# Hive on Tez



# Overview

Tez is a new application framework built on Hadoop Yarn that can execute complex directed acyclic graphs of general data processing tasks. In many ways it can be thought of as a more flexible and powerful successor of the map-reduce framework.

It generalizes map and reduce tasks by exposing interfaces for generic data processing tasks, which consist of a triplet of interfaces: input, output and processor. These tasks are the vertices in the execution graph. Edges (i.e.: data connections between tasks) are first class citizens in Tez and together with the input/output interfaces greatly increase the flexibility of how data is transferred between tasks.

Tez also greatly extends the possible ways of which individual tasks can be linked together; In fact any arbitrary DAG can be executed directly in Tez.

In Tez parlance a map-reduce job is basically a simple DAG consisting of a single map and reduce vertice connected by a “bipartite” edge (i.e.: the edge connects every map task to every reduce task). Map input and reduce outputs are HDFS inputs and outputs respectively. The map output class locally sorts and partitions the data by a certain key, while the reduce input class merge-sorts its data on the same key.

Tez also provides what basically is a map-reduce compat layer that let’s one run MR jobs on top of the new execution layer by implementing Map/Reduce concepts on the new execution framework.

More information about Tez can be found here:

- Tez website: http://tez.apache.org/
- Tez design doc: https://issues.apache.org/jira/browse/TEZ-65
- Tez presentation: http://www.youtube.com/watch?v=9ZLLzlsz7h8
- Tez slides: http://www.slideshare.net/Hadoop_Summit/murhty-saha-june26255pmroom212

Hive uses map-reduce as its execution engine. Any query will produce a graph of MR jobs potentially interspersed with some local/client-side work. This leads to many inefficiencies in the planning and execution of queries. Here are some examples that can be improved by using the more flexible Tez primitives:

## Multiple reduce stages

Whenever a query has multiple reduce sinks (that cannot be combined, i.e.: no correlation between the partition keys), Hive will break the plan apart and submit one MR job per sink. All of the MR jobs in this chain need to be scheduled one-by-one and each one has to re-read the output of the previous job from HDFS and shuffle it. In Tez several reduce sinks can be linked directly and data can be pipelined without the need of temporary HDFS files. This pattern is referred to as MRR (Map - reduce - reduce*).

## Pipelining

More than just MRR, Tez allows for sending the entire query plan at once thus enabling the framework to allocate resources more intelligently as well as pipelining data through the various stages. This is a huge improvement for more complicated queries as it eliminates IO/sync barriers and scheduling overhead between individual stages. An example would be a query that aggregates two tables in subqueries in the from clause and joins the resulting relations.

### In memory versus disk writes

Currently any shuffle is performed the same way regardless of the data size. Sorted partitions are written to disk, pulled by the reducers, merge-sorted and then fed into the reducers. Tez allows for small datasets to be handled entirely in memory, while no such optimization is available in map-reduce. Many warehousing queries sort or aggregate small datasets after the heavy lifting is done. These would benefit from an in memory shuffle.

### Joins

Distributed join algorithms are difficult to express in map-reduce. A regular shuffle join for instance has to process different inputs in the same map task, use tags to be written to disk for distinguishing tables, use secondary sort to get the rows from each table in a predictable order, etc. Tez is a much more natural platform to implement these algorithms.

For example: It is possible to have one Tez task take multiple bipartite edges as input thus exposing the input relations directly to the join implementation. The case where multiple tasks feed into the same shuffle join task will be referred to as multi-parent shuffle join.

### Fine-tuned algorithms

All sorting in map-reduce happens using the same binary sort, regardless of the data type. Hive might for instance choose to use a more effective integer-only sort when possible. Tez makes that available.

Since Hive uses map-reduce to compute aggregations, processing will always boil down to a sort-merge even though we're not actually interested in the sort order. Tez will allow for more efficient hash-based algorithms to do the same.

### Limit processing

Tez allows complete control over the processing, including being able to stop processing when limits are met (without simply skipping records or relying on file formats/input formats.) It's also possible to define specific edge semantics, which could be used to provide a generic top-k edge to simplify "limit" processing.

## Scope

The rest of this document describes the first phase of Hive/Tez integration. The goals are:

- Bring Tez concepts and primitives into Hive, make them available to all Hive developers
- Take advantage of TEZ through MRR (Multiple reduce-stage jobs)
- Take advantage of TEZ through MPJ (multi-parent shuffle joins)

Limiting the integration to the fairly simple MRR/MPJ pattern will require minimal changes to the planner and execution framework while speeding up a wide variety of queries. At the same time it will allow us to build a solid foundation for future improvements.

## Functional requirements of phase I

- Hive continues to work **as is** on clusters that do not have TEZ.
  - MR revisions 20, 20S, 23 continue to work unchanged.
- Hive can optionally submit MR jobs to TEZ without any additional improvements.
  - Hive can treat TEZ like just another Hadoop 23 instance.
- Hive can optionally detect chains of MR jobs and optimize them to a single DAG of the form MR* and submit it to TEZ.
- Hive can optionally detect when a join has multiple parent tasks and combine them into a single DAG of a tree shape.
- Hive will display the MRR optimization in explain plans.
- Hive will give appropriate feedback to the user about progress and completion status of the query when running MRR queries.
- The user will be able to get statistics and diagnostic information as before (counters, logs, debug info on the console).
- Hive has unit tests to cover all new functionality.

