For Wikipedia:

Below are the plans printed during runtime using the explain(true) function on the SQL Query on the dataframes.

1) Dataframe for all pages with inlinks

The query `SELECT page_id FROM row1_and_row2_table` extracts the `inlinks_page_id` from `row1_and_row2` where row 1 and row 2 represents the left and right hand sides of the dataset separated by a colon ":" in the input file in the format (i.e. 3: 344 234 1) where the value on the lhs of the colon represents page id with inlinks from the rhs values. Hence, in this case the given query extracts everything on the **left hand side of the colon** i.e. the **pages with inlinks**.

```
== Parsed Logical Plan ==
Aggregate [page_id#0], [page_id#0]
 Project [page_id#0]
  Subquery row1_and_row2_table
   LogicalRDD [page_id#0,outlinks_page_id#1], MapPartitionsRDD[4] at createDataFrame at Wikipedia.scala:59

== Analyzed Logical Plan ==
page_id: string
Aggregate [page_id#0], [page_id#0]
 Project [page_id#0]
  Subquery row1_and_row2_table
   LogicalRDD [page_id#0,outlinks_page_id#1], MapPartitionsRDD[4] at createDataFrame at Wikipedia.scala:59

== Optimized Logical Plan ==
Aggregate [page_id#0], [page_id#0]
 Project [page_id#0]
  LogicalRDD [page_id#0,outlinks_page_id#1], MapPartitionsRDD[4] at createDataFrame at Wikipedia.scala:59

== Physical Plan ==
TungstenAggregate(key=[page_id#0], functions=[], output=[page_id#0])
 TungstenExchange hashpartitioning(page_id#0)
  TungstenAggregate(key=[page_id#0], functions=[], output=[page_id#0])
   TungstenProject [page_id#0]
    Scan PhysicalRDD[page_id#0,outlinks_page_id#1]
```

In the above physical plan, the operators used are:

- Physical Scan (on the entire table, both relations)
- Project (the `inlinks_page_id` relation)
- Aggregate (using `inlinks_page_id` relation)
- Exchange (hash partitioned `inlinks_page_id`'s for processing)
- Aggregate (using `inlinks_page_id` relation for creating a new relation)

2) Dataframe for all pages with outlinks

Similarly, the another query `SELECT outlinks_page_id FROM row1_and_row2_table` extracts all the values on the **right hand of the colon** i.e. the **pages with outlinks**. The plans are shown below:

```
== Parsed Logical Plan ==
'Project [unresolvedalias('outlinks_page_id)]
 'UnresolvedRelation [row1_and_row2_table], None

== Analyzed Logical Plan ==
outlinks_page_id: string
Project [outlinks_page_id#1]
 Subquery row1_and_row2_table
  LogicalRDD [page_id#0,outlinks_page_id#1], MapPartitionsRDD[4] at createDataFrame at Wikipedia.scala:59

== Optimized Logical Plan ==
Project [outlinks_page_id#1]
 LogicalRDD [page_id#0,outlinks_page_id#1], MapPartitionsRDD[4] at createDataFrame at Wikipedia.scala:59

== Physical Plan ==
TungstenProject [outlinks_page_id#1]
 Scan PhysicalRDD[page_id#0,outlinks_page_id#1]
```

In the above physical plan, same steps are done as the in the previous case, except that `outlinks_page_id's` are now stored in a relational format (RowRDD) and a physical scan is performed.

3) Dataframe for all pages

In the final phase, the titles file is used to extract all the indices which would be the total number of pages. Then, all the pages (where index = page id) are stored in form of a dataframe, and they are selected with the query `SELECT _1 FROM titles` (plans below)

```
== Parsed Logical Plan ==
'Project [unresolvedalias('_1)]
 'UnresolvedRelation [titles], None

== Analyzed Logical Plan ==
_1: bigint
Project [_1#5L]
 Subquery titles
  LogicalRDD [_1#5L,_2#6], MapPartitionsRDD[16] at rddToDataFrameHolder at Wikipedia.scala:102

== Optimized Logical Plan ==
Project [_1#5L]
 LogicalRDD [_1#5L,_2#6], MapPartitionsRDD[16] at rddToDataFrameHolder at Wikipedia.scala:102

== Physical Plan ==
TungstenProject [_1#5L]
 Scan PhysicalRDD[_1#5L,_2#6]
```

In the above plan which extracts all the indices from titles input file as page id's, `_1`, which is the indices before the title string in the file, are stored as a relation and a physical plan gives all the values.

Alternative Logical plan:

Instead of scanning both relations first, only scan the values before the colon (inlinks_page_id, #0) in the logicalRDD, on the subquery, project them and aggregate. This would be a better plan than the one spark picked.

Alternative Physical plan:

An optimized scan on row1_and_row2 would detect the values on either side of the colon, and only return the values on the left i.e. inlinks_page_id (without scanning the right hand side i.e. outlinks_page_id). This would save the cost for processing the values after the colon (greater in number, i.e. outlinks_page_id). A TNLJ could also be performed?

