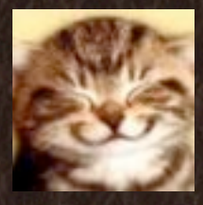

# Hive Source Code Reading

*Hadoop Source Code Reading Vol.9(5/30)*

*http://hadoop-scr9th.eventbrite.com/*

*@wyukawa*



# Overview

- Motivation
- What component did I read?
- Set up Development Environment
- Useful document for Code Reading
- Let's try!
- My understanding
- Where does Hive tune performance?
- My impression

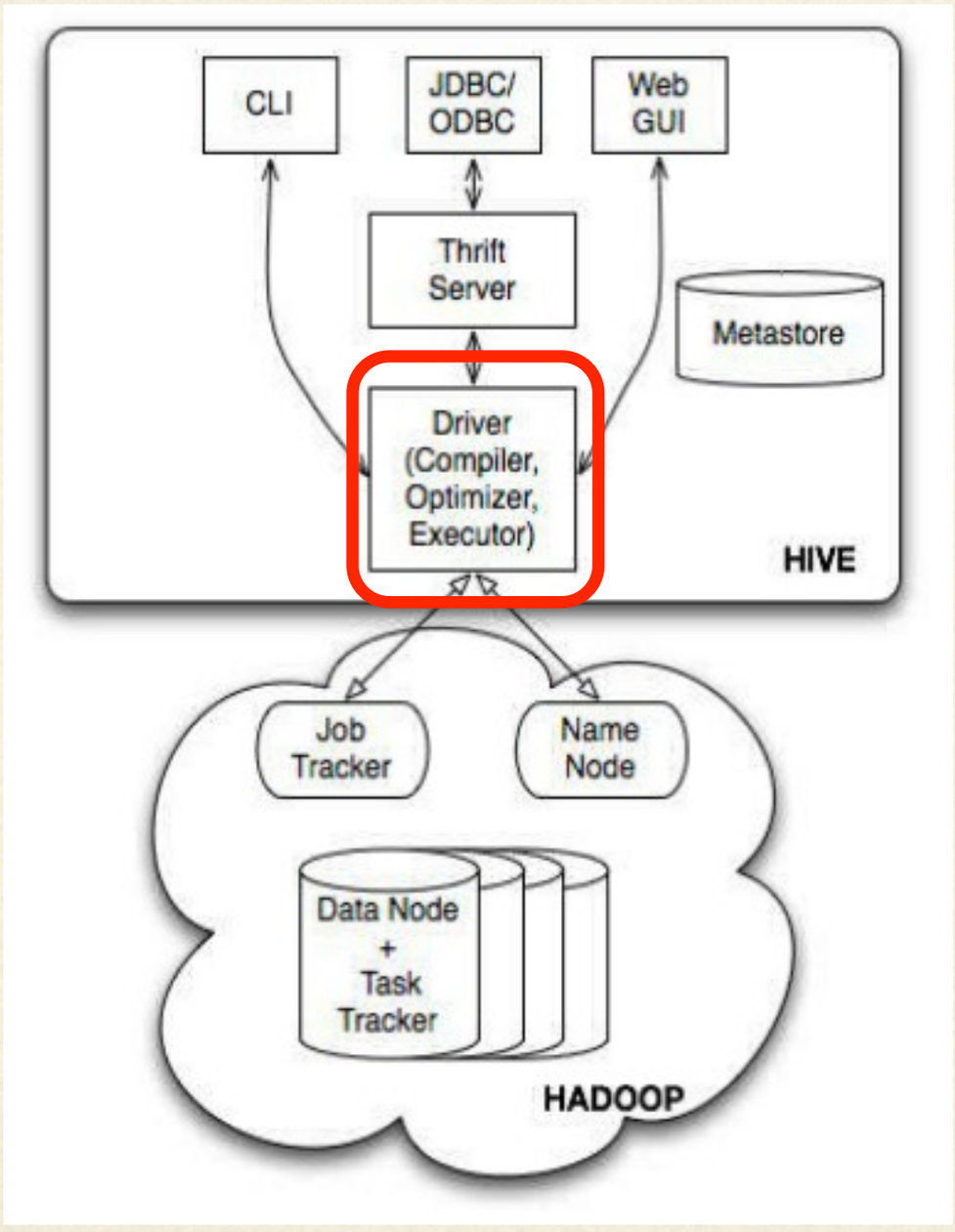
# Motivation

- Curiosity
  - How does Hive convert HiveQL to MapReduce jobs?
  - What optimizing processes are adopted?