The following things are out of scope for the first phase:

- Local tasks will still run as MR only.
- Only Map and Reduce Tez tasks with SimpleEdges will be used (i.e.: no new tasks, new input/output/processors, no new edge types).
- No multi-output task optimizations will be introduced.

One new configuration variable will be introduced:

- ~~hive.optimize.tez~~
  hive.execution.engine (changed in HIVE-6103)
  - ~~True~~
    tez: Submit native TEZ dags, optimized for MRR/MPJ

    - ~~False~~
      mr (default): Submit single map, single reduce plans
- Update: Several configuration variables were introduced in Hive 0.13.0. See the Tez section in Configuration Properties.

Note: It is possible to execute an MR plan against TEZ. In order to do so, one simply has to change the following variable (assuming Tez is installed on the cluster):

- mapreduce.framework.name = yarn-tez

## Example

Here's a TPC-DS query and plans with and without Tez optimizations enabled:

The query (rewritten for Hive):

```
select
  i_item_desc
  ,i_category
  ,i_class
  ,i_current_price
  ,i_item_id
  ,itemrevenue
  ,itemrevenue*100/sum(itemrevenue) over
    (partition by i_class) as revenueratio
from
  (select
     i_item_desc
     ,i_category
     ,i_class
     ,i_current_price
     ,i_item_id
     ,sum(ws_ext_sales_price) as itemrevenue
   from
     web_sales
     join item on (web_sales.ws_item_sk = item.i_item_sk)
     join date_dim on (web_sales.ws_sold_date_sk = date_dim.d_date_sk)
   where
     i_category in ('1', '2', '3')
     and year(d_date) = 2001 and month(d_date) = 10
   group by
     i_item_id
     ,i_item_desc
     ,i_category
     ,i_class
     ,i_current_price) tmp
order by
  i_category
  ,i_class
  ,i_item_id
  ,i_item_desc
  ,revenueratio;
```

### Plan with TEZ

Stage 0:

Local Work: Generate hash table for date dim

Stage 1:

Map: SMB join item + web_sales, mapjoin date_dim + web_sales, map-side group by/aggregate

Reduce 1: Reduce side group by/aggregate, shuffle for windowing

Reduce 2: Compute windowing function, shuffle for order by

Reduce 3: Order by, write to HDFS

### Plan without TEZ

Local Work: Generate hash table for date dim

Stage 1:

Map: SMB join item + web_sales, mapjoin date_dim + web_sales, map-side group by/aggregate

Reduce: Reduce side group by/aggregate, write to HDFS

Stage 2:

Map: Read tmp file, shuffle for windowing

Reduce: Compute windowing function, write to HDFS

Stage 3:

Map: Read tmp file, shuffle for order by

Reduce: Order by, write to HDFS

# Design

## Summary of changes

Changes that impact current Hive code paths:

- Split MR compilation from SemanticAnalyzer (simple)
- Break MapRedWork into MapWork and ReduceWork (straight forward but a number of changes)
- Move execution specific classes from exec to exec.mr package (simple)
- Move some functions from ExecDriver to exec.Utilities to share between Tez and MR
- Build system: Add Tez dependencies and Tez testing

I believe that all of these are valuable by themselves and make the code cleaner and easier to maintain. Especially the second item will touch quite a few places in the code though. None of them change functionality.

New code paths (only active when running Tez):

- Execution: TezWork, TezTask, TezJobMonitor (small)
- Planning: Compiler to generate TezTasks, Perform physical optimizations on Tez (large)

The following outlines the changes across the various Hive components:

## Execution layer

We've initially investigated to add Tez as a simple shim option to the code base. This didn't work out mostly because Tez' API is very different from the MR api. It does not make much sense to move the entire "execute" infrastructure to the shim layer. That would require large code changes with little benefit. Instead there will be separate "Task" implementations for MR and TEZ and Hive will decide at runtime which implementation to use.

We're planning to have two packages:

- org.apache.hadoop.hive.ql.exec.mr
- org.apache.hadoop.hive.ql.exec.tez

Both will contain implementations of the Task interface, which is used to encapsulate units of work to be scheduled and executed by the Driver class.

Both of these packages will have classes for job monitoring and job diagnostics, although they are package private and do not follow a common interface.

### Job submission

Currently ExecDriver and MapRedTask (both are of type "Task") will submit map-reduce work via JobClient (either via local-job runner or against the cluster). All MR specific job submission concepts are hidden behind these classes.

We will add a TezTask as the entry point for Tez execution. TezTask will hide building of the job DAG and orchestrate monitoring and diagnostics of DAG execution.

Hive's driver will still deal with a graph of Tasks to handle execution. No changes are required to handle this. The only difference is that now the planner might transparently add TezTasks to the mix at runtime.

