SPARK SUMMIT 2017

Scaling Spark applications by connecting code to resource consumption

Vinod Nair

Director of Product Management @ Pepperdata

Pepperdata does performance (for Big Data)

18

Thousan d

Production

50

Million

Jobs/Year

20 0

Trillion

Today's talk will cover…

- Why debugging performance problems is hard
- Data elements needed for a complete view of application performance from separate tools
- Bringing these elements together in a single tool

2 reasons why debugging performance problems is hard

Reason # 1

Same external symptoms, but many possible causes

- code
- data
- configuration
- cluster weather

Reason #2

Existing tools provide limited visibility

- Spark Web UI is the most popular
  - Execution plan with some aggregate performance data
- Ganglia, Ambari, CM etc
  - Time series data about cluster, not specific to Spark apps
- Code execution not connected to resource consumption
- Unhealthy hardware or load from other apps unaccounted

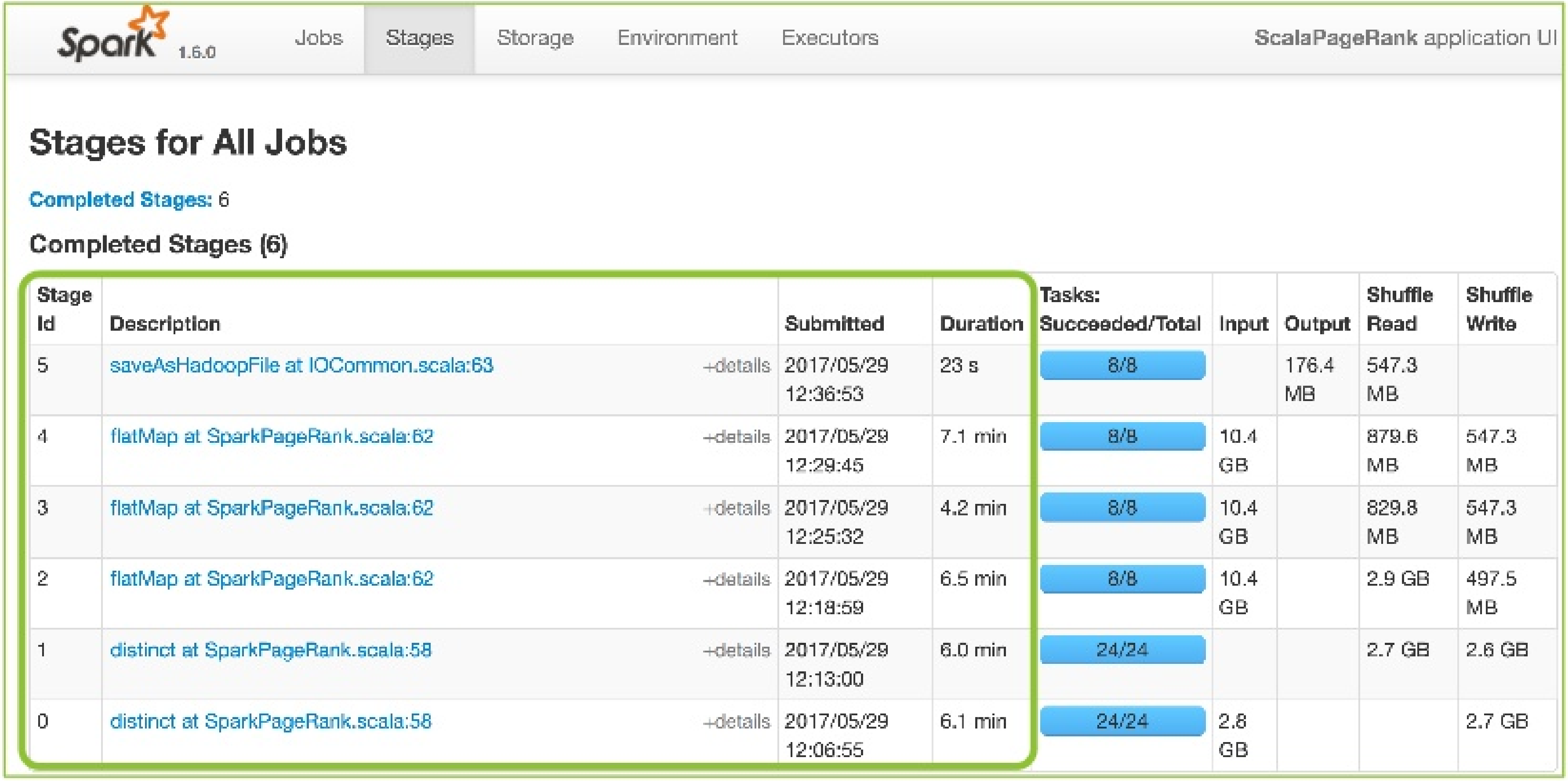
3 data elements form a complete picture of Spark application performance

1. Code execution plan
   - Indicates which block of code is being executed
2. Time series view
   - Visual of resource consumption of application
   - Outliers in resource use very easy to detect
3. Cluster weather
   - A view of all applications that runs on the cluster
   - A view of the health of all the nodes in the cluster

First half of the solution

Spark Web UI

Logical code execution plan from Spark: Jobs / Stages / DAG



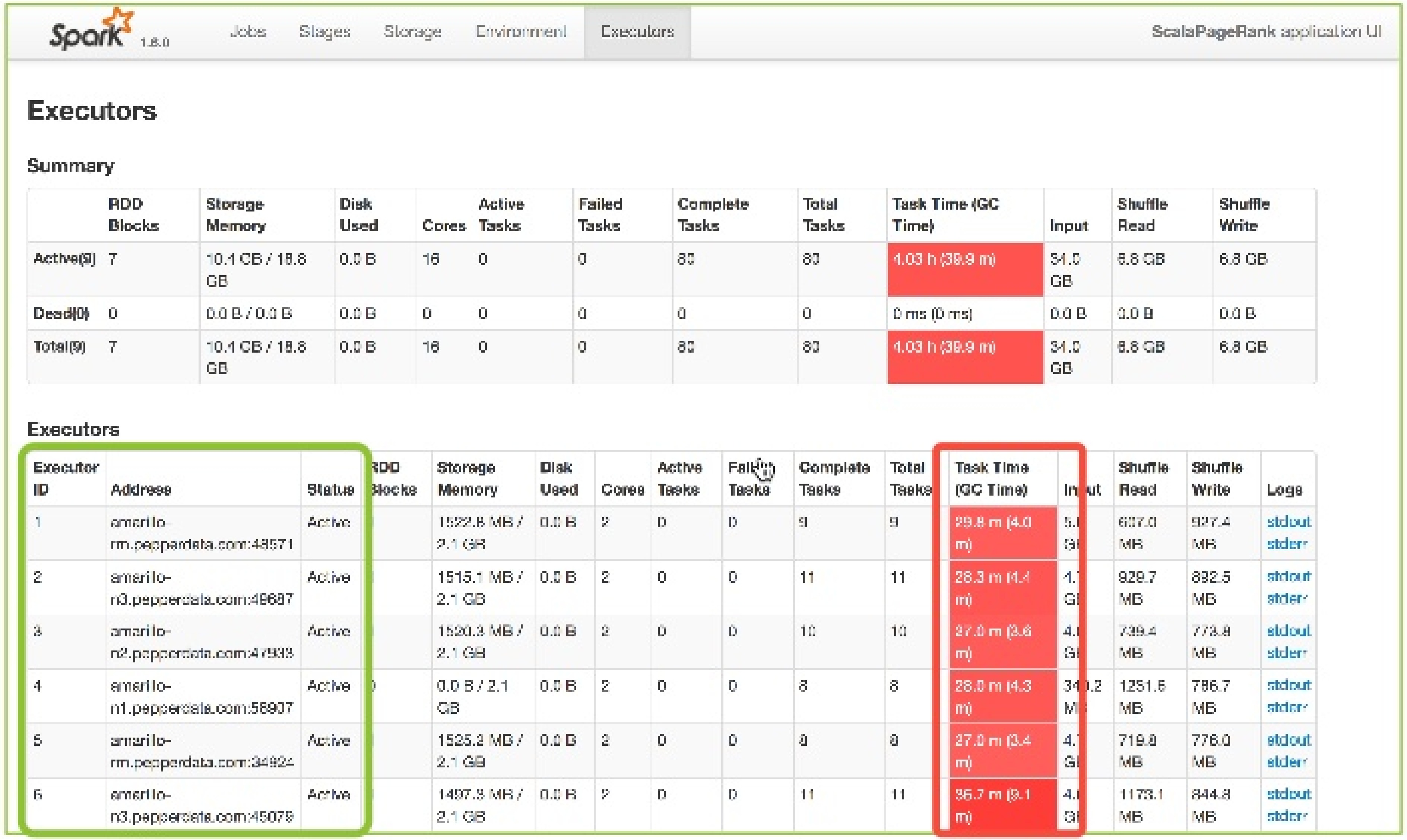
Stages for All Jobs

Completed Stages: 6

Completed Stages (6)

Stage Id	Description		Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
5	saveAsHadoopFile at IOCommon.scala:63	+details	2017/05/29 12:36:53	23 s	8/8		176.4 MB	547.3 MB	
4	flatMap at SparkPageRank.scala:62	+details	2017/05/29 12:29:45	7.1 min	8/8	10.4 GB		879.6 MB	547.3 MB
3	flatMap at SparkPageRank.scala:62	+details	2017/05/29 12:25:32	4.2 min	8/8	10.4 GB		829.8 MB	547.3 MB
2	flatMap at SparkPageRank.scala:62	+details	2017/05/29 12:18:59	6.5 min	8/8	10.4 GB		2.9 GB	497.5 MB
1	distinct at SparkPageRank.scala:58	+details	2017/05/29 12:13:00	6.0 min	24/24			2.7 GB	2.6 GB
0	distinct at SparkPageRank.scala:58	+details	2017/05/29 12:06:55	6.1 min	24/24	2.8 GB			2.7 GB

Physical execution plan from Spark: Executors / Tasks



Executors

Summary

	RDD Blocks	Storage Memory	Disk Used	Cores	Active Tasks	Failed Tasks	Complete Tasks	Total Tasks	Task Time (GC Time)	Input	Shuffle Read	Shuffle Write
Active(9)	7	10.1 GB / 18.8 GB	0.0 B	16	0	0	80	80	4.03 h (38.9 m)	34.0 GB	6.8 GB	6.8 GB
Dead(0)	0	0.0 B / 0.0 B	0.0 B	0	0	0	0	0	0 ms (0 ms)	0.0 B	0.0 B	0.0 B
Total(9)	7	10.1 GB / 18.8 GB	0.0 B	16	0	0	80	80	4.03 h (38.9 m)	34.0 GB	6.8 GB	6.8 GB

