HW #1 Spark SQL

2조 김정훈,한정민,모유찬

(1) SQL query (1점*Query2개)

문제1) select project,title,count from wiki
where project='de' and title<>'Woodkid' and count >=800 and count<1000
order by count desc

문제2) select owner,avg(count) as avg_count from wiki
natural join owner
group by owner

(2) jupyter notebook에 보여지는 Query 실행결과 스크린샷 (1점*Query2개)

문제1)

```
In [30]: # df.createOrReplaceTempView("...")
         # selected = ss.sql("...")
         # selected.show()
         df.createOrReplaceTempView('wiki') # create temp view
         no1 = ss.sql("select project,title,count ₩
                     from wiki ₩
                     where project='de' and title<>'Woodkid' and count >=800 and count<1000 ₩
                     order by count desc")
         no1.show()
```

```
+-------+--------------------+-----+
|project|               title|count|
+-------+--------------------+-----+
|     de|Wikipedia:Auskunf...|  900|
|     de|Spezial:Beobachtu...|  882|
|     de|Spezial:Beobachtu...|  804|
+-------+--------------------+-----+
```

문제2)

```
In [31]: ownertable_raw = [('Woodkid','Lila'),('Sia','Jane'),('Ryuichi_Sakamoto','Sam')]
         ownerrdd = sc.parallelize(ownertable_raw)
         ownerdf = ss.createDataFrame(ownerrdd,['title','owner'])
         ownerdf.show()
```

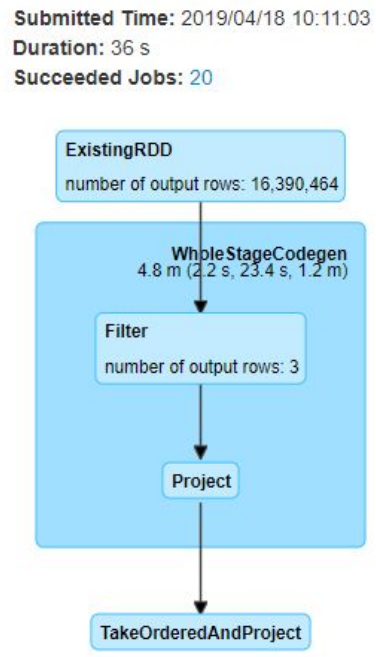
```
+---------------+-----+
|          title|owner|
+---------------+-----+
|        Woodkid| Lila|
|            Sia| Jane|
|Ryuichi_Sakamoto|  Sam|
+---------------+-----+
```

```
In [32]: ownerdf.createOrReplaceTempView('owner')
         no2 = ss.sql('select owner,avg(count) as avg_count₩
                     from wiki₩
                     natural join owner₩
                     group by owner')
         no2.show()
```

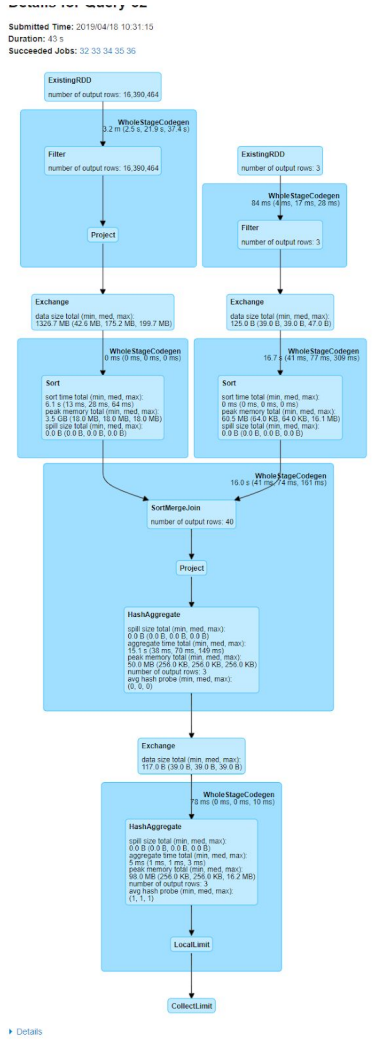
```
+-----+---------+
|owner|avg_count|
+-----+---------+
| Lila|      2.3|
|  Sam|     14.0|
| Jane|     10.0|
+-----+---------+
```

(3) Spark Web UI 'SQL' tab 에서 보여지는 Query plan 그래프의 스크린샷 (1점*Query2개)

