A Brief Introduction to Apache Tez

Introduction

It is a fact that data is basically the new currency of the modern business world. Companies that effectively maximize the value of their data (extract value in a timely fashion) decisively affect their bottom-line. Apache Hadoop represents the de-facto standard for Big Data batch (and some interactive) processing solutions and provides great horizontal scalability at much lower cost than traditional SAN based cluster solutions. In a Hadoop environment, companies can execute parallel analytical workload scenarios and disclose formerly unknown insights that were previously hidden due to technical and/or economical constraints. To reiterate, Hadoop provides rich data value at large scale as well as efficiency via Big Data solutions such as MapReduce (MR), Apache Tez, and YARN (Yet Another Resource Negotiator). Analytical applications perform data processing in a purpose-driven manner that is unique to a specific business problem.

The Hadoop YARN component provides a distributed application hosting, managing, and execution infrastructure (see Figure 2). Next to YARN, an actual application abstraction layer is required and with Hadoop may be provided by either MapReduce or Tez. This primer does not discuss other Apache projects in detail (such as Pig, Hive, or HBase to name a few) that nicely interact with Hadoop. Apache Tez reflects an embedded, extensible framework that easily integrates with YARN and supports the development of interactive as well as batch applications. Tez leverages Hadoop's ability to process massive datasets via a fit-to-purpose data processing logic at a very good response time and throughput level. The Tez performance and scalability focus is beneficial to other Apache projects such as Hive and Pig that ultimately access data that is stored in Hadoop HDFS.

Differences and Similarities Between MapReduce & Tez

Compared to MapReduce, Apache Tez basically depicts a broader, more feature-rich application framework that maintains MapReduce's strengths while overcoming some of its limitations. To illustrate, Tez retains the following MapReduce features:

- Horizontal scalability
- Resource elasticity
- Fault tolerance and failure recovery
- Secure data processing

In a nutshell, Tez basically allows an application to be modeled as a data flow where edges (nodes) reflect the data movement and the vertices represent the data processing tasks. Apache projects such as Pig or Hive do (after processing the query) generate a directed acyclic graph (DAG) that is ready for execution and hence, Tez is well suited as the executor for Pig or Hive jobs. Tez cannot be described as just an engine though as Tez provides common primitives/modules that can be used to develop applications and so the actual design focus is on flexibility and customizability. Developers may implement actual MapReduce jobs via the Tez library.

As a matter of fact, Tez is deployed with an embedded MapReduce implementation that can be utilized to execute any existing MapReduce job (at the Tez efficiency level). While Hadoop MapReduce definitely has its place in batch processing, many companies that focus on extracting value at improved performance levels (and optimized resource utilization) evaluate Tez as an alternative to the general-

purpose engine that is provided by MapReduce. To further illustrate the difference between MapReduce and Hive, the following Hive statement is considered (see Figure 1 for an illustration of the data flow):

```
SELECT a.state, COUNT(*), AVERAGE(c.price)
FROM a
JOIN b ON(a.id = b.id)
JOIN c ON(a.itemId = c.itemId)
GROUP BY a.state
```

While both, MapReduce and Tez basically execute a DAG, Figure 1 depicts the different execution flow for the 2 solutions (MR vs. Tez). To run the simple statement above, Tez requires fewer jobs (1 vs. 3) and no IO synchronization barriers (provided via HDFS for the MR jobs) are required. The constrained DAG for MapReduce (consisting of 1 set of map followed by 1 set of reduce requests) frequently results in executing multiple MapReduce jobs. This MR execution behavior has a detrimental impact on the latency behavior of short queries (due to the encountered overhead of launching multiple jobs) as well as on the throughput potential of large-scale queries (duo to the excessive overhead of materializing intermediate job outputs to the filesystem). With Tez, a much more streamlined (expressive DAG) of tasks is presented that operates within a single application or job (that is better aligned with the required processing task).