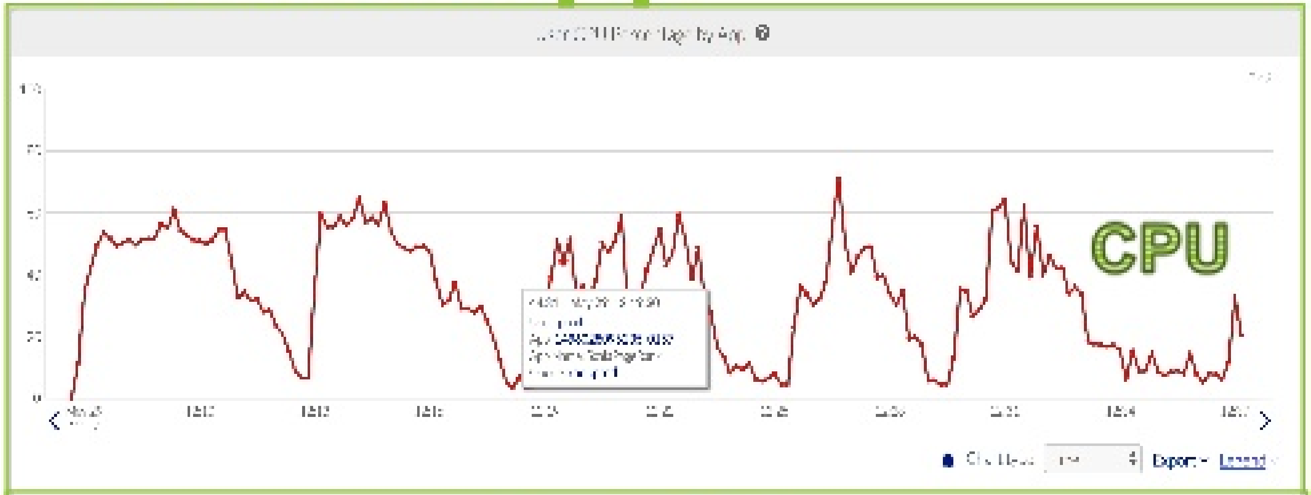
Executors

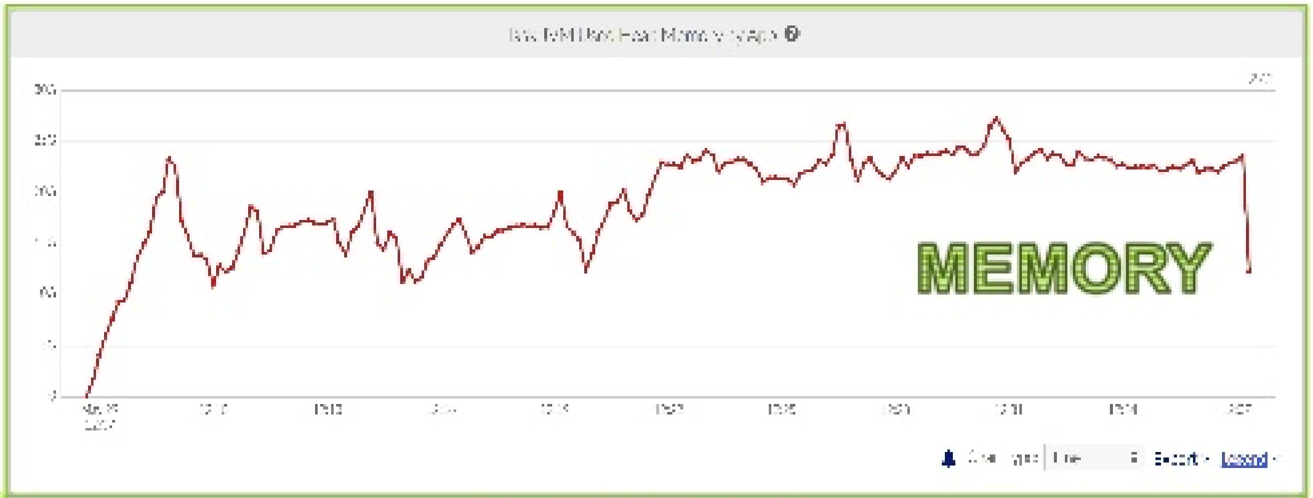
Executor ID	Address	Status	RDD Blocks	Storage Memory	Disk Used	Cores	Active Tasks	Failed Tasks	Complete Tasks	Total Tasks	Task Time (GC Time)	Input	Shuffle Read	Shuffle Write	Logs
1	amarillo-rm.pepperdata.com:48571	Active		1522.8 MB / 2.1 GB	0.0 B	2	0	0	9	9	29.8 m (4.0 m)	5.[illegible] G[illegible]	637.0 MB	927.4 MB	stdout stderr
2	amarillo-n3.pepperdata.com:49687	Active		1515.1 MB / 2.1 GB	0.0 B	2	0	0	11	11	28.3 m (4.4 m)	4.[illegible] G[illegible]	929.7 MB	892.5 MB	stdout stderr
3	amarillo-n2.pepperdata.com:47933	Active		1520.3 MB / 2.1 GB	0.0 B	2	0	0	10	10	27.0 m (3.6 m)	4.[illegible] G[illegible]	739.4 MB	773.8 MB	stdout stderr
4	amarillo-n1.pepperdata.com:56907	Active	0	0.0 B / 2.1 GB	0.0 B	2	0	0	8	8	28.0 m (4.3 m)	34[illegible]0.2 M[illegible]	1251.6 MB	786.7 MB	stdout stderr
5	amarillo-rm.pepperdata.com:34924	Active		1525.2 MB / 2.1 GB	0.0 B	2	0	0	8	8	27.0 m (3.4 m)	4.[illegible] G[illegible]	719.8 MB	776.0 MB	stdout stderr
6	amarillo-n3.pepperdata.com:45079	Active		1497.3 MB / 2.1 GB	0.0 B	2	0	0	11	11	36.7 m (9.1 m)	4.[illegible] G[illegible]	1173.1 MB	844.8 MB	stdout stderr

