Pig Performance Optimization

Introduction

As PIG-2397 pointed out, Pig is slower than Hive for some types of queries. In this doc, we simplify those queries and identify the bottlenecks of Pig's performance, and then we propose some possible optimization.

Environment Setup

- Hadoop 1.0, Pig trunk, Hive trunk
- TPC-H 1GB data
- one EC2 large instance
- 1GB DFS block size (to enforce 1 map task), 1 reducer

Order-By Issues

Description

Currently Pig implements Order-By strictly by first using a MR job to sample the data before sorting. The sampling job can result in poor performance for two reasons:

- it currently forces its previous pipelines to finish before it, which is likely to incur an extra job (PIG 2661);
- for small data set, the Order-By is implemented with one reducer, where the sampling is not useful at all (PIG-483).

Queries

Pig

```
LineItems = LOAD '$input/lineitem' USING PigStorage('|') AS (orderkey:int, partkey, suppkey, linenumber,
quantity, extendedprice, discount, tax, returnflag, linestatus, shipdate, commitdate, receiptdate,
shipinstruct, shipmode, comment);

Result = Order LineItems by orderkey;

STORE Result INTO '$output/OrderBy';
```

Hive

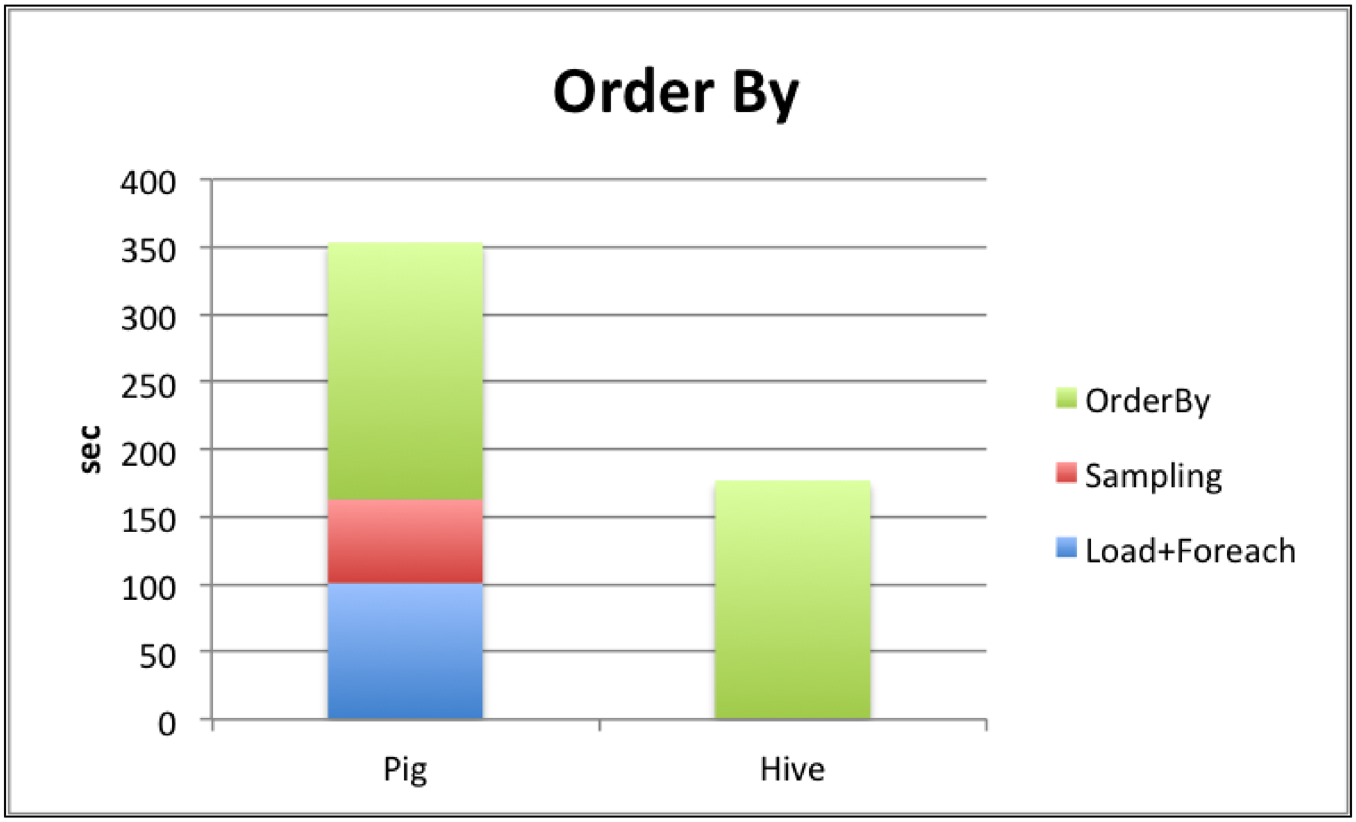
```
DROP TABLE lineitem;
DROP TABLE OrderBy;

-- create tables and load data
Create external table lineitem (L_ORDERKEY INT, L_PARTKEY INT, L_SUPPKEY INT, L_LINENUMBER INT, L_QUANTITY
DOUBLE, L_EXTENDEDPRICE DOUBLE, L_DISCOUNT DOUBLE, L_TAX DOUBLE, L_RETURNFLAG STRING, L_LINESTATUS STRING,
L_SHIPDATE STRING, L_COMMITDATE STRING, L_RECEIPTDATE STRING, L_SHIPINSTRUCT STRING, L_SHIPMODE STRING,
L_COMMENT STRING) ROW FORMAT DELIMITED FIELDS TERMINATED BY '|' STORED AS TEXTFILE LOCATION '/tpch/1G/lineitem';

CREATE TABLE OrderBy (L_ORDERKEY INT, L_PARTKEY INT, L_SUPPKEY INT, L_LINENUMBER INT, L_QUANTITY DOUBLE,
L_EXTENDEDPRICE DOUBLE, L_DISCOUNT DOUBLE, L_TAX DOUBLE, L_RETURNFLAG STRING, L_LINESTATUS STRING, L_SHIPDATE
STRING, L_COMMITDATE STRING, L_RECEIPTDATE STRING, L_SHIPINSTRUCT STRING, L_SHIPMODE STRING, L_COMMENT STRING)
ROW FORMAT DELIMITED FIELDS TERMINATED BY '|' STORED AS TEXTFILE;

-- the query
explain INSERT OVERWRITE TABLE OrderBy
SELECT
  *
FROM
  lineitem
Order By L_ORDERKEY;
```

