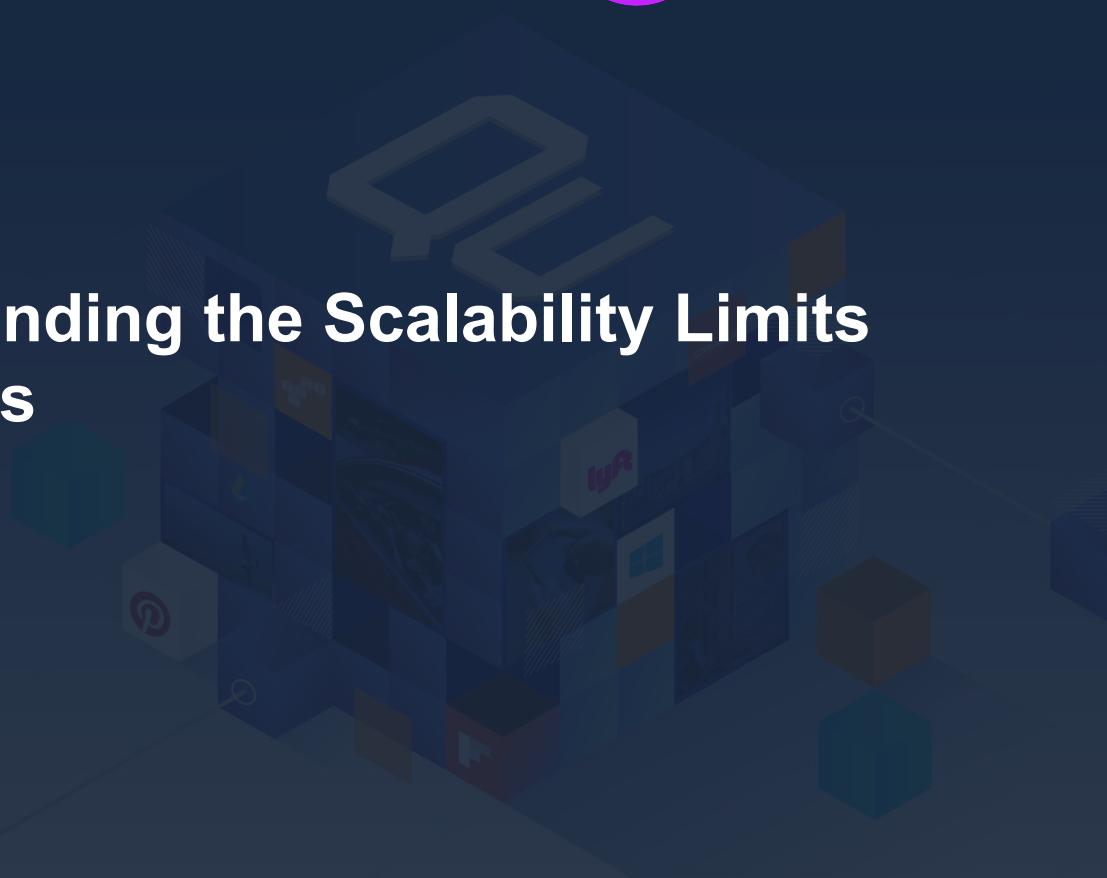
Qubole

Data Council

Sparklens: Understanding the Scalability Limits of Spark Applications

Ashish Dubey, Qubole



ABOUT PRESENTER

Ashish is a Big Data leader and practitioner with more than 15 years of industry experience. Equipped with immense experience involving the design and development of petabyte-scale Big Data applications, he is a seasoned technology architect with variegated experiences in customer interfacing and technical leadership roles.

Ashish heads Qubole's Solutions Architecture team for International Markets, and works with a number of enterprise customers in the EMEA, APAC and India regions. Prior to Qubole, Ashish worked at Microsoft as an engineer in the Windows team. Later, he worked for Claraview (Teradata), while leading their Big Data practice and helped to scale some of their Fortune 500 clients in different industry verticals such as finance, healthcare, retail and multimedia.