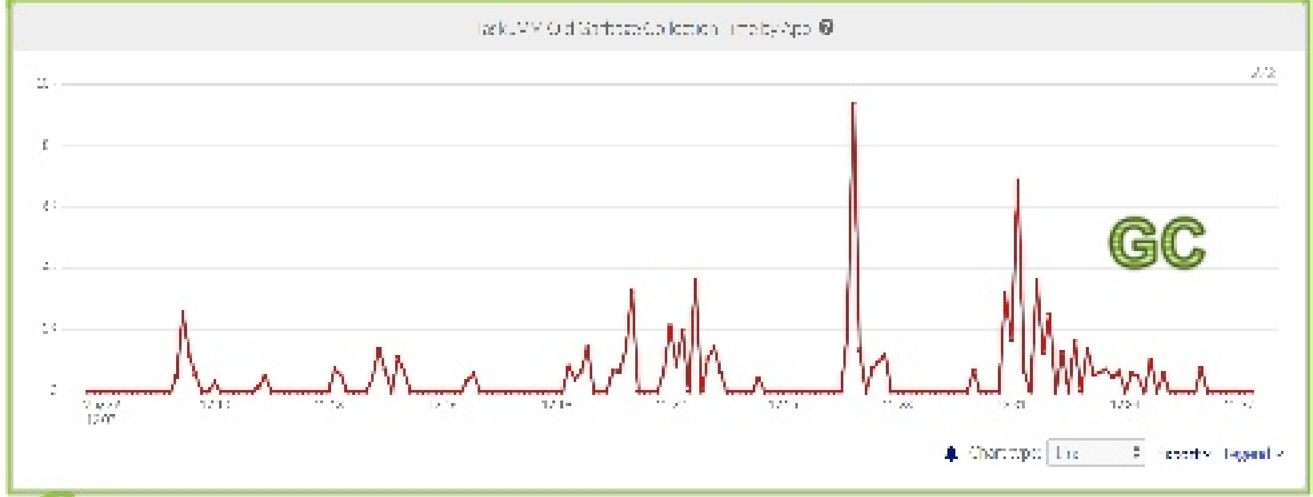
Second half of solution

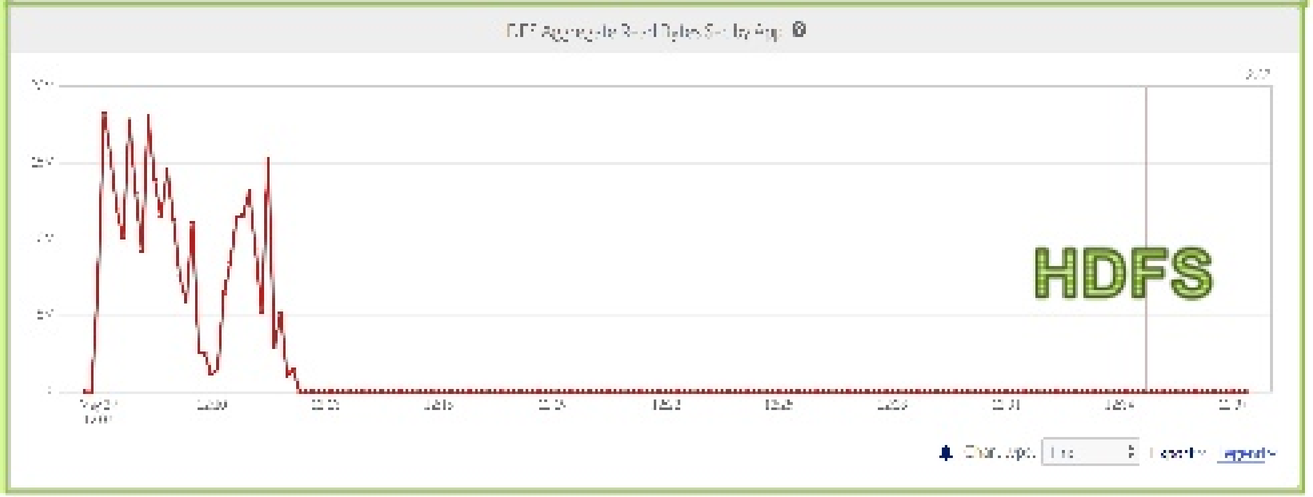
Time series view

Time series view of resource consumption for the App







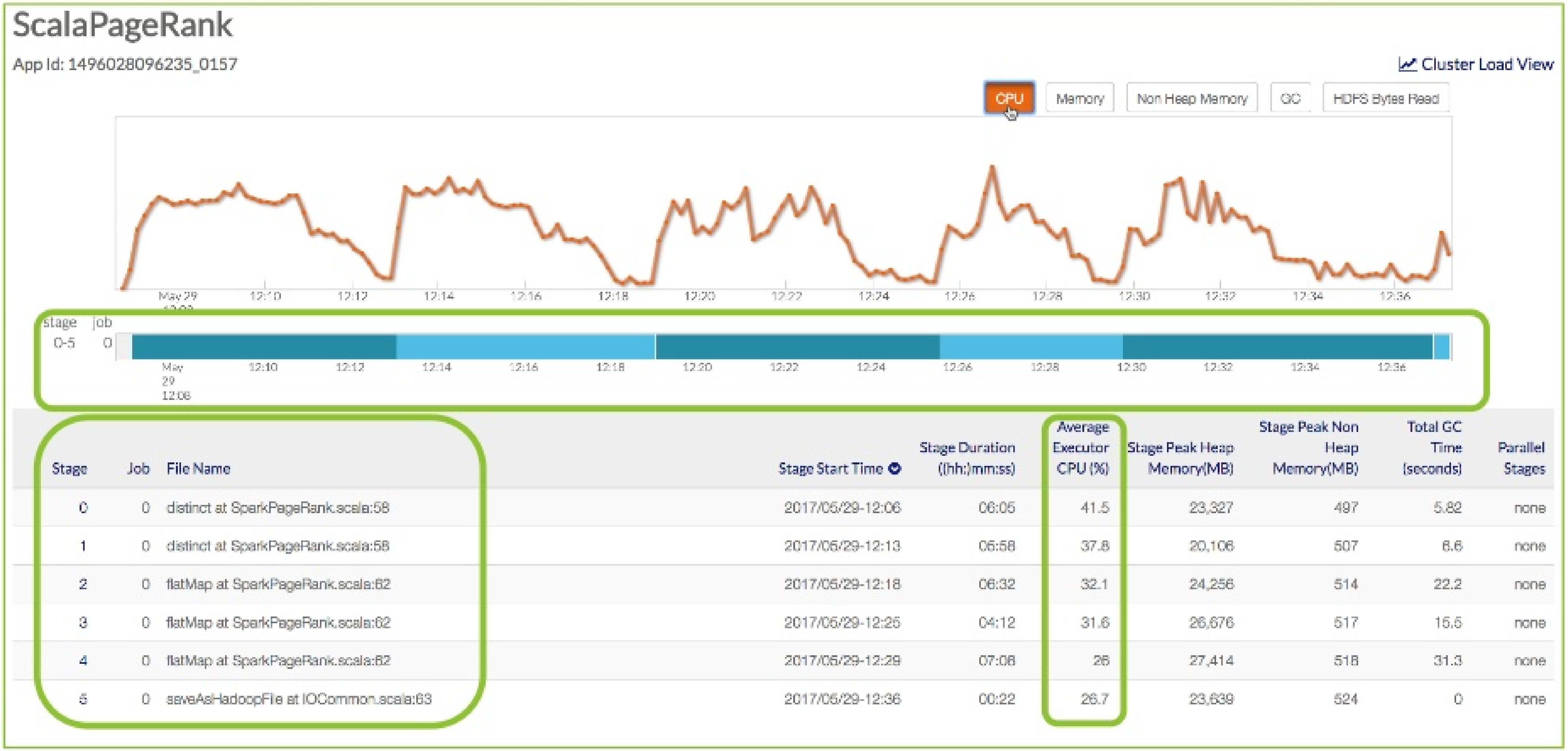


SPARK SUMMIT 2017

Best of both worlds

Bring them together

Bringing it all together: CPU across Stages

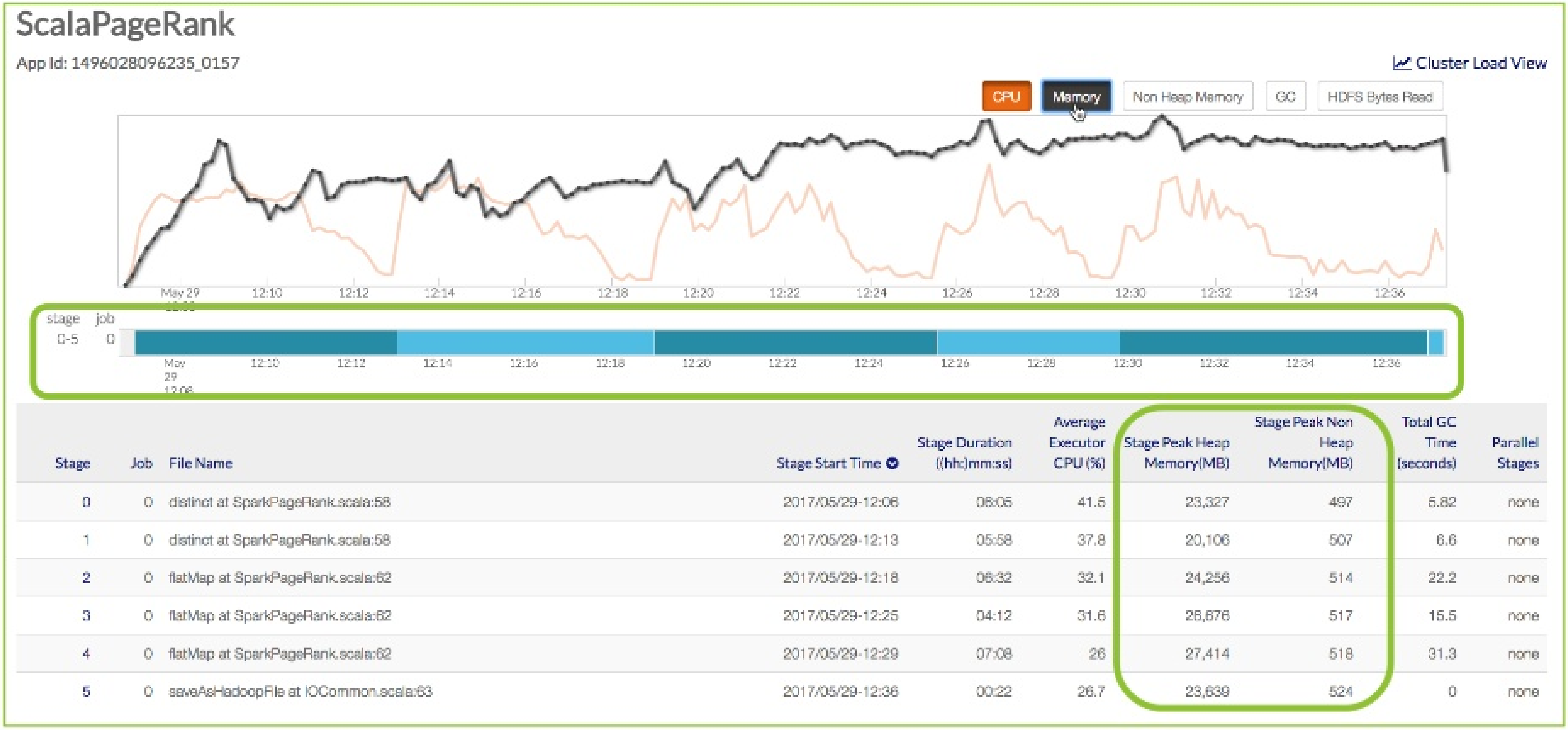


ScalaPageRank

App Id: 1496028096235_0157

Stage	Job	File Name	Stage Start Time	Stage Duration ((hh:)mm:ss)	Average Executor CPU (%)	Stage Peak Heap Memory(MB)	Stage Peak Non Heap Memory(MB)	Total GC Time (seconds)	Parallel Stages
0	0	distinct at SparkPageRank.scala:58	2017/05/29-12:06	06:05	41.5	23,327	497	5.82	none
1	0	distinct at SparkPageRank.scala:58	2017/05/29-12:13	05:58	37.8	20,106	507	6.6	none
2	0	flatMap at SparkPageRank.scala:62	2017/05/29-12:18	06:32	32.1	24,256	514	22.2	none
3	0	flatMap at SparkPageRank.scala:62	2017/05/29-12:25	04:12	31.6	26,676	517	15.5	none
4	0	flatMap at SparkPageRank.scala:62	2017/05/29-12:29	07:08	26	27,414	518	31.3	none
5	0	saveAsHadoopFile at IOCommon.scala:63	2017/05/29-12:36	00:22	26.7	23,639	524	0	none