Result



Proposal

As shown in the result, Pig uses three jobs for the order-by. The first job is the pipeline before the orderby, which is a LOAD and FOREACH in this query. The second job is the sampling. As mentioned in the jira, the sampling job should be dropped if #reducer has been set or estimated to one. Also, the previous pipeline should be merged into both the sampling job and the order-by job instead of being forced to finish in advance.

But note that **Hive doesn't support total ordering sort**. Therefore, Hive always uses one reducer for the Order-By, thus it doesn't have the issues Pig faces.

Join Followed by Group on Same Keys

Description

Join followed by Group-By on the same keys is a common use case in TPC-H (at least four queries). Currently Pig compiles them into two separate queries and is unaware of the common keys between these two operators. Therefore, even though the data has been partitioned by the Join, it will be re-partitioned by the Group-By. Usually the output of Join involves huge amount of data, so the performance of current implementation is undesirable.

A workaround is to rewrite these two operators using CoGroup, which only uses one job.

Queries

Pig

```
Orders = LOAD '$input/orders' USING PigStorage('|') AS (o_orderkey:int, o_custkey, o_orderstatus, o_totalprice,
o_orderdate, o_orderpriority, o_clerk, o_shippriority, o_comment);

LineItem = LOAD '$input/lineitem' USING PigStorage('|') AS (l_orderkey:int, l_partkey, l_suppkey, l_linenumber,
l_quantity, l_extendedprice, l_discount, l_tax:double, l_returnflag, l_linestatus, l_shipdate, l_commitdate,
l_receiptdate, l_shipinstruct, l_shipmode, l_comment);

FOrders = FOREACH Orders GENERATE o_orderkey;
FLineItem = FOREACH LineItem GENERATE l_orderkey, l_tax;

OrderLine = JOIN FOrders BY o_orderkey, FLineItem BY l_orderkey;
GroupOrderLine = GROUP OrderLine BY o_orderkey;

Result = FOREACH GroupOrderLine GENERATE group, SUM(OrderLine.l_tax) as sum_tax;

STORE Result into '$output/JoinGroup';
```

Pig CoGroup

```
Orders = LOAD '$input/orders' USING PigStorage('|') AS (o_orderkey:int, o_custkey, o_orderstatus, o_totalprice,
o_orderdate, o_orderpriority, o_clerk, o_shippriority, o_comment);

LineItem = LOAD '$input/lineitem' USING PigStorage('|') AS (l_orderkey:int, l_partkey, l_suppkey, l_linenumber,
l_quantity, l_extendedprice, l_discount, l_tax:double, l_returnflag, l_linestatus, l_shipdate, l_commitdate,
l_receiptdate, l_shipinstruct, l_shipmode, l_comment);

FOrders = FOREACH Orders GENERATE o_orderkey;
FLineItem = FOREACH LineItem GENERATE l_orderkey, l_tax;

CG = COGROUP FOrders BY o_orderkey, FLineItem BY l_orderkey;

Result = FOREACH CG GENERATE group, SUM(OrderLine.l_tax) as sum_tax;

STORE Result into '$output/CoGroup';
```