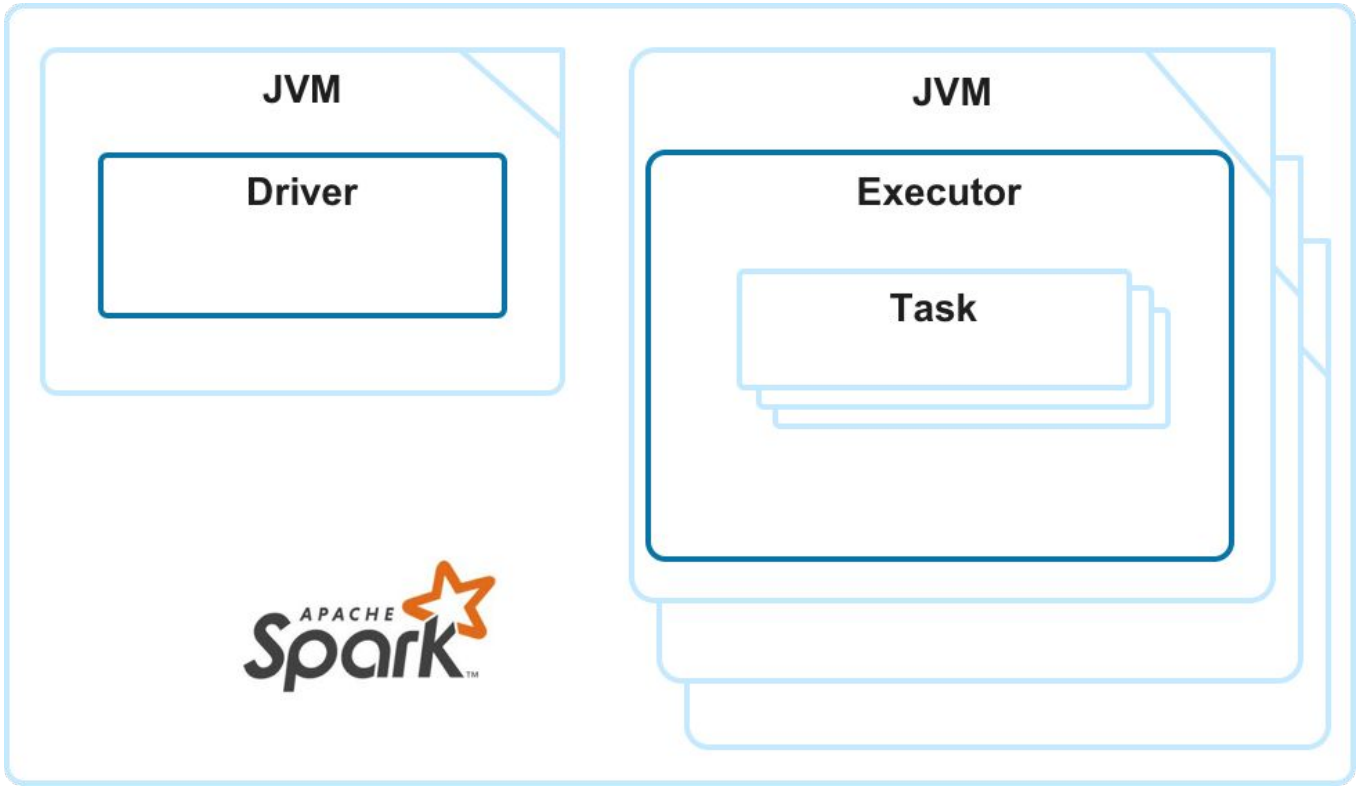
AGENDA

PERFORMANCE TUNING PITFALLS

THEORY BEHIND SPARKLENS

QUBOLE SPARKLENS TUNING EXAMPLE

SPARK APPLICATION STRUCTURE



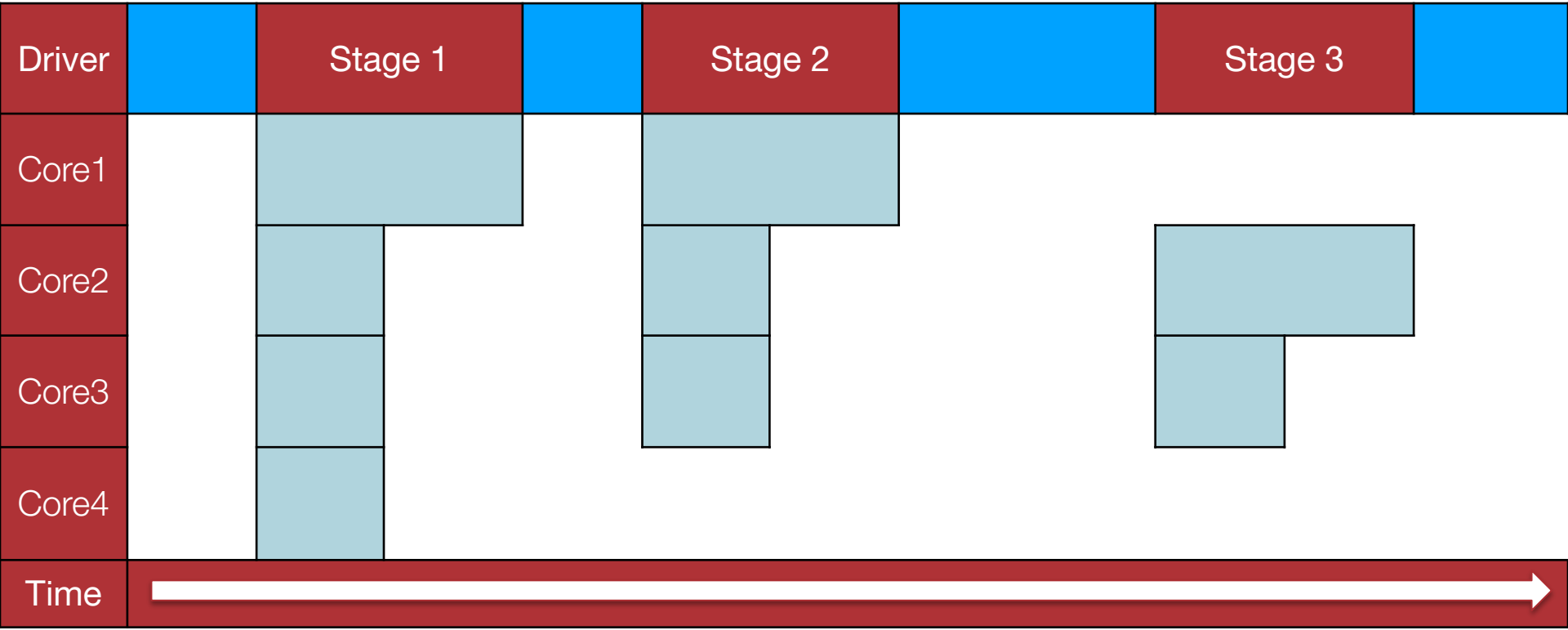
SPARK TUNING: COMMON APPROACHES

Brute-force	Job Diagnosis and Experiments
• Change number of executors • Memory parameter resizing for executors • Driver memory • Shuffle Partitions • Join strategies • And many more …….. *** Very unreliable approach**	• Spark App UI Analysis • Identify major bottlenecks • Driver/Executor log analysis • **Iterative experiments** based on above steps *** Costly in terms of time and developer cost**

