Cody Lieu
11/20/2015
cal53

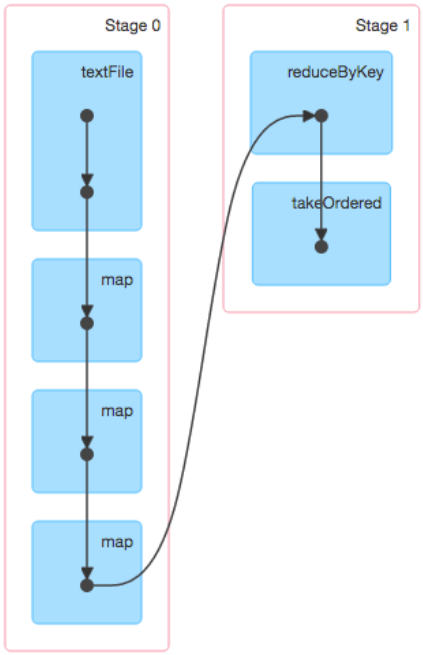
CS 290: Data Engineering HW 5

*** For convenience I've included all of the analysis from HW 2 to more easily make comparisons. The way the document is structured is for each question, the original analysis is shown and then the analysis for DataFrame/SparkSQL is on the following next page.**

TenMostFrequentVisitors.scala without DataFrame/SparkSQL

Completed Stages (2)

Stage Id	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
1	takeOrdered at TenMostFrequentVisitors.scala:24 +details	2015/10/16 04:00:01	3 s	8/8			62.9 MB	
0	map at TenMostFrequentVisitors.scala:23 +details	2015/10/16 03:59:50	11 s	8/8	917.5 MB			62.9 MB

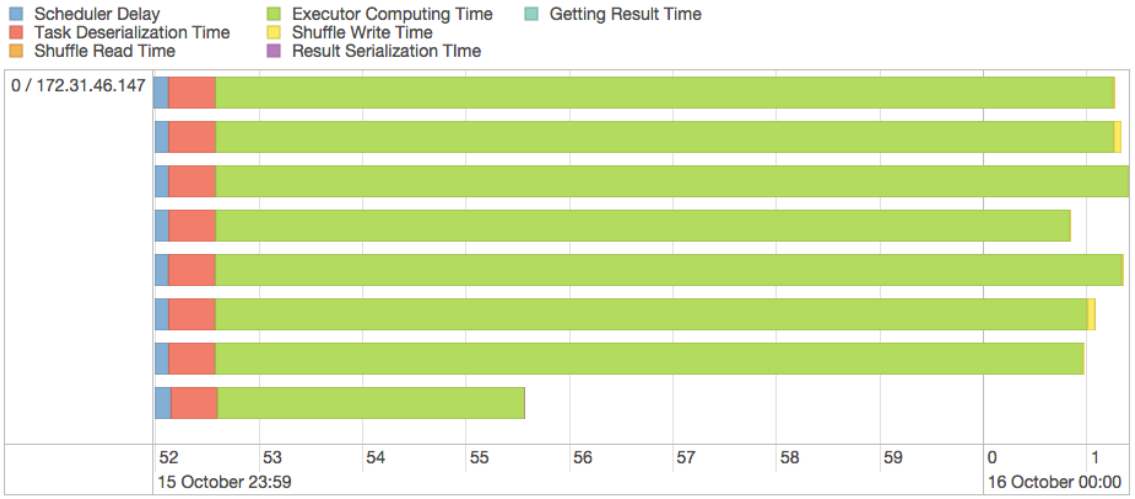


The application runs one job for computing the ten most frequent visitors. The job was comprised of 2 stages, which took a total of 14 seconds with Stage 0 taking 11 seconds and Stage 1 taking 3 seconds. Each stage had 8 tasks. 5 of the tasks in Stage 0 (0-indexed) took approximately 9 seconds to run in parallel and all ran on Executor 0 (0-indexed). All of the details of interest as well as the input and shuffle data sizes produced by each of the longest running tasks are shown below.

Tasks

Index	ID	Attempt	Status	Locality Level	Executor ID / Host	Launch Time	Duration	GC Time	Input Size / Records	Write Time	Shuffle Write Size / Records	Errors
0	0	0	SUCCESS	ANY	0 / 172.31.46.147	2015/10/16 03:59:51	9 s	0.7 s	128.0 MB (hadoop) / 668105	17 ms	8.4 MB / 505136	
1	1	0	SUCCESS	ANY	0 / 172.31.46.147	2015/10/16 03:59:51	9 s	0.7 s	128.0 MB (hadoop) / 668318	72 ms	8.8 MB / 527442	
2	2	0	SUCCESS	ANY	0 / 172.31.46.147	2015/10/16 03:59:51	9 s	0.7 s	128.0 MB (hadoop) / 656531	24 ms	9.6 MB / 545865	
3	3	0	SUCCESS	ANY	0 / 172.31.46.147	2015/10/16 03:59:51	8 s	0.7 s	128.0 MB (hadoop) / 644414	16 ms	8.8 MB / 507149	
5	5	0	SUCCESS	ANY	0 / 172.31.46.147	2015/10/16 03:59:51	9 s	0.7 s	128.0 MB (hadoop) / 664764	74 ms	8.0 MB / 452506	
4	4	0	SUCCESS	ANY	0 / 172.31.46.147	2015/10/16 03:59:51	9 s	0.7 s	128.0 MB (hadoop) / 666103	19 ms	9.1 MB / 521546	
6	6	0	SUCCESS	ANY	0 / 172.31.46.147	2015/10/16 03:59:51	8 s	0.7 s	128.0 MB (hadoop) / 680423	21 ms	8.8 MB / 503828	
7	7	0	SUCCESS	ANY	0 / 172.31.46.147	2015/10/16 03:59:51	3 s	57 ms	21.5 MB (hadoop) / 105705	6 ms	1573.5 KB / 88845	

Stage 0 was comprised of reading in the White House data file and then performing the map functions. Stage 1 was comprised of the reduceByKey call and then the takeOrdered call, which printed the final results. Looking at the bar graph below, Task 2 (0 indexed) of Stage 0 took the longest to run on Executor 0 and had an input size of 128.0 MB and a shuffle size of 9.6 MB.

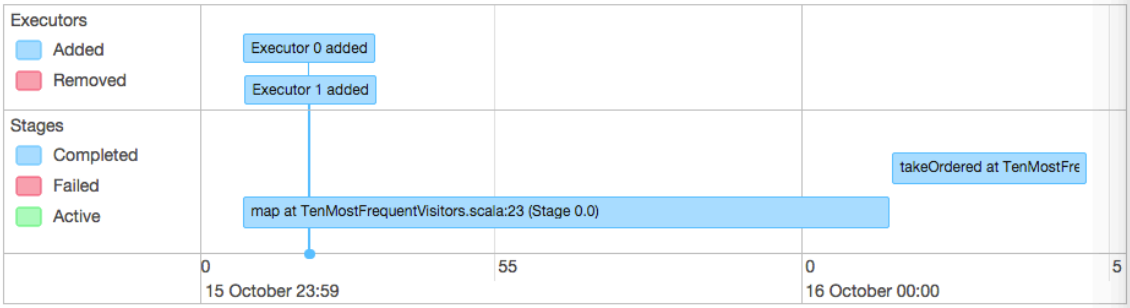


Executor data is shown below. Executor 1 completed 4 tasks and read 31.5 MB total. Executor 0 completed 12 tasks, taking in a total input of 917.5 MB and writing a total of 62.9 MB.

Details for Job 0

Status: SUCCEEDED
Completed Stages: 2

Event Timeline
Enable zooming



Executors (3)

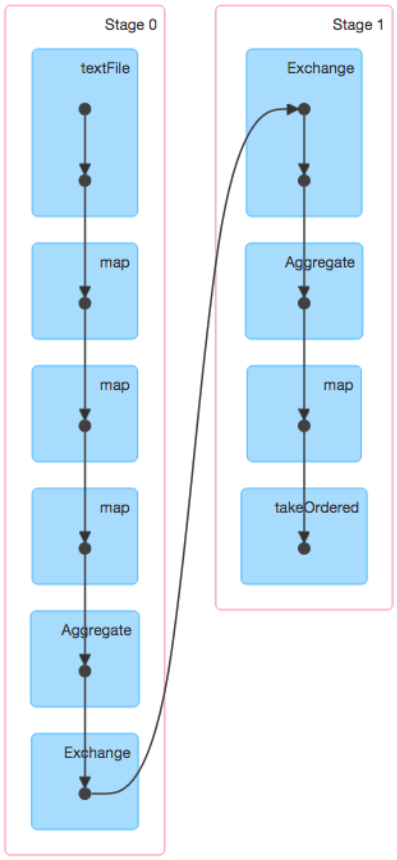
Memory: 0.0 B Used (27.4 GB Total)
Disk: 0.0 B Used

Executor ID	Address	RDD Blocks	Storage Memory	Disk Used	Active Tasks	Failed Tasks	Complete Tasks	Total Tasks	Task Time	Input	Shuffle Read	Shuffle Write	Logs
0	172.31.46.147:50547	0	0.0 B / 13.6 GB	0.0 B	0	0	12	12	1.3 m	917.5 MB	0.0 B	62.9 MB	stdout stderr
1	172.31.46.148:56545	0	0.0 B / 13.6 GB	0.0 B	0	0	4	4	12.3 s	0.0 B	31.5 MB	0.0 B	stdout stderr
driver	172.31.44.186:48480	0	0.0 B / 265.4 MB	0.0 B	0	0	0	0	0 ms	0.0 B	0.0 B	0.0 B	

TenMostFrequentVisitors.scala with DataFrame/SparkSQL

Completed Stages (2)

Stage Id	Description		Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
1	collect at TenMostFrequentVisitors.scala:37	+details	2015/11/16 07:50:02	2 s	200/200			88.9 MB	
0	collect at TenMostFrequentVisitors.scala:37	+details	2015/11/16 07:49:55	7 s	8/8	950.6 MB			88.9 MB

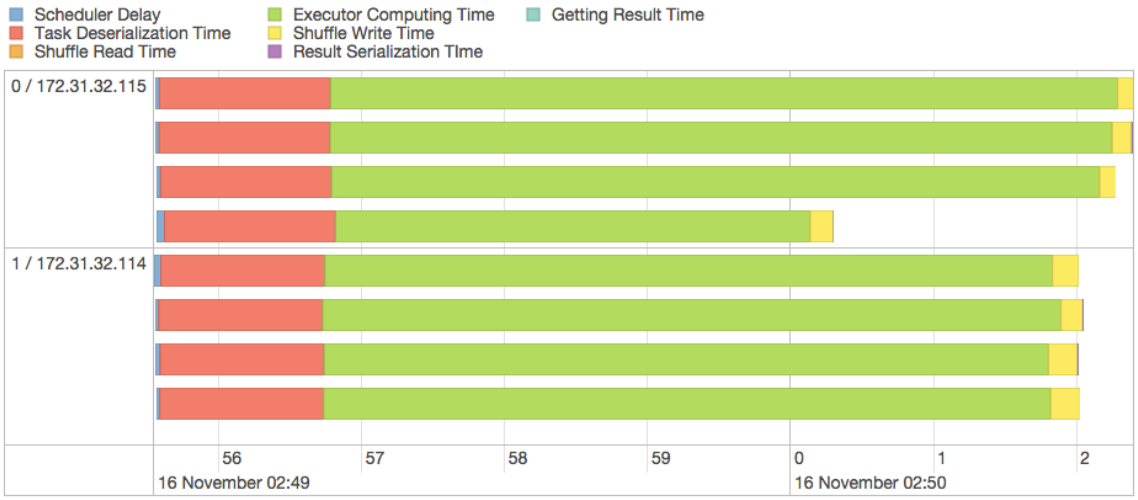


The application runs one job for computing the ten most frequent visitors. The job was comprised of 2 stages, which took a total of 9 seconds with Stage 0 taking 7 seconds and Stage 1 taking 2 seconds. This is shorter than the original, which makes sense since using Spark SQL allows for more efficient queries than RDDs. Stage 0 had 8 tasks, while Stage 1 had 200 tasks. This means that Spark SQL uses much more tasks than the RDDs approach, which could be due to the fact that it registers the RDD to a table like structure and directly queries that. 7 of the 8 tasks in Stage 0 took approximately 5-6 seconds to run in parallel and ran on both Executors 0 and 1. All of the details of interest as well as the input and shuffle data sizes produced by each of the longest running tasks are shown below.

Tasks

Index	ID	Attempt	Status	Locality Level	Executor ID / Host	Launch Time	Duration	GC Time	Input Size / Records	Write Time	Shuffle Write Size / Records	Errors
0	0	0	SUCCESS	ANY	1 / 172.31.32.114	2015/11/16 07:49:55	5 s	0.4 s	128.0 MB (hadoop) / 668105	0.2 s	11.5 MB / 505136	
1	1	0	SUCCESS	ANY	0 / 172.31.32.115	2015/11/16 07:49:55	6 s	0.4 s	128.0 MB (hadoop) / 668318	0.1 s	12.1 MB / 527442	
2	2	0	SUCCESS	ANY	1 / 172.31.32.114	2015/11/16 07:49:55	5 s	0.4 s	128.0 MB (hadoop) / 656531	0.2 s	13.1 MB / 545865	
4	4	0	SUCCESS	ANY	1 / 172.31.32.114	2015/11/16 07:49:55	5 s	0.4 s	128.0 MB (hadoop) / 666103	0.2 s	12.3 MB / 521546	
3	3	0	SUCCESS	ANY	0 / 172.31.32.115	2015/11/16 07:49:55	6 s	0.4 s	128.0 MB (hadoop) / 644414	0.1 s	11.9 MB / 507149	
5	5	0	SUCCESS	ANY	0 / 172.31.32.115	2015/11/16 07:49:55	5 s	0.4 s	128.0 MB (hadoop) / 664764	0.1 s	10.8 MB / 452506	
7	7	0	SUCCESS	ANY	0 / 172.31.32.115	2015/11/16 07:49:55	3 s	77 ms	54.6 MB (hadoop) / 269788	0.2 s	5.2 MB / 216046	
6	6	0	SUCCESS	ANY	1 / 172.31.32.114	2015/11/16 07:49:55	5 s	0.4 s	128.0 MB (hadoop) / 680423	0.2 s	12.0 MB / 503828	

Stage 0 was comprised of reading in the White House data file and then performing the map functions. Stage 1 was composed of the transformation of the data set into a table and executing the query on that table and printing out the top 10 results. Looking at the bar graph below, Task 3 of Stage 0 took the longest to run on Executor 0 and had an input size of 128 MB and a shuffle size of 10.8 MB.



