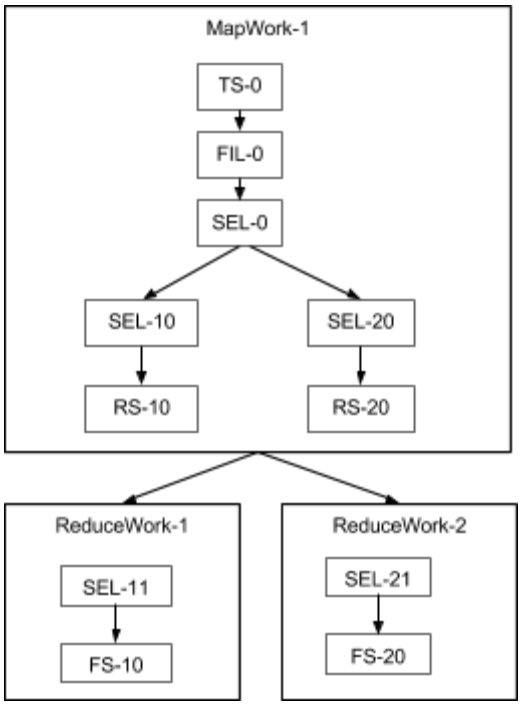
Problem

It is possible that a given MapWork or ReduceWork can output to two or more ReduceWorks.

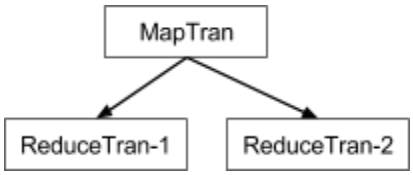
For a below given query:

```
FROM (SELECT * FROM kv WHERE key > 100) src
      INSERT OVERWRITE TABLE kvi1 SELECT * ORDER BY src.key
      INSERT OVERWRITE TABLE kvi2 SELECT * ORDER BY src.value;
```

We get following SparkWork. Operator tree of each work is given within the Work.



In the current SparkPlanGenerator above SparkWork is converted into: (MapWork-1 to MapTran and similarly ReduceWork-1/2 to ReduceTran-1/2)



MapWork or ReduceWork can only take one output collector, so that means both RS-10 and RS-20 from above example get the same output collector and both write to the same output collector. Downstream works ReduceWork-1 and ReduceWork-2, both get the all data that is added to the output collector which is incorrect.

Even if we change the MapWork/ReduceWork to take two or more output collectors, we have a limitation in Spark where each RDD can output only one set out data not multiple output data sets. In the above SparkPlan, MapTran-1 is expected to send each of the ReduceWorks a separate set of data, but in Spark number of output sets of RDD is just one, so we cannot give the above RDD tree for execution on Spark.

Solution we are proposing here is clone the MapWork-1 into MapWork-11 and MapWork-12. MapWork-11 contains the operator that justs outputs data to RedcueWork-1. Similarly MapWork-12 outputs data to ReduceWork-2. As both MapWork-11 and MapWork-12 take data from same HadoopRDD, caching it will help as it avoids reading the data from DFS multiple times.

Design

Preferred step for splitting the (Map/Reduce)Work is after the creation of complete SparkWork, as putting the splitting logic in SparkCompiler will only complicate the existing SparkWork generation. So the multiple output (Map/Reduce)Works will be split before generating the SparkPlan from SparkWork in SparkPlanGenerator.

How to map the leaf operators to ReduceWorks?

In above example, we need to have a way to identify RS-10 output is for ReduceWork-1. Currently when generating the SparkWork, we store this information in GenSparkProcContext, but this info is not passed in SparkWork to SparkPlanGenerator. One idea is to store the parent operator reference or ID in SparkEdgeProperty. However, this approach might be trickier than expected. Since we are cloning the works, storing references doesn't help much. Also, edge property by nature should not have info of the both ends.

One shortcut we can take is to match RS and following ReduceWork by the order in which they appeared in the operator tree and the work tree in SparkWork.

SparkWork traversal and splitting multiple output (Map/Reduce)Work

1. Create a map (`cloneToWork`) of from the cloned work to the original BaseWork record the clone-original relationship. This is because cloned works should share the same input to the original work.
2. Traverse SparkWork graph in BFS starting from roots (which are MapWorks) in SparkWork. For each work:
    a. If the work has less than or equal to one child, do nothing.
    b. Else:
        i. Find the number of child works, say N.
        ii. Clone the current work for N-1 times, and put the them in the map created above.
        iii. Modify the operator to tree in the original work, removing all branches leading to a RS except the first branch that leads to a RS. In another word, keep the first branch that leads to a RS and other branches that are not ending in RS (such as branches ending in FS).
        iv. Disconnect the work in (iii) from all child works except the first one. Assume the disconnect ones are number from 1, 2, …, N-1.

v. For each cloned work from (ii), Ci, where i starts from 1 to N-1, keep only the #i branch that leads to a RS by removing all other branches. Note that #i branch is numbered within all branches that lead to a RS.
vi. Connect cloned work Ci to the disconnected work from the original work in (iv) according to the sequence.
vii. Connect the cloned work Ci to the parent (if any) of the original Work.

Modify plan generation

The plan generation is largely unchanged except that he cloned works should share the input of the original work. The input should be either a MapInput, or the output from a shuffle.

Identify RDDs to cache

If we split a Work then we cache its parent RDD. To clarify, the parent RDD of a ReduceWork is the output the shuffler, not the output of previous work. For this, I think we can isolate shuffle from SparkPlan and create ShuffleTran. What needs to be cached is the output RDD of the ShuffleTran. Both MapInput and ShuffleTran have a caching flag, if true, enabling output caching. All Works generated from splitting a Work are stored in `cloneToWork`. Whenever we are creating a Tran for a Work or creating MapInput from a MapWork, we check if the work is a clone of anothe work by consulting `cloneToWork`. If yes, the clone will share the input of the original work and the caching flag of the input needs to be set accordingly..