Hive

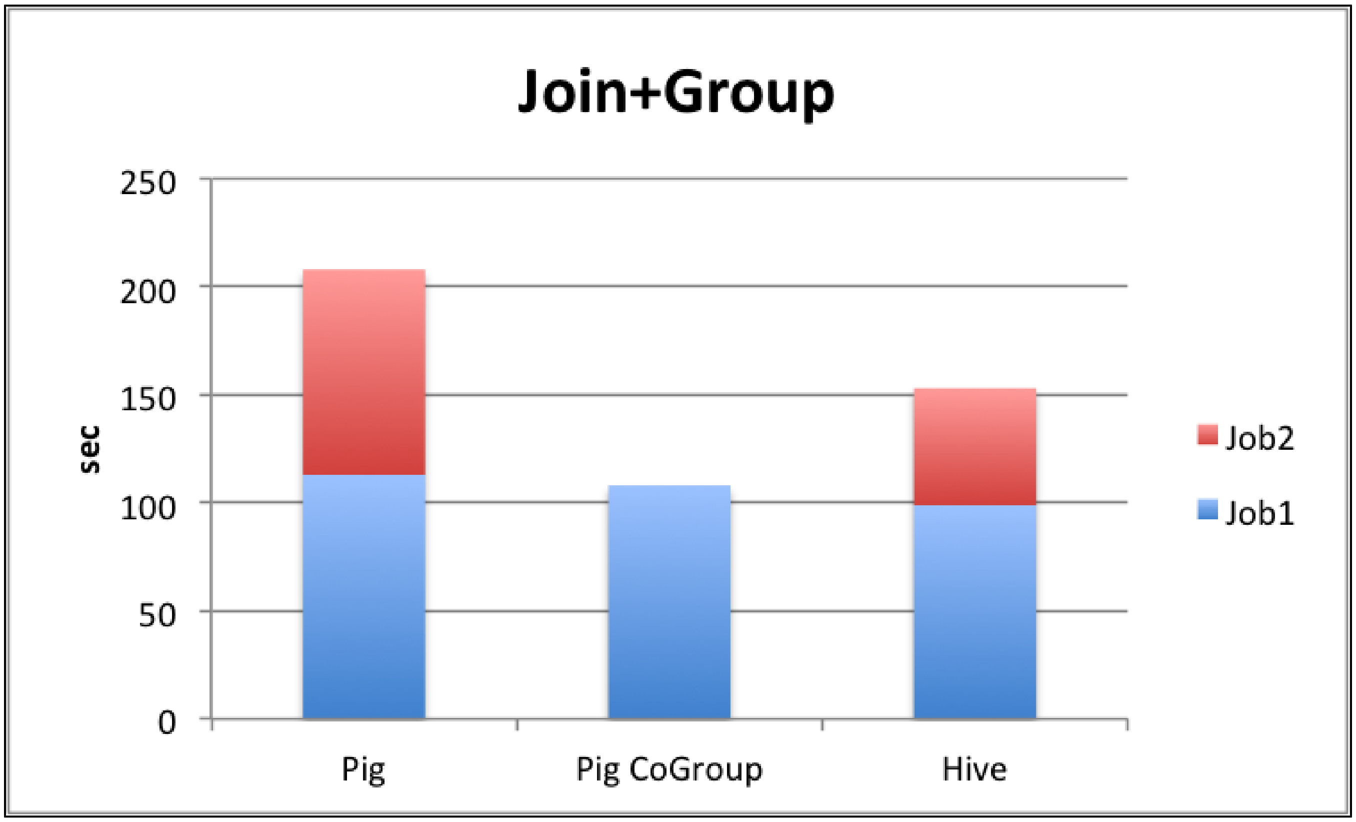
```
DROP TABLE orders;
DROP TABLE lineitem;
DROP TABLE JoinGroup;

-- create tables and load data
create external table orders (O_ORDERKEY INT, O_CUSTKEY INT, O_ORDERSTATUS STRING, O_TOTALPRICE DOUBLE,
O_ORDERDATE STRING, O_ORDERPRIORITY STRING, O_CLERK STRING, O_SHIPPRIORITY INT, O_COMMENT STRING) ROW FORMAT
DELIMITED FIELDS TERMINATED BY '|' STORED AS TEXTFILE LOCATION '/tpch/1G/orders';
Create external table lineitem (L_ORDERKEY INT, L_PARTKEY INT, L_SUPPKEY INT, L_LINENUMBER INT, L_QUANTITY
DOUBLE, L_EXTENDEDPRICE DOUBLE, L_DISCOUNT DOUBLE, L_TAX DOUBLE, L_RETURNFLAG STRING, L_LINESTATUS STRING,
L_SHIPDATE STRING, L_COMMITDATE STRING, L_RECEIPTDATE STRING, L_SHIPINSTRUCT STRING, L_SHIPMODE STRING,
L_COMMENT STRING) ROW FORMAT DELIMITED FIELDS TERMINATED BY '|' STORED AS TEXTFILE LOCATION '/tpch/1G/lineitem';

-- create the target table
CREATE TABLE JoinGroup (O_ORDERKEY INT, SUM_TAX DOUBLE);

-- the query
INSERT OVERWRITE TABLE JoinGroup
select O_ORDERKEY, SUM(lineitem.l_tax) as sum_tax
from
  orders join lineitem
  on orders.O_ORDERKEY == lineitem.L_ORDERKEY
group by O_ORDERKEY;
```

Result



Proposal

If Join keys are same (or more generally, prefix) as Group-By keys, then the Group-By can be implemented as a map-only job, which can be merged into the reduce phase of the Join job. Therefore, only one job is required.

Nested Column Pruning

Description

Currently Pig doesn't go into a bag field to see if any sub-field of the bag can be pruned, which makes huge amount of data being transferred across jobs unnecessarily. This is a known issue PIG-1324.

Users could do pruning explicitly but it could complicate the pig scripts and make Pig look stupid🙂

Queries

Pig

```
LineItems = LOAD '$input/lineitem' USING PigStorage('|') AS (orderkey, partkey, suppkey, linenumber, quantity, 
extendedprice, discount, tax, returnflag, linestatus, shipdate, commitdate, receiptdate, shipinstruct, 
shipmode, comment);

Orders = LOAD '$input/orders' USING PigStorage('|') AS (o_orderkey, o_custkey, o_orderstatus, o_totalprice, 
o_orderdate, o_orderpriority, o_clerk, o_shippriority, o_comment);

OrdersLineItem = JOIN Orders BY o_orderkey, LineItems BY orderkey;

LGroup = GROUP OrdersLineItem by o_custkey;

LSum = FOREACH LGroup GENERATE group, SUM(OrdersLineItem.tax);

STORE LSum INTO '$output/PruneGroup_out';
```

Pig with pruning

```
LineItems = LOAD '$input/lineitem' USING PigStorage('|') AS (orderkey, partkey, suppkey, linenumber, quantity, 
extendedprice, discount, tax, returnflag, linestatus, shipdate, commitdate, receiptdate, shipinstruct, 
shipmode, comment);

Orders = LOAD '$input/orders' USING PigStorage('|') AS (o_orderkey, o_custkey, o_orderstatus, o_totalprice, 
o_orderdate, o_orderpriority, o_clerk, o_shippriority, o_comment);

fLineItems = FOREACH LineItems GENERATE orderkey, tax;
fOrders = FOREACH Orders GENERATE o_orderkey, o_custkey;

OrdersLineItem = JOIN fOrders BY o_orderkey, fLineItems BY orderkey;

LGroup = GROUP OrdersLineItem by o_custkey;

LSum = FOREACH LGroup GENERATE group, SUM(OrdersLineItem.tax);

STORE LSum INTO '$output/PruneGroup_out';
```