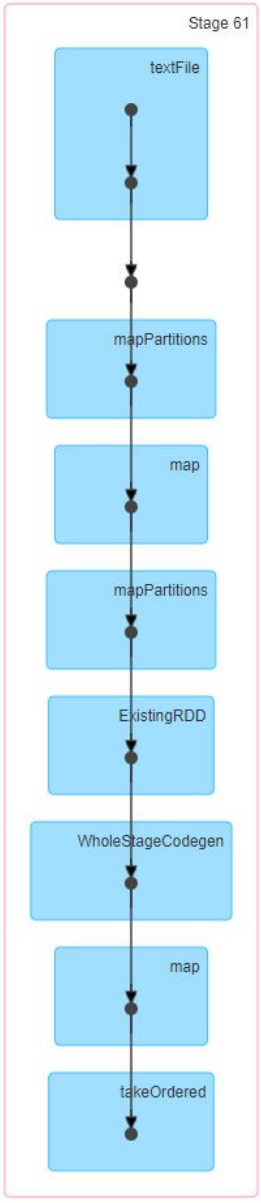
문제1)



문제2)



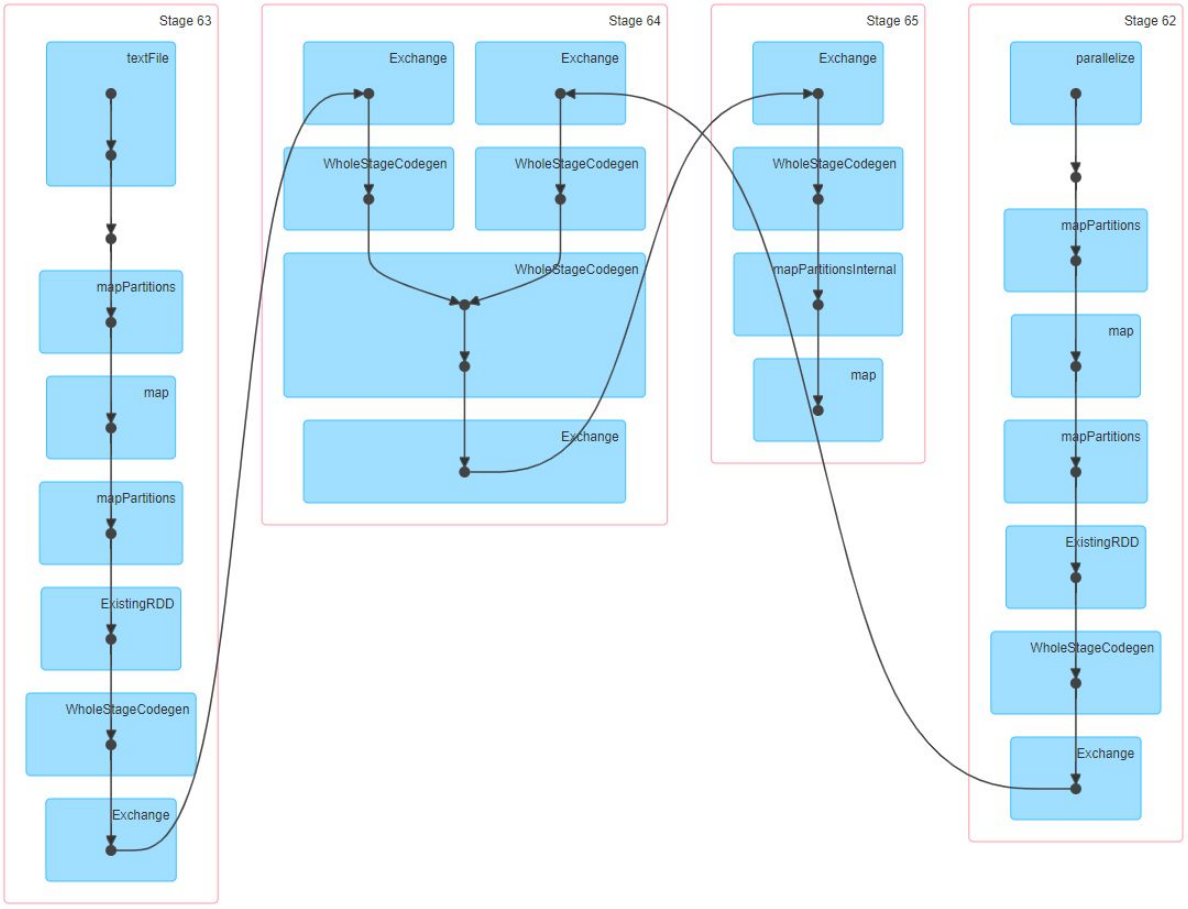
(4) Spark Web UI ‘Jobs’ tab 에서 보여지는 DAG Visualization 그래프의 스크린샷
(1점*Query2개)
문제1)