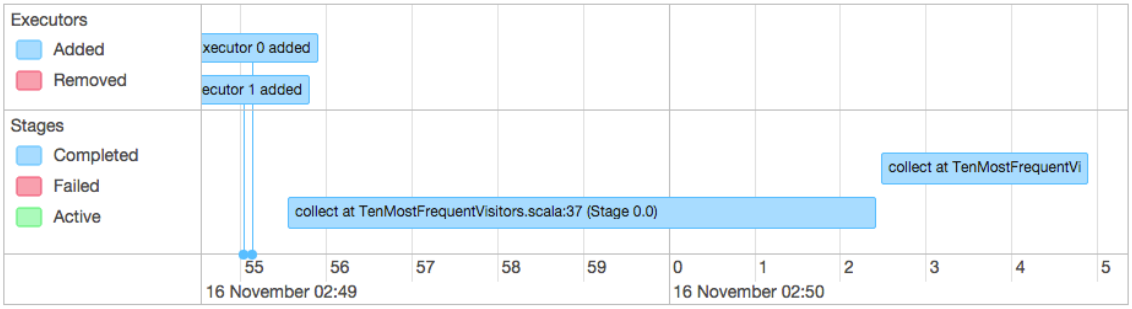
The next screenshot shows how many more short tasks were run with Spark SQL than with RDDs.

Executor data is shown below. Executor 1 completed 113 tasks with an input of 512 MB total and shuffle read and write sizes of 21.8 MB and 48.9 MB respectively. Executor 0 completed 95 tasks with an input of 438.6 MB and shuffle read and write sizes of 22.3 MB and 40 MB respectively.

Details for Job 0

Status: SUCCEEDED
Completed Stages: 2

▾ Event Timeline
☐ Enable zooming



Executors (3)

Memory: 0.0 B Used (27.4 GB Total)
Disk: 0.0 B Used

Executor ID	Address	RDD Blocks	Storage Memory	Disk Used	Active Tasks	Failed Tasks	Complete Tasks	Total Tasks	Task Time	Input	Shuffle Read	Shuffle Write	Logs
0	172.31.32.115:47306	0	0.0 B / 13.6 GB	0.0 B	0	0	95	95	44.2 s	438.6 MB	22.3 MB	40.0 MB	stdout stderr
1	172.31.32.114:37229	0	0.0 B / 13.6 GB	0.0 B	0	0	113	113	45.1 s	512.0 MB	21.8 MB	48.9 MB	stdout stderr
driver	172.31.37.133:54086	0	0.0 B / 265.4 MB	0.0 B	0	0	0	0	0 ms	0.0 B	0.0 B	0.0 B	

Logical and Physical Plans

```
[Hash,Michael,M,775]
[Borzi,Phyllis,C,544]
[Levitis,Jason,A,499]
[Tavenner,Marilyn,n,497]
[Hoff,James,C,486]
[Schultz,William,B,420]
[Mann,Cynthia,R,398]
[Khalid,Aryana,C,395]
[Bowler,Timothy,J,392]
[BrooksLaSure,Chiquita,n,390]
== Parsed Logical Plan ==
'Limit 10
 'Sort ['cnt DESC], true
  'Aggregate ['last_name,'first_name,'middle_name], ['last_name,'first_name,'middle_name,COUNT(1) AS cnt#3L]
   'UnresolvedRelation [visitors], None

== Analyzed Logical Plan ==
last_name: string, first_name: string, middle_name: string, cnt: bigint
Limit 10
 Sort [cnt#3L DESC], true
  Aggregate [last_name#0,first_name#1,middle_name#2], [last_name#0,first_name#1,middle_name#2,COUNT(1) AS cnt#3L]
   Subquery visitors
    LogicalRDD [last_name#0,first_name#1,middle_name#2], MapPartitionsRDD[4] at createDataFrame at TenMostFrequentVisitors.scala:33

== Optimized Logical Plan ==
Limit 10
 Sort [cnt#3L DESC], true
  Aggregate [last_name#0,first_name#1,middle_name#2], [last_name#0,first_name#1,middle_name#2,COUNT(1) AS cnt#3L]
   LogicalRDD [last_name#0,first_name#1,middle_name#2], MapPartitionsRDD[4] at createDataFrame at TenMostFrequentVisitors.scala:33

== Physical Plan ==
TakeOrdered 10, [cnt#3L DESC]
 Aggregate false, [last_name#0,first_name#1,middle_name#2], [last_name#0,first_name#1,middle_name#2,Coalesce(SUM(PartialCount#11L),0) AS cnt#3L]
  Exchange (HashPartitioning 200)
   Aggregate true, [last_name#0,first_name#1,middle_name#2], [last_name#0,first_name#1,middle_name#2,COUNT(1) AS PartialCount#11L]
    PhysicalRDD [last_name#0,first_name#1,middle_name#2], MapPartitionsRDD[4] at createDataFrame at TenMostFrequentVisitors.scala:33
```

Operators in Physical Plan:

- TakeOrdered takes the result and outputs only a subset of the numbers dependent on the parameter.
- Aggregate accepts a Boolean and decides whether it needs to combine the results of multiple partitions.
- Coalesce works on aggregate queries to get the aggregate values after a group by.
- Exchange does the swapping between partitions to get relevant data together. This one is using Hash Partitioning.
- Physical RDD gives the schema of the output and represents the table.
- MapPartitionRDD specifies the location of the RDD in memory (byte addressable).

Alternative Logical Plan:

Another logical plan that Spark could have used is similar to the optimized logical plan except that it doesn't sort the output by first name, middle name, and last name and instead goes through the table outputting the highest numbers while maintaining references to these numbers so that on subsequent passes through the table, the next highest can be outputted. This is much less efficient than the presented logical plan because sorting it allows you to output all of the values in one pass.

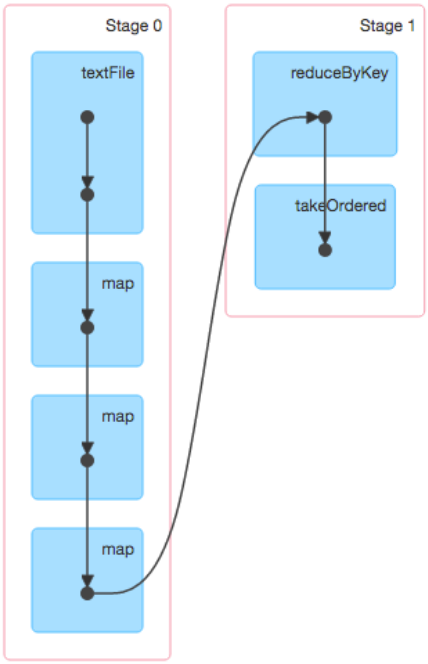
Alternative Physical Plan:

Another physical plan that Spark could've used is instead of grouping the data by first name, last name, and middle name into partitions with the same group by clause, it could've ranged partitioned the data and then computed the necessary exchanges. This is also much less efficient than the implemented plan since a lot of computing would have to be spent just shuffling the data so that the counts can be computed.

TenMostFrequentVisitees.scala without DataFrame/SparkSQL

Completed Stages (2)

Stage Id	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
1	takeOrdered at TenMostFrequentVisitees.scala:24 +details	2015/10/16 03:40:41	1 s	8/8			713.5 KB	
0	map at TenMostFrequentVisitees.scala:23 +details	2015/10/16 03:40:35	6 s	8/8	917.5 MB			713.5 KB

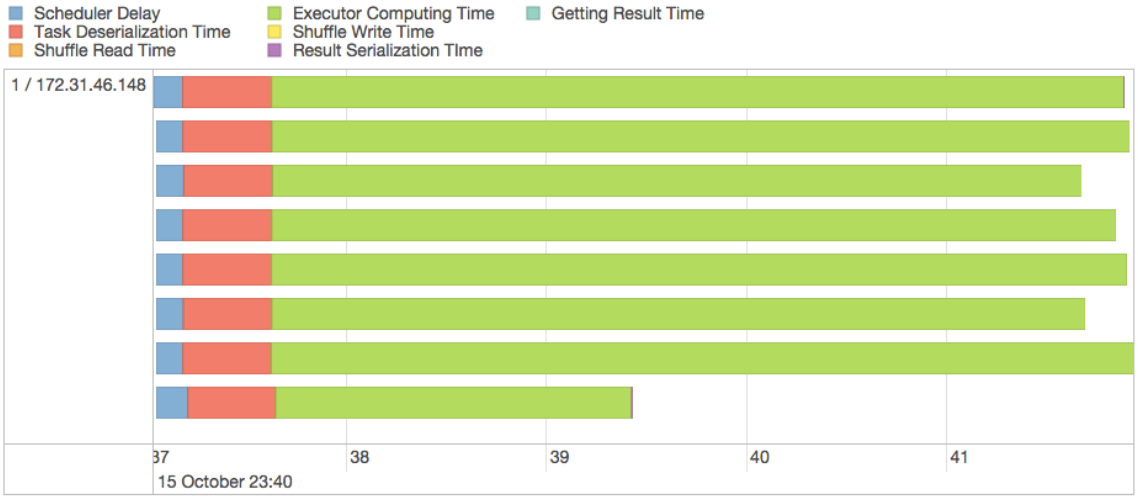


The application runs one job for computing the ten most frequent visitees. The job was comprised of 2 stages, which took a total of 7 seconds with Stage 0 taking 6 seconds and Stage 1 taking 1 second. Each stage had 8 tasks. 7 of the tasks in Stage 0 (0-indexed) took approximately 4 seconds to run in parallel and all ran on Executor 1 (0-indexed). All of the details of interest as well as the input and shuffle data sizes produced by each of the longest running tasks are shown below.

Tasks

Index	ID	Attempt	Status	Locality Level	Executor ID / Host	Launch Time	Duration	GC Time	Input Size / Records	Write Time	Shuffle Write Size / Records	Errors
0	0	0	SUCCESS	ANY	1 / 172.31.46.148	2015/10/16 03:40:37	4 s	55 ms	128.0 MB (hadoop) / 668105		74.8 KB / 4422	
1	1	0	SUCCESS	ANY	1 / 172.31.46.148	2015/10/16 03:40:37	4 s	55 ms	128.0 MB (hadoop) / 668318		73.9 KB / 4372	
2	2	0	SUCCESS	ANY	1 / 172.31.46.148	2015/10/16 03:40:37	4 s	55 ms	128.0 MB (hadoop) / 656531		103.6 KB / 5838	
3	3	0	SUCCESS	ANY	1 / 172.31.46.148	2015/10/16 03:40:37	4 s	55 ms	128.0 MB (hadoop) / 644414		88.3 KB / 4986	
5	5	0	SUCCESS	ANY	1 / 172.31.46.148	2015/10/16 03:40:37	4 s	55 ms	128.0 MB (hadoop) / 664764		114.5 KB / 6567	
4	4	0	SUCCESS	ANY	1 / 172.31.46.148	2015/10/16 03:40:37	4 s	55 ms	128.0 MB (hadoop) / 666103		95.3 KB / 5423	
6	6	0	SUCCESS	ANY	1 / 172.31.46.148	2015/10/16 03:40:37	4 s	55 ms	128.0 MB (hadoop) / 680423		123.2 KB / 6958	
7	7	0	SUCCESS	ANY	1 / 172.31.46.148	2015/10/16 03:40:37	2 s	27 ms	21.5 MB (hadoop) / 105705	1 ms	39.9 KB / 2146	

Stage 0 was comprised of reading in the White House data file and then performing the map functions. Stage 1 was comprised of the reduceByKey call and then the takeOrdered call, which printed the final results. Looking at the bar graph below, Task 6 (0 indexed) of Stage 0 took the longest to run on Executor 1 and had an input size of 128.0 MB and a shuffle size of 123.2 KB.

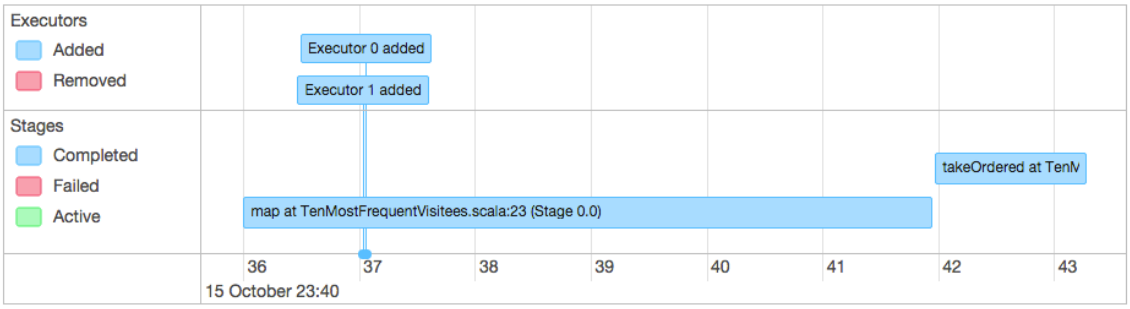


Executor data is shown below. Executor 1 completed 12 tasks, taking in a total input of 917.5 MB and writing a total of 713.5 KB. Executor 0 completed 4 tasks and read 353 KB of data total.

Details for Job 0

Status: SUCCEEDED
Completed Stages: 2

▾ Event Timeline
☐ Enable zooming



Executors (3)

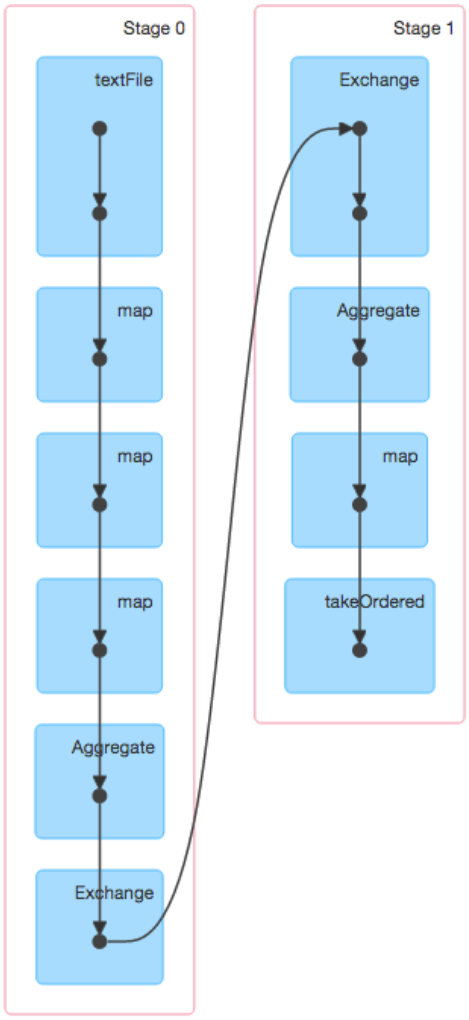
Memory: 0.0 B Used (27.4 GB Total)
Disk: 0.0 B Used

Executor ID	Address	RDD Blocks	Storage Memory	Disk Used	Active Tasks	Failed Tasks	Complete Tasks	Total Tasks	Task Time	Input	Shuffle Read	Shuffle Write	Logs
0	172.31.46.147:51591	0	0.0 B / 13.6 GB	0.0 B	0	0	4	4	4.9 s	0.0 B	353.0 KB	0.0 B	stdout stderr
1	172.31.46.148:60622	0	0.0 B / 13.6 GB	0.0 B	0	0	12	12	37.2 s	917.5 MB	0.0 B	713.5 KB	stdout stderr
driver	172.31.44.186:42171	0	0.0 B / 265.4 MB	0.0 B	0	0	0	0	0 ms	0.0 B	0.0 B	0.0 B	

TenMostFrequentVisitees.scala with DataFrame/SparkSQL

Completed Stages (2)

Stage Id	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
1	collect at TenMostFrequentVisitees.scala:37 +details	2015/11/16 07:50:49	1 s	200/200			1081.1 KB	
0	collect at TenMostFrequentVisitees.scala:37 +details	2015/11/16 07:50:44	5 s	8/8	950.6 MB			1091.5 KB