select item, sum(value) from item_tbl group by item;

map reduce

# What component did I read?




http://www.slideshare.net/ragho/hive-user-meeting-august-2009-facebook

# Set up Development Environment

See http://glowing-moon-7493.herokuapp.com/questions/171/hive

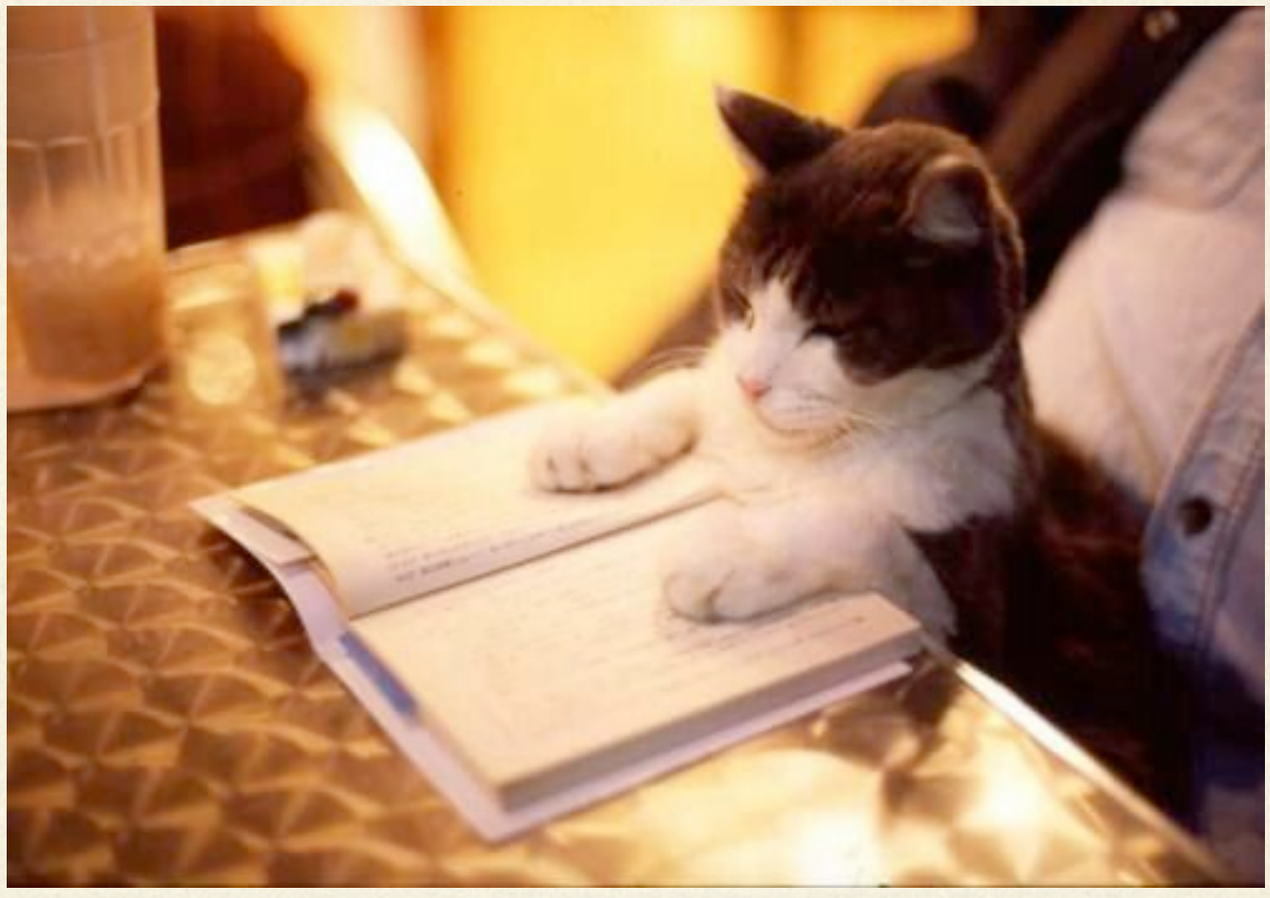
Setting up is troublesome...(-_-;)

I read r1342841.