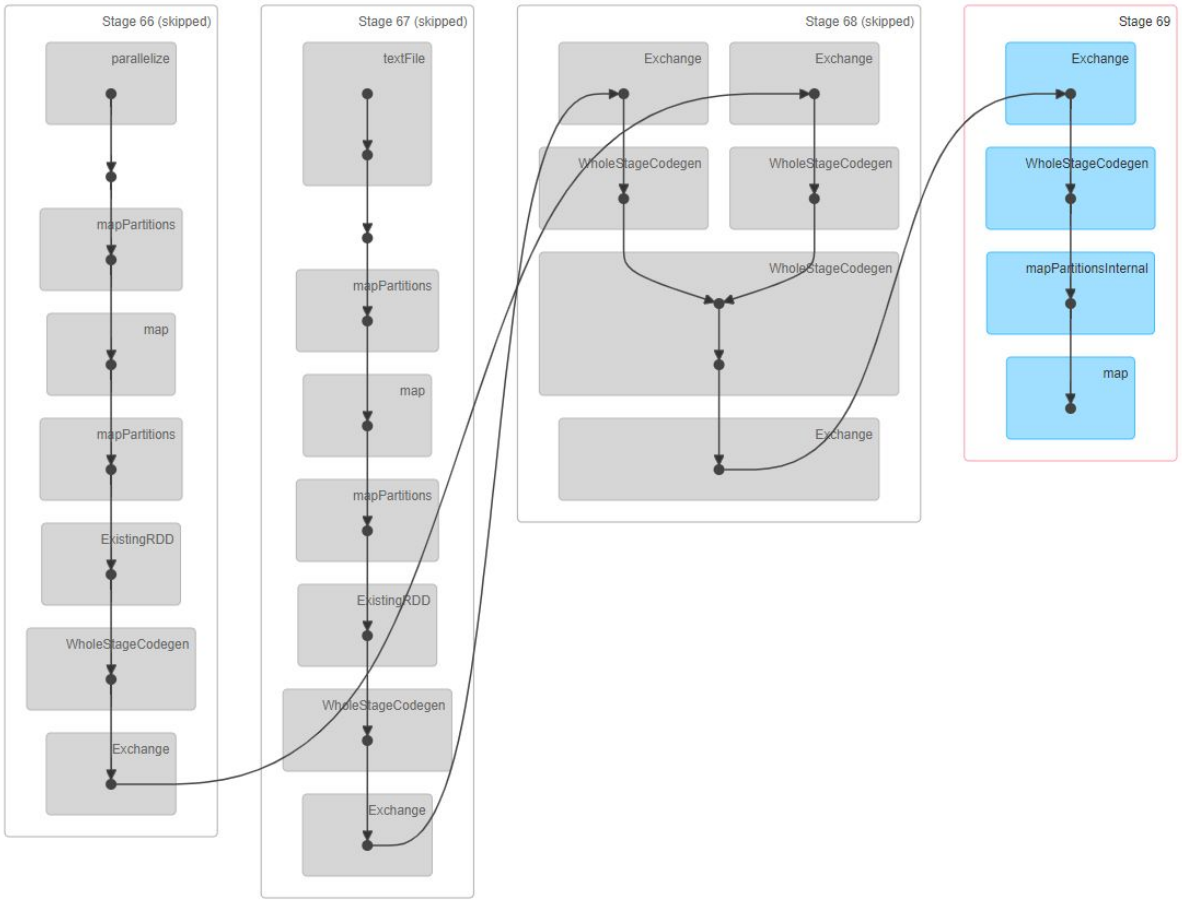
문제2) 총 5개의 Job 으로 진행됨

Job Id ▾	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
36	showString at NativeMethodAccessorImpl.java:0 showString at NativeMethodAccessorImpl.java:0	2019/04/18 10:31:58	0.2 s	1/1 (3 skipped)	75/75 (215 skipped)
35	showString at NativeMethodAccessorImpl.java:0 showString at NativeMethodAccessorImpl.java:0	2019/04/18 10:31:57	0.2 s	1/1 (3 skipped)	100/100 (215 skipped)
34	showString at NativeMethodAccessorImpl.java:0 showString at NativeMethodAccessorImpl.java:0	2019/04/18 10:31:57	48 ms	1/1 (3 skipped)	20/20 (215 skipped)
33	showString at NativeMethodAccessorImpl.java:0 showString at NativeMethodAccessorImpl.java:0	2019/04/18 10:31:57	25 ms	1/1 (3 skipped)	4/4 (215 skipped)
32	showString at NativeMethodAccessorImpl.java:0 showString at NativeMethodAccessorImpl.java:0	2019/04/18 10:31:15	42 s	4/4	216/216