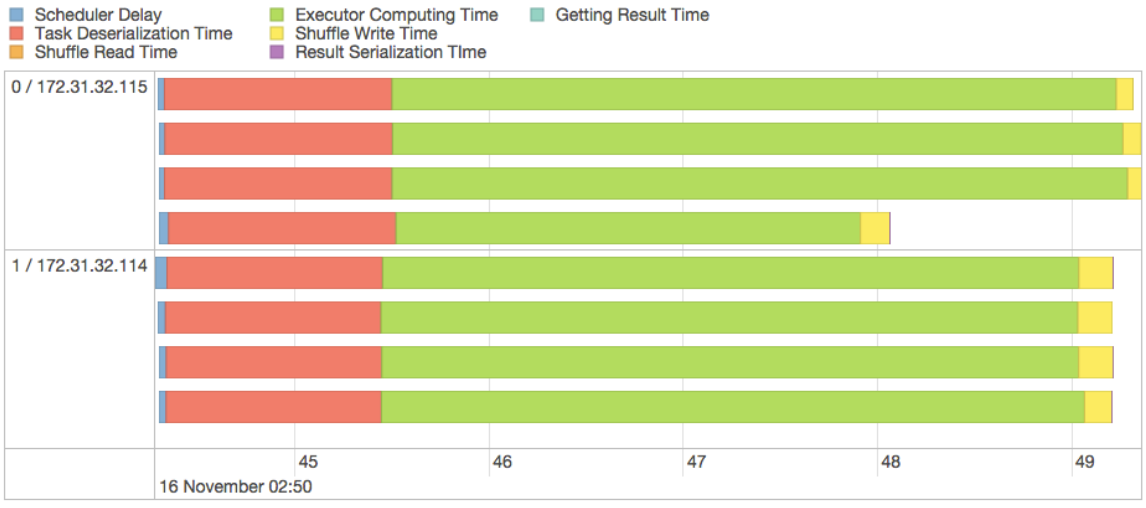
The application runs one job for computing the ten most frequent visitees. The job was comprised of 2 stages, which took a total of 6 seconds with Stage 0 taking 5 seconds and Stage 1 taking 1 second. This is really similar to the performance of the RDD implementation, which could be attributed to the fact that the data sets for visitees requires more computations (the final output numbers are much larger), so most of the latency came from processing this data as opposed to the differences between using Spark SQL and DataFrame. 7 of the 8 tasks took approximately 4 seconds to run in parallel. Similar to the previous question, Stage 1 had 200 tasks, which corresponds with the expectation. All of the details of interest

as well as the input and shuffle data sizes produced by each of the longest running tasks are shown below.

Tasks

Index	ID	Attempt	Status	Locality Level	Executor ID / Host	Launch Time	Duration	GC Time	Input Size / Records	Write Time	Shuffle Write Size / Records	Errors
0	0	0	SUCCESS	ANY	1 / 172.31.32.114	2015/11/16 07:50:44	4 s	48 ms	128.0 MB (hadoop) / 668105	0.2 s	113.1 KB / 4422	
1	1	0	SUCCESS	ANY	0 / 172.31.32.115	2015/11/16 07:50:44	4 s	66 ms	128.0 MB (hadoop) / 668318	86 ms	112.6 KB / 4372	
2	2	0	SUCCESS	ANY	1 / 172.31.32.114	2015/11/16 07:50:44	4 s	48 ms	128.0 MB (hadoop) / 656531	0.2 s	150.6 KB / 5838	
3	3	0	SUCCESS	ANY	0 / 172.31.32.115	2015/11/16 07:50:44	4 s	66 ms	128.0 MB (hadoop) / 644414	94 ms	130.0 KB / 4986	
4	4	0	SUCCESS	ANY	1 / 172.31.32.114	2015/11/16 07:50:44	4 s	48 ms	128.0 MB (hadoop) / 666103	0.2 s	140.7 KB / 5423	
5	5	0	SUCCESS	ANY	0 / 172.31.32.115	2015/11/16 07:50:44	4 s	66 ms	128.0 MB (hadoop) / 664764	83 ms	166.9 KB / 6567	
6	6	0	SUCCESS	ANY	1 / 172.31.32.114	2015/11/16 07:50:44	4 s	48 ms	128.0 MB (hadoop) / 680423	0.1 s	177.9 KB / 6958	
7	7	0	SUCCESS	ANY	0 / 172.31.32.115	2015/11/16 07:50:44	3 s	44 ms	54.6 MB (hadoop) / 269788	0.2 s	99.9 KB / 3731	

Stage 0 was comprised of reading in the White House data file and then performing the map functions. Stage 1 was comprised of the transformation of the RDD into a temporary table and executing the SQL query on it. Looking at the bar graph below, Task 2 took the longest to run on Executor 1 and had an input size of 128 MB and a shuffle size of 150.6 KB.

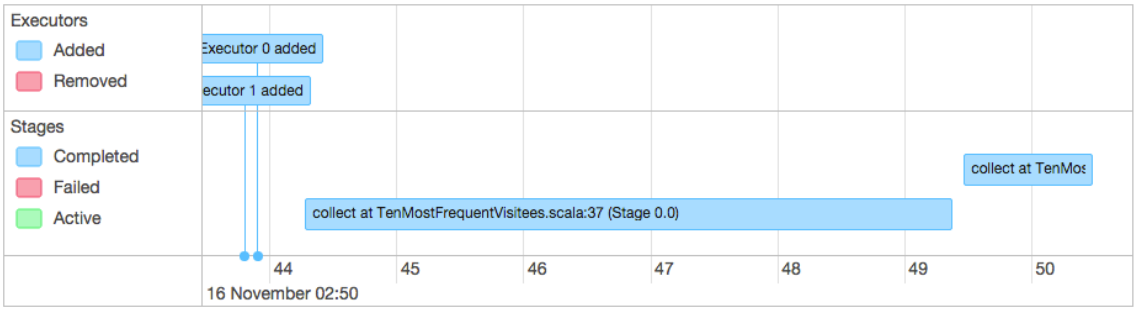


Executor data is shown below. Executor 1 completed 110 tasks, taking in a total input of 512 MB with shuffle read and write sizes of 263.3 KB and 582.2 KB respectively. Executor 0 completed 98 tasks, taking in a total input of 438.5 MB with shuffle read and write sizes of 270.3 KB and 509.3 KB respectively.

Details for Job 0

Status: SUCCEEDED
Completed Stages: 2

▾ Event Timeline
☐ Enable zooming



Executors (3)

Memory: 0.0 B Used (27.4 GB Total)
Disk: 0.0 B Used

Executor ID	Address	RDD Blocks	Storage Memory	Disk Used	Active Tasks	Failed Tasks	Complete Tasks	Total Tasks	Task Time	Input	Shuffle Read	Shuffle Write	Logs
0	172.31.32.115:35886	0	0.0 B / 13.6 GB	0.0 B	0	0	98	98	27.0 s	438.6 MB	270.3 KB	509.3 KB	stdout stderr
1	172.31.32.114:50884	0	0.0 B / 13.6 GB	0.0 B	0	0	110	110	27.9 s	512.0 MB	263.3 KB	582.2 KB	stdout stderr
driver	172.31.37.133:37936	0	0.0 B / 265.4 MB	0.0 B	0	0	0	0	0 ms	0.0 B	0.0 B	0.0 B	

Logical and Physical Plans

```
[OFFICE,VISITORS,2972310]
[,POTUS,139908]
[POTUS,,75855]
[,POTUS/FLOTUS,38775]
[,,24782]
[Lierman,Kyle,18789]
[Lambrew,Jeanne,18448]
[,FLOTUS,15332]
[,potus,15018]
[OFFICE,VISITORS ,12063]
== Parsed Logical Plan ==
'Limit 10
 'Sort ['cnt DESC], true
  'Aggregate ['last_name,'first_name], ['last_name,'first_name,COUNT(1) AS cnt#2L]
   'UnresolvedRelation [visitees], None

== Analyzed Logical Plan ==
last_name: string, first_name: string, cnt: bigint
Limit 10
 Sort [cnt#2L DESC], true
  Aggregate [last_name#0,first_name#1], [last_name#0,first_name#1,COUNT(1) AS cnt#2L]
   Subquery visitees
    LogicalRDD [last_name#0,first_name#1], MapPartitionsRDD[4] at createDataFrame at TenMostFrequentVisitees.scala:33

== Optimized Logical Plan ==
Limit 10
 Sort [cnt#2L DESC], true
  Aggregate [last_name#0,first_name#1], [last_name#0,first_name#1,COUNT(1) AS cnt#2L]
   LogicalRDD [last_name#0,first_name#1], MapPartitionsRDD[4] at createDataFrame at TenMostFrequentVisitees.scala:33

== Physical Plan ==
TakeOrdered 10, [cnt#2L DESC]
 Aggregate false, [last_name#0,first_name#1], [last_name#0,first_name#1,Coalesce(SUM(PartialCount#10L),0) AS cnt#2L]
  Exchange (HashPartitioning 200)
   Aggregate true, [last_name#0,first_name#1], [last_name#0,first_name#1,COUNT(1) AS PartialCount#10L]
    PhysicalRDD [last_name#0,first_name#1], MapPartitionsRDD[4] at createDataFrame at TenMostFrequentVisitees.scala:33
```

Operators in Physical Plan:

- TakeOrdered takes the result and outputs only a subset of the numbers dependent on the parameter.
- Aggregate accepts a Boolean and decides whether it needs to combine the results of multiple partitions.
- Coalesce works on aggregate queries to get the aggregate values after a group by.
- Exchange does the swapping between partitions to get relevant data together. This one is using Hash Partitioning.
- Physical RDD gives the schema of the output and represents the table.
- MapPartitionRDD specifies the location of the RDD in memory (byte addressable).

Alternative Logical Plan:

Another logical plan that Spark could have used is similar to the optimized logical plan except that it doesn't sort the output by visitee first name and visitee last name and instead goes through the table outputting the highest numbers while maintaining references to these numbers so that on subsequent passes through the table, the next highest can be outputted. This is much less efficient than the presented logical plan because sorting it allows you to output all of the values in one pass.

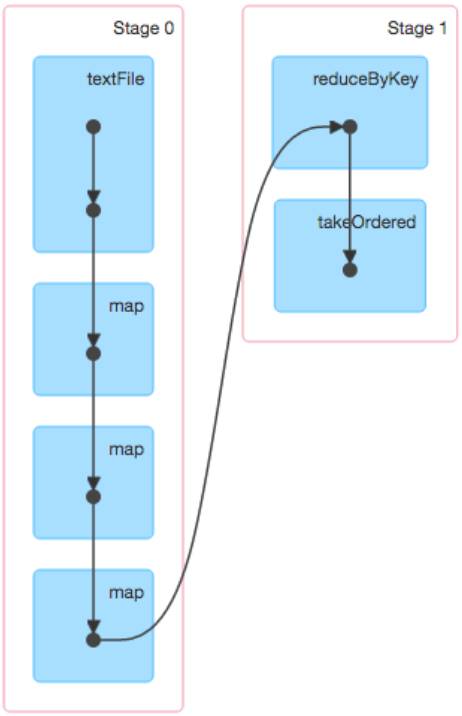
Alternative Physical Plan:

Another physical plan that Spark could've used is instead of grouping the data by visitee first name and visitee last name into partitions with the same group by clause, it could've ranged partitioned the data and then computed the necessary exchanges. This is also much less efficient than the implemented plan since a lot of computing would have to be spent just shuffling the data so that the counts can be computed.

TenMostFrequentVisitorVisitees.scala without DataFrame/SparkSQL

Completed Stages (2)

Stage Id	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
1	takeOrdered at TenMostFrequentVisitorVisitees.scala:25 +details	2015/10/16 02:58:56	4 s	8/8			94.9 MB	
0	map at TenMostFrequentVisitorVisitees.scala:24 +details	2015/10/16 02:58:41	15 s	8/8	917.5 MB			94.9 MB

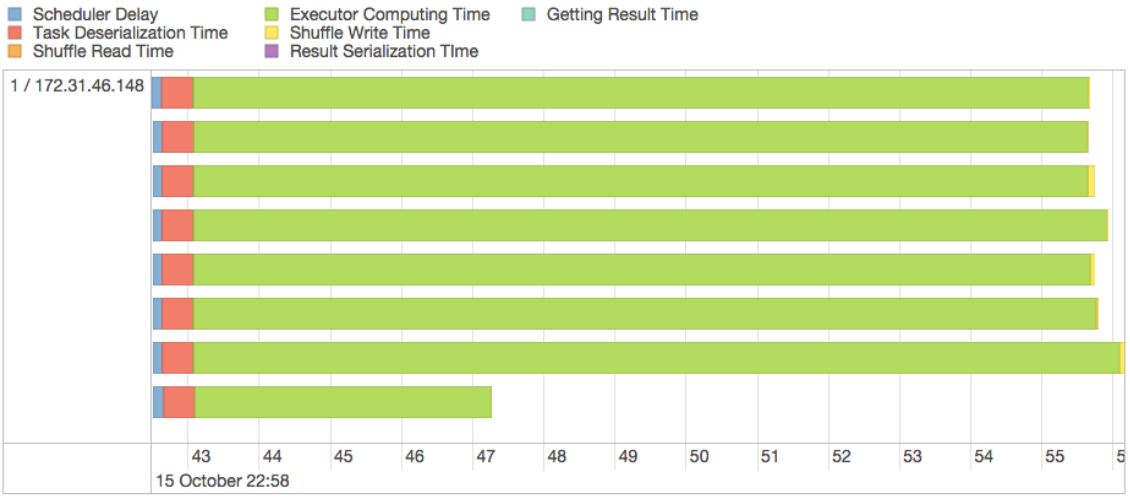


The application runs one job for computing the ten most frequent visitor-visitee combinations. The job was comprised of 2 stages, which took a total of 19 seconds with Stage 0 taking 4 seconds and Stage 1 taking 15 seconds. Each stage had 8 tasks. 7 of the tasks in Stage 0 (0-indexed) took approximately 13 seconds to run in parallel and all ran on Executor 1 (0-indexed). All of the details of interest as well as the input and shuffle data sizes produced by each of the longest running tasks are shown below.