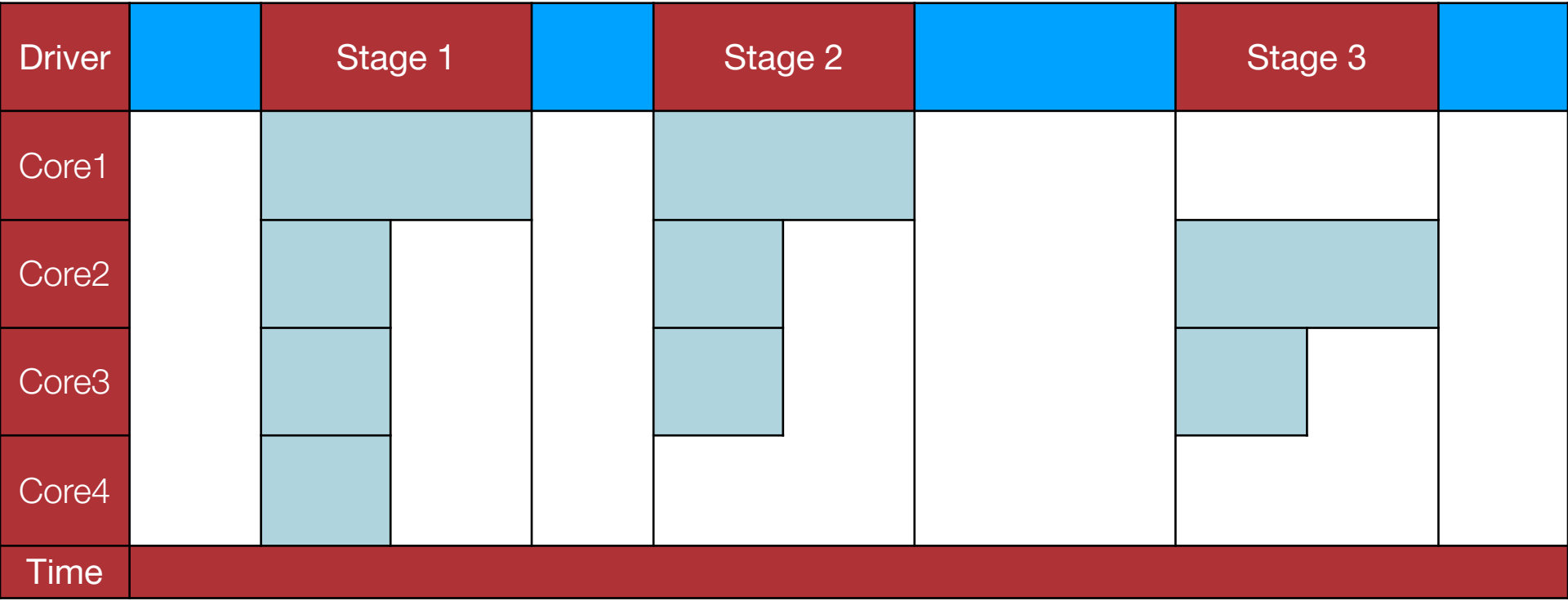
SPARK TUNING: PERFORMANCE KEY FACTORS

- Resource Utilization(Memory/CPU)
- Driver-only phases (Executors sitting idle)
- Tasks vs Num of Executors/Cores
- Skewed Tasks
- Scalability Limits (e.g. num-executors)