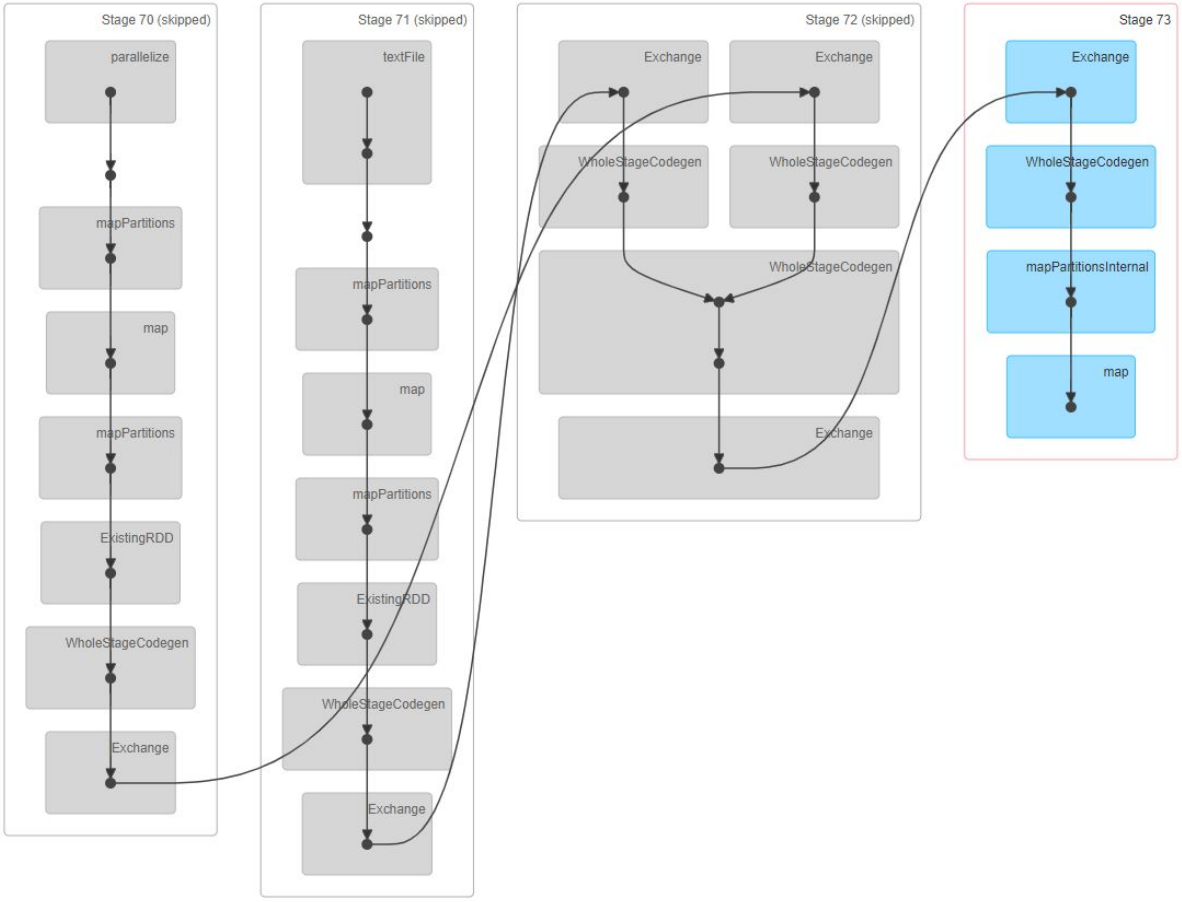
JobID 32 (Stages 4/4)