Tasks

Index	ID	Attempt	Status	Locality Level	Executor ID / Host	Launch Time	Duration	GC Time	Input Size / Records	Write Time	Shuffle Write Size / Records	Errors
0	0	0	SUCCESS	ANY	1 / 172.31.46.148	2015/10/16 02:58:42	13 s	1 s	128.0 MB (hadoop) / 668105	30 ms	12.5 MB / 562473	
1	1	0	SUCCESS	ANY	1 / 172.31.46.148	2015/10/16 02:58:42	13 s	1 s	128.0 MB (hadoop) / 668318	22 ms	12.5 MB / 581179	
2	2	0	SUCCESS	ANY	1 / 172.31.46.148	2015/10/16 02:58:42	13 s	1 s	128.0 MB (hadoop) / 656531	0.1 s	13.4 MB / 590105	
3	3	0	SUCCESS	ANY	1 / 172.31.46.148	2015/10/16 02:58:42	13 s	1 s	128.0 MB (hadoop) / 644414	30 ms	13.0 MB / 570418	
5	5	0	SUCCESS	ANY	1 / 172.31.46.148	2015/10/16 02:58:42	13 s	1 s	128.0 MB (hadoop) / 664764	31 ms	13.7 MB / 552483	
4	4	0	SUCCESS	ANY	1 / 172.31.46.148	2015/10/16 02:58:42	13 s	1 s	128.0 MB (hadoop) / 666103	74 ms	13.4 MB / 583218	
6	6	0	SUCCESS	ANY	1 / 172.31.46.148	2015/10/16 02:58:42	13 s	1 s	128.0 MB (hadoop) / 680423	0.1 s	14.1 MB / 587774	
7	7	0	SUCCESS	ANY	1 / 172.31.46.148	2015/10/16 02:58:42	4 s	0.5 s	21.5 MB (hadoop) / 105705	8 ms	2.3 MB / 96679	

Stage 0 was comprised of reading in the White House data file and then performing the map functions. Stage 1 was comprised of the reduceByKey call and then the takeOrdered call, which printed the final results. Looking at the bar graph below, Task 6 (0 indexed) of Stage 0 took the longest to run on Executor 1 and had an input size of 128.0 MB and a shuffle size of 14.1 MB.

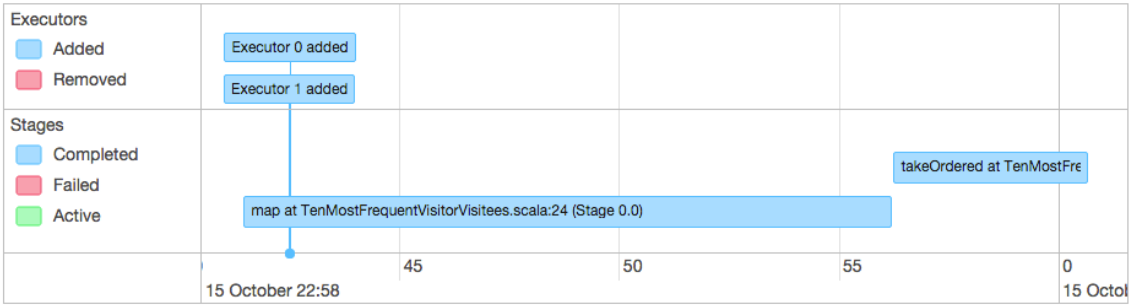


Executor data is shown below. Executor 1 completed 12 tasks, taking in a total input of 917.5 MB and writing a total of 94.9 MB. Executor 0 completed 4 tasks and read 47.4 MB of data total.

Details for Job 0

Status: SUCCEEDED
Completed Stages: 2

▾ Event Timeline
☐ Enable zooming



Executors (3)

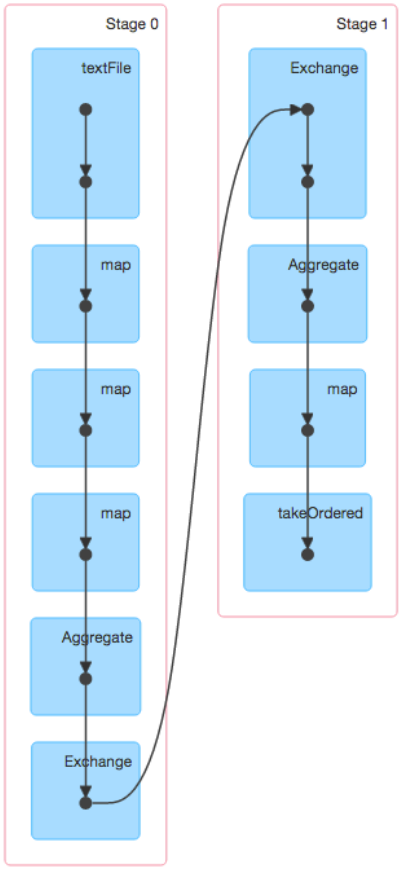
Memory: 0.0 B Used (27.4 GB Total)
Disk: 0.0 B Used

Executor ID	Address	RDD Blocks	Storage Memory	Disk Used	Active Tasks	Failed Tasks	Complete Tasks	Total Tasks	Task Time	Input	Shuffle Read	Shuffle Write	Logs
0	172.31.46.147:44107	0	0.0 B / 13.6 GB	0.0 B	0	0	4	4	17.1 s	0.0 B	47.4 MB	0.0 B	stdout stderr
1	172.31.46.148:45632	0	0.0 B / 13.6 GB	0.0 B	0	0	12	12	1.8 m	917.5 MB	0.0 B	94.9 MB	stdout stderr
driver	172.31.44.186:51587	0	0.0 B / 265.4 MB	0.0 B	0	0	0	0	0 ms	0.0 B	0.0 B	0.0 B	

TenMostFrequentVisitorVisitees.scala with DataFrame/SparkSQL

Completed Stages (2)

Stage Id	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
1	collect at TenMostFrequentVisitorVisitees.scala:38 +details	2015/11/16 07:50:28	3 s	200/200			132.6 MB	
0	collect at TenMostFrequentVisitorVisitees.scala:38 +details	2015/11/16 07:50:21	8 s	8/8	950.6 MB			132.6 MB

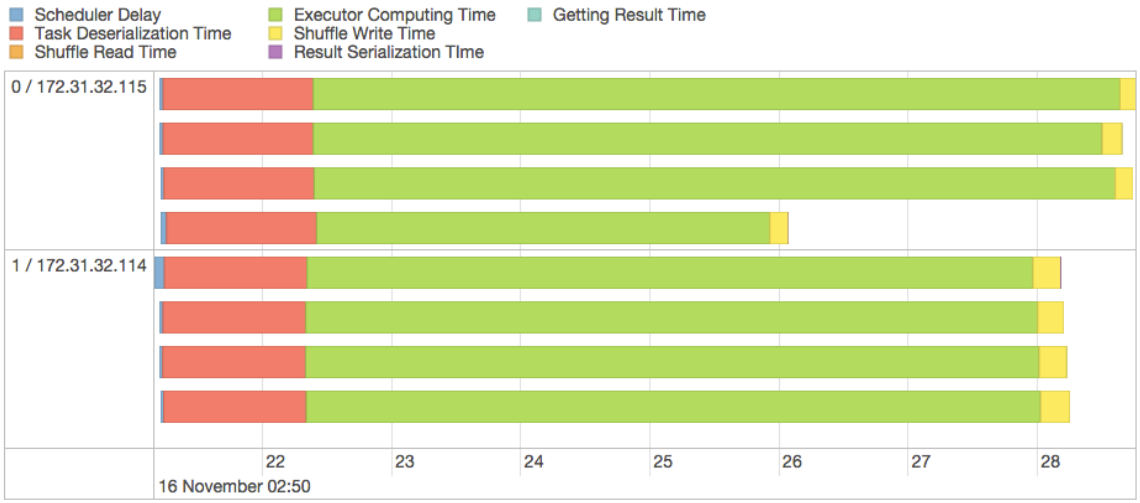


The application runs one job for computing the ten most frequent visitor-visitee combinations. The job was comprised of 2 stages, which took a total of 11 seconds with Stage 0 taking 3 seconds and Stage 1 taking 8 seconds. The distribution of time spent matches with the RDD implementation. It takes an overall shorter time, which matches with the results from TenMostFrequentVisitors since the data set is able to be processed quicker due to filtering out more things, which makes the bottleneck mostly due to the implementation of RDDs vs. Spark SQL and DataFrames. 7 of the 8 tasks in Stage 0 took approximately 6 seconds to run in parallel and ran on both Executors. All of the details of interest as well as the input and shuffle data sizes produced by each of the longest running tasks are shown below.

Tasks

Index	ID	Attempt	Status	Locality Level	Executor ID / Host	Launch Time	Duration	GC Time	Input Size / Records	Write Time	Shuffle Write Size / Records	Errors
0	0	0	SUCCESS	ANY	1 / 172.31.32.114	2015/11/16 07:50:21	6 s	0.5 s	128.0 MB (hadoop) / 668105	0.2 s	16.9 MB / 562473	
1	1	0	SUCCESS	ANY	0 / 172.31.32.115	2015/11/16 07:50:21	6 s	0.5 s	128.0 MB (hadoop) / 668318	0.1 s	17.3 MB / 581179	
2	2	0	SUCCESS	ANY	1 / 172.31.32.114	2015/11/16 07:50:21	6 s	0.5 s	128.0 MB (hadoop) / 656531	0.2 s	18.3 MB / 590105	
4	4	0	SUCCESS	ANY	1 / 172.31.32.114	2015/11/16 07:50:21	6 s	0.5 s	128.0 MB (hadoop) / 666103	0.2 s	18.1 MB / 583218	
3	3	0	SUCCESS	ANY	0 / 172.31.32.115	2015/11/16 07:50:21	6 s	0.5 s	128.0 MB (hadoop) / 644414	0.2 s	17.6 MB / 570418	
5	5	0	SUCCESS	ANY	0 / 172.31.32.115	2015/11/16 07:50:21	6 s	0.5 s	128.0 MB (hadoop) / 664764	0.1 s	17.8 MB / 552483	
7	7	0	SUCCESS	ANY	0 / 172.31.32.115	2015/11/16 07:50:21	4 s	0.1 s	54.6 MB (hadoop) / 269788	0.1 s	7.8 MB / 241355	
6	6	0	SUCCESS	ANY	1 / 172.31.32.114	2015/11/16 07:50:21	6 s	0.5 s	128.0 MB (hadoop) / 680423	0.2 s	18.8 MB / 587774	

Stage 0 was comprised of reading in the White house data file and then performing the map functions. Stage 1 was comprised of the transformation of the RDD into the temporary table and then executing the query on that table and printing out the results. Looking at the bar graph below, Task 2 took the longest to run on Executor 1, and had an input size of 128 MB with a shuffle write size of 18.3 MB.



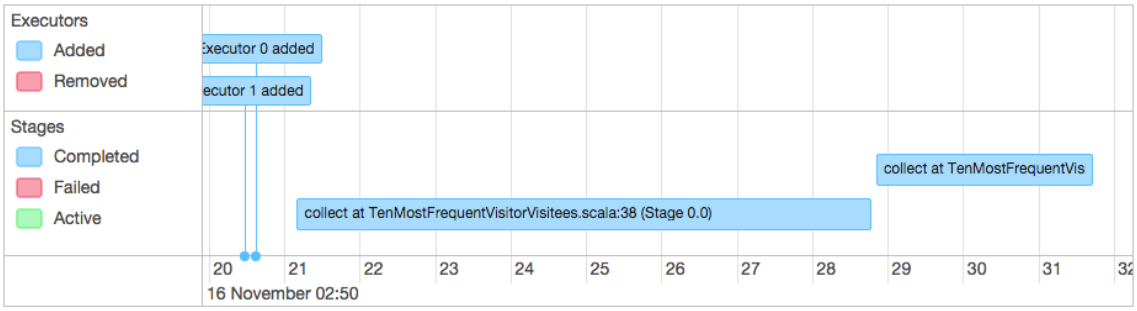
Executor data is shown below. Executor 1 completed 102 tasks, taking in a total input of 512 MB with shuffle read and write sizes of 29.6 MB and 72.1 MB

respectively. This is much smaller than the other implementation. Executor 0 completed 106 tasks with shuffle read and write sizes of 36.8 MB and 60.5 MB respectively. This is also shorter.

Details for Job 0

Status: SUCCEEDED
Completed Stages: 2

▾ Event Timeline
☐ Enable zooming



Executors (3)

Memory: 0.0 B Used (27.4 GB Total)
Disk: 0.0 B Used

Executor ID	Address	RDD Blocks	Storage Memory	Disk Used	Active Tasks	Failed Tasks	Complete Tasks	Total Tasks	Task Time	Input	Shuffle Read	Shuffle Write	Logs
0	172.31.32.115:40719	0	0.0 B / 13.6 GB	0.0 B	0	0	106	106	50.2 s	438.6 MB	36.8 MB	60.5 MB	stdout stderr
1	172.31.32.114:40113	0	0.0 B / 13.6 GB	0.0 B	0	0	102	102	51.0 s	512.0 MB	29.6 MB	72.1 MB	stdout stderr
driver	172.31.37.133:36912	0	0.0 B / 265.4 MB	0.0 B	0	0	0	0	0 ms	0.0 B	0.0 B	0.0 B	

Logical and Physical Plans

```
[Hash,Michael,M,Lambrew,Jeanne,583]
[Tavenner,Marilyn,n,Lambrew,Jeanne,422]
[Hoff,James,C,Hoff,Joanne,407]
[Levitis,Jason,A,Lambrew,Jeanne,362]
[BrooksLaSure,Chiquita,n,Lambrew,Jeanne,346]
[Mann,Cynthia,R,Lambrew,Jeanne,332]
[Khalid,Aryana,C,Lambrew,Jeanne,326]
[Borzi,Phyllis,C,Lambrew,Jeanne,313]
[Fontenot,Yvette,E,Lambrew,Jeanne,313]
[Turner,Amy,J,Lambrew,Jeanne,306]
== Parsed Logical Plan ==
'Limit 10
 'Sort ['cnt DESC], true
  'Aggregate ['visitor_last_name,'visitor_first_name,'visitor_middle_name,'visitee_last_name,'visitee_first_name], ['visitor_last_name,
'visitor_first_name,'visitor_middle_name,'visitee_last_name,'visitee_first_name,COUNT(1) AS cnt#5L]
   'UnresolvedRelation [visitorVisitees], None

== Analyzed Logical Plan ==
visitor_last_name: string, visitor_first_name: string, visitor_middle_name: string, visitee_last_name: string, visitee_first_name: stri
ng, cnt: bigint
Limit 10
 Sort [cnt#5L DESC], true
  Aggregate [visitor_last_name#0,visitor_first_name#1,visitor_middle_name#2,visitee_last_name#3,visitee_first_name#4], [visitor_last_na
me#0,visitor_first_name#1,visitor_middle_name#2,visitee_last_name#3,visitee_first_name#4,COUNT(1) AS cnt#5L]
   Subquery visitorVisitees
    LogicalRDD [visitor_last_name#0,visitor_first_name#1,visitor_middle_name#2,visitee_last_name#3,visitee_first_name#4], MapPartitions
RDD[4] at createDataFrame at TenMostFrequentVisitorVisitees.scala:34

== Optimized Logical Plan ==
Limit 10
 Sort [cnt#5L DESC], true
  Aggregate [visitor_last_name#0,visitor_first_name#1,visitor_middle_name#2,visitee_last_name#3,visitee_first_name#4], [visitor_last_na
me#0,visitor_first_name#1,visitor_middle_name#2,visitee_last_name#3,visitee_first_name#4,COUNT(1) AS cnt#5L]
   LogicalRDD [visitor_last_name#0,visitor_first_name#1,visitor_middle_name#2,visitee_last_name#3,visitee_first_name#4], MapPartitionsR
DD[4] at createDataFrame at TenMostFrequentVisitorVisitees.scala:34

== Physical Plan ==
TakeOrdered 10, [cnt#5L DESC]
 Aggregate false, [visitee_last_name#3,visitor_last_name#0,visitee_first_name#4,visitor_middle_name#2,visitor_first_name#1], [visitor_l
ast_name#0,visitor_first_name#1,visitor_middle_name#2,visitee_last_name#3,visitee_first_name#4,Coalesce(SUM(PartialCount#13L),0) AS cnt
#5L]
  Exchange (HashPartitioning 200)
   Aggregate true, [visitor_last_name#0,visitor_first_name#1,visitor_middle_name#2,visitee_last_name#3,visitee_first_name#4], [visitee_
last_name#3,visitor_last_name#0,visitee_first_name#4,visitor_middle_name#2,visitor_first_name#1,COUNT(1) AS PartialCount#13L]
    PhysicalRDD [visitor_last_name#0,visitor_first_name#1,visitor_middle_name#2,visitee_last_name#3,visitee_first_name#4], MapPartition
sRDD[4] at createDataFrame at TenMostFrequentVisitorVisitees.scala:34
```

Operators in Physical Plan:

- TakeOrdered takes the result and outputs only a subset of the numbers dependent on the parameter.
- Aggregate accepts a Boolean and decides whether it needs to combine the results of multiple partitions.
- Coalesce works on aggregate queries to get the aggregate values after a group by.
- Exchange does the swapping between partitions to get relevant data together. This one is using Hash Partitioning.
- Physical RDD gives the schema of the output and represents the table.
- MapPartitionRDD specifies the location of the RDD in memory (byte addressable).