MINIMIZE DOING NOTHING

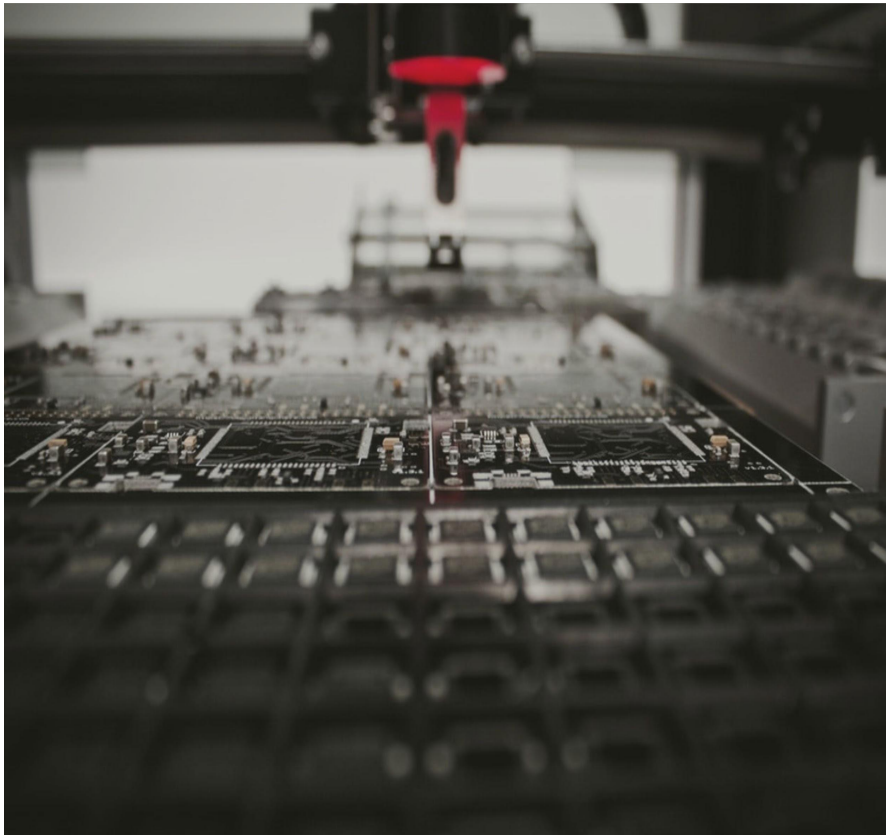


DRIVER SIDE COMPUTATIONS

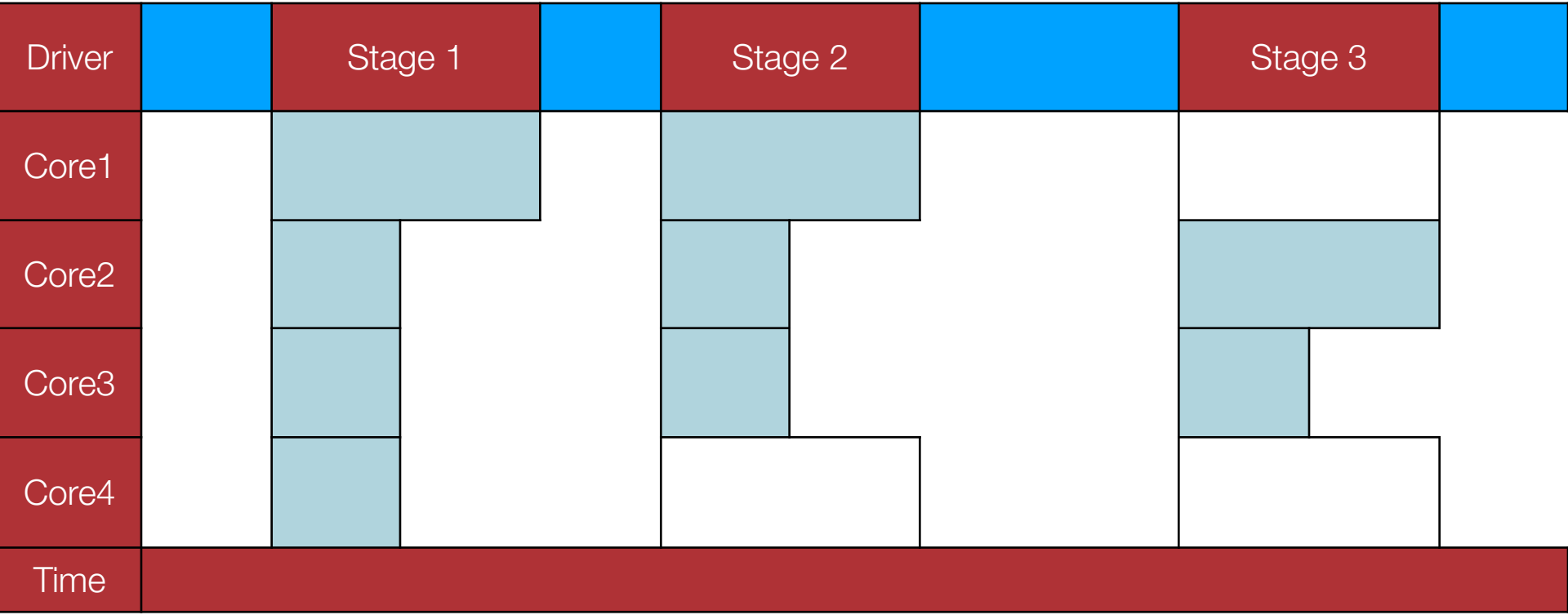


WHAT DRIVER DOES

- File listing & split computation
- Loading of hive tables
- FOC
- Collect
- df.toPandas()