Hive

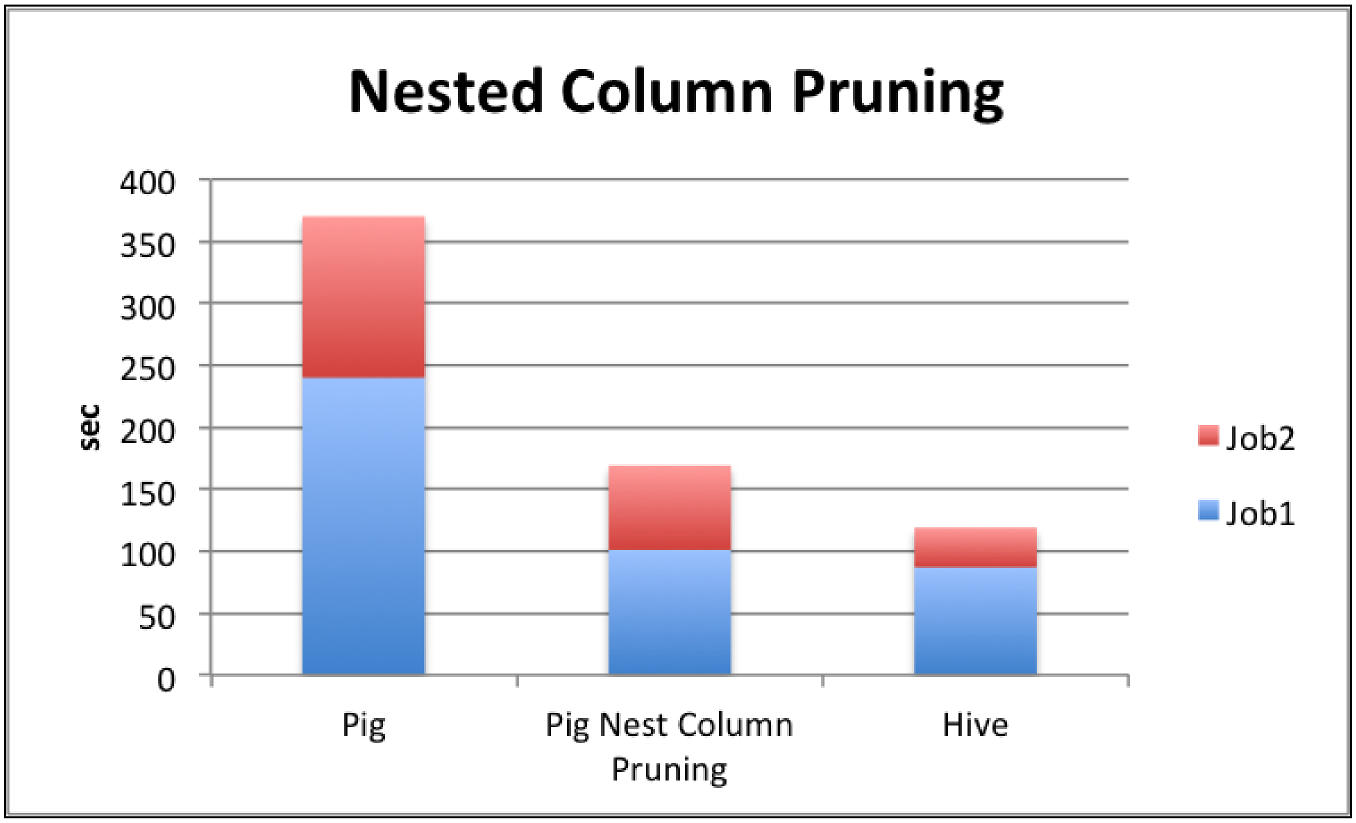
```
DROP TABLE lineitem;
DROP TABLE orders;
DROP TABLE prune_group;

-- create the tables and load the data
Create external table lineitem (L_ORDERKEY INT, L_PARTKEY INT, L_SUPPKEY INT, L_LINENUMBER INT, L_QUANTITY
DOUBLE, L_EXTENDEDPRICE DOUBLE, L_DISCOUNT DOUBLE, L_TAX DOUBLE, L_RETURNFLAG STRING, L_LINESTATUS STRING,
L_SHIPDATE STRING, L_COMMITDATE STRING, L_RECEIPTDATE STRING, L_SHIPINSTRUCT STRING, L_SHIPMODE STRING,
L_COMMENT STRING) ROW FORMAT DELIMITED FIELDS TERMINATED BY '|' STORED AS TEXTFILE LOCATION '/tpch/1G/lineitem';
create external table orders (O_ORDERKEY INT, O_CUSTKEY INT, O_ORDERSTATUS STRING, O_TOTALPRICE DOUBLE,
O_ORDERDATE STRING, O_ORDERPRIORITY STRING, O_CLERK STRING, O_SHIPPRIORITY INT, O_COMMENT STRING) ROW FORMAT
DELIMITED FIELDS TERMINATED BY '|' STORED AS TEXTFILE LOCATION '/tpch/1G/orders';

-- create the result table
create table prune_group (O_CUSTKEY INT, SUM_TAX DOUBLE);

-- the query
insert overwrite table prune_group
select
  O_CUSTKEY, sum(L_TAX) as sum_tax
from orders join lineitem
on orders. O_ORDERKEY == lineitem. L_ORDERKEY
group by O_CUSTKEY;
```

Result



Proposal

Currently Pig inserts a Foreach to drop unnecessary columns **after** each operator if possible. For Group we want to drop columns **in advance** so they won't hit the disk during the Group job. Also, we need to track used nested columns so we can drop them if possible.

Lazy Type Conversion

Description

As observed in PIG-410, right now any time user declares types for loaded data, we insert a generate after the load to produce data of the right type. It would be more efficient to daley conversion to the point where each individual field is used.