Alternative Logical Plan:

Another logical plan that Spark could have used is similar to the optimized logical plan except that it doesn't sort the output by first name, middle name, last name, visitee first name, and visitee last name and instead goes through the table outputting the highest numbers while maintaining references to these numbers so that on subsequent passes through the table, the next highest can be outputted. This

is much less efficient than the presented logical plan because sorting it allows you to output all of the values in one pass.

Alternative Physical Plan:

Another physical plan that Spark could've used is instead of grouping the data by first name, last name, middle name, visitee first name, and visitee last name into partitions with the same group by clause, it could've ranged partitioned the data and then computed the necessary exchanges. This is also much less efficient than the implemented plan since a lot of computing would have to be spent just shuffling the data so that the counts can be computed.

WikipediaPagesWithNoOutlinks.scala without DataFrame/SparkSQL

Completed Jobs (2)

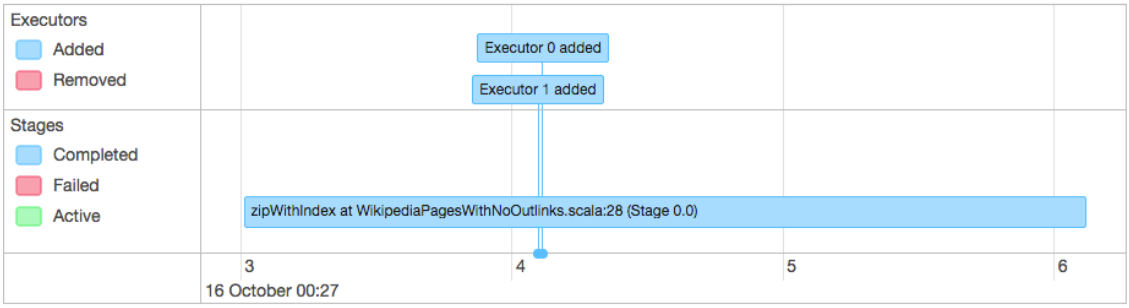
Job Id	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
1	count at WikipediaPagesWithNoOutlinks.scala:37	2015/10/16 04:27:06	17 s	5/5	16/16
0	zipWithIndex at WikipediaPagesWithNoOutlinks.scala:28	2015/10/16 04:27:02	3 s	1/1	1/1

The application runs two jobs with a total of 6 Stages for computing all of the Wikipedia pages with no outlinks. Job 0 was comprised of 1 stage of 1 task and took 3 seconds total. Job 1 was comprised of 5 stages and 16 tasks total, which took a total of 17 seconds.

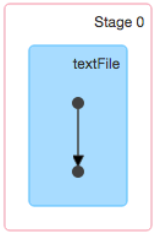
Details for Job 0

Status: SUCCEEDED
Completed Stages: 1

▾ Event Timeline
☐ Enable zooming

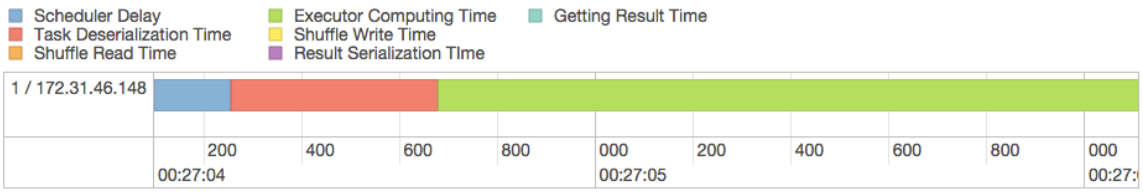


▾ DAG Visualization



Completed Stages (1)

Stage Id	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
0	zipWithIndex at WikipediaPagesWithNoOutlinks.scala:28 +details	2015/10/16 04:27:03	3 s	1/1	53.2 MB			



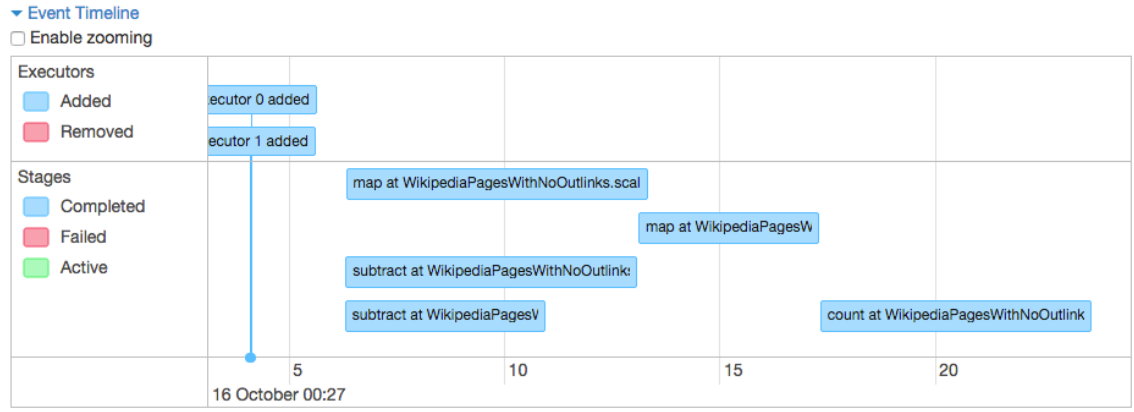
Tasks

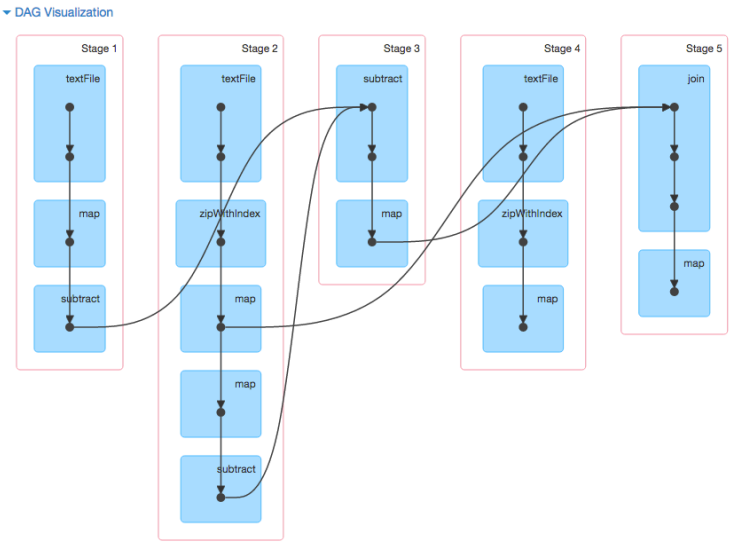
Index	ID	Attempt	Status	Locality Level	Executor ID / Host	Launch Time	Duration	GC Time	Input Size / Records	Errors
0	0	0	SUCCESS	ANY	1 / 172.31.46.148	2015/10/16 04:27:04	1 s		53.2 MB (hadoop) / 2915094	

Stage 0 took 3 seconds with 1 task. Task 0 handled the zipWithIndex call and took approximately 1.5 seconds with an input size of 53.2 MB on Executor 1. This information is summarized above.

Details for Job 1

Status: SUCCEEDED
Completed Stages: 5

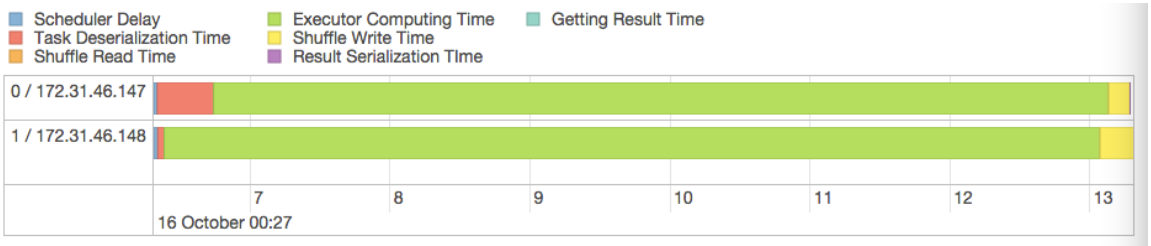




Completed Stages (5)

Stage Id	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
5	count at WikipediaPagesWithNoOutlinks.scala:37 +details	2015/10/16 04:27:17	6 s	2/2			87.5 MB	
3	map at WikipediaPagesWithNoOutlinks.scala:35 +details	2015/10/16 04:27:13	4 s	2/2			55.6 MB	63.1 KB
4	map at WikipediaPagesWithNoOutlinks.scala:28 +details	2015/10/16 04:27:06	7 s	2/2	106.4 MB			87.5 MB
2	subtract at WikipediaPagesWithNoOutlinks.scala:33 +details	2015/10/16 04:27:06	7 s	2/2	106.4 MB			27.8 MB
1	subtract at WikipediaPagesWithNoOutlinks.scala:33 +details	2015/10/16 04:27:06	5 s	8/8	1009.4 MB			27.8 MB

Job 1 was comprised of 5 Stages. Stage 1 is comprised of 8 tasks and Stages 2 through 5 are comprised of 2 tasks. Stage 1 reads in links-simple-sorted.txt, which it maps and then performs a subtract on after getting the relevant data together in Stage 3. Stage 2 reads in titles-sorted.txt and then performs zipWithIndex, two maps, and a subtract on after getting the relevant data together in Stage 3. Stage 3 handles the subtract of data from Stage 1 and Stage 2 then maps it. Stage 4 reads in titles-sorted.txt and performs a zipWithIndex and map on it. Stage 5 takes the result of Stage 3 and one of the intermediate mapped RDDs from Stage 2 and joins them together then performs another map on it to get the final result. Tasks 0 and 1 of Stage 4 took the longest to run at 7 seconds and ran on Executors 0 and 1 respectively. Task 0 had an input size of 53.2 MB and shuffle size of 44.4 MB. Task 1 had an input size of 53.2 MB and shuffle size of 43.1 MB. This information is summarized below.



Tasks

Index	ID	Attempt	Status	Locality Level	Executor ID / Host	Launch Time	Duration ▲	GC Time	Input Size / Records	Write Time	Shuffle Write Size / Records	Error
0	11	0	SUCCESS	ANY	0 / 172.31.46.147	2015/10/16 04:27:06	7 s	43 ms	53.2 MB (hadoop) / 2915094	0.2 s	44.4 MB / 2915094	
1	12	0	SUCCESS	ANY	1 / 172.31.46.148	2015/10/16 04:27:06	7 s	16 ms	53.2 MB (hadoop) /	0.3 s	43.1 MB / 2801714	

Executor data is shown below. Executor 0 completed 8 tasks, with a total input size of 603.8 MB, shuffle read size of 35.3 MB, and shuffle write size of 72.5 MB. Executor 1 completed 9 tasks, with a total input size of 671.6 MB, shuffle read size of 36.2 MB, and shuffle write size of 70.6 MB.

Executors (3)

Memory: 0.0 B Used (27.4 GB Total)
Disk: 0.0 B Used

Executor ID	Address	RDD Blocks	Storage Memory	Disk Used	Active Tasks	Failed Tasks	Complete Tasks	Total Tasks	Task Time	Input	Shuffle Read	Shuffle Write	Logs
0	172.31.46.147:41448	0	0.0 B / 13.6 GB	0.0 B	0	0	8	8	41.2 s	603.8 MB	35.3 MB	72.5 MB	stdout stderr
1	172.31.46.148:40038	0	0.0 B / 13.6 GB	0.0 B	0	0	9	9	37.8 s	671.6 MB	36.2 MB	70.6 MB	stdout stderr
driver	172.31.44.186:54247	0	0.0 B / 265.4 MB	0.0 B	0	0	0	0	0 ms	0.0 B	0.0 B	0.0 B	

WikipediaPagesWithNoOutlinks.scala with DataFrame/SparkSQL

Completed Jobs (2)

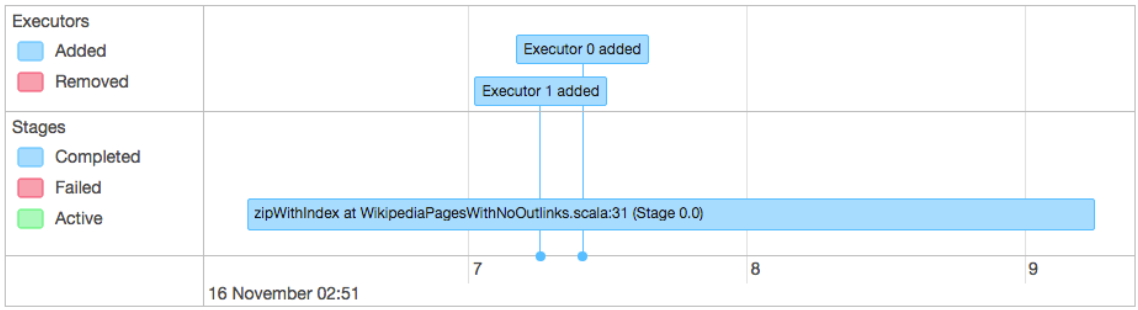
Job Id	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
1	count at WikipediaPagesWithNoOutlinks.scala:46	2015/11/16 07:51:10	29 s	6/6	215/215
0	zipWithIndex at WikipediaPagesWithNoOutlinks.scala:31	2015/11/16 07:51:06	3 s	1/1	1/1

The application runs two jobs with a total of 7 Stages for computing all of the Wikipedia pages with no outlinks. Job 0 was comprised of 1 stage of 1 task and took 3 seconds total. This is exactly the same as the original implementation, which makes sense since it's just reading in the text file. Job 1 was comprised of 6 Stages and 215 tasks total, which took a total 0f 29 seconds. This is much longer than the original implementation with more stages and tasks, which can be attributed to the join required to compute the page titles with no outlinks. If only IDs were required, this would've made this query much more efficient