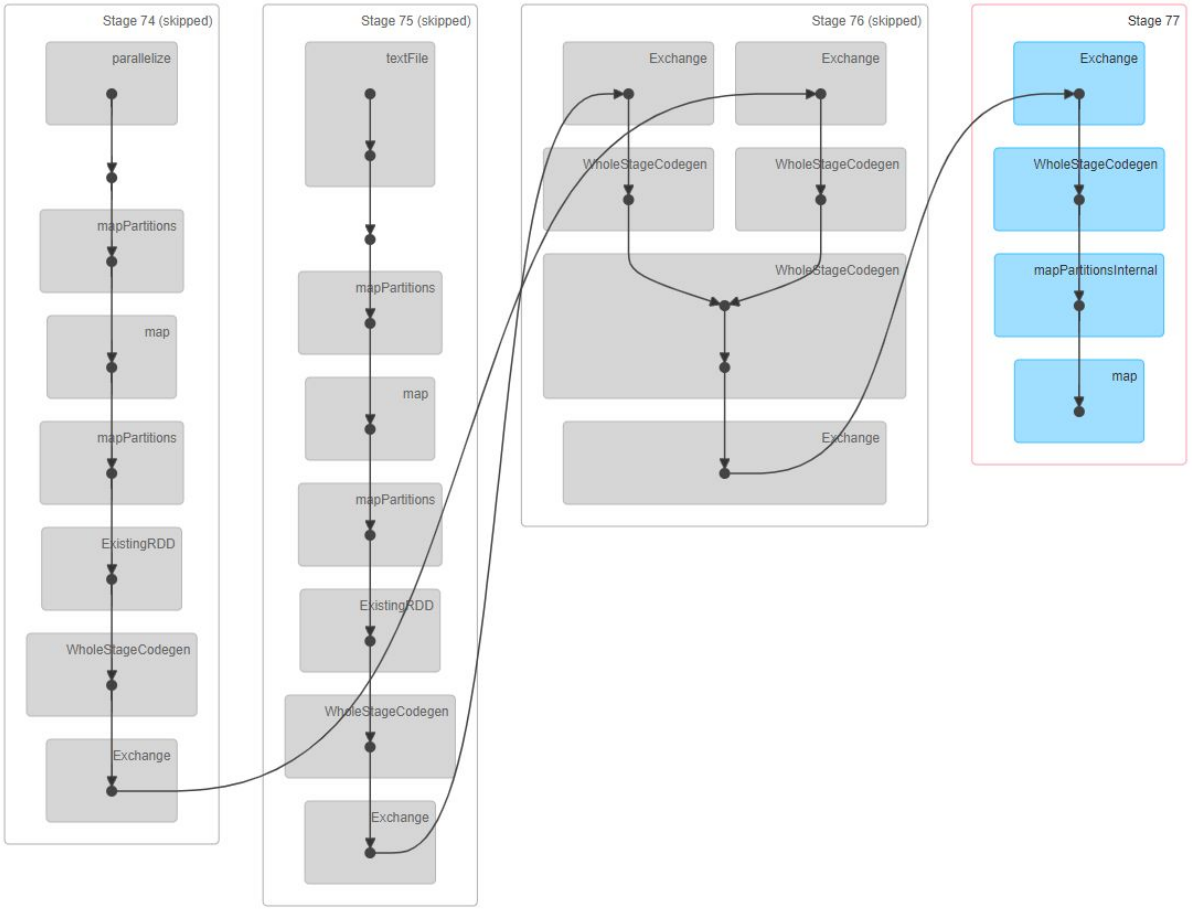
JobID 33 (1/1 (3 skipped))



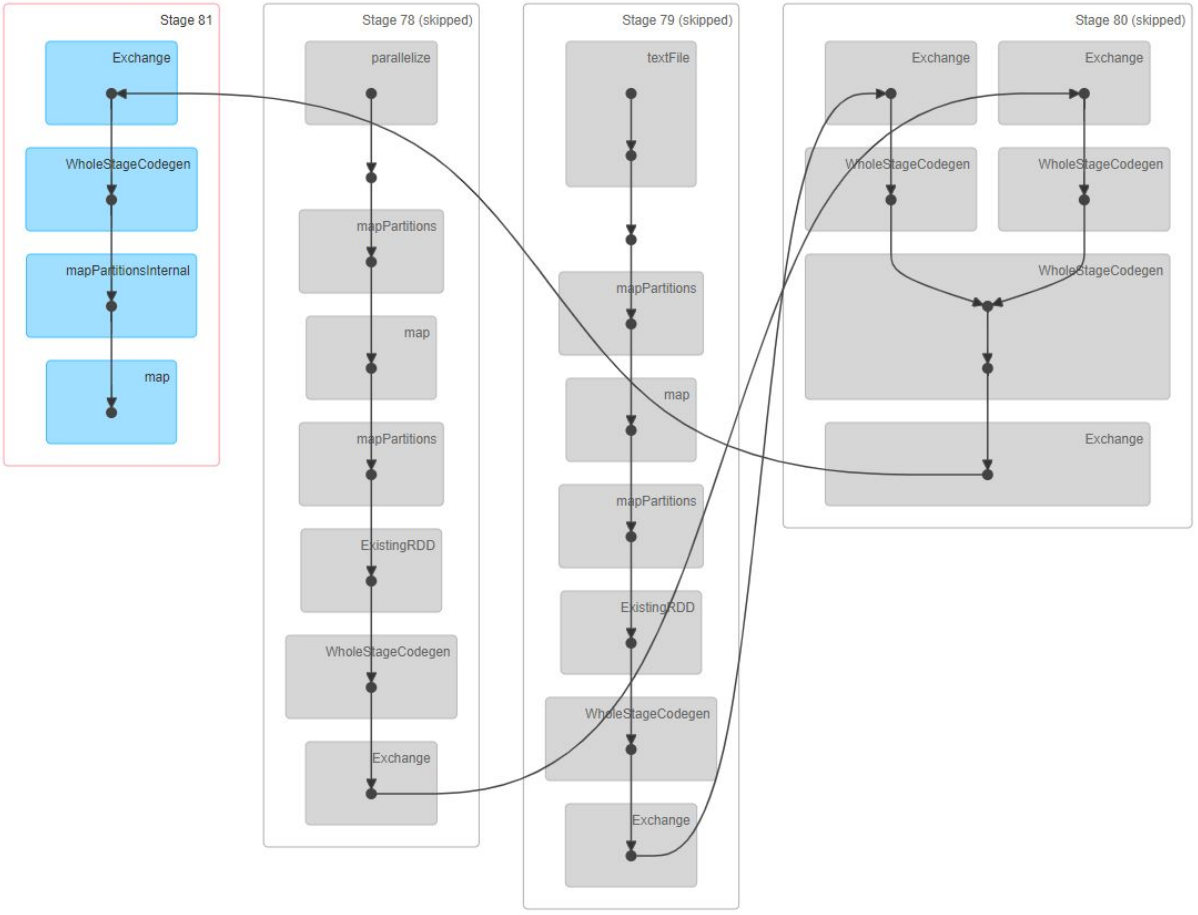
JobID 34 (1/1 (3 skipped))