Queries

Pig

```
LineItems = LOAD '$input/lineitem' USING PigStorage('|') AS (orderkey:int, partkey:int, suppkey:int, linenumber:
int, quantity:double, extendedprice:double, discount:double, tax:double, returnflag:chararray, linestatus:
chararray, shipdate:chararray, commitdate:chararray, receiptdate:chararray, shipinstruct:chararray, shipmode:
chararray, comment:chararray);
SubLineItems = FILTER LineItems BY shipdate == '2012-09-02';
STORE SubLineItems INTO '$output/out';
```

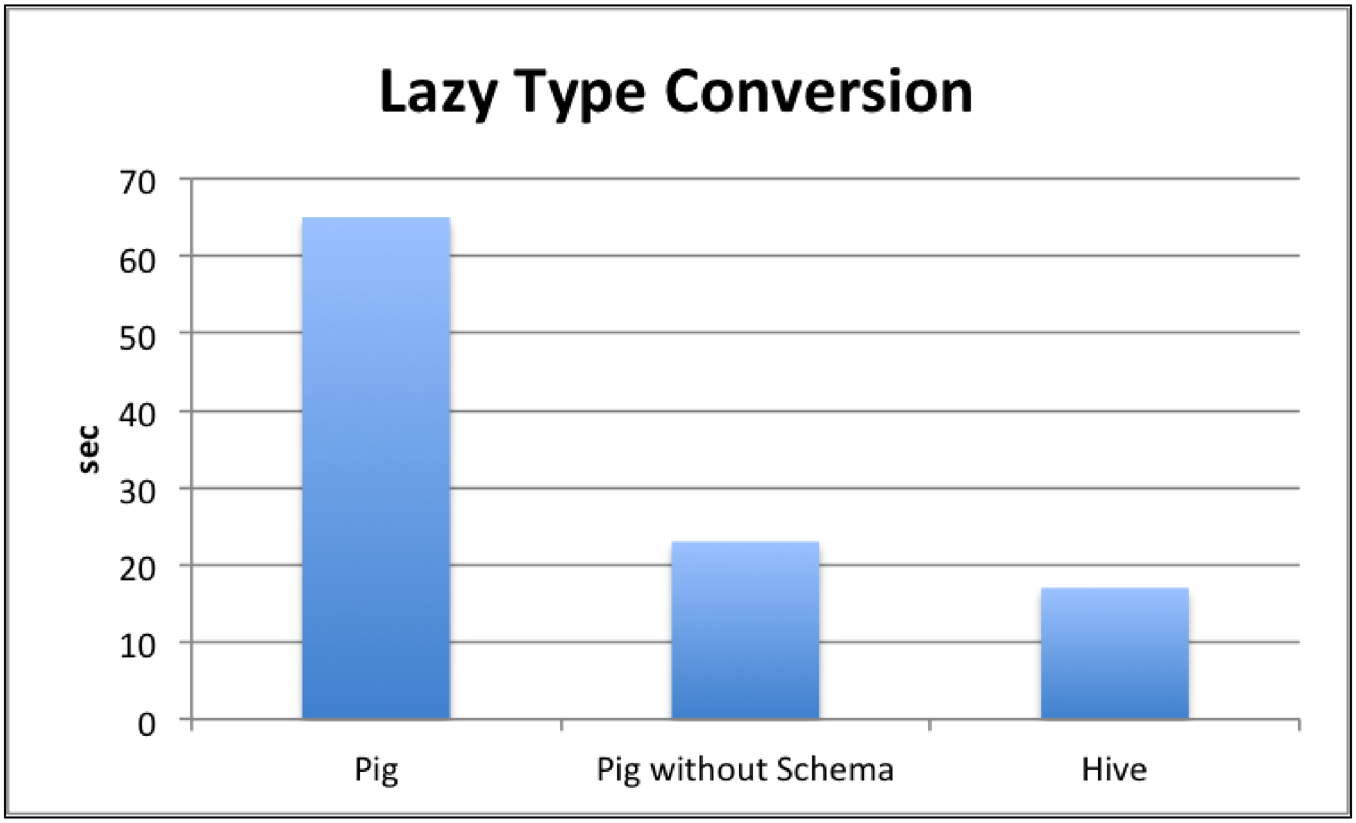
Pig without Schema

```
LineItems = LOAD '$input/lineitem' USING PigStorage('|') AS (orderkey, partkey, suppkey, linenumber, quantity,
extendedprice, discount, tax, returnflag, linestatus, shipdate, commitdate, receiptdate, shipinstruct,
shipmode, comment);
SubLineItems = FILTER LineItems BY shipdate == '2012-09-02';
STORE SubLineItems INTO '$output/out';
```

Hive

```
DROP TABLE lineitem;
DROP TABLE q1_pricing_summary_report;
-- create tables and load data
Create external table lineitem (L_ORDERKEY INT, L_PARTKEY INT, L_SUPPKEY INT, L_LINENUMBER INT, L_QUANTITY
DOUBLE, L_EXTENDEDPRICE DOUBLE, L_DISCOUNT DOUBLE, L_TAX DOUBLE, L_RETURNFLAG STRING, L_LINESTATUS STRING,
L_SHIPDATE STRING, L_COMMITDATE STRING, L_RECEIPTDATE STRING, L_SHIPINSTRUCT STRING, L_SHIPMODE STRING,
L_COMMENT STRING) ROW FORMAT DELIMITED FIELDS TERMINATED BY '|' STORED AS TEXTFILE LOCATION '/tpch/1G/lineitem';
-- create the target table
CREATE TABLE q1_pricing_summary_report (L_ORDERKEY INT, L_PARTKEY INT, L_SUPPKEY INT, L_LINENUMBER INT,
L_QUANTITY DOUBLE, L_EXTENDEDPRICE DOUBLE, L_DISCOUNT DOUBLE, L_TAX DOUBLE, L_RETURNFLAG STRING, L_LINESTATUS
STRING, L_SHIPDATE STRING, L_COMMITDATE STRING, L_RECEIPTDATE STRING, L_SHIPINSTRUCT STRING, L_SHIPMODE STRING,
L_COMMENT STRING) ROW FORMAT DELIMITED FIELDS TERMINATED BY '|' STORED AS TEXTFILE;
-- the query
INSERT OVERWRITE TABLE q1_pricing_summary_report
SELECT
  *
FROM
  lineitem
WHERE
  L_SHIPDATE=='2012-09-02';
```

Result



Note that in this query, all the records were filtered out so there was no output. In other words, every record was fully deserialized and then thrown away. By the way Pig has an optimization rule ("PushUpFilter") to avoid deserializing for those tuples being thrown away later, but it was somehow not applied here.