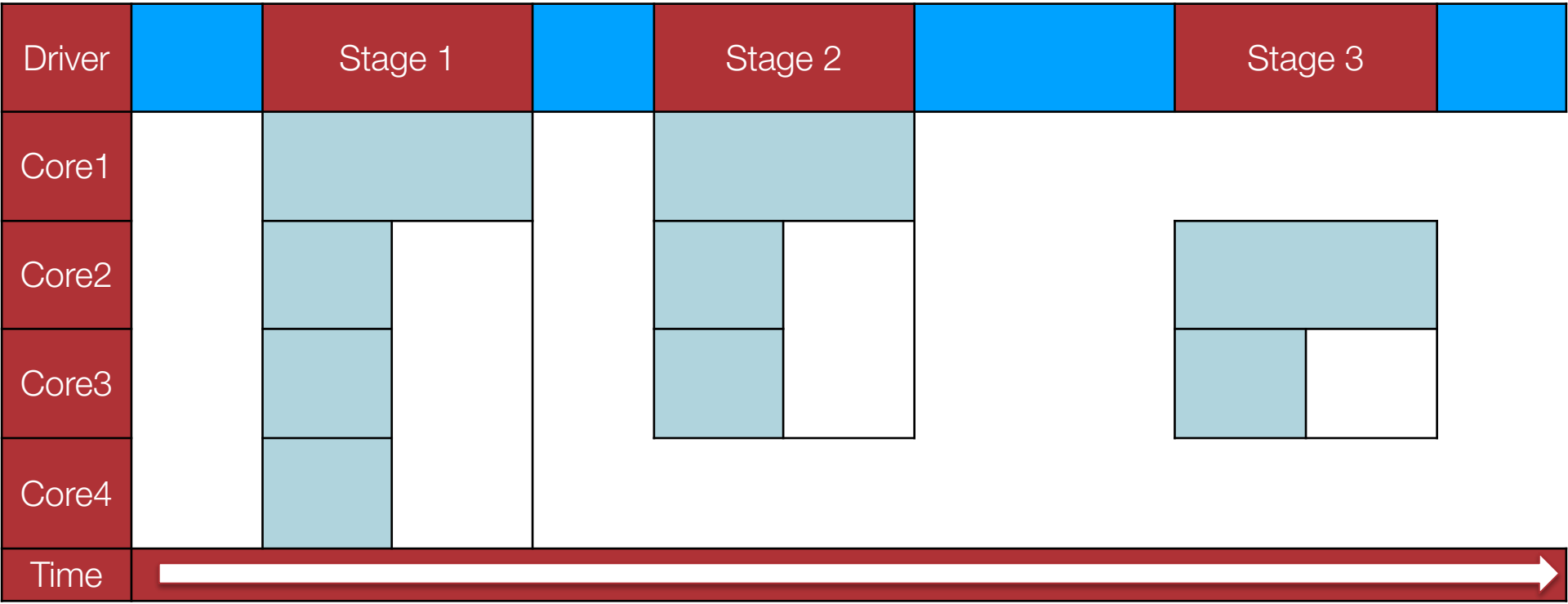
NOT ENOUGH TASKS



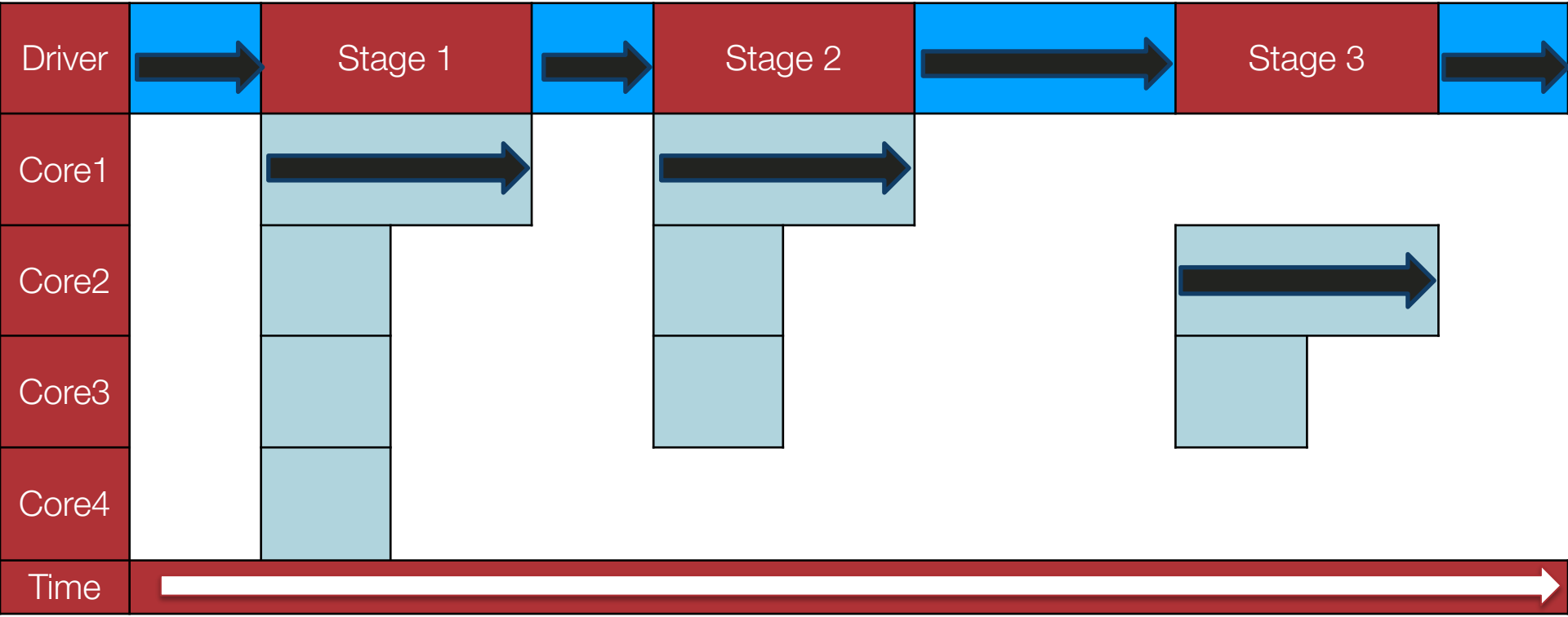
CONTROLLING NUMBER OF TASKS

- HDFS block size
- Min/max split size
- Default Parallelism
- Shuffle Partitions
- Repartitions