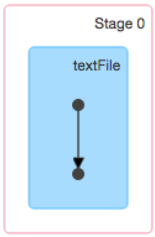
Details for Job 0

Status: SUCCEEDED
Completed Stages: 1

▾ Event Timeline
☐ Enable zooming



▾ DAG Visualization

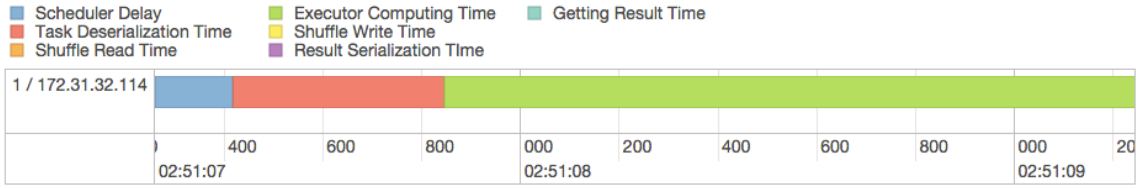


Completed Stages (1)

Stage Id	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
0	zipWithIndex at WikipediaPagesWithNoOutlinks.scala:31 +details	2015/11/16 07:51:06	3 s	1/1	53.2 MB			

Tasks

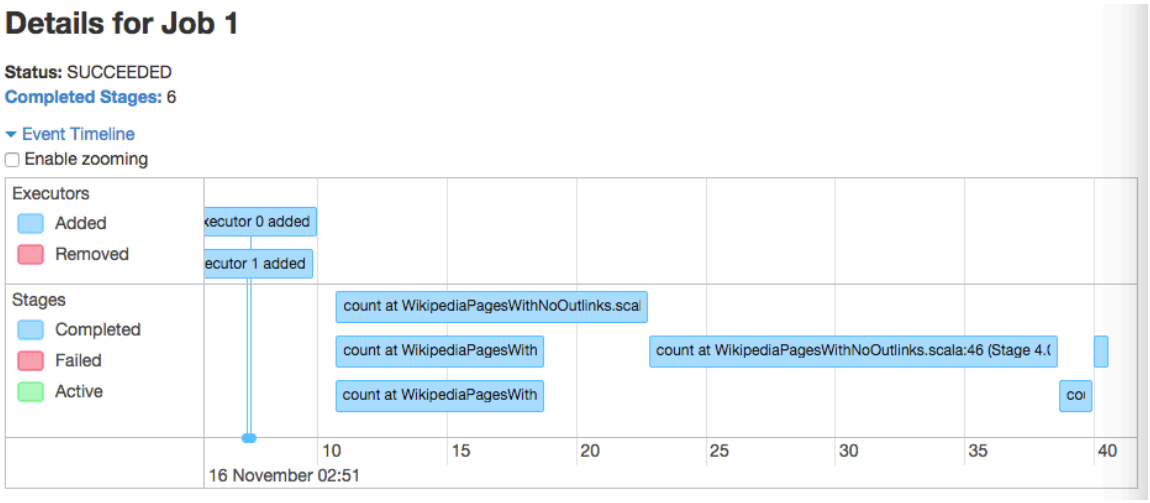
Index	ID	Attempt	Status	Locality Level	Executor ID / Host	Launch Time	Duration	GC Time	Input Size / Records	Errors
0	0	0	SUCCESS	ANY	1 / 172.31.32.114	2015/11/16 07:51:07	1 s		53.2 MB (hadoop) / 2915094	

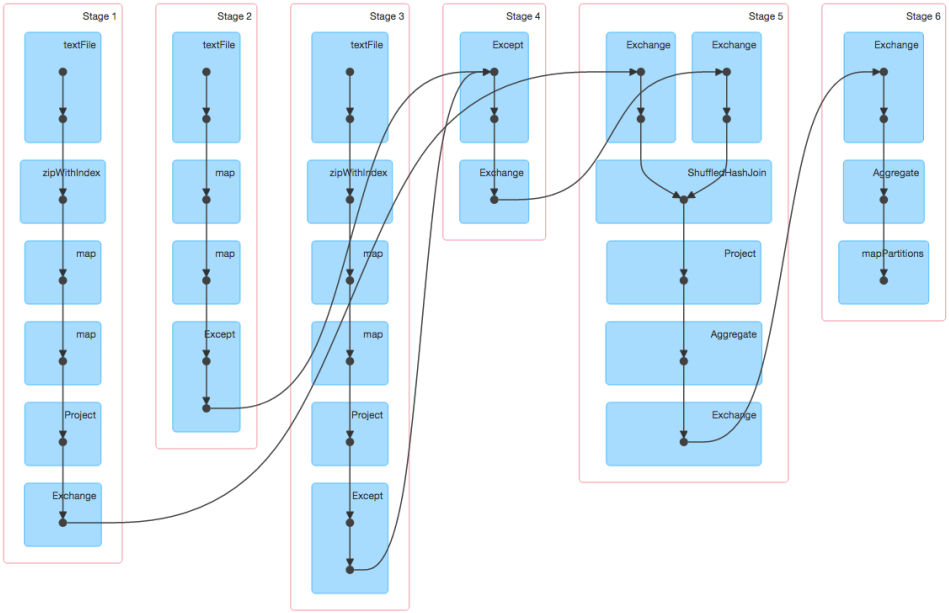


Stage 0 took 3 seconds with 1 task. Task 0 handled the zipWithIndex call and took approximately 1 second with an input size of 53.2 MB on Executor 1. This information is summarized above.

Details for Job 1

Status: SUCCEEDED
Completed Stages: 6





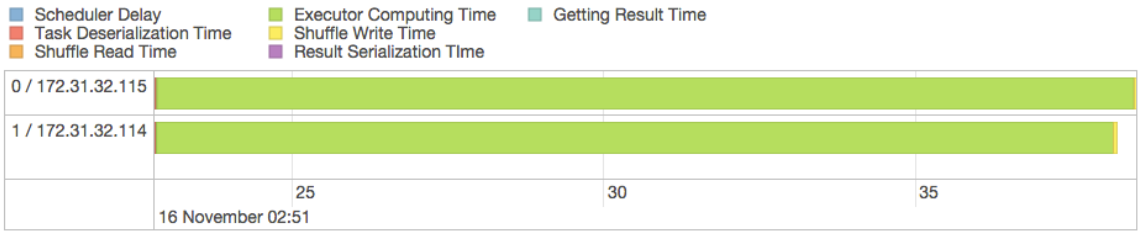
Completed Stages (6)

Stage Id	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
6	count at WikipediaPagesWithNoOutlinks.scala:46 +details	2015/11/16 07:51:39	0.1 s	1/1			6.1 KB	
5	count at WikipediaPagesWithNoOutlinks.scala:46 +details	2015/11/16 07:51:38	1 s	200/200			50.8 MB	6.1 KB
4	count at WikipediaPagesWithNoOutlinks.scala:46 +details	2015/11/16 07:51:22	16 s	2/2			71.3 MB	130.3 KB
3	count at WikipediaPagesWithNoOutlinks.scala:46 +details	2015/11/16 07:51:10	12 s	2/2	106.4 MB			30.9 MB
2	count at WikipediaPagesWithNoOutlinks.scala:46 +details	2015/11/16 07:51:10	8 s	8/8	1009.4 MB			40.4 MB
1	count at WikipediaPagesWithNoOutlinks.scala:46 +details	2015/11/16 07:51:10	8 s	2/2	106.4 MB			50.7 MB

Job 1 was comprised of 6 Stages. Stage 1 is comprised of 2 tasks and reads in titles-sorted.txt and then performs zipWithIndex, which it maps then projects and exchanges with data from Stage 5. Stage 2 reads in links-simple-sorted.txt and then performs two maps and exchanges it with data from Stage 4. Stage 3 takes the titles-sorted.txt RDD and also projects out the titles so that all of the IDs are the only thing left in the table and then exchanges data with Stage 4. Stage 4 performs the subtract between the titles ID table and the links ID table to get the IDs of pages with no outlinks and then passes this data to Stage 5. Stage 5 performs a Hash Join on its two input data sets to get all of the IDs and titles of Wikipedia pages with no outlinks and passes it to Stage 6. Stage 6 prints out the results. Stage 4 took the longest, which makes sense since it's performing the subtraction on the full data sets. Even though Stage 5 performs the join, it is performing the join on a much smaller data set, which makes it faster than Stage 4. Task 0 of Stage 4 took the longest to run at 16 seconds and ran on executor 0 . Task 0 had shuffle read and write sizes of 35.7 MB and 64.8 MB and no input data since it didn't have to read any text files in. This operation takes much longer than compared to the corresponding part of the algorithm in the original implementation because set subtractions on tables are expensive. This information is summarized below.

Tasks

Index	ID	Attempt	Status	Locality Level	Executor ID / Host	Launch Time	Duration	GC Time	Shuffle Read Size / Records	Write Time	Shuffle Write Size / Records	Errors
0	13	0	SUCCESS	PROCESS_LOCAL	0 / 172.31.32.115	2015/11/16 07:51:22	16 s	0.2 s	35.7 MB / 5711474	72 ms	64.8 KB / 5332	
1	14	0	SUCCESS	PROCESS_LOCAL	1 / 172.31.32.114	2015/11/16 07:51:22	15 s	0.2 s	35.6 MB / 5711404	64 ms	65.5 KB / 5406	



Executor data is shown below. Executor 0 completed 88 tasks, with a total input size of 603.8 MB with shuffle read and write sizes of 27.6 MB and 61.9 MB respectively. Executor 1 coplted 128 tasks with a total input size of 671.6 MB and shuffle read and write sizes of 33.5 MB and 60.2 MB respectively. This corroborates the trend that has been seen so far of much higher tasks for the Spark SQL and DataFrame implementations.

Executors (3)

Memory: 0.0 B Used (29.2 GB Total)
Disk: 0.0 B Used

Executor ID	Address	RDD Blocks	Storage Memory	Disk Used	Active Tasks	Failed Tasks	Complete Tasks	Total Tasks	Task Time	Input	Shuffle Read	Shuffle Write	Logs
0	172.31.32.115:42556	0	0.0 B / 13.6 GB	0.0 B	0	0	88	88	1.3 m	603.8 MB	27.6 MB	61.9 MB	stdout stderr
1	172.31.32.114:44999	0	0.0 B / 13.6 GB	0.0 B	0	0	128	128	1.1 m	671.6 MB	33.5 MB	60.2 MB	stdout stderr
driver	172.31.37.133:57430	0	0.0 B / 2.1 GB	0.0 B	0	0	0	0	0 ms	0.0 B	0.0 B	0.0 B	

Logical and Physical Plans

```
== Parsed Logical Plan ==
Except
 Project [id#0]
  LogicalRDD [id#0,title#1], MapPartitionsRDD[4] at createDataFrame at WikipediaPagesWithNoOutlinks.scala:34
 LogicalRDD [id#2], MapPartitionsRDD[8] at createDataFrame at WikipediaPagesWithNoOutlinks.scala:39

== Analyzed Logical Plan ==
id: string
Except
 Project [id#0]
  LogicalRDD [id#0,title#1], MapPartitionsRDD[4] at createDataFrame at WikipediaPagesWithNoOutlinks.scala:34
 LogicalRDD [id#2], MapPartitionsRDD[8] at createDataFrame at WikipediaPagesWithNoOutlinks.scala:39

== Optimized Logical Plan ==
Except
 Project [id#0]
  LogicalRDD [id#0,title#1], MapPartitionsRDD[4] at createDataFrame at WikipediaPagesWithNoOutlinks.scala:34
 LogicalRDD [id#2], MapPartitionsRDD[8] at createDataFrame at WikipediaPagesWithNoOutlinks.scala:39

== Physical Plan ==
Except
 Project [id#0]
  PhysicalRDD [id#0,title#1], MapPartitionsRDD[4] at createDataFrame at WikipediaPagesWithNoOutlinks.scala:34
 PhysicalRDD [id#2], MapPartitionsRDD[8] at createDataFrame at WikipediaPagesWithNoOutlinks.scala:39
```

Operators in Physical Plan:

- Except takes two tables and performs a set subtraction on them.
- Project takes an input of column names and a table and returns the table with only the specified columns.
- Physical RDD gives the schema of the output and represents the table.
- MapPartitionRDD specifies the location of the RDD in memory (byte addressable).

Alternative Logical Plan:

Another logical plan that Spark could've used is to get all of the tables with outlinks and then instead of doing a subtract with all of the ids to get the pages with no outlinks, to use the NOT IN keywords such that you would just output a tuple if its ID is not in the table of pages with outlinks. Implemented with a subquery, this would be more efficient than the current solution because it wouldn't involve a join, but this couldn't be implemented because neither SQLContext nor HiveContext currently have support for subqueries.

Alternative Physical Plan:

The optimized logical plan and physical plan shown above are exactly the same, so what I mentioned previously about the alternative logical plan would also apply. Additionally, however, different types of joins could be used. For instance, a sort-merge join would probably be really efficient since IDs in each data set are unique so both data can fit into memory making it preferable to a reduce side join which would have to iterate and sort through the combined data of both tables.

WikipediaPagesWithNoInlinks.scala without DataFrame/SparkSQL

Completed Jobs (2)

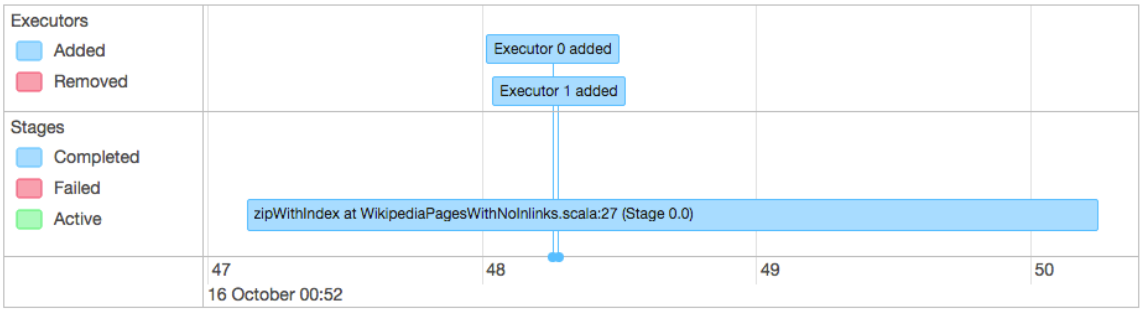
Job Id	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
1	count at WikipediaPagesWithNoInlinks.scala:37	2015/10/16 04:52:50	32 s	6/6	24/24
0	zipWithIndex at WikipediaPagesWithNoInlinks.scala:27	2015/10/16 04:52:47	3 s	1/1	1/1

The application runs two jobs with a total of 7 Stages for computing all of the Wikipedia pages with no inlinks. Job 0 was comprised of 1 stage of 1 task and took 3 seconds total. Job 1 was comprised of 6 stages and 24 tasks total, which took a total of 32 seconds.