Proposal

Some fundamental redesign of the type system may be needed, and we may learn from Hive folks as they put a lot of efforts in this regard.

A naive proposal would be to use a rule similar to the column pruning to prune all the unnecessary cast operations. A cast operation can be pruned if the type of the column never matters. For example, in the pig queries above, only the shipdate need to be cast to chararray.

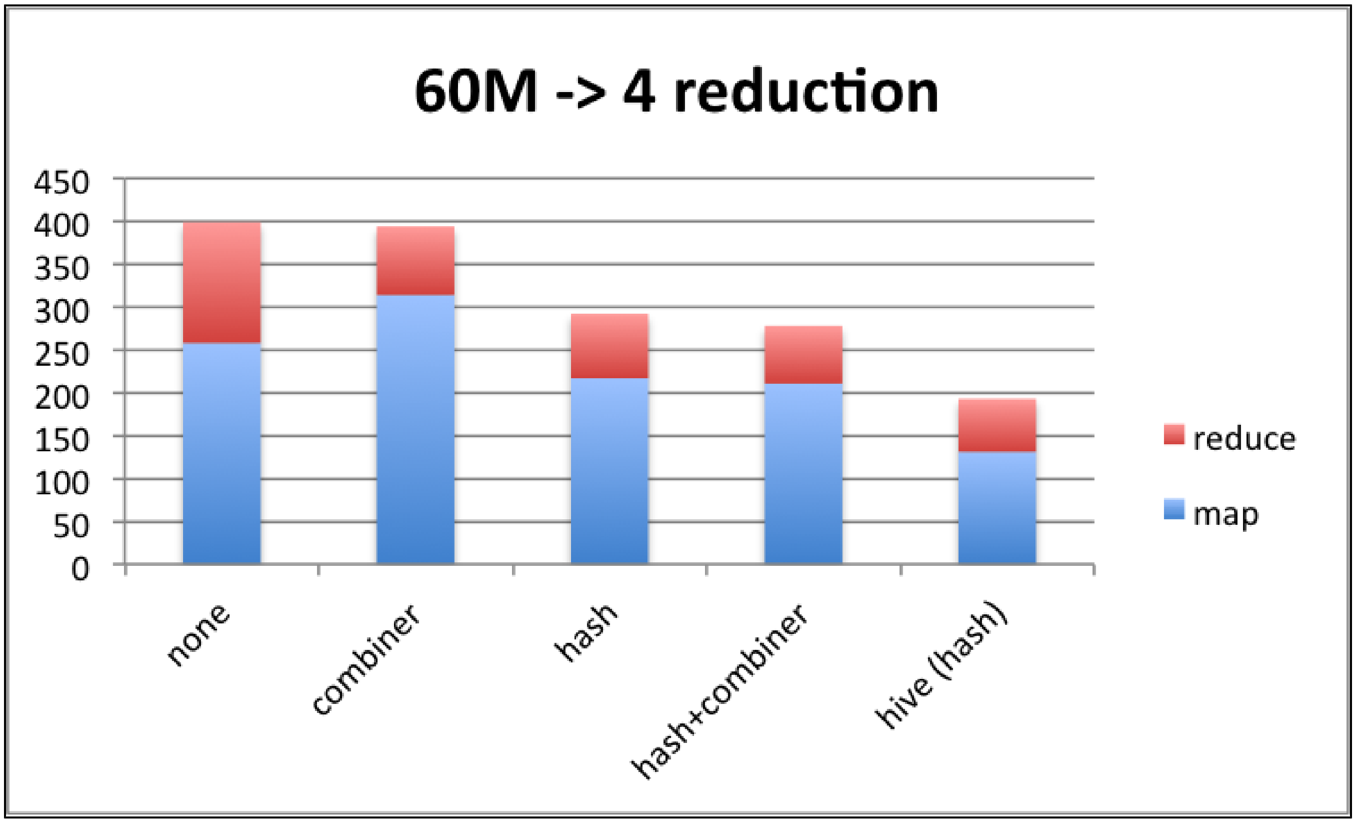
It's also common that some columns are used in subsequent jobs so they don't need to be cast immediately after being loaded, otherwise they need to be cast back and forth between each pair of successive jobs. In such case, we need to move the original Cast to the job in which it's actually needed.

Here are some previous thoughts on this regard http://wiki.apache.org/pig/AvoidingSedes.

HashAgg vs. Combiner

HashAgg (Hash Aggregation, aka in-map combiner) is a new feature in Pig 0.10 that will perform aggregation within map function. The main advantage against combiner is it avoids de/serializing and sorting the data. More benefits can be found in this proposal. Currently it's disabled by default, where in Hive it's turned on by default.