# Useful document

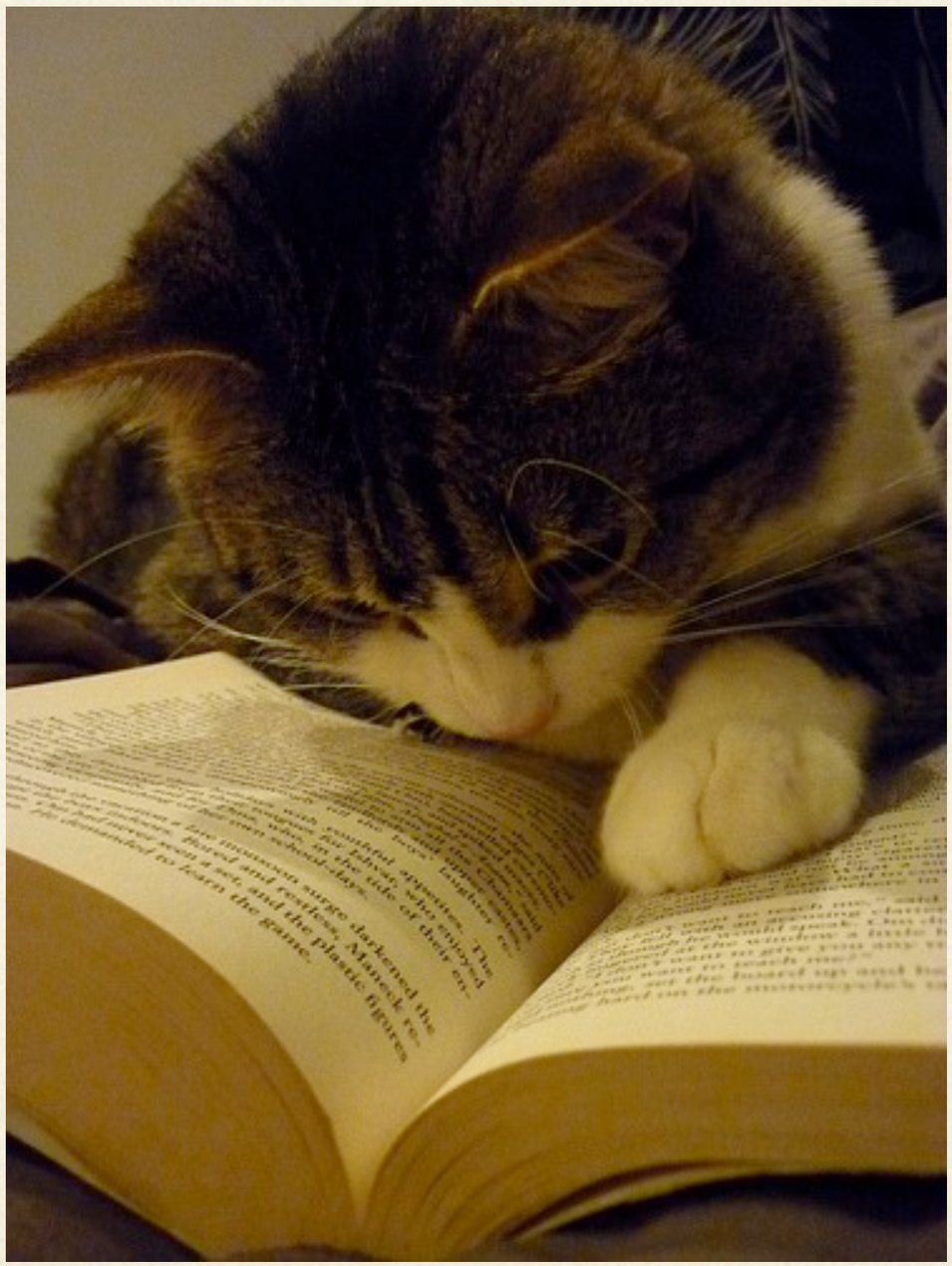
- Hive Anatomy
  - http://www.slideshare.net/nzhang/hive-anatomy
- Internal Hive
  - http://www.slideshare.net/recruitcojp/internal-hive

# Let's try!



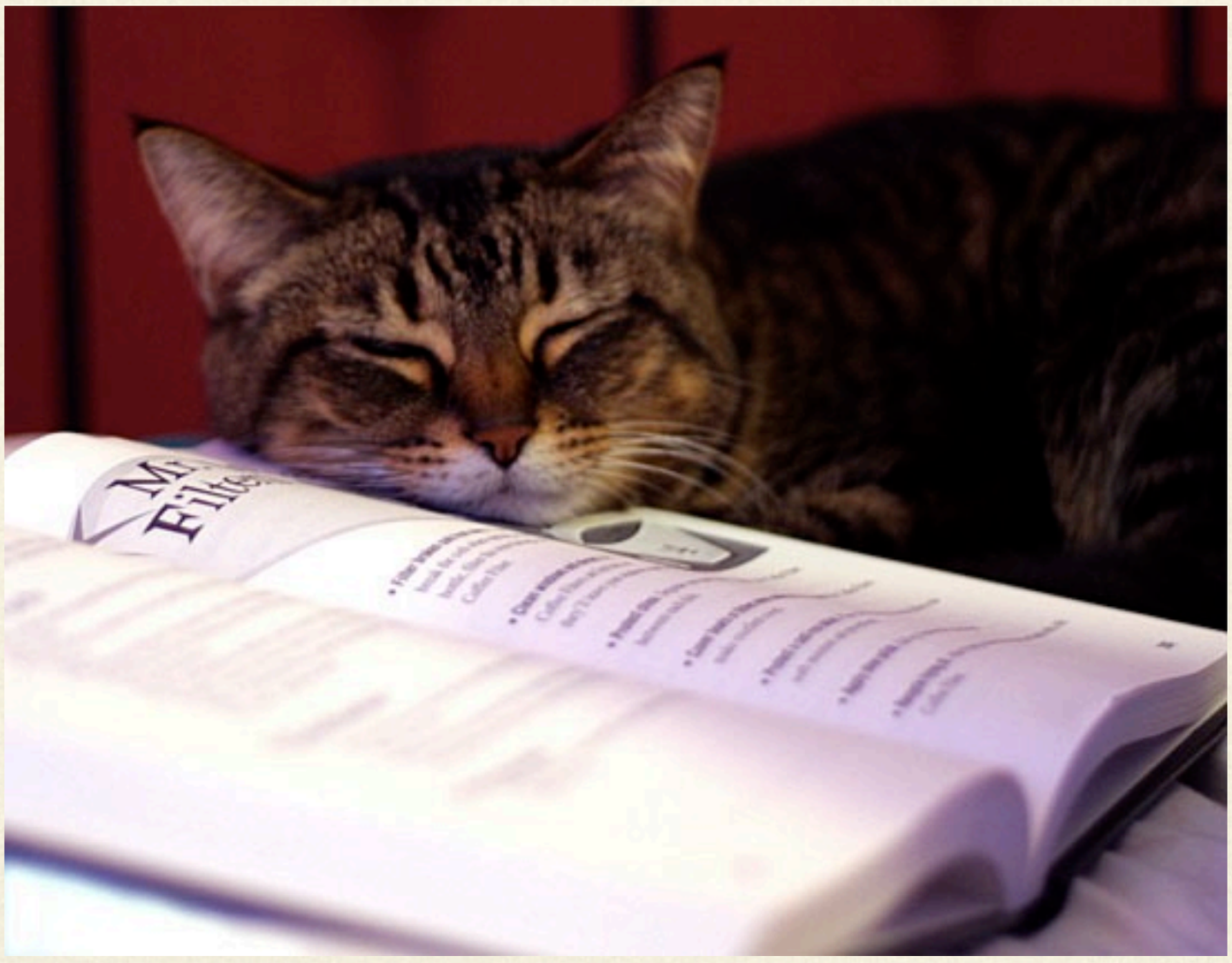
http://matome.naver.jp/odai/2133708130778775801/2133714001783351003