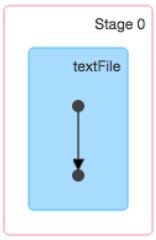
Details for Job 0

Status: SUCCEEDED
Completed Stages: 1

▾ Event Timeline
☐ Enable zooming

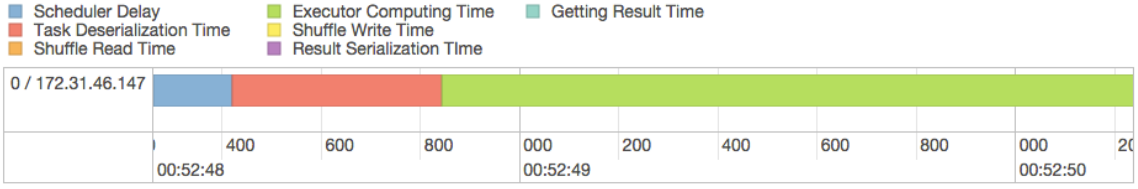


▾ DAG Visualization



Completed Stages (1)

Stage Id	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
0	zipWithIndex at WikipediaPagesWithNoInlinks.scala:27 +details	2015/10/16 04:52:47	3 s	1/1	53.2 MB			



Tasks

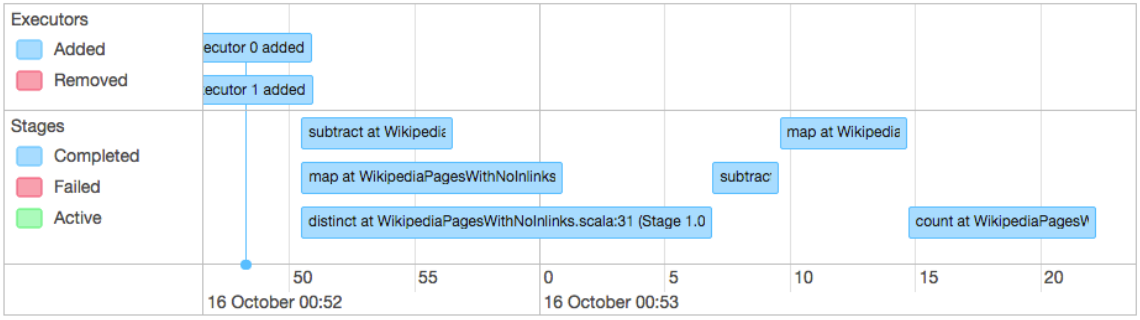
Index	ID	Attempt	Status	Locality Level	Executor ID / Host	Launch Time	Duration	GC Time	Input Size / Records	Errors
0	0	0	SUCCESS	ANY	0 / 172.31.46.147	2015/10/16 04:52:48	1 s		53.2 MB (hadoop) / 2915094	

Stage 0 took 3 seconds with 1 task. Task 0 handled the zipWithIndex call and took approximately 1.5 seconds with an input size of 53.2 MB on Executor 0. This information is summarized above.

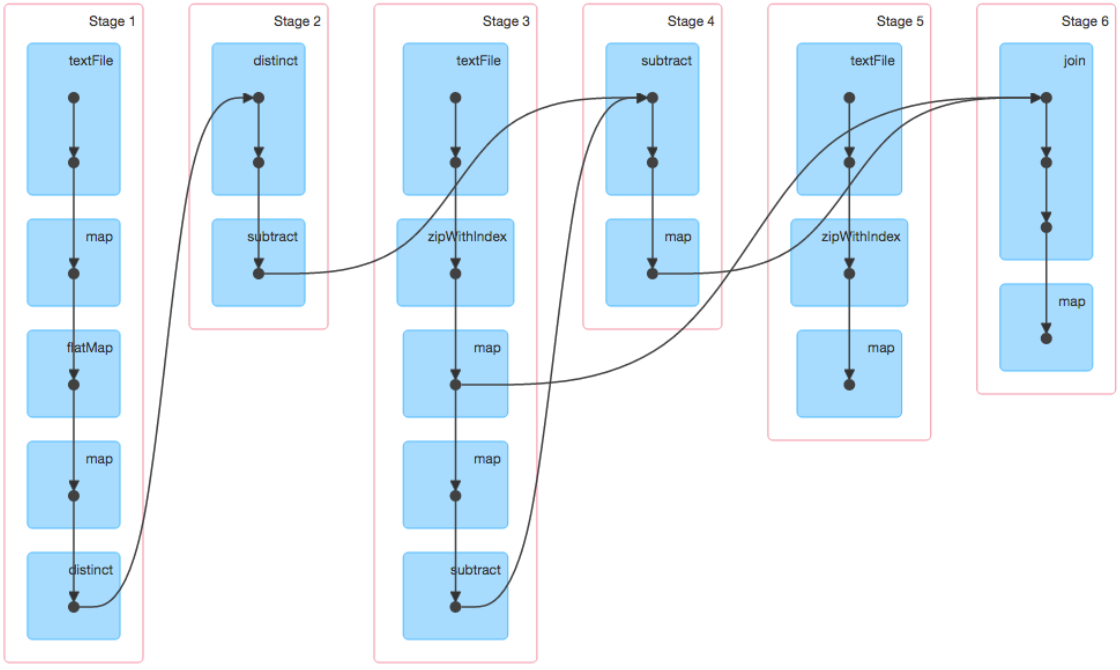
Details for Job 1

Status: SUCCEEDED
Completed Stages: 6

▾ Event Timeline
☐ Enable zooming



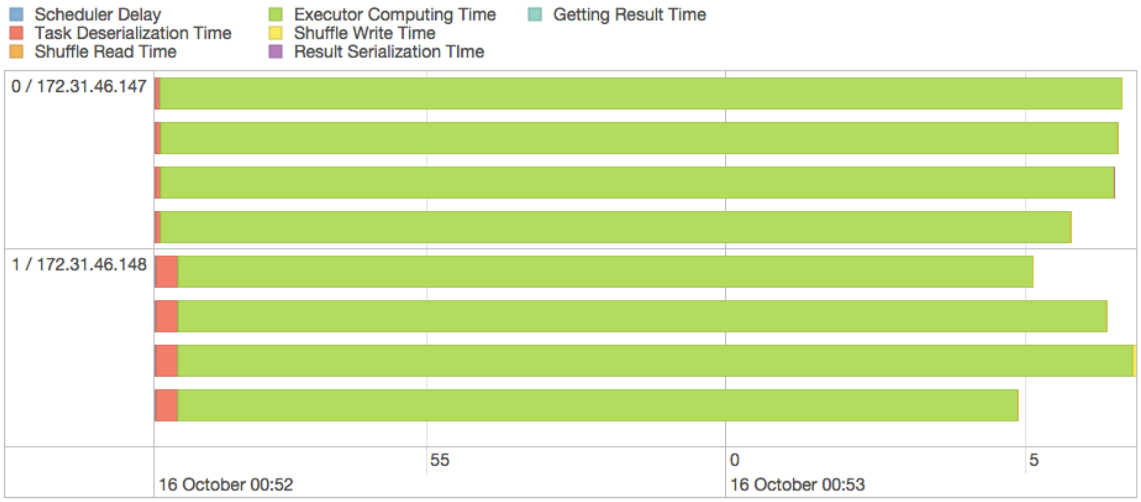
▾ DAG Visualization



Completed Stages (6)

Stage Id	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
6	count at WikipediaPagesWithNoInlinks.scala:37 +details	2015/10/16 04:53:14	7 s	2/2			97.5 MB	
4	map at WikipediaPagesWithNoInlinks.scala:35 +details	2015/10/16 04:53:09	5 s	2/2			49.5 MB	10.0 MB
2	subtract at WikipediaPagesWithNoInlinks.scala:33 +details	2015/10/16 04:53:06	3 s	8/8			72.7 MB	21.7 MB
5	map at WikipediaPagesWithNoInlinks.scala:27 +details	2015/10/16 04:52:50	10 s	2/2	106.4 MB			87.5 MB
3	subtract at WikipediaPagesWithNoInlinks.scala:33 +details	2015/10/16 04:52:50	6 s	2/2	106.4 MB			27.8 MB
1	distinct at WikipediaPagesWithNoInlinks.scala:31 +details	2015/10/16 04:52:50	16 s	8/8	1009.4 MB			72.7 MB

Job 1 was comprised of 6 Stages. Stages 1 and 2 are comprised of 8 tasks and Stages 3 through 6 are comprised of 2 tasks. Stage 1 reads in links-simple-sorted.txt and then performs a map, flatMap, another map, and distinct on the data. The resulting data is used in Stage 2 and subtracted with the result of Stage 3 in Stage 4. Stage 3 reads in titles-sorted.txt and then performs a zipWithIndex, two maps, then eventually a subtract in Stage 4 with the result of Stage 2. Stage 4 performs the subtract of data from Stages 2 and 3 then maps it. Stage 5 reads in titles-sorted.txt and then performs a zipWithIndex and map on it. Stage 6 takes the result from Stage 4 and an intermediate mapped RDD from Stage 3 and joins them together then performs a final map to get the resulting RDD. All of the longest tasks are in Stage 1, with 5 tasks taking approximately 16 seconds to complete. Task 0 of Stage 1 took the longest to run on Executor 0 by fractions of a second. Task 0 had an input size of 128 MB and a shuffle size of 8.5 MB. This information is summarized below.



Tasks

Index	ID	Attempt	Status	Locality Level	Executor ID / Host	Launch Time	Duration ▾	GC Time	Input Size / Records	Write Time	Shuffle Write Size / Records	Errors
0	1	0	SUCCESS	ANY	0 / 172.31.46.147	2015/10/16 04:52:50	16 s	0.5 s	128.0 MB (hadoop) / 634912	15 ms	8.5 MB / 1740228	
5	6	0	SUCCESS	ANY	1 / 172.31.46.148	2015/10/16 04:52:50	16 s	0.6 s	128.0 MB (hadoop) / 787260	90 ms	9.0 MB / 1845995	
2	3	0	SUCCESS	ANY	0 / 172.31.46.147	2015/10/16 04:52:50	16 s	0.5 s	128.0 MB (hadoop) / 813206	15 ms	9.0 MB / 1850944	
4	5	0	SUCCESS	ANY	0 / 172.31.46.147	2015/10/16 04:52:50	16 s	0.5 s	128.0 MB (hadoop) / 522056	19 ms	11.3 MB / 2324823	
3	4	0	SUCCESS	ANY	1 / 172.31.46.148	2015/10/16 04:52:50	16 s	0.6 s	128.0 MB (hadoop) / 759860	14 ms	8.9 MB / 1825635	
6	7	0	SUCCESS	ANY	0 / 172.31.46.147	2015/10/16 04:52:50	15 s	0.5 s	128.0 MB (hadoop) / 775011	16 ms	8.9 MB / 1827668	
1	2	0	SUCCESS	ANY	1 / 172.31.46.148	2015/10/16 04:52:50	14 s	0.6 s	128.0 MB (hadoop) / 741608	14 ms	8.9 MB / 1819931	
7	8	0	SUCCESS	ANY	1 / 172.31.46.148	2015/10/16 04:52:50	14 s	0.6 s	113.4 MB (hadoop) / 672157	14 ms	8.3 MB / 1694130	

Executor data is shown below. Executor 0 completed 13 tasks, with a total input size of 671.6 MB, shuffle read size of 61.7 MB, and shuffle write size of 112.2 MB. Executor 1 completed 12 tasks, with a total input size of 603.8 MB, shuffle read size of 64 MB, and shuffle write size of 107.5 MB.

Executors (3)

Memory: 0.0 B Used (27.4 GB Total)
Disk: 0.0 B Used

Executor ID	Address	RDD Blocks	Storage Memory	Disk Used	Active Tasks	Failed Tasks	Complete Tasks	Total Tasks	Task Time	Input	Shuffle Read	Shuffle Write	Logs
0	172.31.46.147:56909	0	0.0 B / 13.6 GB	0.0 B	0	0	13	13	1.7 m	671.6 MB	61.7 MB	112.2 MB	stdout stderr
1	172.31.46.148:59739	0	0.0 B / 13.6 GB	0.0 B	0	0	12	12	1.7 m	603.8 MB	64.0 MB	107.5 MB	stdout stderr
driver	172.31.44.186:57431	0	0.0 B / 265.4 MB	0.0 B	0	0	0	0	0 ms	0.0 B	0.0 B	0.0 B	

WikipediaPagesWithNoInlinks.scala with DataFrame/SparkSQL

Completed Jobs (2)

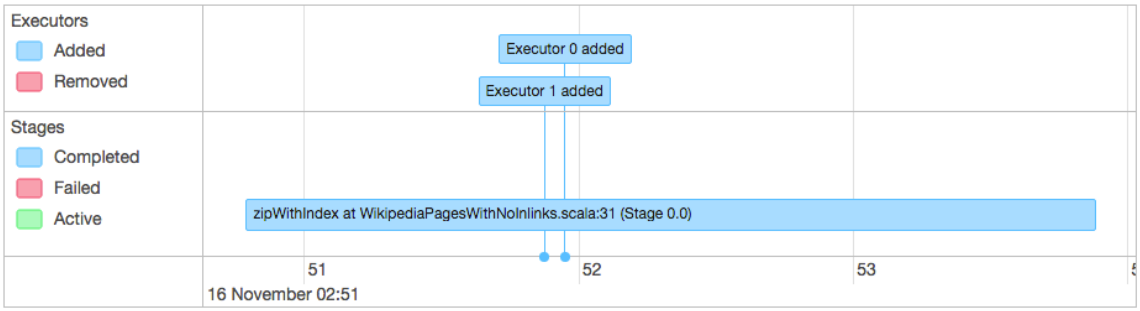
Job Id	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
1	count at WikipediaPagesWithNoInlinks.scala:47	2015/11/16 07:51:55	43 s	7/7	223/223
0	zipWithIndex at WikipediaPagesWithNoInlinks.scala:31	2015/11/16 07:51:50	3 s	1/1	1/1

The application runs two jobs with a total of 8 Stages for computing all of the Wikipedia pages with no inlinks. Job 0 was comprised of 1 stage of 1 task and took 3 seconds total. Job 1 was comprised of 7 Stages and 223 tasks total, which took a total of 43 seconds. This implementation required one more stage than the original implementation.