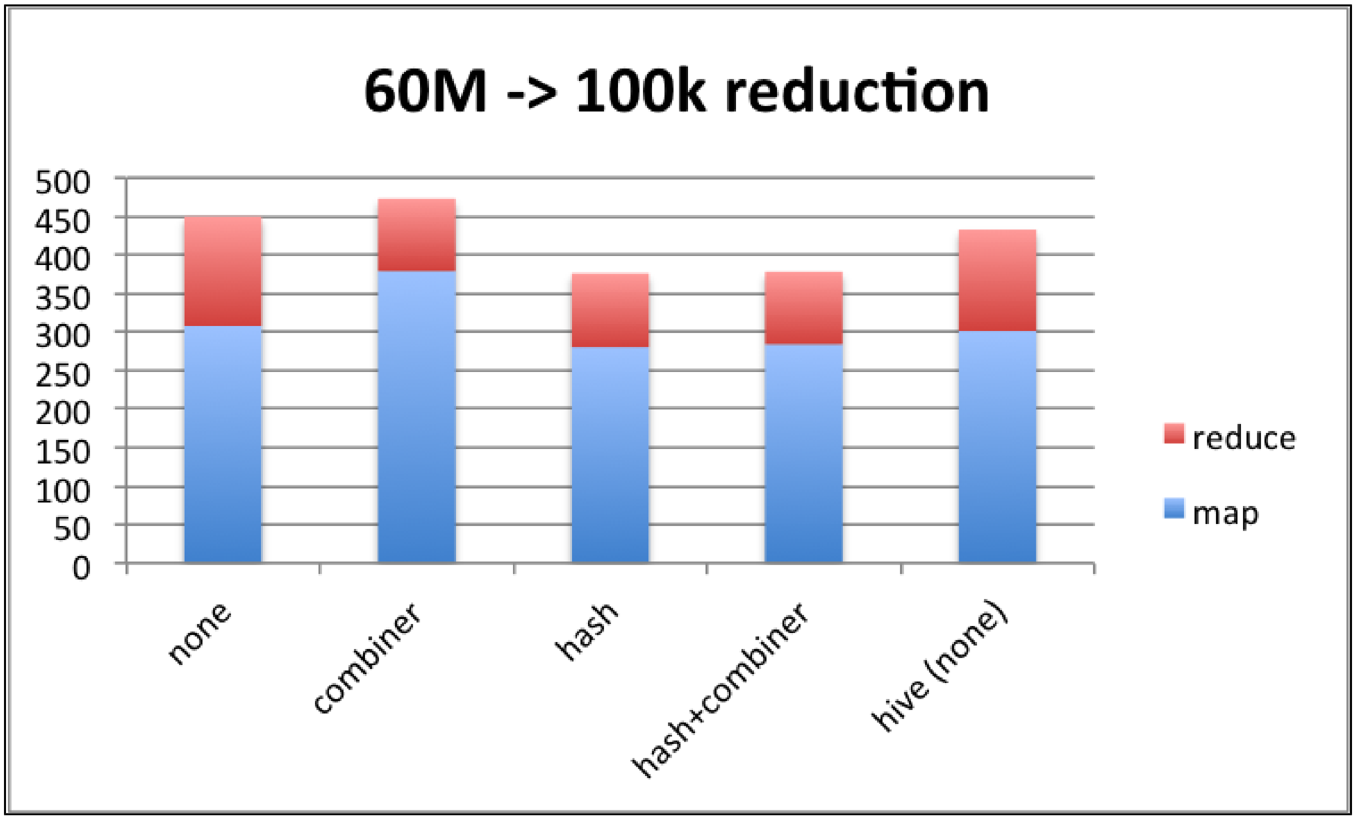
Below is a simple benchmark to study whether we should use HashAgg more aggresively. We perform a group-by aggregation on different keys and with combiner/HashAgg turned on/off respectively. In order to reduce the cache effect, we increase the dataset size to 10GB, which has 60 million records. The number of groups depends on the keys.



In this case, there are only four groups so the combiner should be very effective. However, we can see the overhead of the combiner to the map phase is significant and offsets the benefit from the reduce phase. HashAgg, however, not only reduces the reduce phase time, but also reduces the map phase time, as it reduces the data to be sorted.

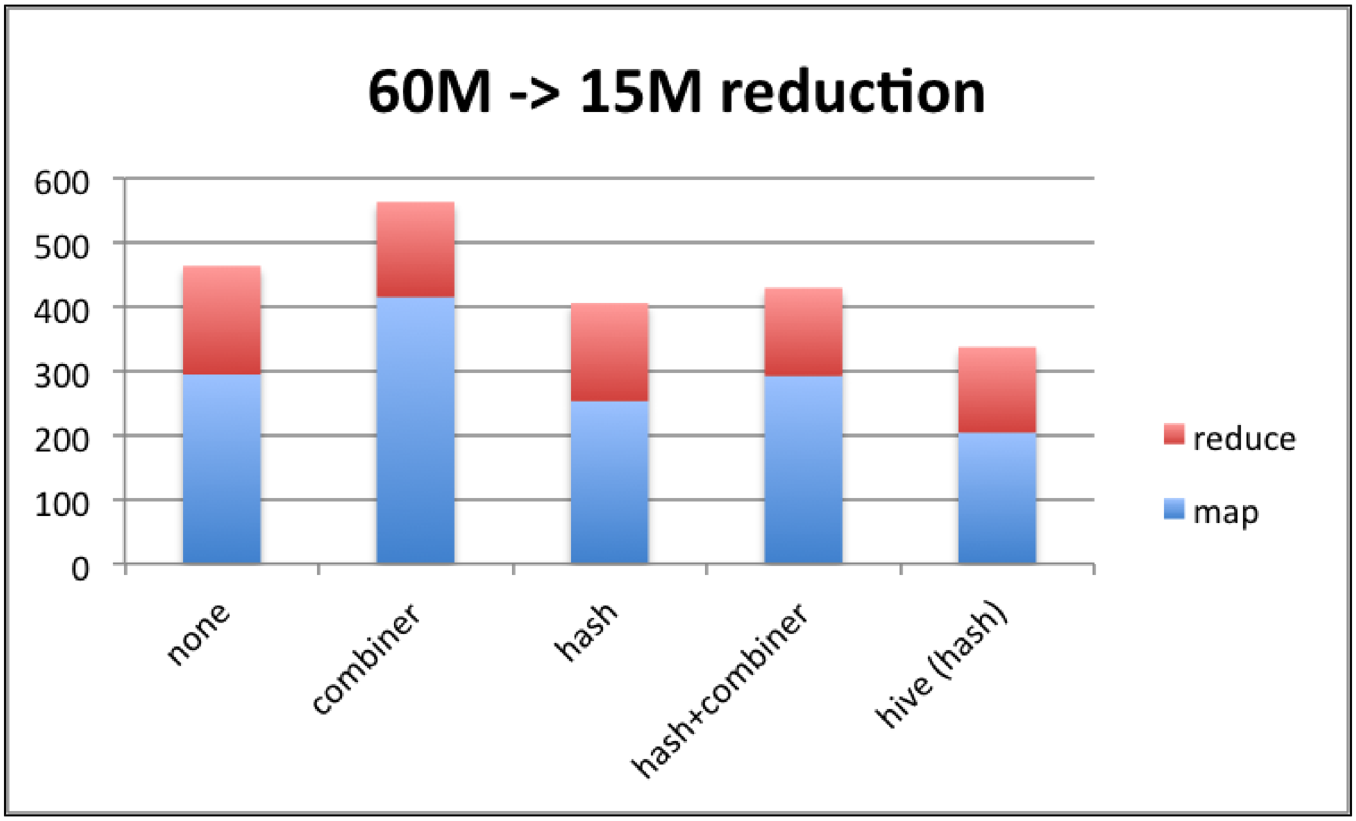
Note the current implementation of HashAgg depends on the combiner, i.e. **we can't use HashAgg only without the combiner**, so we did a trick in the source code to perform HashAgg only. In this case, as HashAgg reduces most of the data, the following combiner doesn't help much.