# a few minutes later



http://matome.naver.jp/odai/2133708130778775801/2133714001583346003

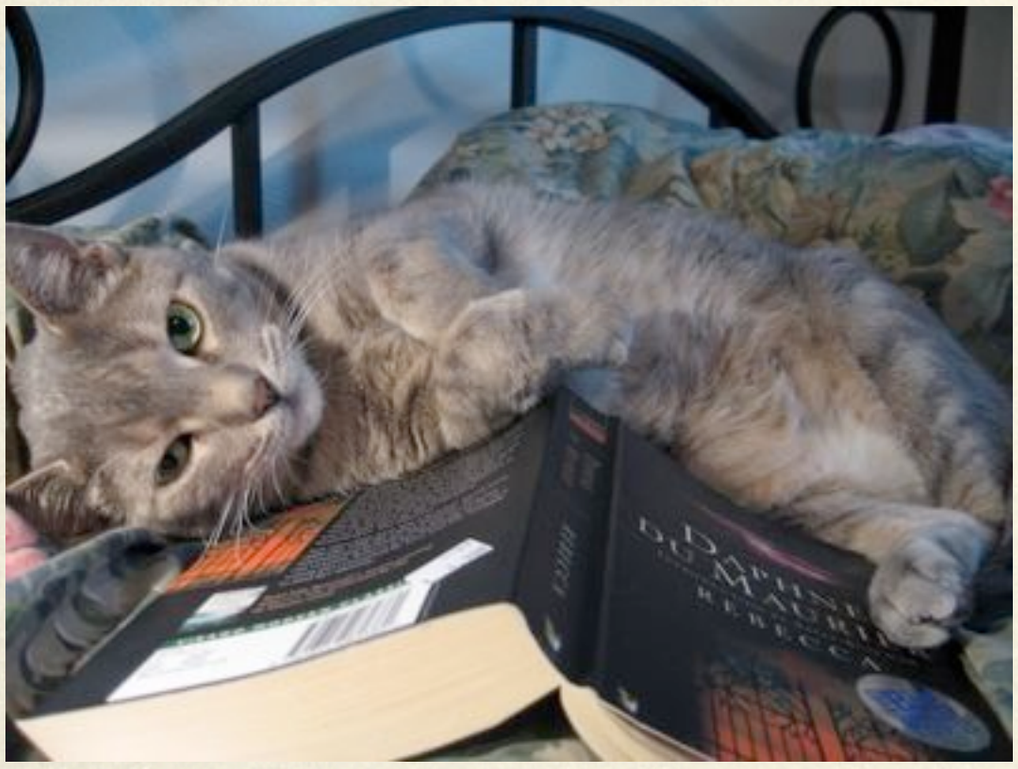
# a few hours later



http://matome.naver.jp/odai/2133708130778775801/2133714001683348003

# It's difficult...

- Complex
- Method length is long



```
} else {

  if (topOps.size() == 1) {
    TableScanOperator ts = (TableScanOperator) topOps.values().toA

    // check if the pruner only contains partition columns
    if (PartitionPruner.onlyContainsPartnCols(topToTable.get(ts),
        opToPartPruner.get(ts))) {

      PrunedPartitionList partsList = null;
      try {
        partsList = opToPartList.get(ts);
        if (partsList == null) {
          partsList = PartitionPruner.prune(topToTable.get(ts),
              opToPartPruner.get(ts), conf, (String) topOps.keySet
              .toArray()[0], prunedPartitions);
          opToPartList.put(ts, partsList);
        }
      } catch (HiveException e) {
        // Has to use full name to make sure it does not conflict
        // org.apache.commons.lang.StringUtils
        LOG.error(org.apache.hadoop.util.StringUtils.stringifyExce
        throw new SemanticException(e.getMessage(), e);
      }

      // If there is any unknown partition, create a map-reduce jo
      // the filter to prune correctly
      if ((partsList.getUnknownPartns().size() == 0)) {
        List<String> listP = new ArrayList<String>();
        List<PartitionDesc> partP = new ArrayList<PartitionDesc>()

        Set<Partition> parts = partsList.getConfirmedPartns();
```

http://matome.naver.jp/odai/2133708130778775801/2133714001683349103

# And there is this ticket...

http://matome.naver.jp/odai/2133708130778775801/2133714001583345803