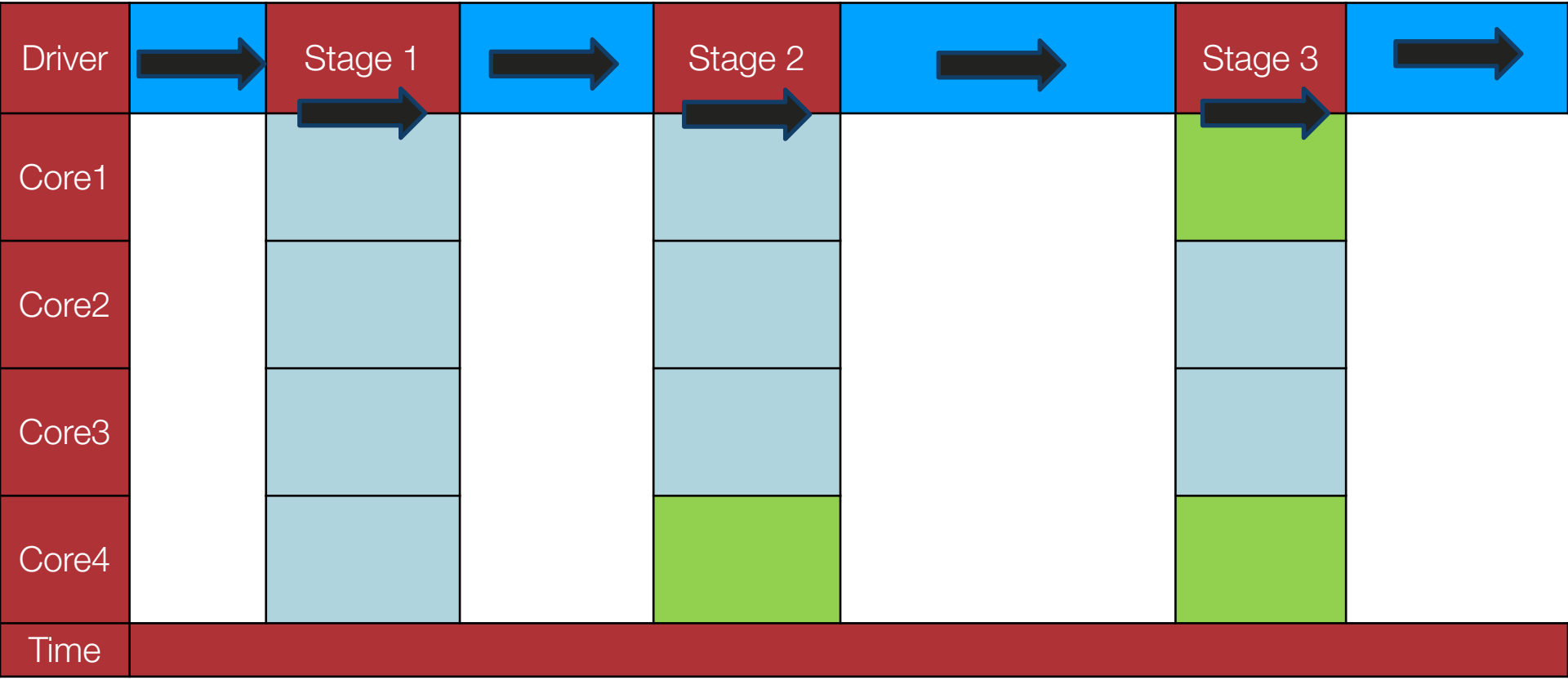
NON-UNIFORM TASKS: SKEW



CRITICAL PATH: LIMIT TO SCALABILITY

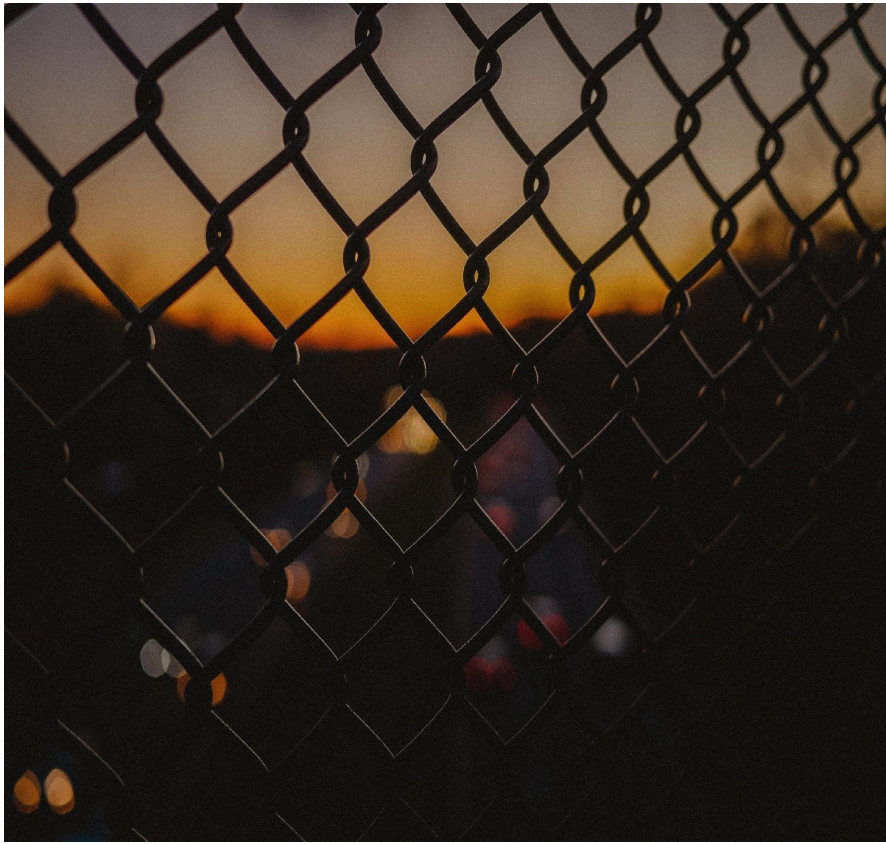


IDEAL APPLICATION TIME



CONTROLLING NUMBER OF TASKS

- Spark application is either executing in driver or in parallel in executors
- Child stage is not executed until all parent stages are complete
- Stage is not complete until all tasks of stage are complete



- An Open Source Spark Profiling Tool
- Runs with any Spark Deployment (Any Cloud, On-Prem or Distribution)
- Helps you take the right decision without many experiments (or trial and error)

https://github.com/qubole/sparklens

```
–packages qubole:sparklens:0.3.0-s_2.11
–conf spark.extraListener=com.qubole.sparklens.QuboleJobListener
```

For inline processing, add following extra command line options to spark-submit

Old event log files (history server)

```
–packages qubole:sparklens:0.3.0-s_2.11 --class
com.qubole.sparklens.app.ReporterApp dummy-arg <eventLogFile>
source=history
```

Special Sparklens output files (very small file with all the relevant data)

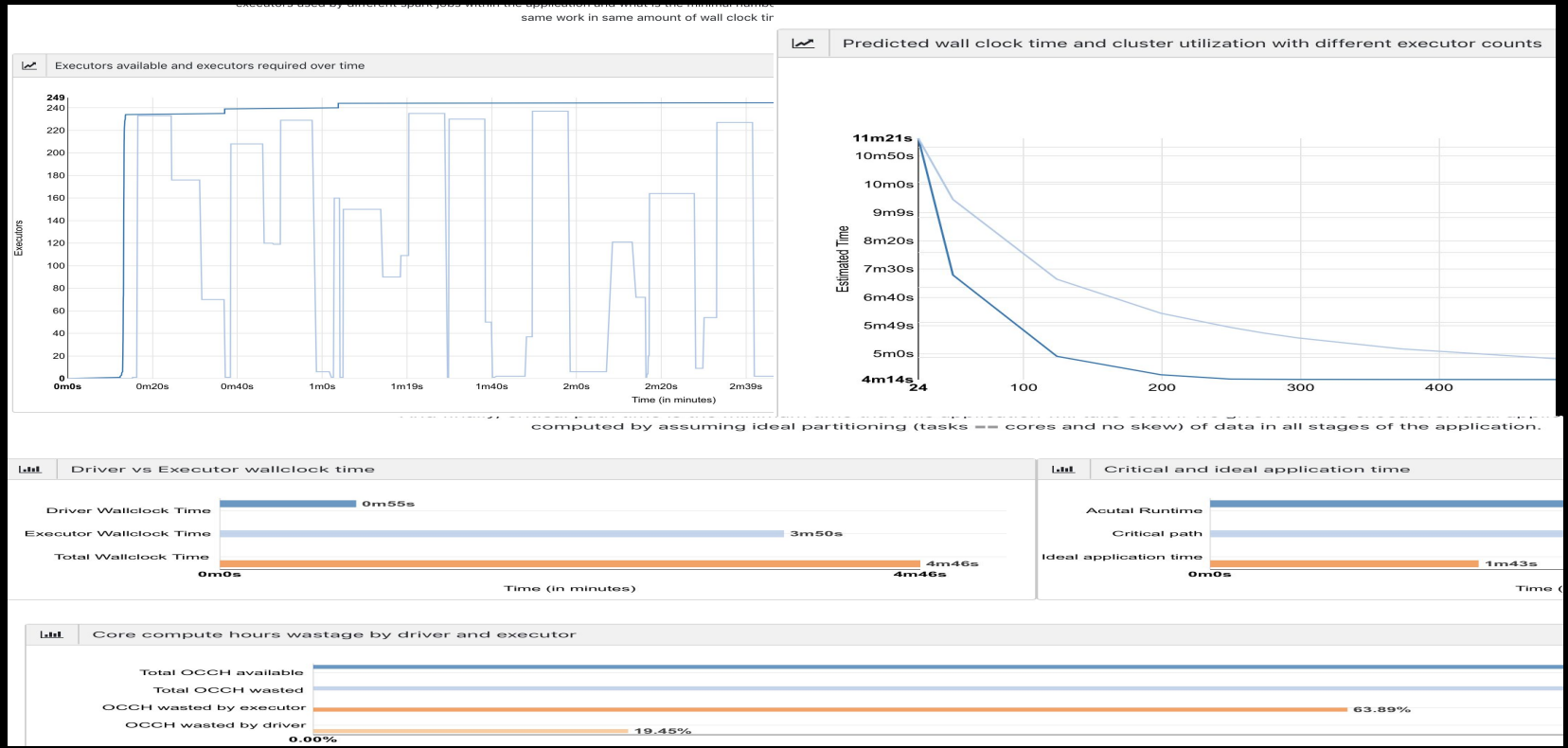
```
–packages qubole:sparklens:0.3.0-s_2.11 --class
com.qubole.sparklens.app.ReporterApp dummy-arg <eventLogFile>
```