Memory across all Stages of App

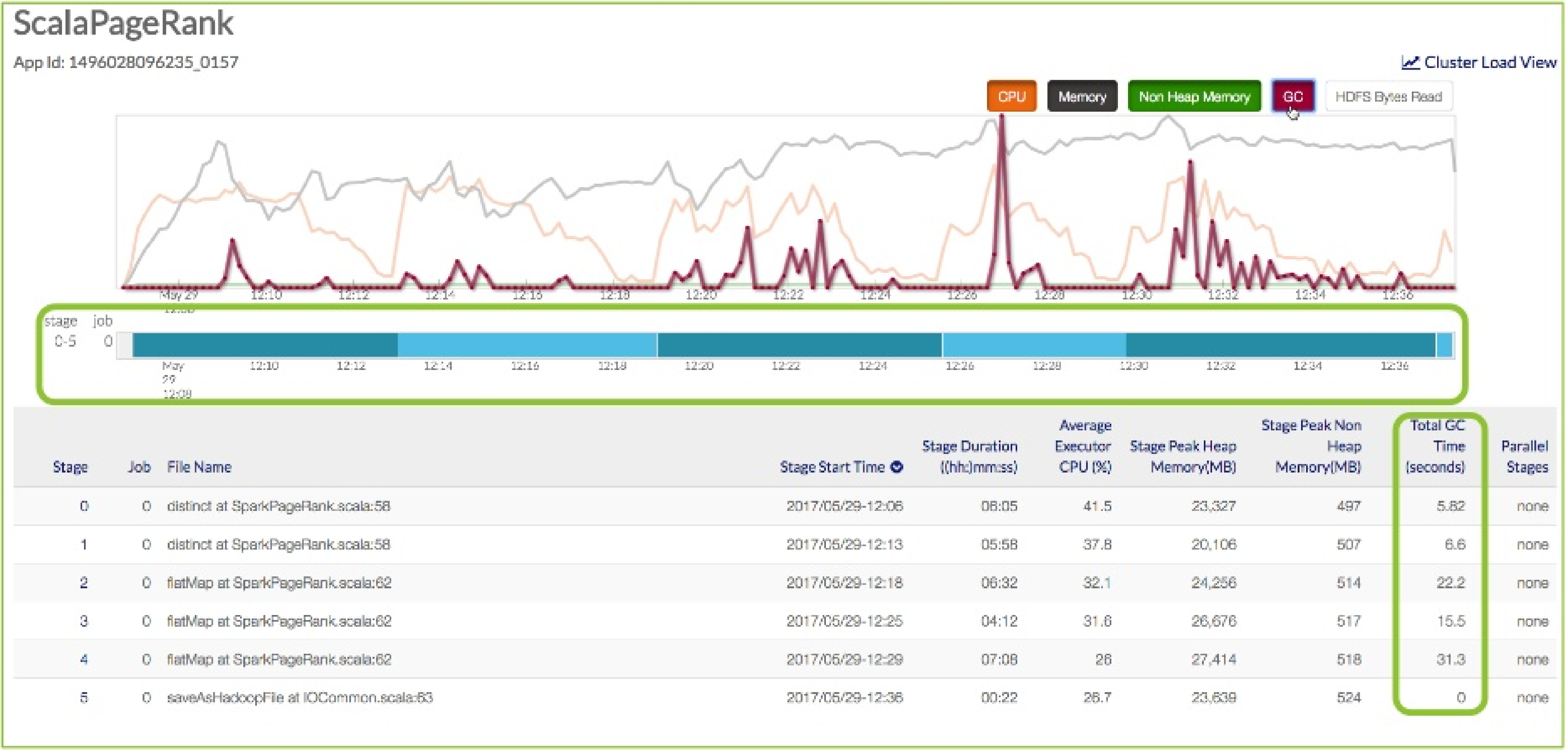


ScalaPageRank

App Id: 1496028096235_0157

Stage	Job	File Name	Stage Start Time	Stage Duration ((hh:)mm:ss)	Average Executor CPU (%)	Stage Peak Heap Memory(MB)	Stage Peak Non Heap Memory(MB)	Total GC Time (seconds)	Parallel Stages
0	0	distinct at SparkPageRank.scala:58	2017/05/29-12:06	06:05	41.5	23,327	497	5.82	none
1	0	distinct at SparkPageRank.scala:58	2017/05/29-12:13	05:58	37.8	20,106	507	6.6	none
2	0	flatMap at SparkPageRank.scala:62	2017/05/29-12:18	06:32	32.1	24,256	514	22.2	none
3	0	flatMap at SparkPageRank.scala:62	2017/05/29-12:25	04:12	31.6	26,676	517	15.5	none
4	0	flatMap at SparkPageRank.scala:62	2017/05/29-12:29	07:08	26	27,414	518	31.3	none
5	0	saveAsHadoopFile at IOCommon.scala:63	2017/05/29-12:36	00:22	26.7	23,639	524	0	none

GC across all Stages of App

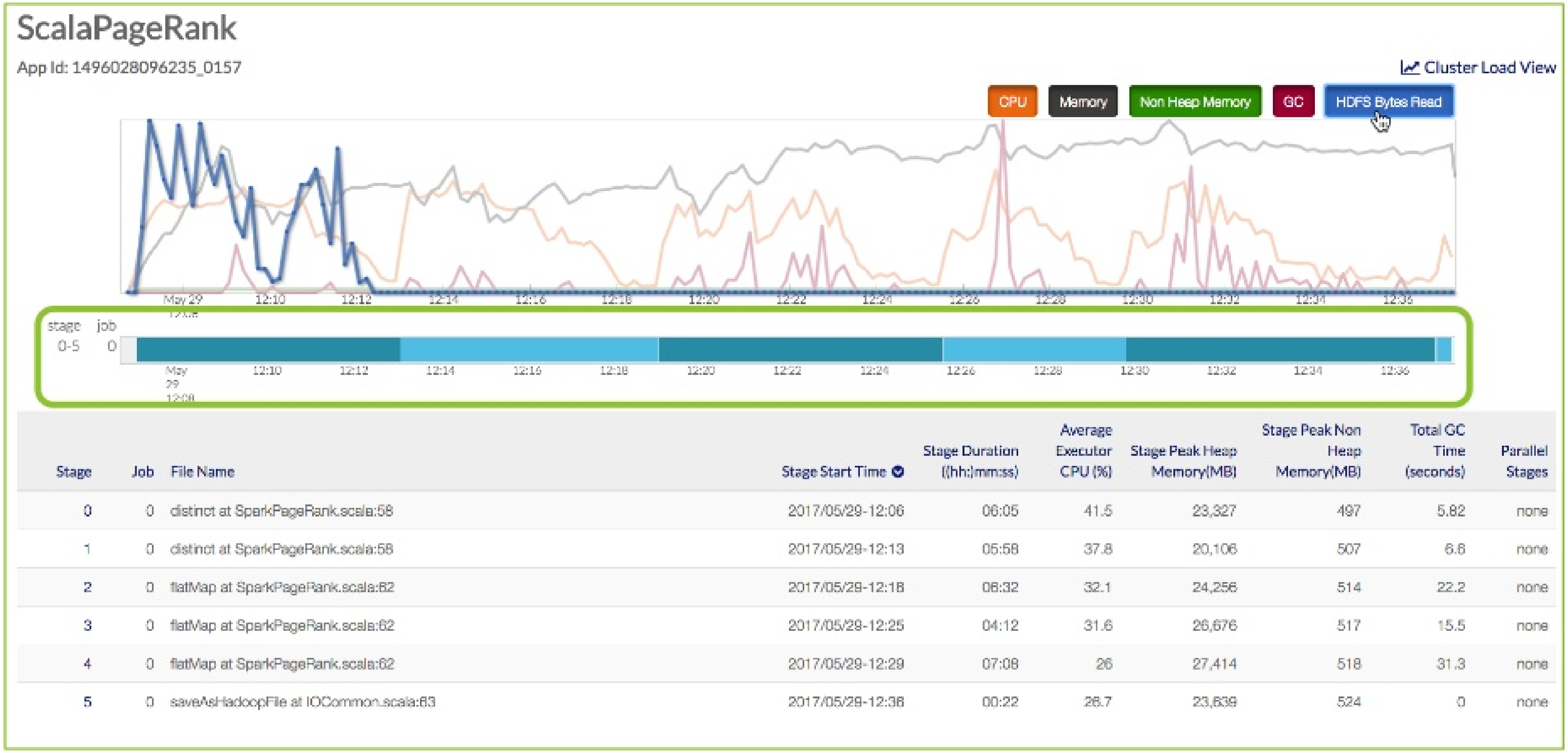


ScalaPageRank

App Id: 1496028096235_0157

Stage	Job	File Name	Stage Start Time	Stage Duration ((hh:)mm:ss)	Average Executor CPU (%)	Stage Peak Heap Memory(MB)	Stage Peak Non Heap Memory(MB)	Total GC Time (seconds)	Parallel Stages
0	0	distinct at SparkPageRank.scala:58	2017/05/29-12:06	06:05	41.5	23,327	497	5.82	none
1	0	distinct at SparkPageRank.scala:58	2017/05/29-12:13	05:58	37.8	20,106	507	6.6	none
2	0	flatMap at SparkPageRank.scala:62	2017/05/29-12:18	06:32	32.1	24,256	514	22.2	none
3	0	flatMap at SparkPageRank.scala:62	2017/05/29-12:25	04:12	31.6	26,676	517	15.5	none
4	0	flatMap at SparkPageRank.scala:62	2017/05/29-12:29	07:08	26	27,414	518	31.3	none
5	0	saveAsHadoopFile at IOCommon.scala:63	2017/05/29-12:36	00:22	26.7	23,639	524	0	none

