PigInMapCombinerProposal

Proposal for hash based aggregation in map

Introduction

Pig does (sort based) partial aggregation in map side through the use of combiner. MR serializes the output of map to a buffer, sorts it on the keys, deserializes and passes the values grouped on the keys to combiner phase. The same work of combiner can be done in the map phase itself by using a hash-map on the keys. This hash based (partial) aggregation can be done with or without a combiner phase.

Benefits

It will send fewer records to combiner and thereby -

- Save on cost of serializing and de-serializing
- Save on cost of lock calls on the combiner input buffer. (I have found this to be a significant cost for a query that was doing multiple group-by's in a single MR job. -Thejas)
- For some types of queries it is useful to turn off combiner in reduce side. This is not possible with hadoop combiner. By doing this, the problem of running out of memory in reduce side, for queries like COUNT(distinct col) can be avoided. The OOM issue happens because very large records get created after the combiner run on merged reduce input, which does not fit into MR framework's buffers. Usually the size after partial aggregation on map side does not cause records to be too large to cause a problem.
- Pig does not use hadoop combiner when there is a non algebraic function on the grouped bag column or if that column is projected. This is because the data size reduction in those cases are usually not significant enough to justify the additional (de)serialization costs. But hash based aggregation can be used in such cases because there is no (de)serialization overhead.
- It is possible to turn off the in-map combine automatically if there is not enough 'combination' that is taking place to justify the overhead of the in-map combiner. (Idea borrowed from Hive jira.)
- If input data is sorted, it is possible to do efficient map side (partial) aggregation with in-map combiner.

Concerns

Memory used by the in-map combiner will have to be managed carefully, to avoid any out of memory errors.

Open questions

It is not clear, if both MR-combiner and in-map combiner should be enabled in a pig MR job. If in-map combiner is used, the data reduction that would happen because of MR-combiner might not be sufficient to justify the costs. But MR-combiner might help in further reducing the data that gets written to disk, if multiple spill files get merged or if multiple waves of sort-merge happen on reduce side.

Changes

There will be a new physical operator, POHashAgg, that will do the hash based aggregation. This will be the last node before the LORearrange in the map plan of MR job for the group operation. Its input will be same as the input to LORearrange in plan that uses MR combiner - a foreach statement that computes the key and UDF$Initial.exec() .

POHashAgg will have the plans used in combiner to generate intermediate values. For every new input record, POHashAgg will store the key and input values in a hashmap.
When there are two, or a small number of values for a group key (set), it will compute the new partial aggregate value and store back in the hash-map as intermediate result. The memory management of the hash-map will be similar to that of InternalCacheBag - it will estimate its memory footprint. It will flush some % (5%?) of entries and write them to output when it exceeds configurable memory limit. The least recently used keys can be chosen to be flushed (keeping track of few most recently used might be enough). These flushed entries will be written as map output.

It might be useful to have a new udf interface that accepts a tuple at a time to compute a partial aggregate, so that new bags don't have to be created for each new tuple that needs to be aggregated. But the bag creation overhead and overhead of calling the udf multiple times could be reduced by calling the udf only after few values have been accumulated in the hash-map.

Initial Implementation

In the initial implementation, combiner will be supported only when all projections are either expressions on the group column or expressions on algebraic UDFs. This is because column pruning does not currently discard unused columns within a grouped-bag, and in such cases there will not be data size reduction happening because of the use of in-map combiner.

For the query -

```
l = load 'x'   as (a,b,c);
g = group l by a;
f = foreach g generate  group, COUNT(l.b);
```

The existing plan -

```
Map Plan
g: Local Rearrange[tuple]{bytearray}(false) - scope-73
|   |
|   Project[bytearray][0] - scope-74
|
|---f: New For Each(false,false)[bag] - scope-61
    |   |
    |   Project[bytearray][0] - scope-62
    |   |
    |   POUserFunc(org.apache.pig.builtin.COUNT$Initial)[tuple] - scope-63
    |   |
    |   |---Project[bag][1] - scope-64
    |       |
    |       |---Project[bag][1] - scope-65
    |
    |---Pre Combiner Local Rearrange[tuple]{Unknown} - scope-75
        |
        |---l: New For Each(false,false,false)[bag] - scope-47
```

Will change to -

```
Map Plan
g: Local Rearrange[tuple]{bytearray}(false) - scope-73
|   |
|   Project[bytearray][0] - scope-74
|
|---f: HashAgg
    |   |
    |   Project[bytearray][0] - scope-62
    |   |
    |   POUserFunc(org.apache.pig.builtin.COUNT$Intermediate)[tuple] - scope-63
    |   |
    |   |---Project[bag][1] - scope-64
    |
    |---f: New For Each(false,false)[bag] - scope-61
        |   |
        |   Project[bytearray][0] - scope-62
        |   |
        |   POUserFunc(org.apache.pig.builtin.COUNT$Initial)[tuple] - scope-63
        |   |
        |   |---Project[bag][1] - scope-64
        |       |
        |       |---Project[bag][1] - scope-65
        |
        |---Pre Combiner Local Rearrange[tuple]{Unknown} - scope-75
            |
            |---l: New For Each(false,false,false)[bag] - scope-47
```

The MR combiner will also be supported and by default in-map combiner will not be used. There will be a property that will need to be set to enable it. There will be another property that will control use of MR combiner along with in-map combiner. After sufficient testing is done, we can change the default execution mode and properties.

Query Plan
The CombinerOptimizer will generate the in-map plan as well. Except for the introduction of a new operator, the plan will remain the same.

Plan execution
1. Evaluating aggregate results
For every new input record, POHashAgg will do the following -

1. compute the key using the index for the key. (the key is computed in the foreach statement with UDF$Initial calls that precedes the POHashAgg).
2. lookup the hashmap for entry corresponding to key
3. if the key is a key not seen in previous iteration (currentKey), or if the currentKey has many values (> 100 (configurable)), for each aggregate plan, create a bag with the existing value(s) and new value, and compute the aggregated value.

2. Memory Management
Similar to the current strategy of InternalCachedBag, find the average size of the entries and estimate the size held by current number of entries. If the size exceeds the internal cached bag size threshold, it will write a portion of the hashmap entries to output.
In case of multi-query, there will be multiple such bags, and the memory limit will be shared equally between them.