Hive / HIVE-3027

# The optimizer architecture of Hive is terrible, need code refactoring

Log In

## Details

| | | | |
|---|---|---|---|
| Type: | Improvement | Status: | |
| Priority: | Major | Resolution: | |
| Affects Version/s: | 0.4.0, 0.4.1, 0.5.0, 0.6.0, 0.7.0, 0.7.1, 0.8.0, 0.8.1 | Fix Version/s: | |
| Component/s: | Query Processor | | |
| Labels: | architecture optimizer ysmart | | |

## Description

Now I want to add a complete cost-based optimization for hive. but when I begin the work, I found it very difficult to do using current hive optimiz
awful design and makes me mad. For example, the map-side optimization, it scans the whole operators' DAG and try to find the operators that c
expands to 1000 lines, and only implements the map-side optimizations!!!
In my opinion, optimization shouldn't be done in a separated step, differnt optimization should be done in appropriate time. For example, join reor
Operator for each join according to the cost estimation. And, in the process, we can do join and aggreagation merge, and, we shoud push down p
corresponding predicate of each base table for estimating JOIN cost. How concise and graceful the code will be if we do the optimization this wa
amazing redundancy, and, the code is very very difficult to debug!!!! Now there is a patch of cost-based JOIN reorder and merge optimizer callec
The optimizer architecture of Hive is terrible, How can I do now?

## Issue Links

is related to

- HIVE-33 [Hive]: Add ability to compute statistics on hive tables
- HIVE-1938 Cost Based Query optimization for Joins in Hive

## Activity

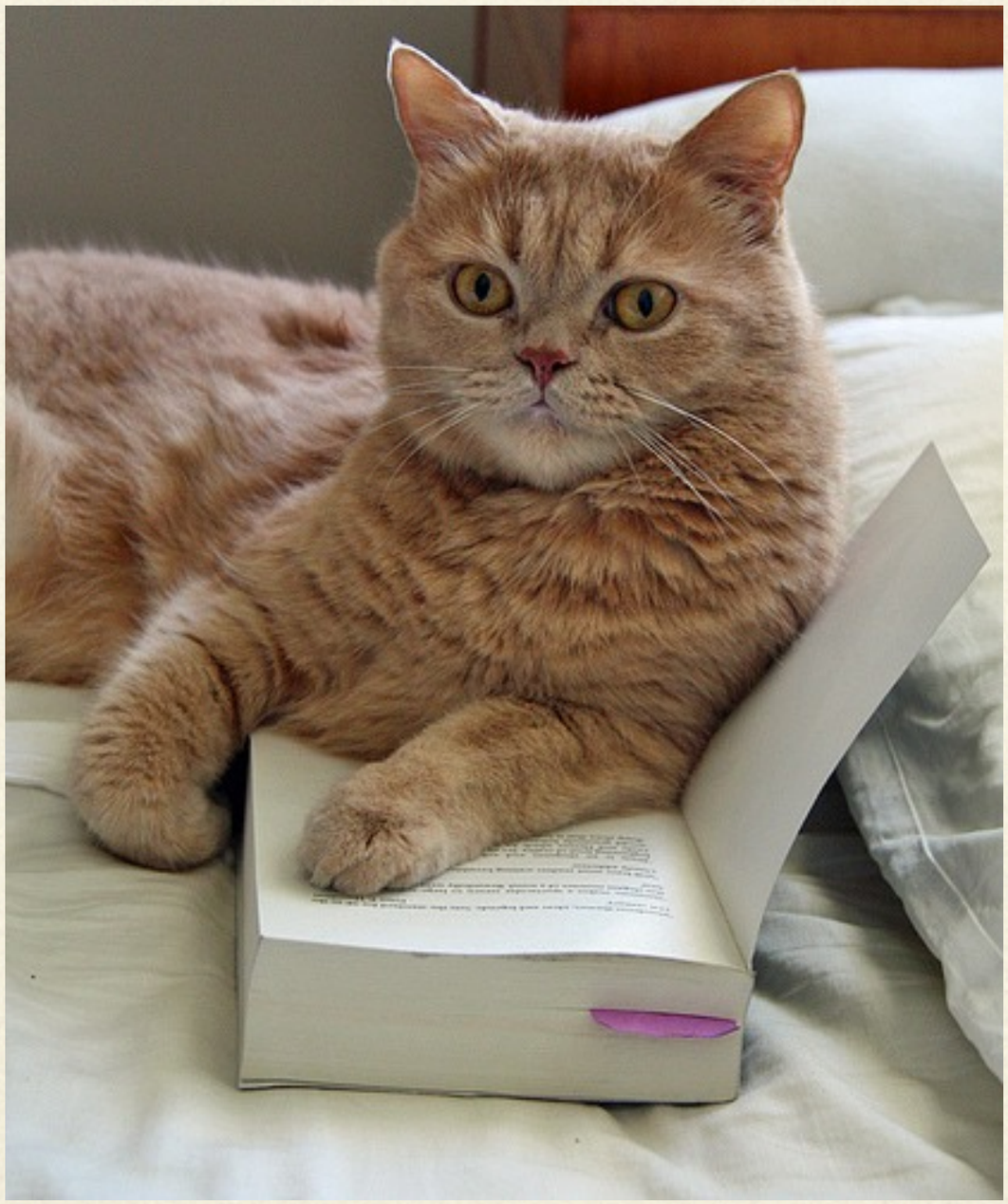
All | Comments | Work Log | History | Activity | Subversion Commits