HDFS Reads across all Stages of App

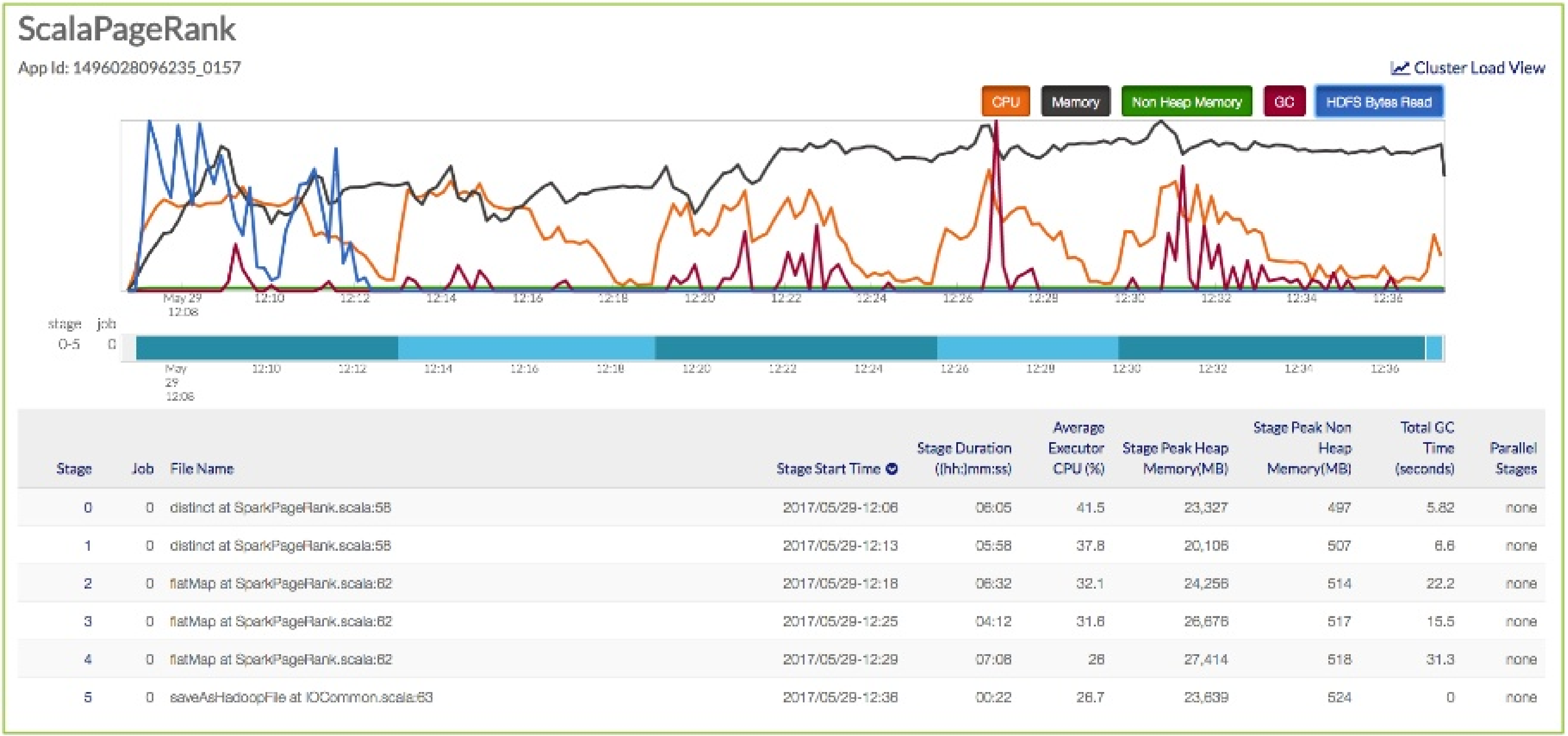


ScalaPageRank

App Id: 1496028096235_0157

Stage	Job	File Name	Stage Start Time	Stage Duration ((hh:)mm:ss)	Average Executor CPU (%)	Stage Peak Heap Memory(MB)	Stage Peak Non Heap Memory(MB)	Total GC Time (seconds)	Parallel Stages
0	0	distinct at SparkPageRank.scala:58	2017/05/29-12:06	06:05	41.5	23,327	497	5.82	none
1	0	distinct at SparkPageRank.scala:58	2017/05/29-12:13	05:58	37.8	20,106	507	6.6	none
2	0	flatMap at SparkPageRank.scala:62	2017/05/29-12:18	06:32	32.1	24,256	514	22.2	none
3	0	flatMap at SparkPageRank.scala:62	2017/05/29-12:25	04:12	31.6	26,676	517	15.5	none
4	0	flatMap at SparkPageRank.scala:62	2017/05/29-12:29	07:08	26	27,414	518	31.3	none
5	0	saveAsHadoopFile at IOCommon.scala:63	2017/05/29-12:36	00:22	26.7	23,639	524	0	none

Bringing it all together

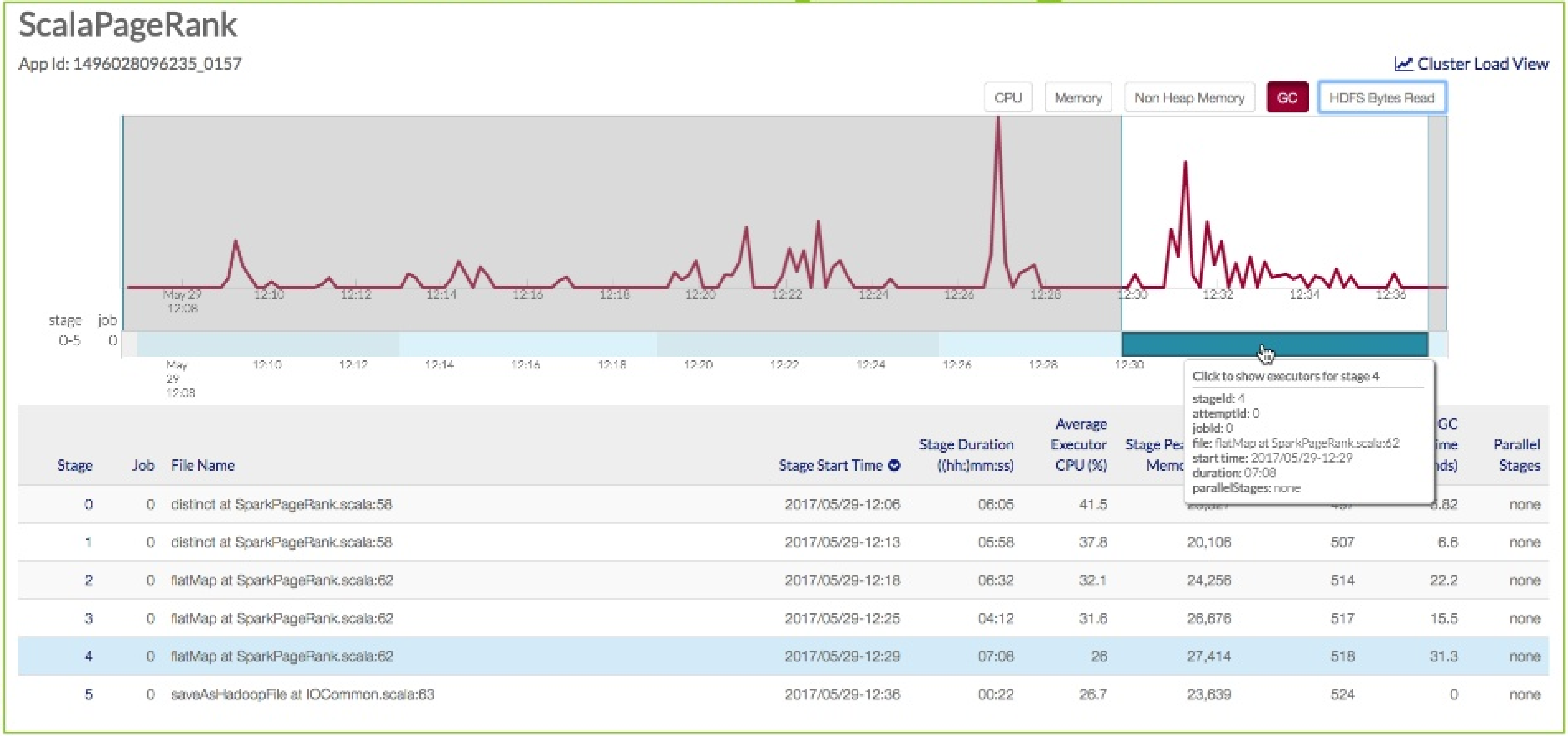


ScalaPageRank

App Id: 1496028096235_0157

Stage	Job	File Name	Stage Start Time	Stage Duration ([hh:]mm:ss)	Average Executor CPU (%)	Stage Peak Heap Memory(MB)	Stage Peak Non Heap Memory(MB)	Total GC Time (seconds)	Parallel Stages
0	0	distinct at SparkPageRank.scala:58	2017/05/29-12:08	06:05	41.5	23,327	497	5.82	none
1	0	distinct at SparkPageRank.scala:58	2017/05/29-12:13	05:58	37.8	20,106	507	6.6	none
2	0	flatMap at SparkPageRank.scala:62	2017/05/29-12:18	06:32	32.1	24,256	514	22.2	none
3	0	flatMap at SparkPageRank.scala:62	2017/05/29-12:25	04:12	31.8	26,676	517	15.5	none
4	0	flatMap at SparkPageRank.scala:62	2017/05/29-12:29	07:08	28	27,414	518	31.3	none
5	0	saveAsHadoopFile at IOCommon.scala:63	2017/05/29-12:36	00:22	26.7	23,639	524	0	none

Let’s examine GC activity in Stage 4



ScalaPageRank

App Id: 1496028096235_0157

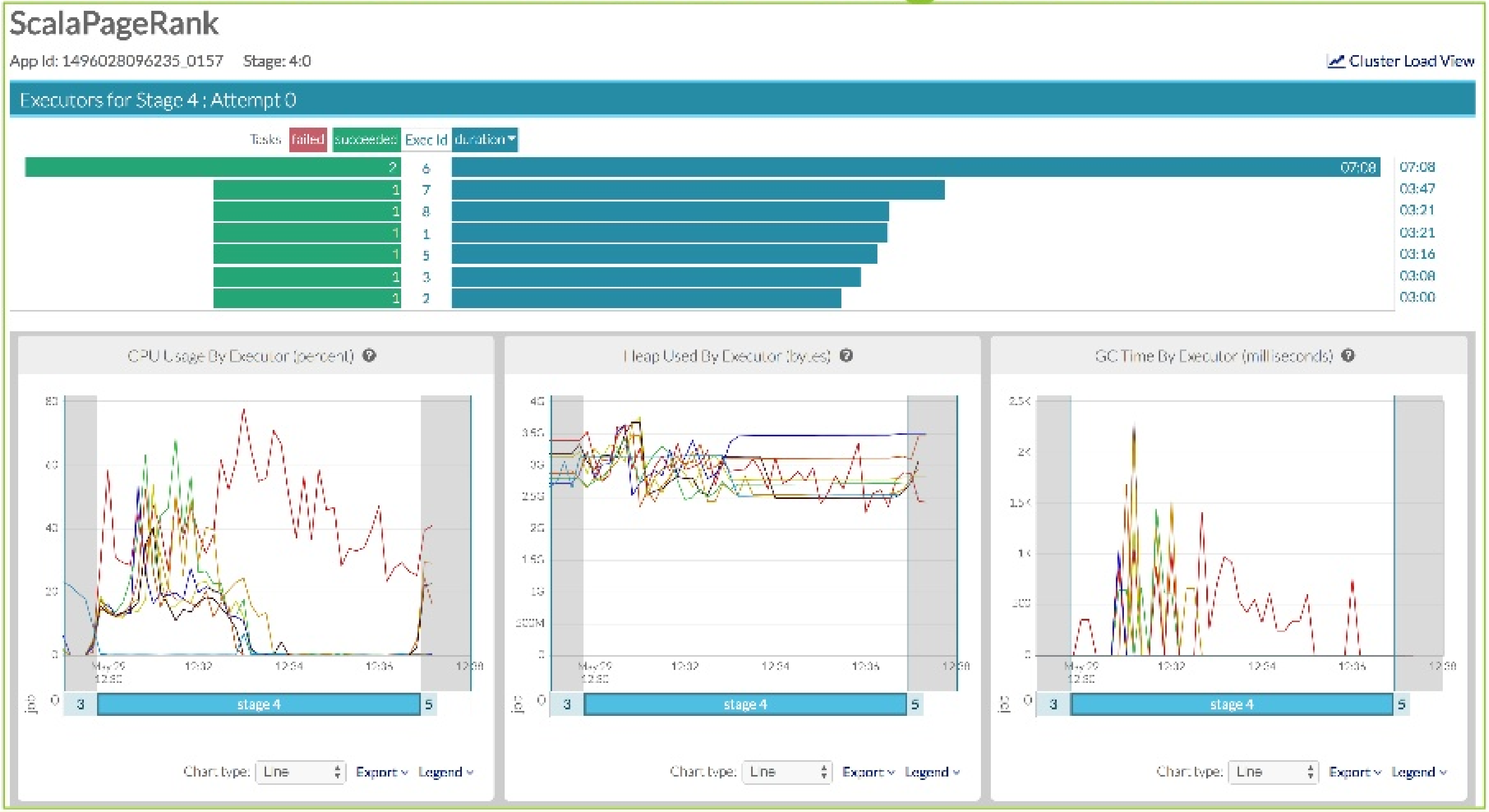
Cluster Load View

Click to show executors for stage 4

stageId: 4
attemptId: 0
jobId: 0
file: flatMap at SparkPageRank.scala:62
start time: 2017/05/29-12:29
duration: 07:08
parallelStages: none

