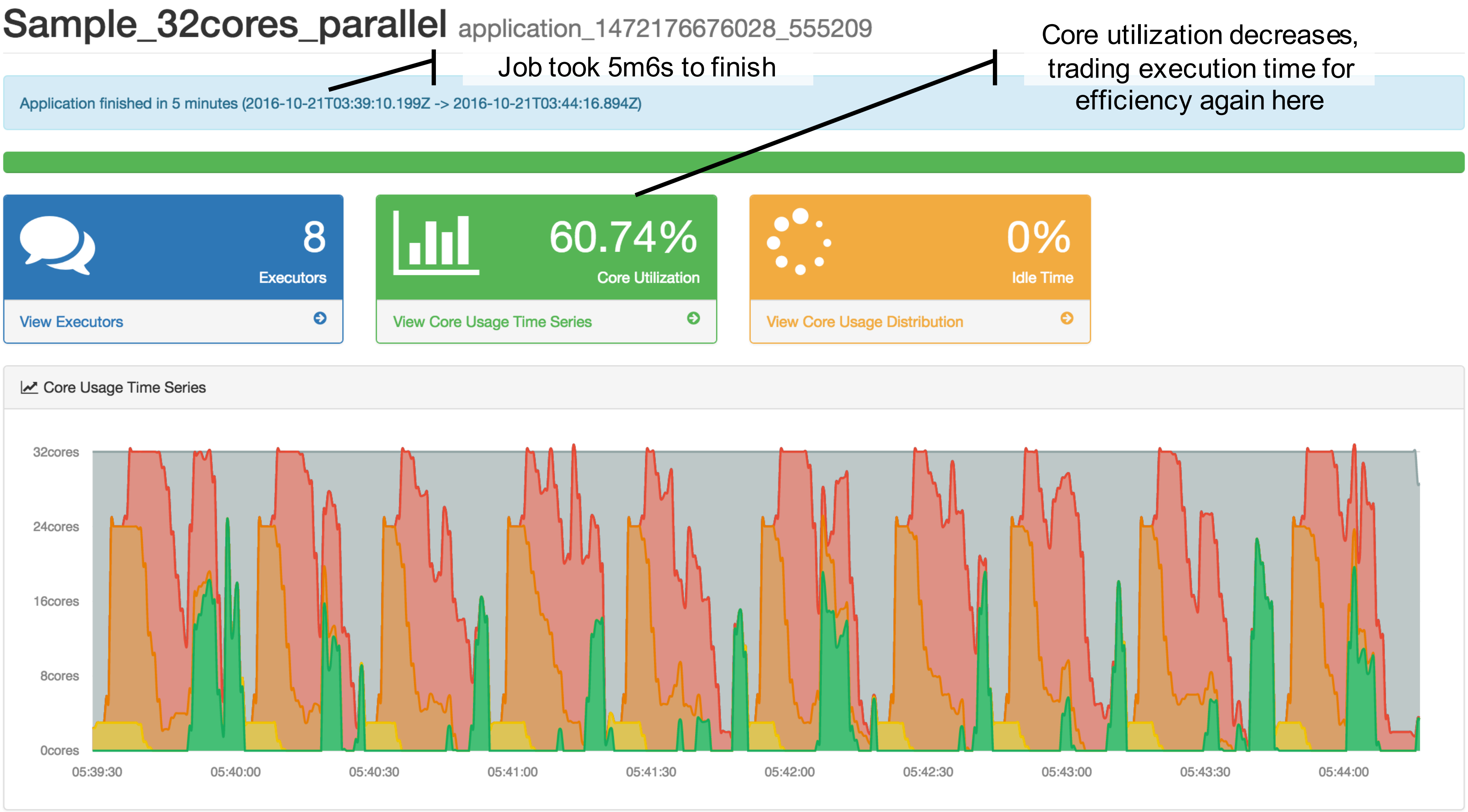
Core Usage Time Series

Job took 7m34s to finish

Core utilization way up, with lower execution time

Flat tops show we are becoming CPU bound

Parallel execution is clearly visible in overlapping stages



Sample_32cores_parallel application_1472176676028_555209

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

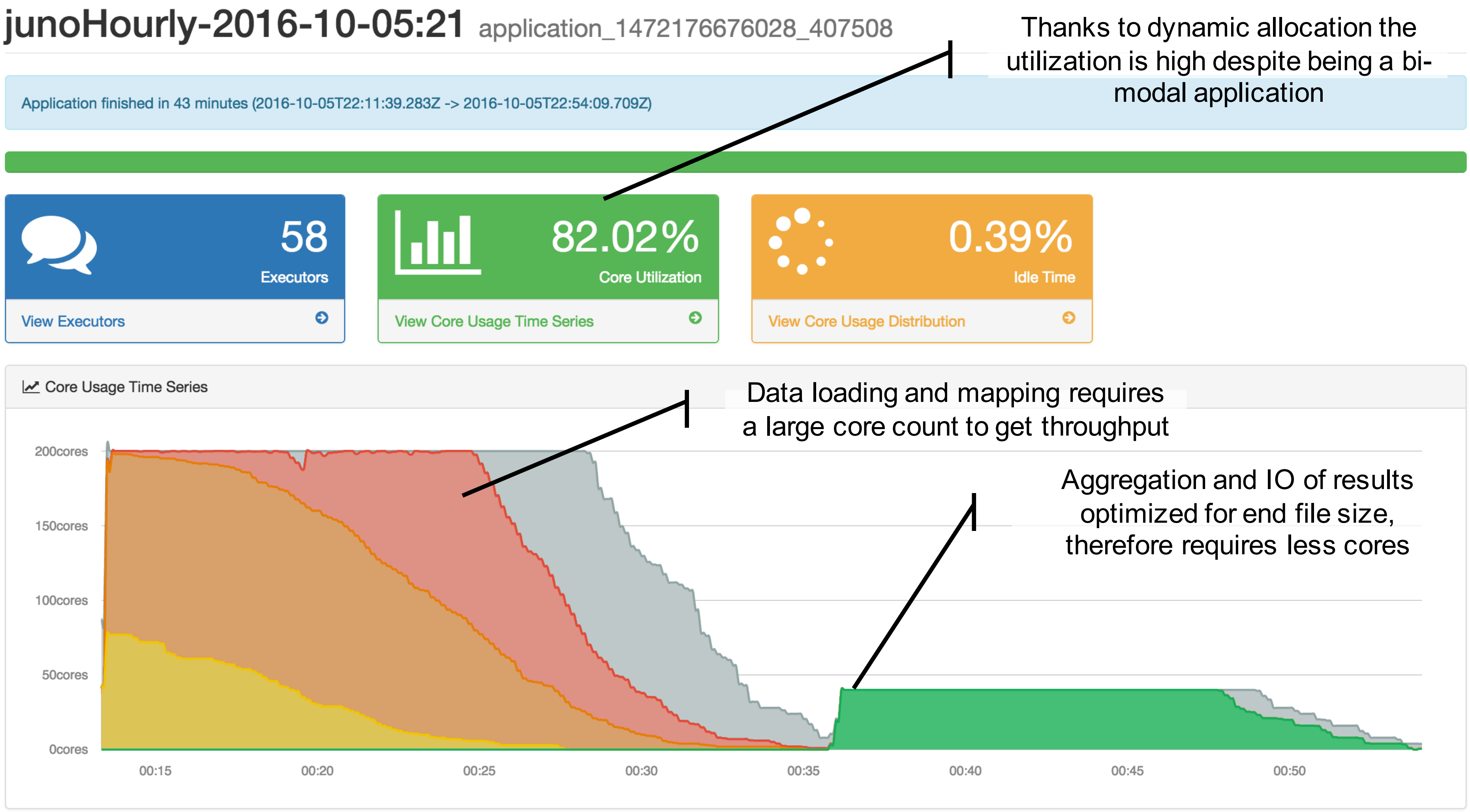
View Core Usage Distribution

Core Usage Time Series

Job took 5m6s to finish

Core utilization decreases, trading execution time for efficiency again here

SPARK SUMMIT EUROPE 2016



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)

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

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

Data loading and mapping requires a large core count to get throughput

Aggregation and IO of results optimized for end file size, therefore requires less cores

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