TODO: investigate the gap between HashAgg and Hive.

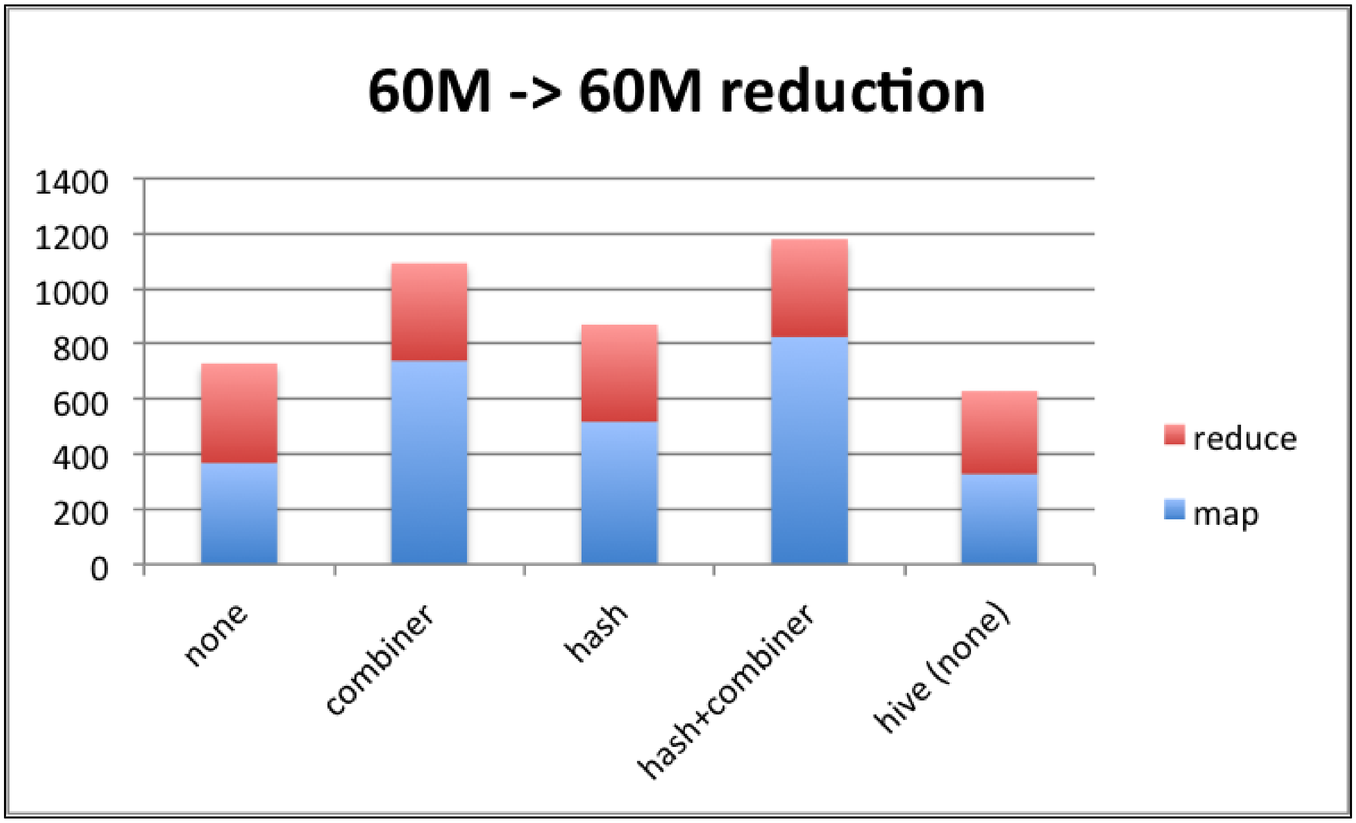


In this case the combiner will slow down the performance even there is 600x reduction. Note both Pig and Hive would auto-disable HashAgg because of "low" reduction rate, and for Pig we set the pig.exec.mapPartAgg.minReduction to 1 to avoid auto-disabling.

TODO: refine the criteria to auto-disable HashAgg.



With only 4x reduction, the advantage of HashAgg against combiner is more obvious.



If all records have different group-by key, then the combiner should be definitely turned off. We can observe that HashAgg's overhead is lower than the combiner's, and note that HashAgg can also auto-disable itself when observing such case, while the combiner can't.

So, it seems HashAgg is always a better choice than the combiner. One hypothetical case when combiner might be better is that, combiner can also happen in reduce's shuffle phase, while HashAgg currently can't.