Stage	Job	File Name	Stage Start Time	Stage Duration ((hh:)mm:ss)	Average Executor CPU (%)	Stage Pe Memo		GC ime nds)	Parallel Stages
0	0	distinct at SparkPageRank.scala:58	2017/05/29-12:06	06:05	41.5	[illegible]	[illegible]	[illegible].82	none
1	0	distinct at SparkPageRank.scala:58	2017/05/29-12:13	05:58	37.8	20,106	507	6.6	none
2	0	flatMap at SparkPageRank.scala:62	2017/05/29-12:18	06:32	32.1	24,256	514	22.2	none
3	0	flatMap at SparkPageRank.scala:62	2017/05/29-12:25	04:12	31.6	26,676	517	15.5	none
4	0	flatMap at SparkPageRank.scala:62	2017/05/29-12:29	07:08	26	27,414	518	31.3	none
5	0	saveAsHadoopFile at IOCommon.scala:63	2017/05/29-12:36	00:22	26.7	23,639	524	0	none

Executor skew increased Stage duration 2x



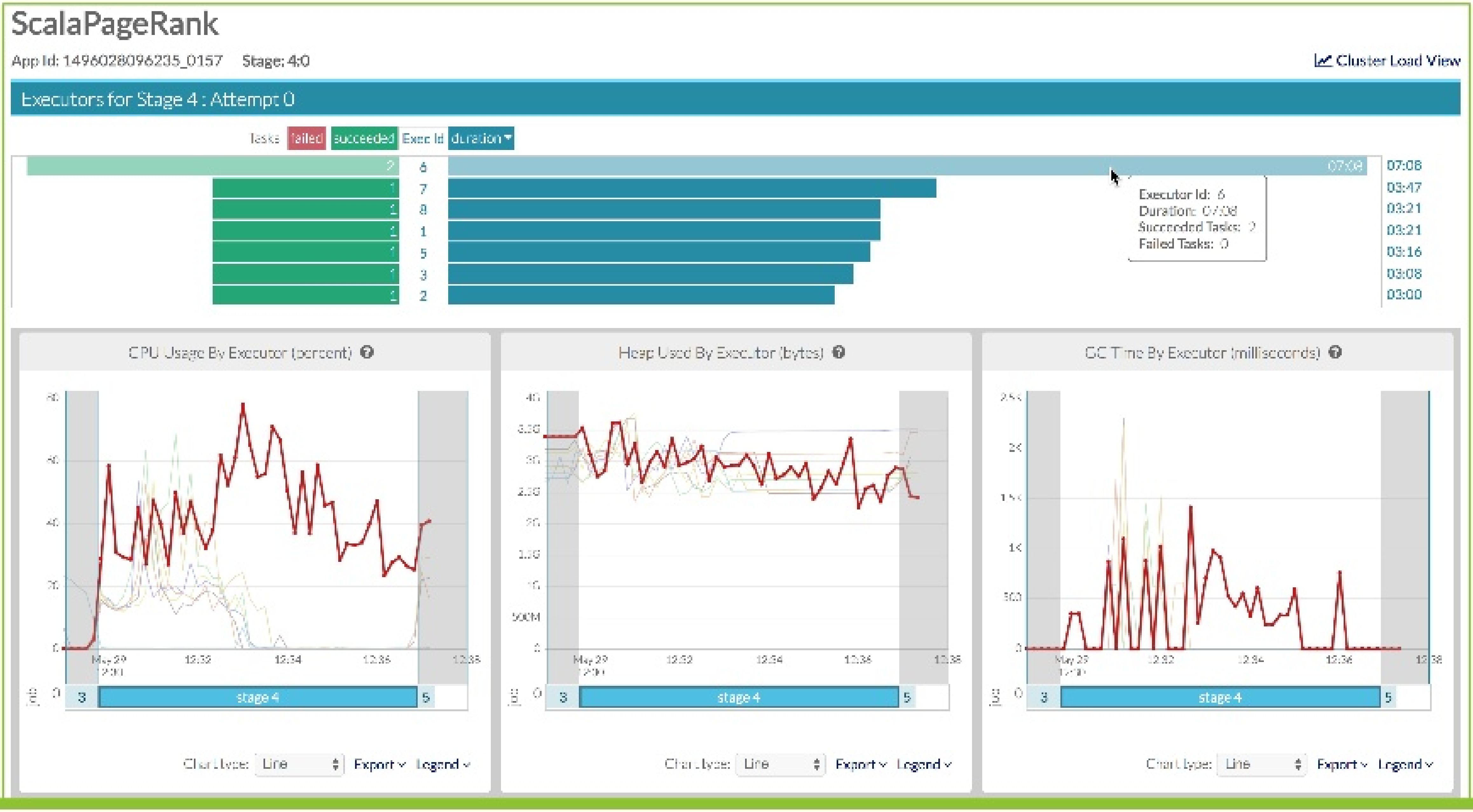
ScalaPageRank

App Id: 1496028096235_0157 Stage: 4:0

Executors for Stage 4 : Attempt 0

Tasks failed	succeeded	Exec Id	duration
	2	6	07:08
	1	7	03:47
	1	8	03:21
	1	1	03:21
	1	5	03:16
	1	3	03:08
	1	2	03:00

Possible solution: increase number of partitions



ScalaPageRank

App Id: 1496028096235_0157 Stage: 4:0

Cluster Load View

Executors for Stage 4 : Attempt 0

Tasks	Exec Id	duration
2	6	07:08
1	7	03:47
1	8	03:21
1	1	03:21
1	5	03:16
1	3	03:08
1	2	03:00

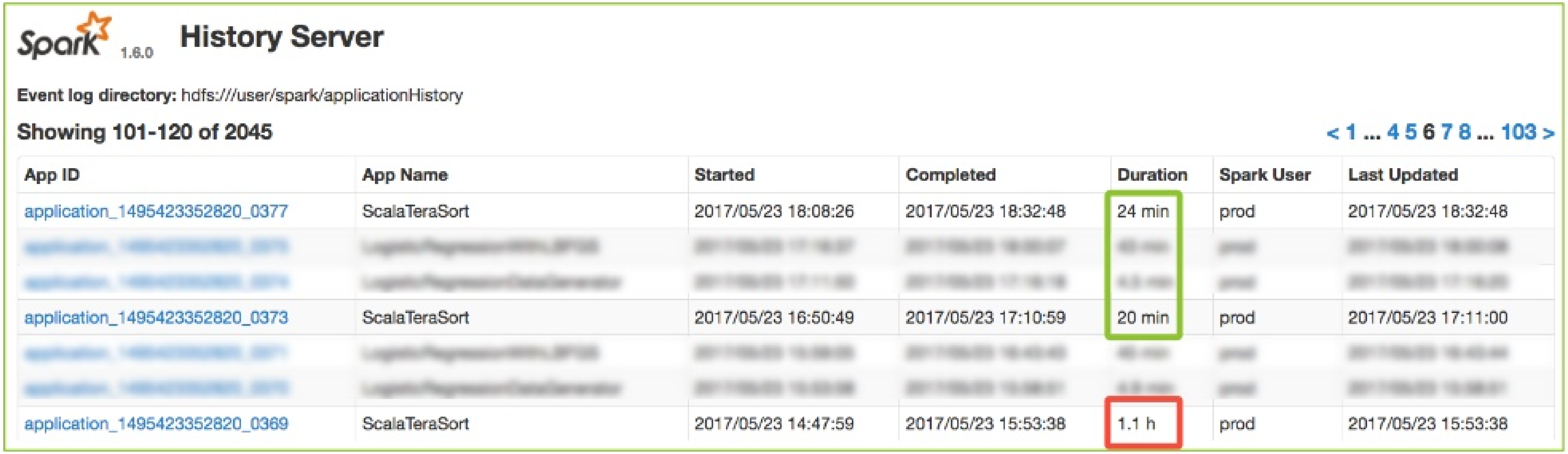
Executor Id: 6
Duration: 07:08
Succeeded Tasks: 2
Failed Tasks: 0

Chart type: Line

Cluster weather

What if it’s not your fault?

How does cluster weather impact your app ?



History Server

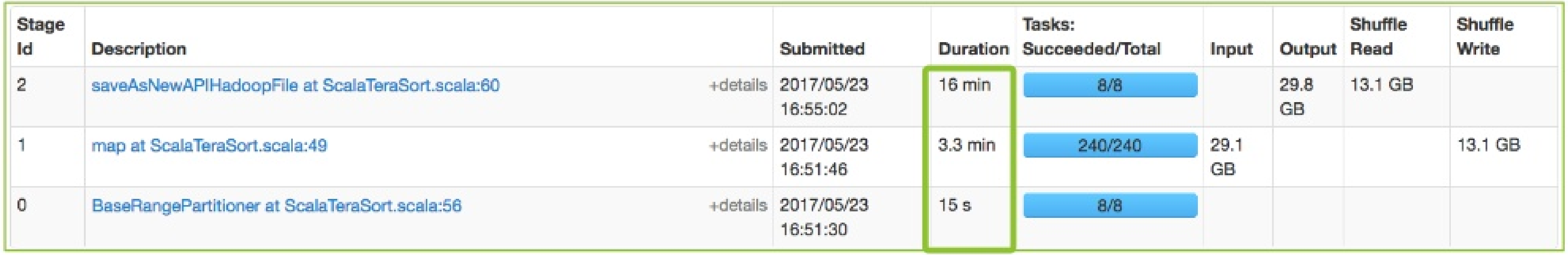
Event log directory: hdfs:///user/spark/applicationHistory