Wall Clock Time

Critical Path Time

Ideal* Application Time

http://sparklens.qubole.net/



Qubole

SPARKLENS IN ACTION - I

PERFORMANCE TUNING - A SIMPLE SPARK SQL JOIN

Untitled(288247003) no tags

Spark Command

SQL Query Statement

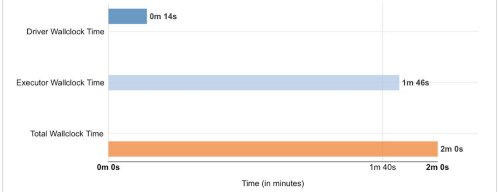
```
1 select b.s_state, count(*) as cnt from tpcds_orc_1000.store_sales a join tpcds_orc_1000.store b on (a.ss_store_sk=b.s_store_sk
2 group by  b.s_state
```

Uploaded At : 2019-07-14 9:00:14 AM
Original File Size : 76 KB
Original File name : application_1563088643227_0026.sparklens.json

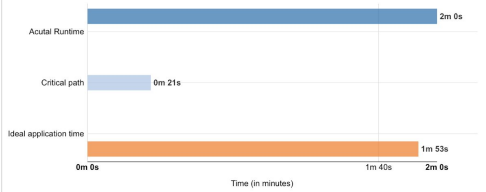
Efficiency Statistics | Simulation | Per Stage Metrics | Ideal Executors | Aggregate Metrics

The total spark application wallclock time can be divided into time spent in driver and time spent in executors. When a spark application spends too much time in the driver, it wastes the executors compute time. Executors can waste compute time, because of lack of tasks or skew. And finally, critical path time is the minimum time that this application will take even if we give it infinite executors. Ideal application time is computed by assuming i partitioning (tasks == cores and no skew) of data in all stages of the application.

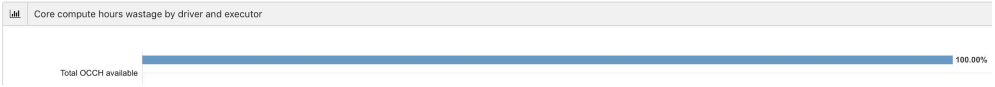
Driver vs Executor wallclock time



Critical and ideal application time



Core compute hours wastage by driver and executor



Uploaded At : 2019-07-14 9:00:14 AM
Original File Size : 76 KB
Original File name : application_1563088643227_0026.sparklens.json

Efficiency Statistics | Simulation | Per Stage Metrics | Ideal Executors | Aggregate Metrics

Not all stages are equally important. Start by looking at stages which occupy most of the wall clock time. Specifically look for lower PRatio and higher TaskSkew and fix accordingly.

Per stage metrics

PRatio : Number of tasks in stage divided by number of cores. Represents degree of parallelism in the stage

TaskSkew : Duration of largest task in stage divided by duration of median task.Represents degree of skew in the stage

Stage-ID	WallClock%	Task Count	WallClockTime Measured	MaxTaskMem	PRatio	TaskSkew	IO%
0	2.00	1	00m 02s	0.0 KB	0.13	1.00	0.0
1	2.00	1	00m 02s	0.0 KB	0.13	1.00	0.0
2	94.00	850	01m 39s	1.2 MB	106.25	3.87	99.7
3	0.00	200	00m 00s	257.0 MB	25.00	110.50	0.0

SPARK JOIN SQL (Modified)

Untitled(288247031) no tags

Spark Command | Save | Run | default | fsp

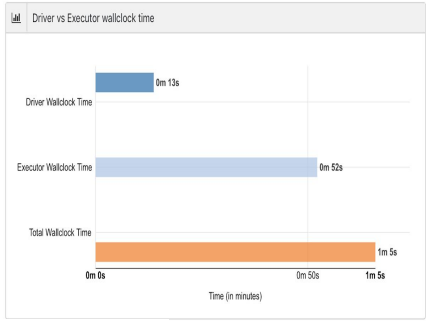
SQL | Query Statement

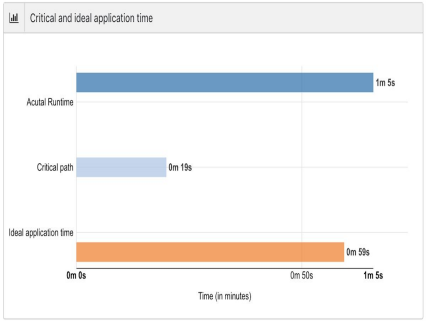
```
1 select b.s_state, sum(c) as cnt from
2 ( select ss_store_sk, count(*) as c from tpcds_orc_1000.store_sales group by ss_store_sk ) a
3 join tpcds_orc_1000.store b on (a.ss_store_sk=b.s_store_sk )
4 group by  b.s_state
```

Uploaded At : 2019-07-14 9:00:19 AM
Original File Size : 87 KB
Original File name : application_1563088643227_0028.sparklens.json

Efficiency Statistics | Simulation | Per Stage Metrics | Ideal Executors | Aggregate Metrics

The total spark application wallclock time can be divided into time spent in driver and time spent in executors. When a spark application spends too much time in the driver, it wastes the executors compute time. Executors can also waste compute time, because of lack of tasks or skew. And finally, critical path time is the minimum time that this application will take even if we give it infinite executors. Ideal application time is computed by assuming ideal partitioning (tasks == cores and no skew) of data in all stages of the application.





Uploaded At : 2019-07-14 9:00:19 AM
Original File Size : 87 KB
Original File name : application_1563088643227_0028.sparklens.json

Efficiency Statistics | Simulation | Per Stage Metrics | Ideal Executors | Aggregate Metrics

Not all stages are equally important. Start by looking at stages which occupy most of the wall clock time. Specifically look for lower PRatio and higher TaskSkew and fix accordingly.