Edward Capriolo added a comment - 26/May/12 00:48
Patches welcome. I am sure if you re factor the code and make it better no one will be adverse .

# Terrible…It's not good…

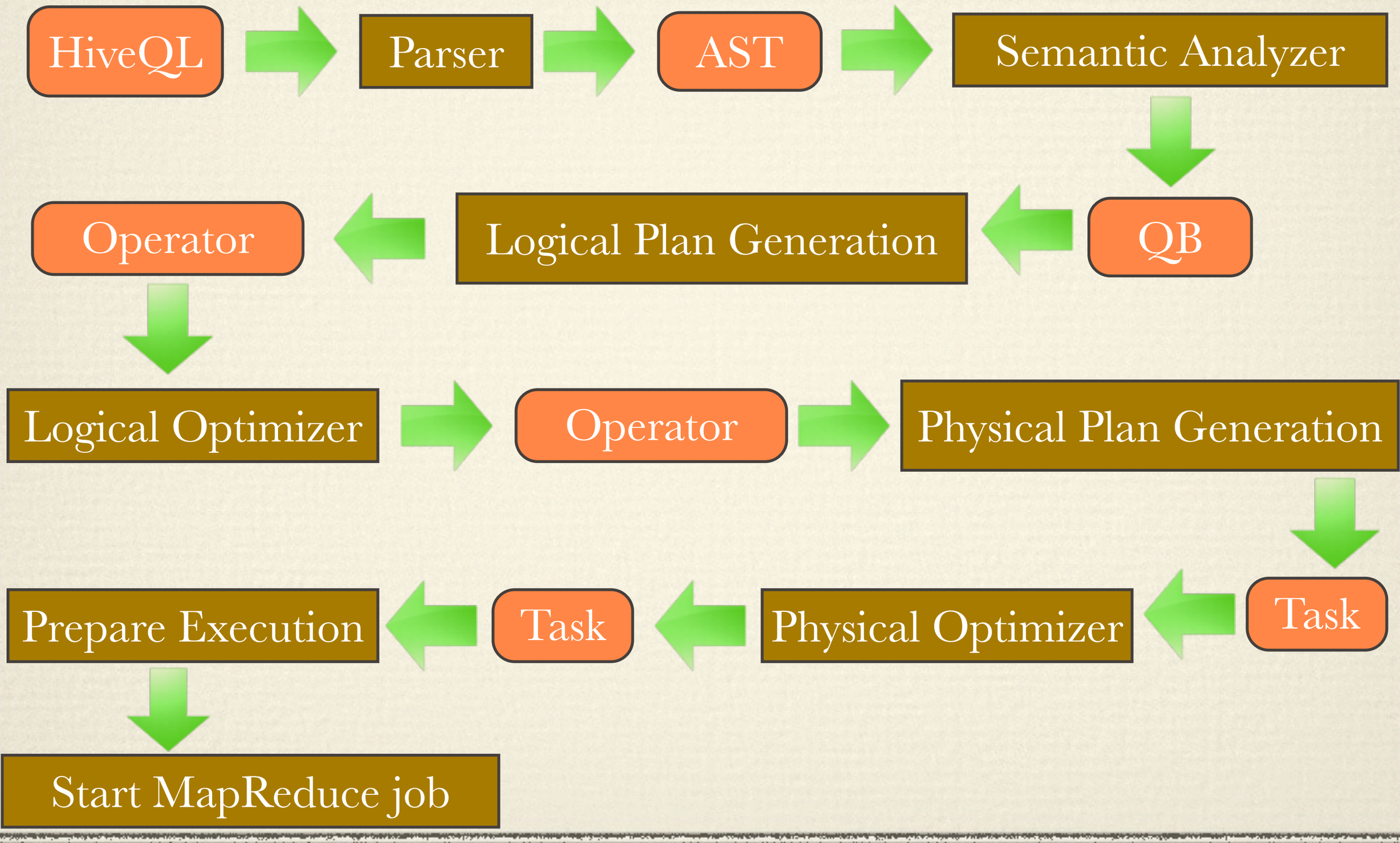
http://matome.naver.jp/odai/2133622524999684001/2133810307761770103