Showing 101-120 of 2045

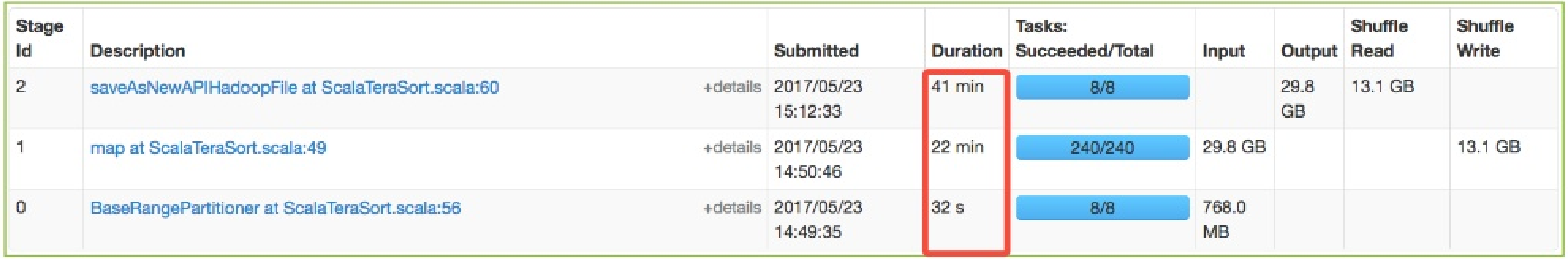
< 1 ... 4 5 6 7 8 ... 103 >

App ID	App Name	Started	Completed	Duration	Spark User	Last Updated
application_1495423352820_0377	ScalaTeraSort	2017/05/23 18:08:26	2017/05/23 18:32:48	24 min	prod	2017/05/23 18:32:48
[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
application_1495423352820_0373	ScalaTeraSort	2017/05/23 16:50:49	2017/05/23 17:10:59	20 min	prod	2017/05/23 17:11:00
[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
application_1495423352820_0369	ScalaTeraSort	2017/05/23 14:47:59	2017/05/23 15:53:38	1.1 h	prod	2017/05/23 15:53:38

No apparent reason for delay from Spark Web UI

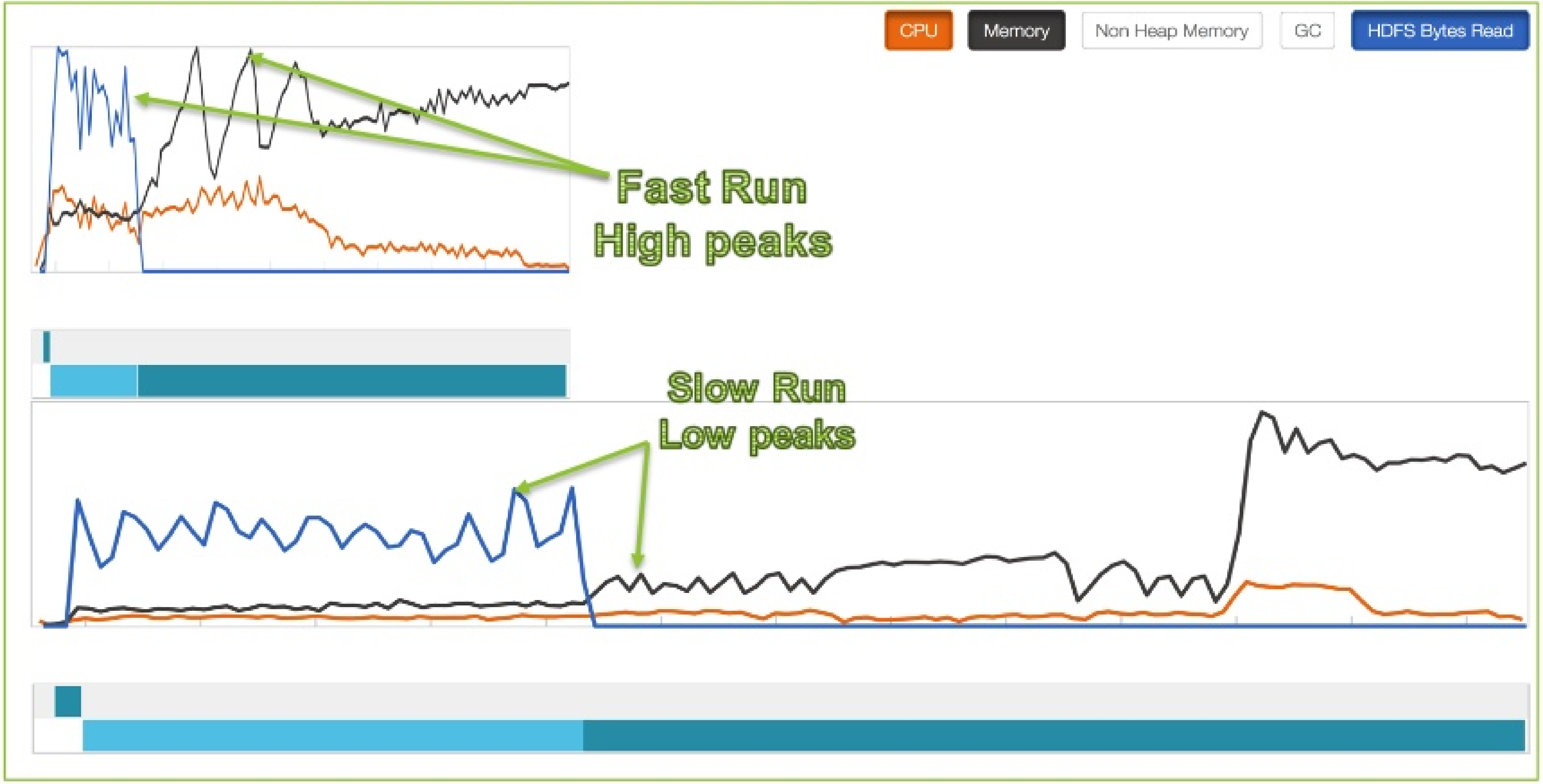


Stage Id	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
2	saveAsNewAPIHadoopFile at ScalaTeraSort.scala:60 +details	2017/05/23 16:55:02	16 min	8/8		29.8 GB	13.1 GB	
1	map at ScalaTeraSort.scala:49 +details	2017/05/23 16:51:46	3.3 min	240/240	29.1 GB			13.1 GB
0	BaseRangePartitioner at ScalaTeraSort.scala:56 +details	2017/05/23 16:51:30	15 s	8/8				



Stage Id	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
2	saveAsNewAPIHadoopFile at ScalaTeraSort.scala:60 +details	2017/05/23 15:12:33	41 min	8/8		29.8 GB	13.1 GB	
1	map at ScalaTeraSort.scala:49 +details	2017/05/23 14:50:46	22 min	240/240	29.8 GB			13.1 GB
0	BaseRangePartitioner at ScalaTeraSort.scala:56 +details	2017/05/23 14:49:35	32 s	8/8	768.0 MB			

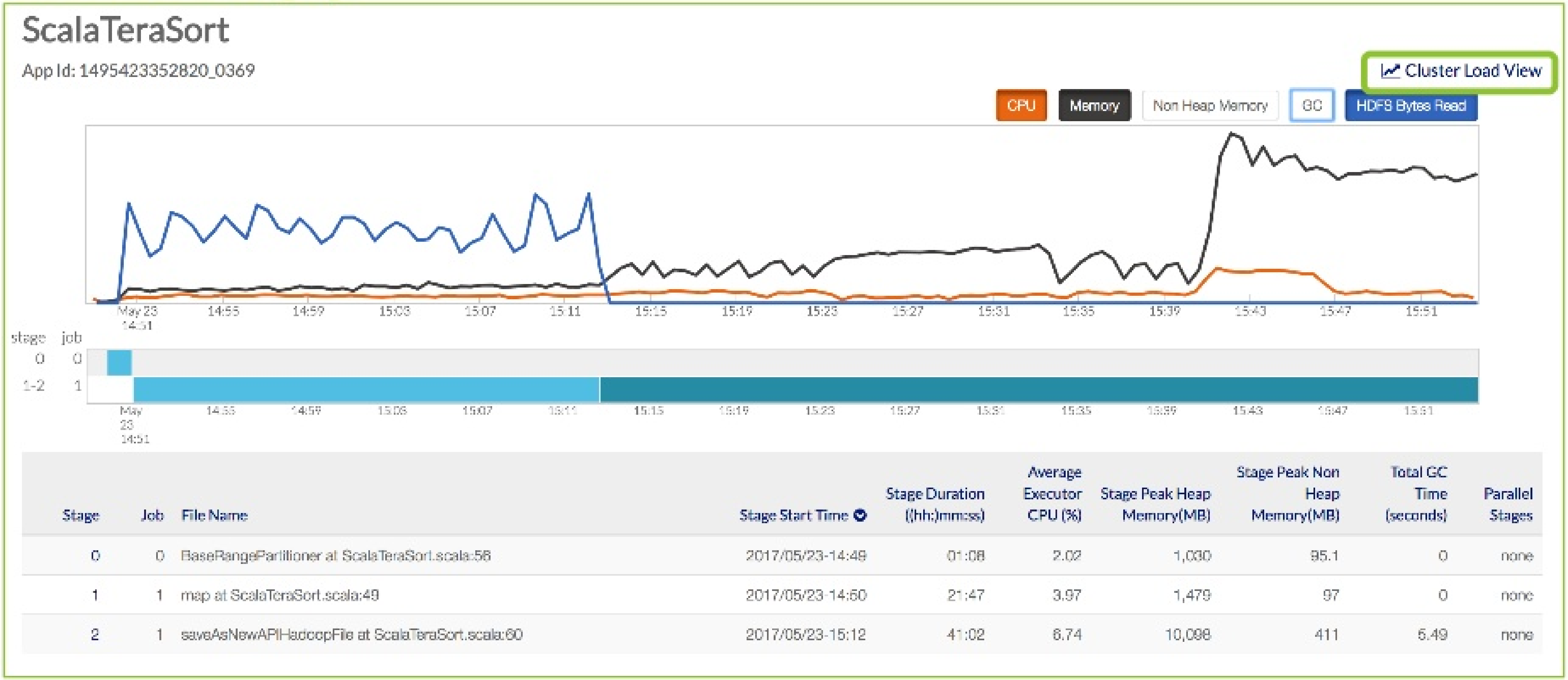
Time series shows slower run of App with much lower resources