JobID 35 (1/1 (3 skipped))



JobID 36 (1/1 (3 skipped))



(5) Spark Web UI 'Stages' tab에서 보여지는, 해당 Query에 대응하는 Job에 속하는 모든 Stage들의 'Summary Metrics for N Completed Tasks' 테이블의 스크린샷 (1점*Query2개)

문제1)

62	showString at NativeMethodAccessorImpl.java:0 +details	2019/04/18 10:31:15	98 ms	6/6				258.0 B

Summary Metrics for 9 Completed Tasks

Metric	Min	25th percentile	Median	75th percentile	Max
Duration	2 s	7 s	24 s	32 s	34 s
GC Time	40 ms	0.1 s	0.3 s	0.4 s	0.4 s
Input Size / Records	39.2 MB / 148050	96.7 MB / 521200	128.1 MB / 2133942	128.1 MB / 3159349	128.1 MB / 3261669

문제2)

Stage Id	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
81	showString at NativeMethodAccessorImpl.java:0 +details	2019/04/18 10:31:58	0.1 s	75/75			78.0 B	
77	showString at NativeMethodAccessorImpl.java:0 +details	2019/04/18 10:31:57	0.2 s	100/100			156.0 B	
73	showString at NativeMethodAccessorImpl.java:0 +details	2019/04/18 10:31:57	40 ms	20/20				
69	showString at NativeMethodAccessorImpl.java:0 +details	2019/04/18 10:31:57	21 ms	4/4				
65	showString at NativeMethodAccessorImpl.java:0 +details	2019/04/18 10:31:57	15 ms	1/1				
64	showString at NativeMethodAccessorImpl.java:0 +details	2019/04/18 10:31:54	3 s	200/200			550.8 MB	234.0 B
63	showString at NativeMethodAccessorImpl.java:0 +details	2019/04/18 10:31:15	39 s	9/9	966.5 MB			550.8 MB

Details for Stage 63 (Attempt 0)

Total Time Across All Tasks: 3.2 min
Locality Level Summary: Any: 9
Input Size / Records: 966.5 MB / 16390464
Shuffle Write: 550.8 MB / 16390464

- DAG Visualization
- Show Additional Metrics
- Event Timeline

Summary Metrics for 9 Completed Tasks

Metric	Min	25th percentile	Median	75th percentile	Max
Duration	3 s	8 s	22 s	32 s	38 s
GC Time	57 ms	0.2 s	0.5 s	0.6 s	0.6 s
Input Size / Records	39.2 MB / 148050	96.7 MB / 521200	128.1 MB / 2133942	128.1 MB / 3159349	128.1 MB / 3261669
Shuffle Write Size / Records	12.5 MB / 148050	30.0 MB / 521200	73.8 MB / 2133942	92.1 MB / 3159349	93.6 MB / 3261669

Details for Stage 64 (Attempt 0)

Total Time Across All Tasks: 18 s
Locality Level Summary: Node local: 200
Shuffle Read: 550.8 MB / 16390467
Shuffle Write: 234.0 B / 3

- DAG Visualization
- Show Additional Metrics
- Event Timeline

Summary Metrics for 200 Completed Tasks

Metric	Min	25th percentile	Median	75th percentile	Max
Duration	47 ms	77 ms	86 ms	98 ms	0.2 s
GC Time	0 ms	0 ms	0 ms	0 ms	74 ms
Shuffle Read Size / Records	2.7 MB / 80568	2.7 MB / 81610	2.8 MB / 81978	2.8 MB / 82249	2.8 MB / 83177
Shuffle Write Size / Records	0.0 B / 0	0.0 B / 0	0.0 B / 0	0.0 B / 0	78.0 B / 1