# anyway...



http://matome.naver.jp/odai/2133708130778775801/2133714001583346703

# My understanding



# Operator Example

- FetchOperator
- SelectOperator
- FilterOperator
- GroupByOperator
- TableScanOperator
- ReduceSinkOperator
- JoinOperator

# Explain Example 1

```
hive> explain select * from aaa;
...

STAGE DEPENDENCIES:
  Stage-0 is a root stage

STAGE PLANS:
  Stage: Stage-0
    Fetch Operator
      limit: -1
```

# Explain Example 2

```
hive> explain select a from aaa where a=1;
OK
ABSTRACT SYNTAX TREE:
  (TOK_QUERY (TOK_FROM (TOK_TABREF (TOK_TABNAME aaa))) (TOK_INSERT (TOK_DESTINATION (TOK_DIR TOK_TMP_FILE)) (T
OK_SELECT (TOK_SELEXPR (TOK_TABLE_OR_COL a))) (TOK_WHERE (= (TOK_TABLE_OR_COL a) 1))))

STAGE DEPENDENCIES:
  Stage-1 is a root stage
  Stage-0 is a root stage

STAGE PLANS:
  Stage: Stage-1
    Map Reduce
      Alias -> Map Operator Tree:
        aaa
          TableScan
            alias: aaa
            Filter Operator
              predicate:
                  expr: (a = 1)
                  type: boolean
              Select Operator
                expressions:
                      expr: a
                      type: int
                outputColumnNames: _col0
                File Output Operator
                  compressed: false
                  GlobalTableId: 0
                  table:
                      input format: org.apache.hadoop.mapred.TextInputFormat
                      output format: org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat

Stage: Stage-0
  Fetch Operator
    limit: -1
```

# Optimizer

- Rule-based
- Optimizer is a set of Transformation rules on the operator tree
- The transformation rules are specified by a regexp pattern on the tree.
- The rule with minimum cost are executed.
- The cost is regexp pattern match length.

# Cost Calculation Example

- HiveQL is “select key from src;”
- Optimizer is ColumnPruner.
- SelectOperator is expressed as “SEL%”.
- Regexp pattern of ColumnPrunerSelectProc rule is expressed as “SEL%”.
- The cost is 4.
- Pattern.compile(“SEL%”).matcher(“SEL%”).group().length()

# Logical Optimizer Example

- ColumnPruner
- PredicatePushDown
- PartitionPruner
- GroupByOptimizer
- UnionProcessor
- JoinReorder

# Physical Optimizer Example

- SkewJoinResolver
- CommonJoinResolver
  - If hive.auto.convert.join=true, resolver start.
- IndexWhereResolver
- MetadataOnlyOptimizer

# Break

http://matome.naver.jp/odai/2133708130778775801/2133714001583346703

# Where does Hive tune performance?

- hive.map.aggr
- hive.auto.convert.join
- hive.exec.parallel

Logical Plan Generation

Physical Optimizer

Prepare Execution

# hive.map.aggr

- default true

Logical Plan Generation

- map side aggregation(in-mapper combining)

```
1: class Mapper
2:     method Map(docid a, doc d)
3:         H ← new AssociativeArray
4:         for all term t ∈ doc d do
5:             H{t} ← H{t} + 1                ▷ Tally counts for entire document
6:         for all term t ∈ H do
7:             Emit(term t, count H{t})
```

**Figure 3.2:** Pseudo-code for the improved MapReduce word count algorithm that uses an associative array to aggregate term counts on a per-document basis. Reducer is the same as in Figure 3.1.

http://www.umiacs.umd.edu/~jimmylin/MapReduce-book-final.pdf

# hive.auto.convert.join

- default false
- If hive.auto.convert.join=true, map join start.