### Job monitoring

We will add a TezJobMonitor class that handles printing of status as well as reporting the final result. This class provides similar functions than HadoopJobExecHelper used for MR processing. TezJobMonitor will also retrieve and print the top level exception thrown at execution time.

### Job diagnostics

Basic ‘job succeeded/failed’ as well as progress will be as discussed in “Job monitoring”.  Hive’s current way of trying to fetch additional information about failed jobs will not be available in phase I.

Currently Tez offers limited debugging support once a job is complete. The only way to get access to detailed logs, counters, etc is to look at the log of the AM, find the appropriate url for specific task logs and access them through copy and paste. This will change over time and historical logging information will be accessible, but for the first phase debugging support will be limited.

### Counters

API for retrieving counters will be different in Tez and we will thus add a shim api for that. Incrementing counters at execution time will work unchanged.

### Job execution

The basic execution flow will remain unchanged in phase I. ExecMapper/ExecReducer will be used through Tez’ MR compat layer. The operator plan + settings will be communicated via ‘scratch dir’ as before. ExecMapper/Reducer will load and configure themselves accordingly.

## Query planning

### MapRedWork

MapRedWork is where we currently record all information about the MR job during compile/optimize time.

This class will be refactored to capture Map information and reduce information separately (in MapWork and ReduceWork). This requires quite a few - albeit straight-forward - changes to both planning and execution.

The refactor has benefits for pure MR execution as well. It removes the need for Mappers/Reducers to load and de-serialize information they don’t use and it makes it easier to read and maintain the code, because it clearly delineates what information is used at what stage.

MapWork and ReduceWork will be shared by both MapRedWork and TezWork. MapRedWork is basically a composition of 1M + 0-1R, while TezWork is a tree of Map/ReduceWork with MapWork classes as leaves only.

As discussed above, TezTask will use TezWork, while MapRedTask and ExecDriver will use MapReduceWork.

### Semantic analysis and logical optimizations

Neither semantic analyzer nor any logical optimizations will change. Physical optimizations and MR plan generation are currently also done in the SemanticAnalyzer and will be moved out to separate classes.

### Physical Optimizations and Task generation

The MapReduceCompiler does two things at the same time right now. After breaking down the operator plan into map-reduce tasks, it optimizes the number of tasks and also performs physical optimizations (picks join implementations, total order sort, etc).

In order to limit the impact of Tez, we will provide a separate implementation: TezCompiler. The Tez compiler will attempt to perform most physical optimizations at the plan level, leaving the breakdown of the plan into Tez jobs for a second round of optimizations.

Later we may decide to use that physical optimizer (at the plan level) for both MR and Tez, while leaving specific optimizations in the two layers.

### Local Job Runner

In the short term Tez will not support a “LocalTezDagRunner”. That means that Hive will always have to submit MR jobs when executing locally. In order to avoid replanning the query after execution has started in Tez mode some optimizations for converting stages to local jobs will not be available.

### Number of tasks

Some MR jobs have a predetermined number of reducers. This happens for order by (numReducers = 1) and scenarios where bucketing is used (numReducers = numBuckets). The user can also set the number of reducers manually. The same numbers will be used for each reduce tasks. Initially there will be no way for the user to set different numbers of reducers for each of the separate reduce stages. There is already a ticket (HIVE-3946) to address this shortcoming which can be used for both Tez and MR.

In most cases Hive will determine the number of reducers by looking at the input size of a particular MR job. Hive will then guess the correct number of reducers. The same guess will be used for subsequent reduce phases in a Tez plan. Ultimately, this number will have to be determined using statistics which is out of scope, but applies equally to MR and Tez.

### Explain statements

Explain statements are driven (in part) off of fields in the MapReduceWork. Part of extending/refactoring MapReduceWork will be to add sufficient information to print the correct operator trees in explain for Tez.

### Hive variables

The “set” mechanism for Hive variables will not change. The variables will be passed through to the execution engine as before. However, Hive will not shim or map any mapreduce variables. If a variable is not supported in Hive it will be silently ignored.

## Build infrastructure

There will be a new “ql” dependency on Tez. The jars will be handled the same way Hadoop jars are handled, i.e.: They will be used during compile, but not included in the final distribution. Rather we will depend on them being installed separately. The jars will only have to be present to run Tez jobs, they are not needed for regular MR execution.

## Testing

### Mini Tez Cluster

Mini Tez Cluster will initially be the only way to run Tez during unit tests. LocalRunner is not yet available. If mr.rev is set to tez all MiniMr tests will run against Tez.

# Installation and Configuration

For information about how to set up Tez on a Hadoop 2 cluster, see https://github.com/apache/incubator-tez/blob/branch-0.2.0/INSTALL.txt.

For information about how to configure Hive 0.13.0+ for Tez, see the release notes for HIVE-6098, Merge Tez branch into trunk. Also see Configuration Properties: Tez for descriptions of all the Tez parameters.

### Hive-Tez Compatibility

For a list of Hive and Tez releases that are compatible with each other, see Hive-Tez Compatibility.