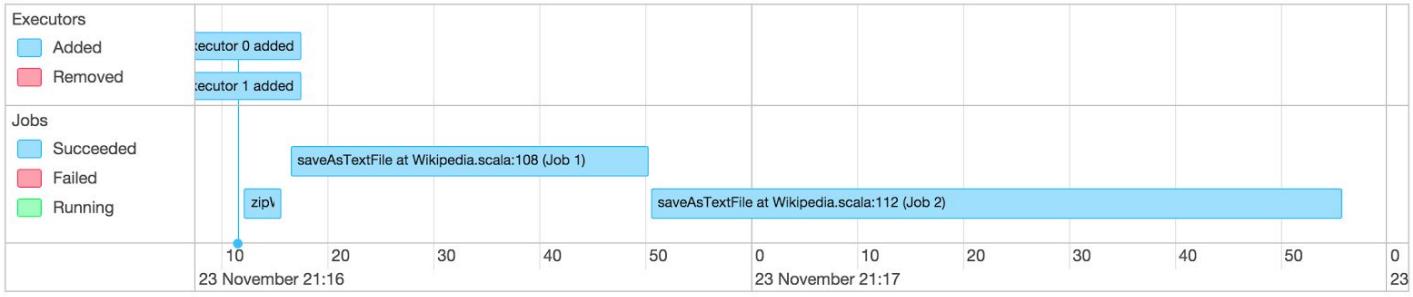
Also, if the scanning is optimized, the aggregation can be done once (saves cost too)

Spark 1.5.0 | Jobs | Stages | Storage | Environment | Executors | **hw5 (in progress)** application UI

Spark Jobs (?)

Scheduling Mode: FIFO
Completed Jobs: 3

▾ Event Timeline
☐ Enable zooming



Completed Jobs (3)

Job Id	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
2	saveAsTextFile at Wikipedia.scala:112	2015/11/24 02:16:50	1.1 min	4/4	206/206
1	saveAsTextFile at Wikipedia.scala:108	2015/11/24 02:16:16	34 s	4/4	206/206
0	zipWithIndex at Wikipedia.scala:97	2015/11/24 02:16:12	4 s	1/1	1/1

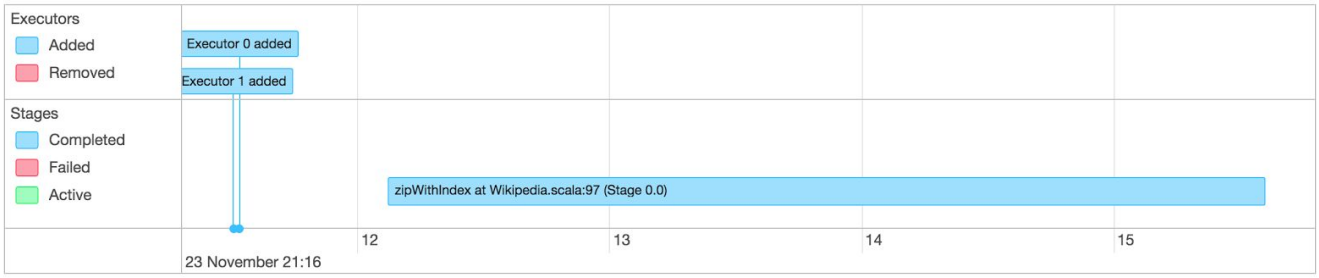
Spark 1.5.0 | Jobs | Stages | Storage | Environment | Executors | **hw5 (in progress)** application UI

Details for Job 0

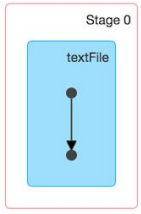
Status: SUCCEEDED
Completed Stages: 1

▾ Event Timeline

☐ Enable zooming



▾ DAG Visualization



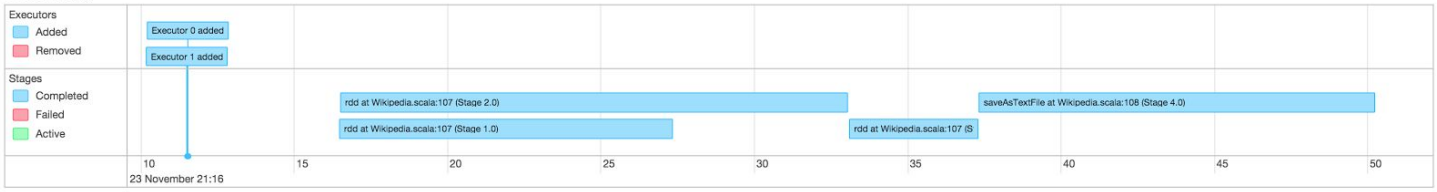
Completed Stages (1)