Details for Stage 65 (Attempt 0)

Total Time Across All Tasks: 6 ms
Locality Level Summary: Process local: 1

- DAG Visualization
- Show Additional Metrics
- Event Timeline

Summary Metrics for 1 Completed Tasks

Metric	Min	25th percentile	Median	75th percentile	Max
Duration	6 ms	6 ms	6 ms	6 ms	6 ms
GC Time	0 ms	0 ms	0 ms	0 ms	0 ms

Details for Stage 69 (Attempt 0)

Total Time Across All Tasks: 21 ms
Locality Level Summary: Process local: 4

- DAG Visualization
- Show Additional Metrics
- Event Timeline

Summary Metrics for 4 Completed Tasks

Metric	Min	25th percentile	Median	75th percentile	Max
Duration	1 ms	3 ms	5 ms	12 ms	12 ms
GC Time	0 ms	0 ms	0 ms	0 ms	0 ms

Details for Stage 73 (Attempt 0)

Total Time Across All Tasks: 60 ms
Locality Level Summary: Process local: 20

- DAG Visualization
- Show Additional Metrics
- Event Timeline

Summary Metrics for 20 Completed Tasks

Metric	Min	25th percentile	Median	75th percentile	Max
Duration	1 ms	2 ms	2 ms	4 ms	10 ms
GC Time	0 ms	0 ms	0 ms	0 ms	0 ms

Details for Stage 77 (Attempt 0)

Total Time Across All Tasks: 0.3 s
Locality Level Summary: Node local: 2; Process local: 98
Shuffle Read: 156.0 B / 2

- DAG Visualization
- Show Additional Metrics
- Event Timeline

Summary Metrics for 100 Completed Tasks

Metric	Min	25th percentile	Median	75th percentile	Max
Duration	1 ms	1 ms	2 ms	3 ms	9 ms
GC Time	0 ms	0 ms	0 ms	0 ms	0 ms
Shuffle Read Size / Records	0.0 B / 0	0.0 B / 0	0.0 B / 0	0.0 B / 0	78.0 B / 1

Details for Stage 81 (Attempt 0)

Total Time Across All Tasks: 0.2 s
Locality Level Summary: Node local: 1; Process local: 74
Shuffle Read: 78.0 B / 1

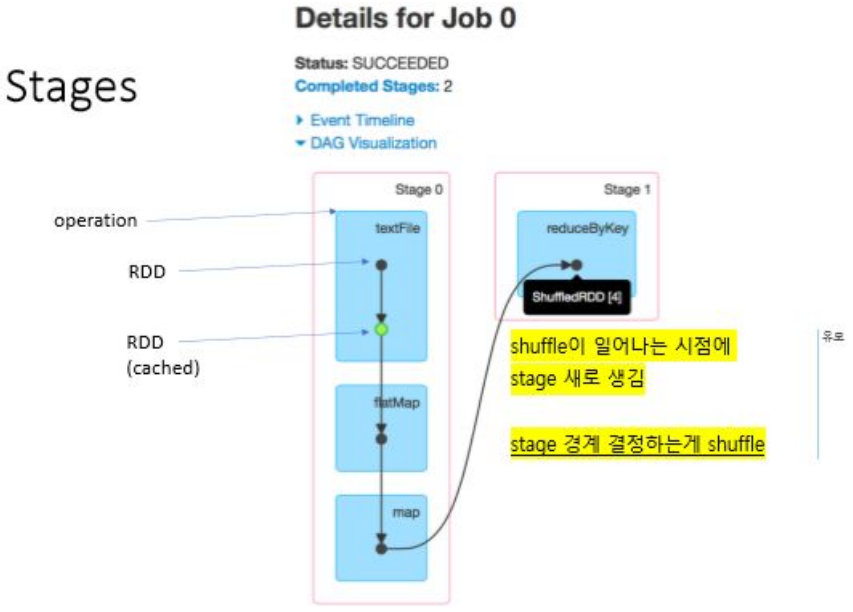
- DAG Visualization
- Show Additional Metrics
- Event Timeline