Physical Optimizer

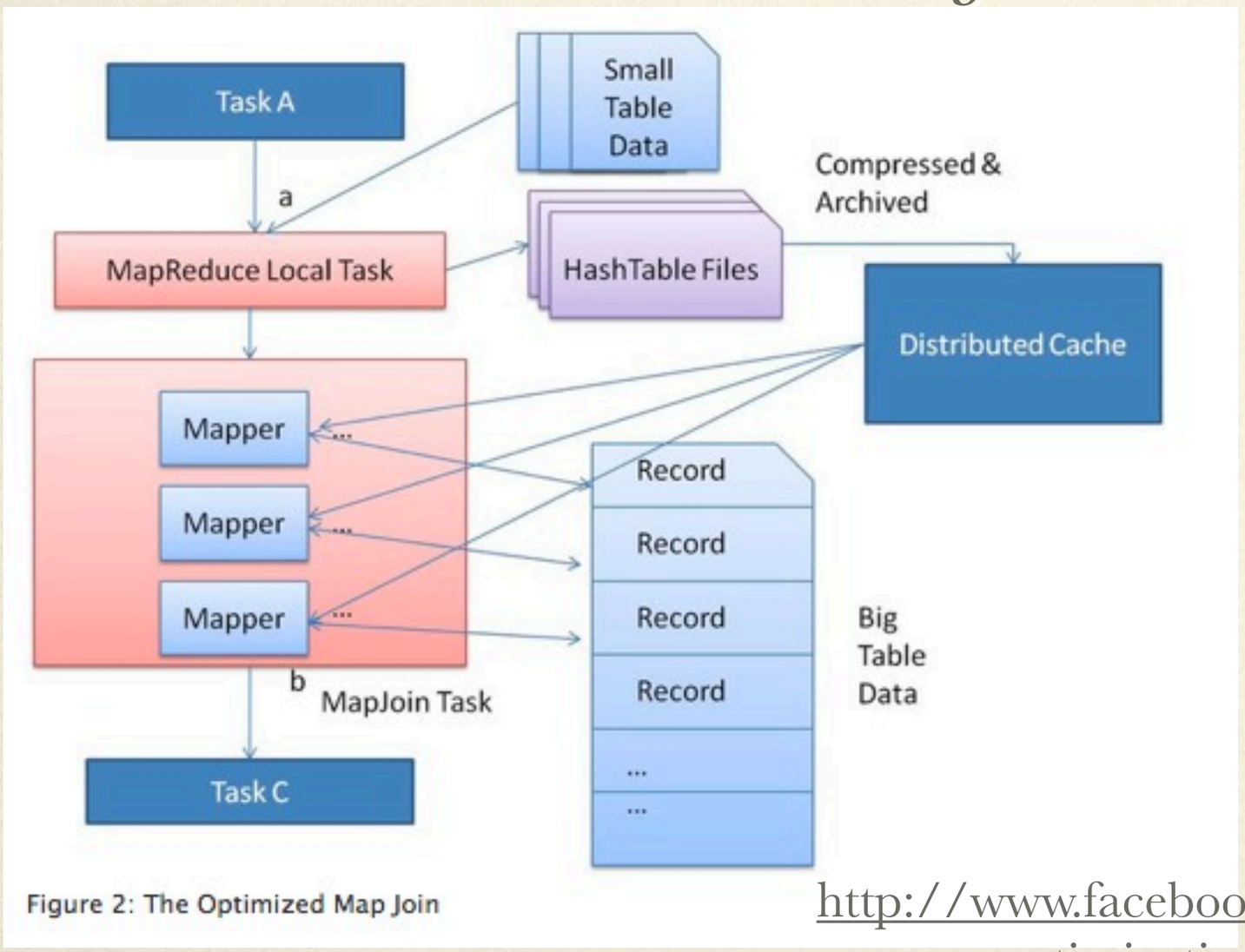


Figure 2: The Optimized Map Join

http://www.facebook.com/notes/facebook-engineering/join-optimization-in-apache-hive/470667928919

# hive.exec.parallel

- default false

Prepare Execution

- If hive.exec.parallel=true, parallel launch(like parallel query in Oracle) start. It is effective in multi table insert.

```
2012-04-03 19:34:09,346 Stage-4 map = 0%,  reduce = 0%
2012-04-03 19:34:16,576 Stage-5 map = 0%,  reduce = 0%
2012-04-03 19:34:47,630 Stage-4 map = 100%,  reduce = 0%
2012-04-03 19:34:57,716 Stage-5 map = 100%,  reduce = 0%
2012-04-03 19:35:40,147 Stage-4 map = 100%,  reduce = 67%
2012-04-03 19:35:46,430 Stage-4 map = 100%,  reduce = 100%
2012-04-03 19:35:46,443 Stage-5 map = 100%,  reduce = 100%
```

# My impression

- There are a few choices of performance tuning. (hive.auto.convert.join, hive.exec.parallel)
- Data model, ETL flow is more important than performance tuning.
- Current optimization in Hive is rule-based. If index and statistics and cost-based optimization are more developed, there will be more choices of performance tuning.