Per stage metrics

PRatio : Number of tasks in stage divided by number of cores. Represents degree of parallelism in the stage

TaskSkew : Duration of largest task in stage divided by duration of median task.Represents degree of skew in the stage

Stage-ID	WallClock%	Task Count	WallClockTime Measured	MaxTaskMem	PRatio	TaskSkew	IO%
0	5.00	1	00m 02s	0.0 KB	0.13	1.00	0.0
1	5.00	1	00m 02s	0.0 KB	0.13	1.00	0.0
2	84.00	850	00m 43s	1.0 MB	106.25	2.74	97.4
3	4.00	200	00m 02s	258.2 MB	25.00	5.35	0.0
4	0.00	200	00m 00s	257.0 MB	25.00	19.00	0.0

Qubole

SPARKLENS IN ACTION - II

PERFORMANCE TUNING **603** LINES OF UNFAMILIAR SCALA CODE

SPARKLENS: FIRST PASS

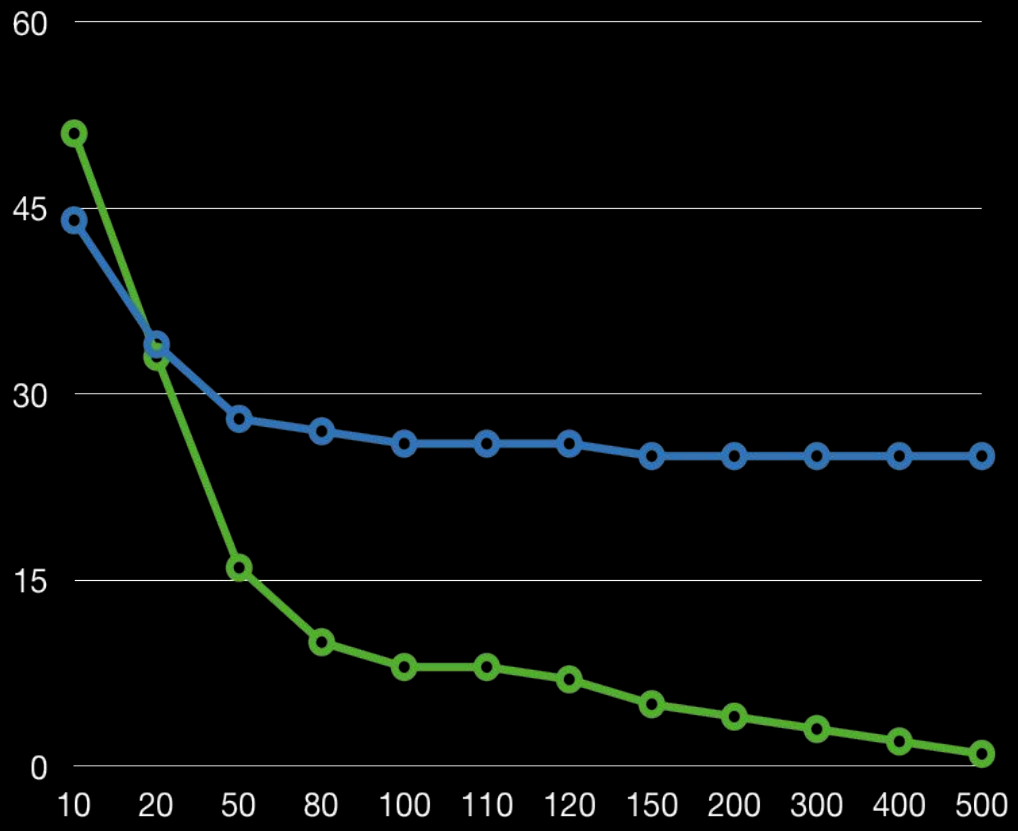
```
Driver WallClock      41m 40s   26%
Executor WallClock 117m 03s   74%
Total WallClock     158m 44s

Critical Path       127m 41s
Ideal Application    43m 32s
```

- The application had too many stages (697)
- The Critical Path Time was 3X the Ideal Application Time
- Instead of letting spark write to hive table, the code was doing serial writes to each partition, in a loop
- We changed the code to let spark write to partitions in parallel

Driver WallClock	02m 28s	9%
Executor WallClock	24m 03s	91%
Total WallClock	26m 32s	
Critical Path	25m 27s	
Ideal Application	04m 48s	

Qubole

SPARKLENS PERFORMANCE PREDICTION



Count	Time	Utilisation
10	44m	51%
20	34m	33%
50	28m	16%
80	27m	10%
100	**26m**	8%
110	26m	8%
120	26m	7%
150	25m	5%
200	25m	4%
300	25m	3%
400	25m	2%
500	25m	1%

```
ECCH available 320h 50m
ECCH used       31h 00m  9%
ECCH wasted    289h 50m 91%
```

ECCH: Executor Core Compute Hour

PER STAGE METRICS

Stage-ID	WallClock Stage%	Core ComputeHours	Task Count	PRatio	-----Task------ Skew	StageSkew
0	0.27	00h 00m	2	0.00	1.00	0.78
1	0.37	00h 00m	10	0.01	1.05	0.85
33	85.84	03h 18m	10	0.01	1.07	1.00

Stage-ID	OIRatio	|* ShuffleWrite%	ReadFetch%	GC% *|
0	0.00	|* 0.00	0.00	3.03 *|
1	0.00	|* 0.00	0.00	2.02 *|
33	0.00	|* 0.00	0.00	0.23 *|

CCH	3h 18m
Task Count	10
Total Cores	800

OBSERVATIONS & ACTIONS

- 85% of time spent in a single stage with very low number of tasks.
- 91% compute wasted on executor side.
- Found that repartition(10) was called somewhere in code, resulting in only 10 tasks. Removed it.
- Also increased the spark.sql.shuffle.partitions from default 200 to 800

SPARKLENS: THIRD PASS

Driver WallClock	02m 34s	26%
Executor WallClock	07m 13s	74%
Total WallClock	09m 48s	
Critical Path	07m 18s	
Ideal Application	07m 09s	

Qubole

THANK YOU