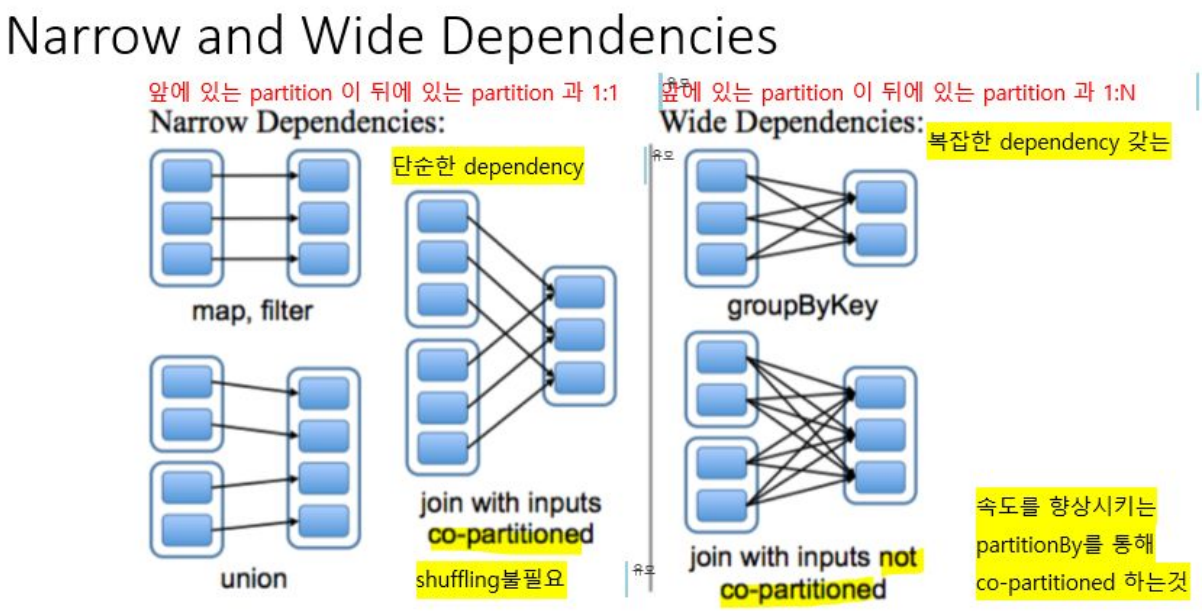
Summary Metrics for 75 Completed Tasks

Metric	Min	25th percentile	Median	75th percentile	Max
Duration	1 ms	1 ms	2 ms	3 ms	12 ms
GC Time	0 ms	0 ms	0 ms	0 ms	0 ms
Shuffle Read Size / Records	0.0 B / 0	0.0 B / 0	0.0 B / 0	0.0 B / 0	78.0 B / 1

(6) (보너스 점수 2점 - 각각 1점) 수행하면서 발견한 재미있는 현상 2개에 대해 기술
발견한 사항)

1) DAG 관점 발견 사항 (문제1 vs 문제2)
   a) 문제1 : 1 Stage 에서 완료됨 => shuffle 이 일어나지 않았다는 점. 그것은 Narrow dependencies(map,filter,union등) 동일한 partition에 의한 연산만 수행된다고 판단됨
   b) 문제2 : 총 5job(총8stage : 12개 stage 는 skipped) 에 거쳐 연산이 수행되는데, 잦은 shuffling이 발생하는 것으로 보아 Wide dependencies 연산으로 보인다. 해당 연산이 많이 발생하는 이유로는 join, groupBy(avg 연산)을 진행할 때 partition 간의 shuffling 이 다수 일어난 것으로 보인다.(하지만 확인 결과 최초 job 외에서는 계획과는 다르게 실제로는 shuffling 이 발생하지 않는 것으로 보인다)





2) 문제2의 DAG Graph의 수많은 Skipped stage와 Summary Metrics 의 Garbage collection time 로 부터 들은 궁금점을 Stack over flow를 통해 해답을 얻었다. 일단 Stage 에서 Skipped 된 것은 후속 stage 에서 HDFS로부터 데이터를 가져오는 것이 아닌 자주 연산되는 것을 caching하여 연산을 최적화 한것으로 보이고, 첫 Stage외에 모든 Garbage Collection time 이 0 인 점이 모든 것을 in mermory 에서 caching하여 후속 처리를 하는 것에 따라 발생 된 결과로 생각된다.

https://stackoverflow.com/questions/34580662/what-does-stage-skipped-mean-in-apache-spark-web-ui

일반적으로 캐시에서 데이터를 가져오고 지정된 스테이지를 다시 실행할 필요가 없음을 의미합니다. 다음 단계에서 셔플이 필요함을 보여주는 DAG와 일치합니다 (`reduceByKey`). 셔플 링이있을 때마다 Spark는 자동으로 생성 된 데이터를 캐시합니다 .

Shuffle은 또한 디스크에 많은 수의 중간 파일을 생성합니다. Spark 1.3에서이 파일들은 해당 RDD가 더 이상 사용되지 않고 가비지 수집 될 때까지 보존됩니다. 이는 계보가 다시 계산 될 때 셔플 파일을 다시 만들 필요가 없도록하기 위해 수행됩니다.