Stage Id	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
0	zipWithIndex at Wikipedia.scala:97 +details	2015/11/24 02:16:12	3 s	1/1	53.2 MB			

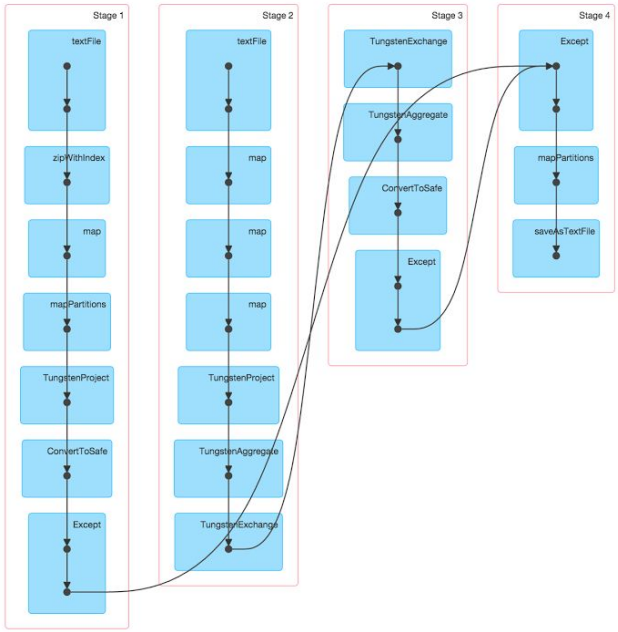
Details for Job 1

Status: SUCCEEDED
Completed Stages: 4

Event Timeline
Enable zooming



DAG Visualization



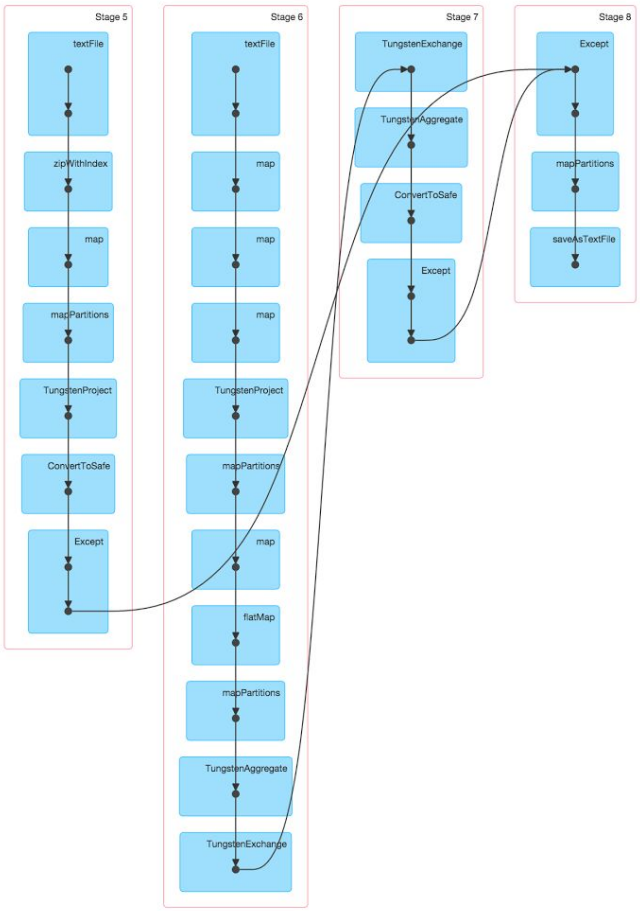
Completed Stages (4)

Stage Id	Description		Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
4	saveAsTextFile at Wikipedia.scala:108	+details	2015/11/24 02:16:37	13 s	2/2			72.7 MB	
3	rdd at Wikipedia.scala:107	+details	2015/11/24 02:16:33	4 s	200/200			37.9 MB	41.6 MB
2	rdd at Wikipedia.scala:107	+details	2015/11/24 02:16:16	17 s	2/2	1009.4 MB			37.9 MB
1	rdd at Wikipedia.scala:107	+details	2015/11/24 02:16:16	11 s	2/2	106.4 MB			31.1 MB

Details for Job 2

Status: SUCCEEDED
Completed Stages: 4

▸ Event Timeline
▾ DAG Visualization



Completed Stages (4)

Stage Id	Description		Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
8	saveAsTextFile at Wikipedia.scala:112	+details	2015/11/24 02:17:44	11 s	2/2			65.2 MB	
7	rdd at Wikipedia.scala:111	+details	2015/11/24 02:17:41	3 s	200/200			55.9 MB	34.1 MB
6	rdd at Wikipedia.scala:111	+details	2015/11/24 02:16:50	51 s	2/2	1009.4 MB			55.9 MB
5	rdd at Wikipedia.scala:111	+details	2015/11/24 02:16:50	9 s	2/2	106.4 MB			31.1 MB