Fast Run
High peaks

Slow Run
Low peaks

View cluster weather conditions for slower run of App

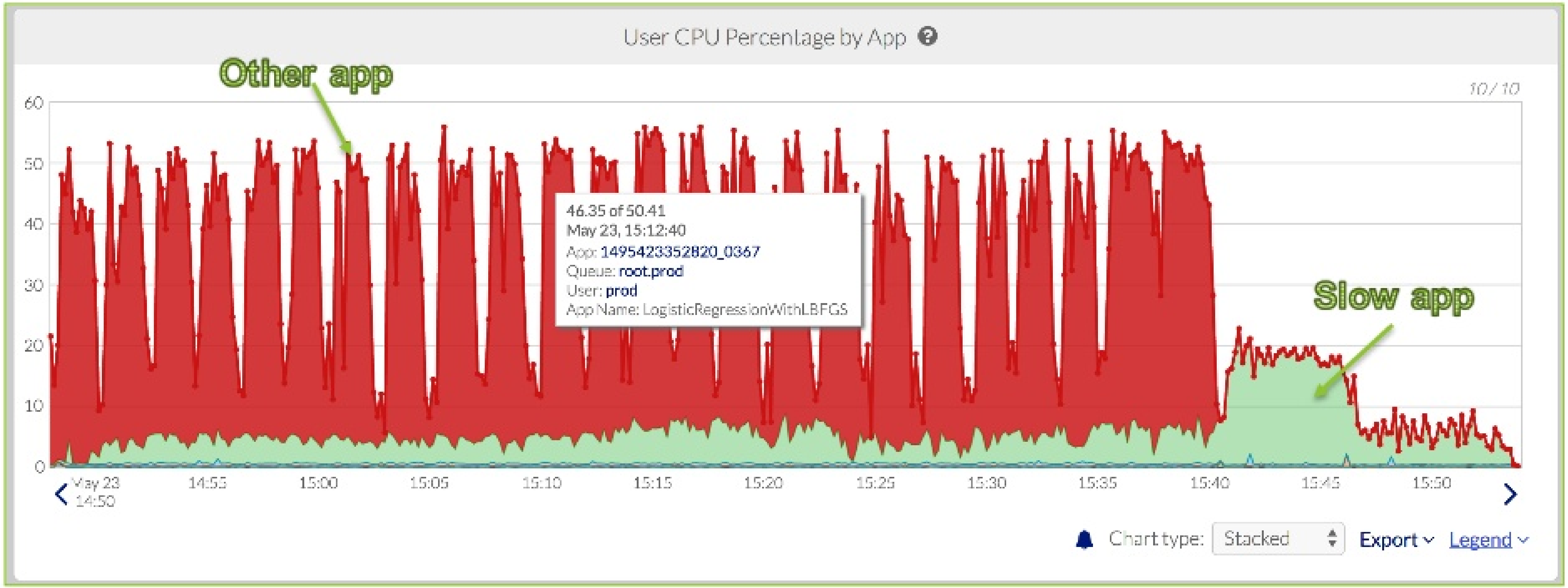


ScalaTeraSort

App Id: 1495423352820_0369

Stage	Job	File Name	Stage Start Time	Stage Duration ((hh:)mm:ss)	Average Executor CPU (%)	Stage Peak Heap Memory(MB)	Stage Peak Non Heap Memory(MB)	Total GC Time (seconds)	Parallel Stages
0	0	BaseRangePartitioner at ScalaTeraSort.scala:56	2017/05/23-14:49	01:08	2.02	1,030	95.1	0	none
1	1	map at ScalaTeraSort.scala:49	2017/05/23-14:50	21:47	3.97	1,479	97	0	none
2	1	saveAsNewAPIHadoopFile at ScalaTeraSort.scala:60	2017/05/23-15:12	41:02	6.74	10,098	411	5.49	none

Cluster weather reveals reason for CPU constraints on slower app



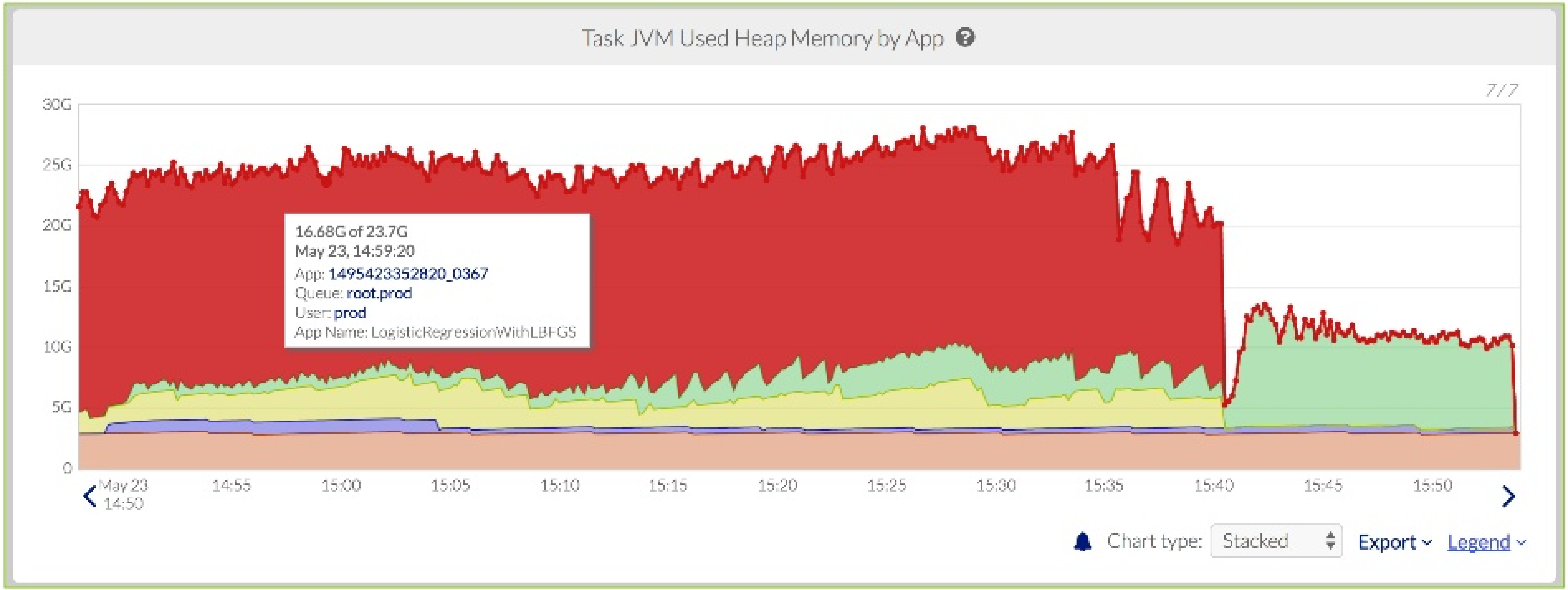
Other app

Slow app

46.35 of 50.41
May 23, 15:12:40
App: 1495423352820_0367
Queue: root.prod
User: prod
App Name: LogisticRegressionWithLBFGS

Chart type: Stacked

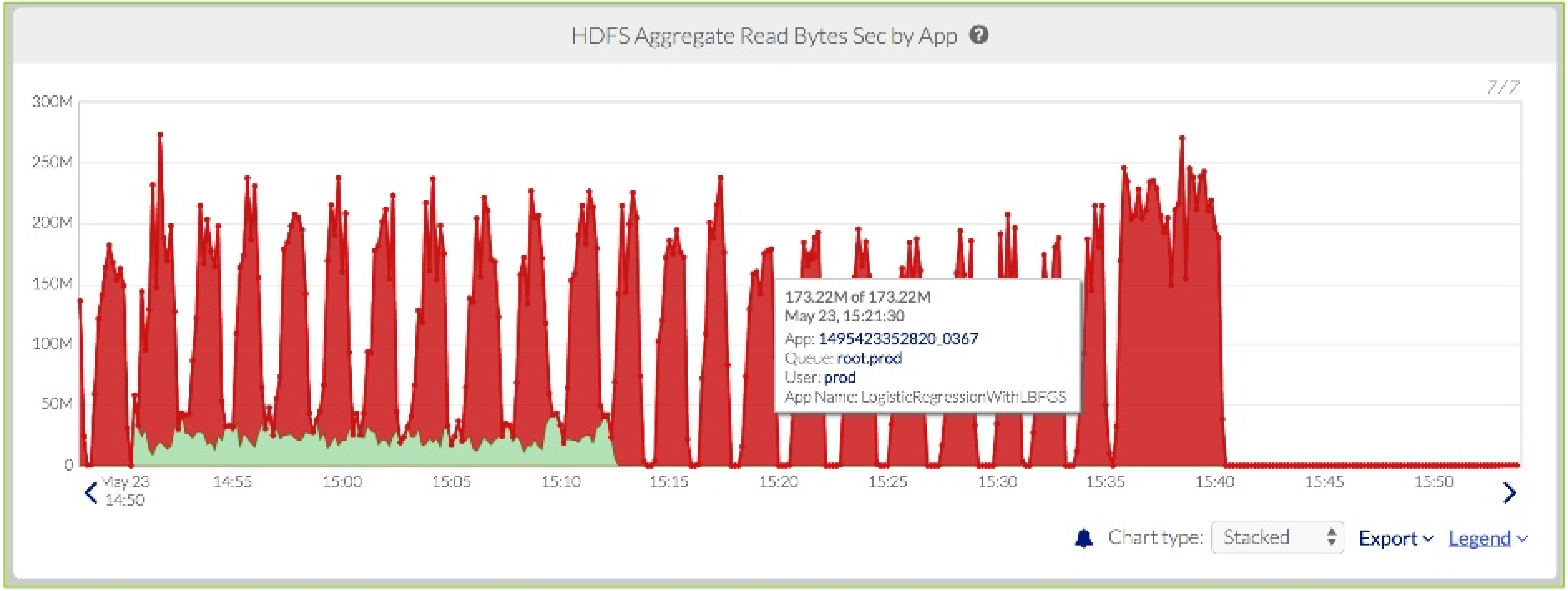
Cluster weather reveals reason for memory constraints on slower app



16.68G of 23.7G
May 23, 14:59:20
App: 1495423352820_0367
Queue: root.prod
User: prod
App Name: LogisticRegressionWithLBFGS

Chart type: Stacked

Cluster weather reveals reason for HDFS constraints on slower app



173.22M of 173.22M
May 23, 15:21:30
App: 1495423352820_0367
Queue: root.prod
User: prod
App Name: LogisticRegressionWithLBFGS

To recap

- Execution plan integrated with time series shines a spot light on problems
- Stage and code section integrated with resources consumed enables focus on most impactful areas for optimization
- Knowing cluster weather can prevent time wasted debugging non-existent performance issues

Code Analyzer for Apache Spark

- Free during Early Access starting June 5th, 2017
- Early Access is for development teams
- To learn more visit pepperdata. booth #101
- My contact vinod@pepperdata.com

pepperdata.com/products/code-analyzer

SPARK
SUMMIT
2017

Thank You.

www.pepperdata.com/products/code-analyzer/