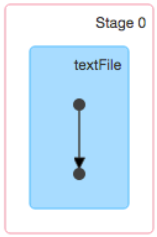
Details for Job 0

Status: SUCCEEDED
Completed Stages: 1

▾ Event Timeline
☐ Enable zooming

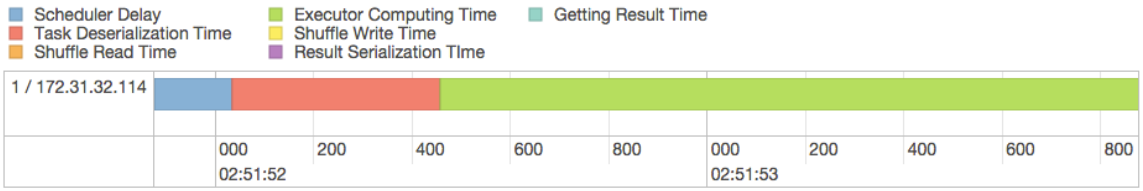


▾ DAG Visualization



Completed Stages (1)

Stage Id	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
0	zipWithIndex at WikipediaPagesWithNoInlinks.scala:31 +details	2015/11/16 07:51:50	3 s	1/1	53.2 MB			



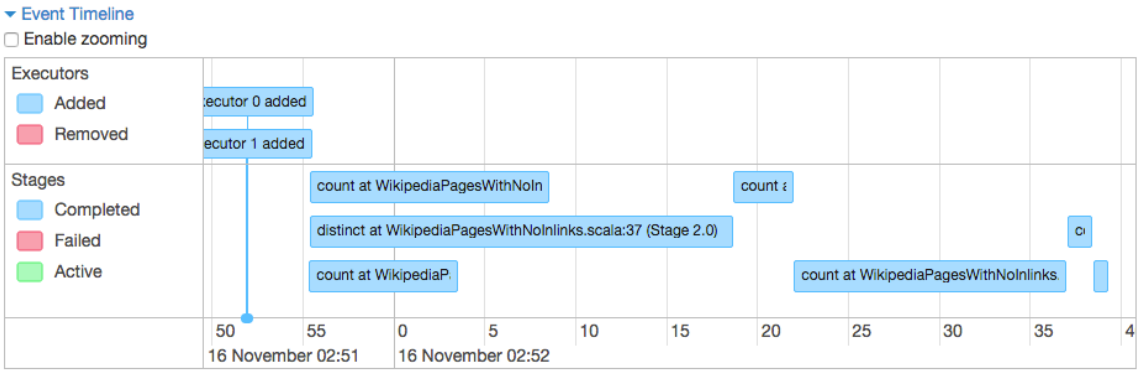
Tasks

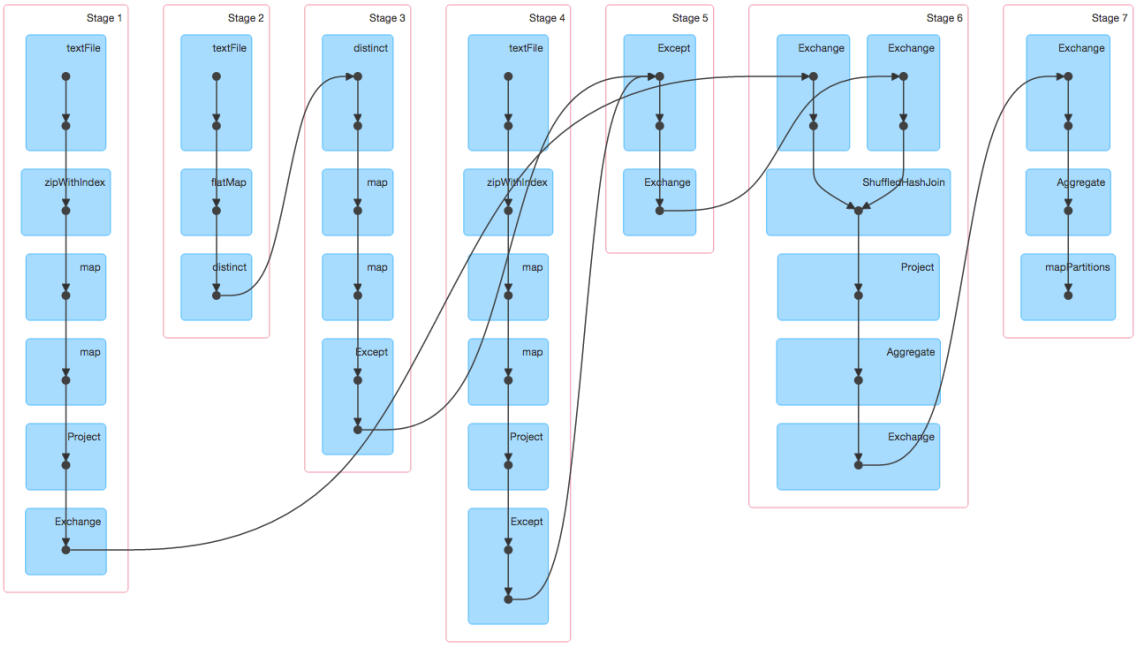
Index	ID	Attempt	Status	Locality Level	Executor ID / Host	Launch Time	Duration	GC Time	Input Size / Records	Errors
0	0	0	SUCCESS	ANY	1 / 172.31.32.114	2015/11/16 07:51:51	1 s		53.2 MB (hadoop) / 2915094	

Stage 0 took 3 seconds with 1 task. Task 0 handled the zipWithIndex call and took approximately 1 second with an input size of 53.2 MB on Executor 1. This information is summarized above.

Details for Job 1

Status: SUCCEEDED
Completed Stages: 7



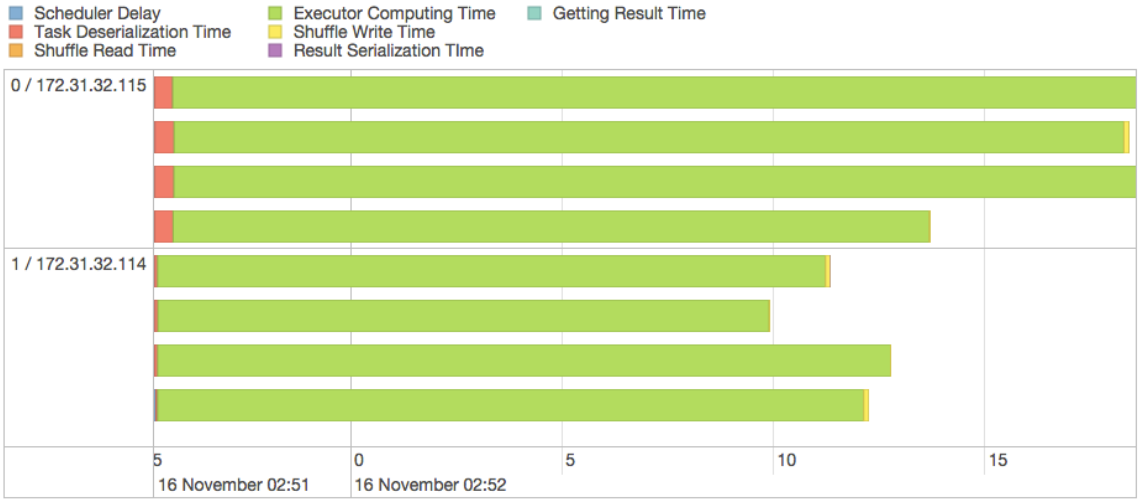


Completed Stages (7)

Stage Id	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
7	count at WikipediaPagesWithNoInlinks.scala:47+details	2015/11/16 07:52:38	0.1 s	1/1			6.1 KB	
6	count at WikipediaPagesWithNoInlinks.scala:47+details	2015/11/16 07:52:37	1 s	200/200			69.4 MB	6.1 KB
5	count at WikipediaPagesWithNoInlinks.scala:47+details	2015/11/16 07:52:21	15 s	2/2			71.0 MB	18.7 MB
3	count at WikipediaPagesWithNoInlinks.scala:47+details	2015/11/16 07:52:18	3 s	8/8			74.7 MB	40.1 MB
4	count at WikipediaPagesWithNoInlinks.scala:47+details	2015/11/16 07:51:55	13 s	2/2	106.4 MB			30.9 MB
2	distinct at WikipediaPagesWithNoInlinks.scala:37 +details	2015/11/16 07:51:55	23 s	8/8	1009.4 MB			74.7 MB
1	count at WikipediaPagesWithNoInlinks.scala:47+details	2015/11/16 07:51:55	8 s	2/2	106.4 MB			50.7 MB

Job 1 was comprised of 7 Stages. Stage 1 was comprised of 2 tasks and read in titles-sorted.txt and then zipped it with an index to represent its ID and passes this information to Stage 4. Stage 2 reads in links-simple-sorted.txt and extracts the IDs on the right of the colon with a flatMap and then passes this to Stage 3 after calling distinct to get rid of duplicate IDs. Stage 3 performs further mapping and processing of the information then passes the RDD to Stage 5. Stage 4 takes titles-sorted.txt after zipWithIndex is applied to it and then projects out the title information and passes it to Stage 5. Stage 5 performs a set subtraction on the title IDs and the link IDs to get all of the IDs of pages with no inlinks and then passes this information to Stage 6. Stage 6 takes the IDs of pages with no inlinks and the titles table with titles and IDs and performs a join to get all of the page titles that have no inlinks and then passes this information to Stage 7. Stage 7 prints the number of pages with no inlinks. Stage 2 took the longest, which makes sense since it has to do

a full table scan over a table of all pages with inlinks that also has a lot of duplicates (many Wikipedia pages are linked from many other pages). Task 4 of Stage 2 took the longest to run at 23 second on Executor 0. Task 4 had an input size of 128 MB and a shuffle size of 11.5 MB. This is different than the original implementation where the longest task came from joining the two RDDs. This information is summarized below.



Tasks

Index	ID	Attempt	Status	Locality Level	Executor ID / Host	Launch Time	Duration	GC Time	Input Size / Records	Write Time	Shuffle Write Size / Records	Error
0	3	0	SUCCESS	ANY	0 / 172.31.32.115	2015/11/16 07:51:55	23 s	1.0 s	128.0 MB (hadoop) / 634912	16 ms	8.7 MB / 1740228	
1	4	0	SUCCESS	ANY	1 / 172.31.32.114	2015/11/16 07:51:55	16 s	1 s	128.0 MB (hadoop) / 741608	0.1 s	9.1 MB / 1819931	
2	5	0	SUCCESS	ANY	0 / 172.31.32.115	2015/11/16 07:51:55	23 s	1.0 s	128.0 MB (hadoop) / 813206	0.1 s	9.3 MB / 1850944	
3	6	0	SUCCESS	ANY	1 / 172.31.32.114	2015/11/16 07:51:55	14 s	0.7 s	128.0 MB (hadoop) / 759860	15 ms	9.2 MB / 1825635	
4	7	0	SUCCESS	ANY	0 / 172.31.32.115	2015/11/16 07:51:55	23 s	1.0 s	128.0 MB (hadoop) / 522056	20 ms	11.5 MB / 2324823	
5	8	0	SUCCESS	ANY	1 / 172.31.32.114	2015/11/16 07:51:55	17 s	1 s	128.0 MB (hadoop) / 787260	15 ms	9.2 MB / 1845995	
6	9	0	SUCCESS	ANY	0 / 172.31.32.115	2015/11/16 07:51:55	18 s	1.0 s	128.0 MB (hadoop) / 775011	23 ms	9.2 MB / 1827668	
7	10	0	SUCCESS	ANY	1 / 172.31.32.114	2015/11/16 07:51:55	17 s	1.0 s	113.4 MB (hadoop) / 672157	0.1 s	8.5 MB / 1694130	

Executor data is shown below. Executor 0 completed 119 tasks with a total input size of 618.4 MB and shuffle read and write sizes of 62.8 MB and 108.2 MB

respectively. Executor 1 completed 105 tasks with a total input size of 657 MB and shuffle read and write sizes of 65.8 MB and 106.9MB respectively. This is about the same as the original implementation except for the much higher number of tasks

Executors (3)

Memory: 0.0 B Used (29.2 GB Total)
Disk: 0.0 B Used

Executor ID	Address	RDD Blocks	Storage Memory	Disk Used	Active Tasks	Failed Tasks	Complete Tasks	Total Tasks	Task Time	Input	Shuffle Read	Shuffle Write	Logs
0	172.31.32.115:58683	0	0.0 B / 13.6 GB	0.0 B	0	0	105	105	2.5 m	618.4 MB	62.8 MB	108.2 MB	stdout stderr
1	172.31.32.114:58704	0	0.0 B / 13.6 GB	0.0 B	0	0	119	119	2.1 m	657.0 MB	65.8 MB	106.9 MB	stdout stderr
driver	172.31.37.133:48790	0	0.0 B / 2.1 GB	0.0 B	0	0	0	0	0 ms	0.0 B	0.0 B	0.0 B	

Logical and Physical Plans

```
== Parsed Logical Plan ==
Project [id#0,title#4]
 Join Inner, Some((id#0 = id#3))
  Except
   Project [id#0]
    LogicalRDD [id#0,title#1], MapPartitionsRDD[4] at createDataFrame at WikipediaPagesWithNoInlinks.scala:34
   LogicalRDD [id#2], MapPartitionsRDD[12] at createDataFrame at WikipediaPagesWithNoInlinks.scala:40
  LogicalRDD [id#3,title#4], MapPartitionsRDD[4] at createDataFrame at WikipediaPagesWithNoInlinks.scala:34

== Analyzed Logical Plan ==
id: string, title: string
Project [id#0,title#4]
 Join Inner, Some((id#0 = id#3))
  Except
   Project [id#0]
    LogicalRDD [id#0,title#1], MapPartitionsRDD[4] at createDataFrame at WikipediaPagesWithNoInlinks.scala:34
   LogicalRDD [id#2], MapPartitionsRDD[12] at createDataFrame at WikipediaPagesWithNoInlinks.scala:40
  LogicalRDD [id#3,title#4], MapPartitionsRDD[4] at createDataFrame at WikipediaPagesWithNoInlinks.scala:34

== Optimized Logical Plan ==
Project [id#0,title#4]
 Join Inner, Some((id#0 = id#3))
  Except
   Project [id#0]
    LogicalRDD [id#0,title#1], MapPartitionsRDD[4] at createDataFrame at WikipediaPagesWithNoInlinks.scala:34
   LogicalRDD [id#2], MapPartitionsRDD[12] at createDataFrame at WikipediaPagesWithNoInlinks.scala:40
  LogicalRDD [id#3,title#4], MapPartitionsRDD[4] at createDataFrame at WikipediaPagesWithNoInlinks.scala:34

== Physical Plan ==
Project [id#0,title#4]
 ShuffledHashJoin [id#0], [id#3], BuildRight
  Exchange (HashPartitioning 200)
   Except
    Project [id#0]
     PhysicalRDD [id#0,title#1], MapPartitionsRDD[4] at createDataFrame at WikipediaPagesWithNoInlinks.scala:34
    PhysicalRDD [id#2], MapPartitionsRDD[12] at createDataFrame at WikipediaPagesWithNoInlinks.scala:40
  Exchange (HashPartitioning 200)
   PhysicalRDD [id#3,title#4], MapPartitionsRDD[4] at createDataFrame at WikipediaPagesWithNoInlinks.scala:34
```

Operators in Physical Plan:

- ShuffleHashJoin specifies using a hash join while exchanging data between partitions to get the correct data together.

- BuildRight specifies to read in data from the right side of the "tree" to perform the ShuffleHashJoin.
- Exchange specifies that HashPartitioning is used to partition the data.
- Except takes two tables and performs a set subtraction on them.
- Project takes an input of column names and a table and returns the table with only the specified columns.
- Physical RDD gives the schema of the output and represents the table.
- MapPartitionRDD specifies the location of the RDD in memory (byte addressable).

Alternative Logical Plan:

Another logical plan that Spark could've used is to get all of the tables with inlinks and then instead of doing a subtract with all of the ids to get the pages with no inlinks, to use the NOT IN keywords such that you would just output a tuple if its ID is not in the table of pages with inlinks. Implemented with a subquery, this would be more efficient than the current solution because it wouldn't involve a join, but this couldn't be implemented because neither SQLContext nor HiveContext currently have support for subqueries.

Alternative Physical Plan:

Another physical plan that Spark could've used is to use an IndexJoin on the two tables with the filtered link IDs as the left subtree and the titles ID as the right subtree. This is hugely inefficient since no indices exist for either of these tables so an IndexJoin isn't truly possible and amounts to a FullTableScan. Having the titles ID on the right as opposed to the filtered link IDs is also hugely inefficient since you want the smaller table on the right, so that you only read in the bigger table once into memory and not repeatedly.