Figure 1: MapReduce vs. Tez Execution Profile

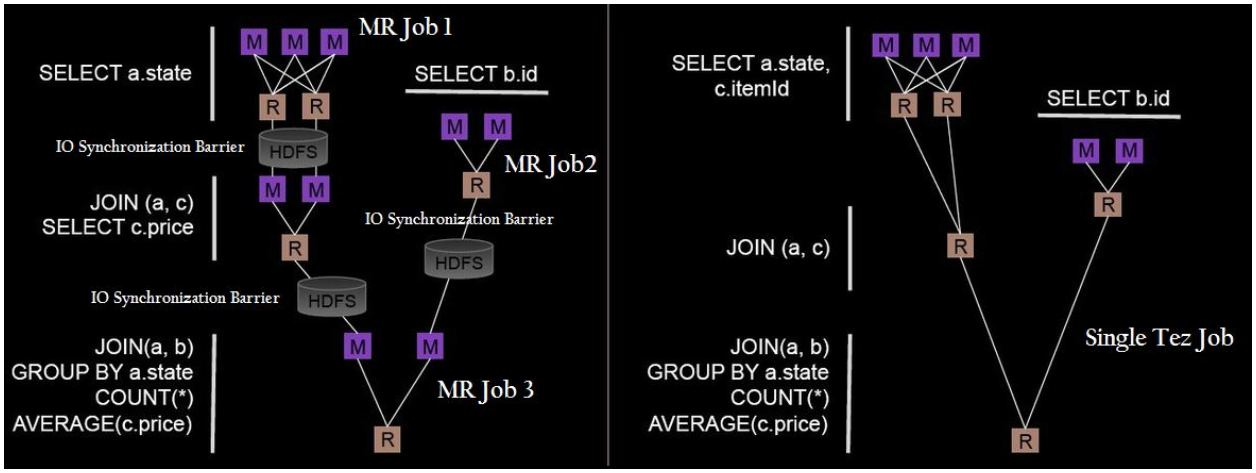

Figure courtesy of Hortonworks

Some Tez Design Considerations

With the introduction of Hadoop 2.0 (YARN), the need for executing an actual workflow (DAG) of MR jobs becomes paramount. Simplified, Hadoop YARN segregates the processing paradigm (MR) and the resource management functionalities (which were interlinked in Hadoop 1.0). So with Hadoop 2.0, the resource management functionality is handled by the YARN resource manager while several other execution frameworks (such as Tez, Storm or Spark) can operate on top of YARN (see Figure 2). All these execution frameworks can co-exist and process data that may be stored in HDFS. With Hadoop 2.0, it is paramount that an application can process a DAG as efficiently and as effectively as possible. With Hadoop 1.0, a DAG was basically described as a chain of individual MR jobs while with Hadoop 2.0 and Tez, the focus is on executing a query or a script as a single DAG task.

Hence, the Apache Tez design focuses on customized data-processing applications that are executed within the Hadoop 2.0 ecosystem. The data is modeled as a data flow graph and Tez provides a good framework for interactive applications that have high response time and throughput requirements (batch processing is supported as well). Each entity in the data flow graph reflects some of the business logic that transforms and/or analyzes the data. The connections among the entities in the graph depict actual data movement. After the application logic has been defined via the DAG, Tez parallelizes the logic and executes it in Hadoop. Any business problem that can be depicted as a DAG may basically be executed in a Tez environment (this holds true for a MapReduce setup as well). To illustrate, any *Extract-Transform-Load* (ETL) application is basically a good fit for Tez. Further, Tez is commonly used in conjunction with query-processing engines such as Hive or scripting languages like Pig.

Figure 2: (Expanded) YARN Framework

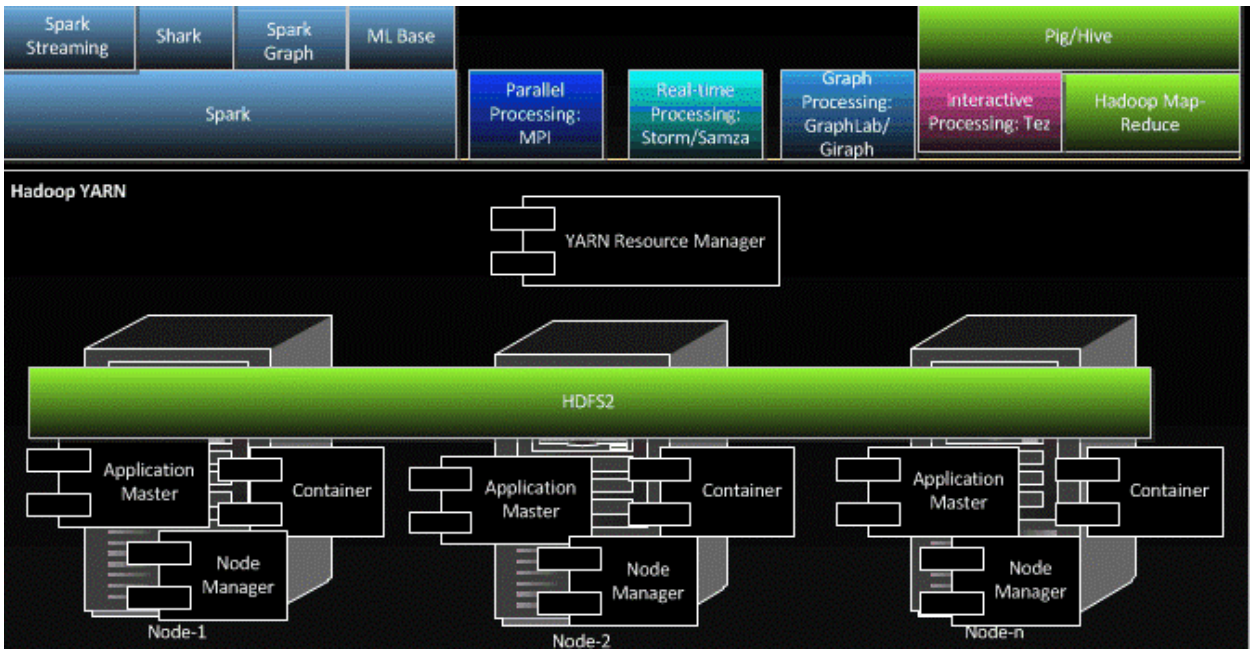

Figure courtesy of Dr. Agneeswaran

The Apache Tez development framework includes Java APIs that provide developers with an intuitive interface to create the data-processing graphs for the applications' data-processing flows. After a flow has been defined, Tez provides additional APIs to inject the (custom) business logic. Modular input, output, and processor elements are provided. Applications developed via these APIs execute efficiently in a Hadoop environment while the actual Tez framework provides the interaction with other stack components. Tez applications basically represent customized, optimized, and YARN integrated solutions that take advantage of Hadoop's scalable and fault-tolerant runtime environment.

As with any other Big Data application, a sound design and a comprehensive understanding of the actual data flow and the business (logic) requirements is necessary though. After the actual business mandate is understood and the decision has been made that a Tez application is a feasible approach to extract value and solve the business problem, it is necessary to define the actual data flow that addresses the problem. More than 1 data flow graph may be identified to solve the problem and choosing the appropriate one is paramount from a performance perspective. As an example, Apache Hive performance is greatly improved if optimal joining graphs via the Tez APIs are defined. Further, by utilizing the Tez input, output, and processor modules, the business logic (that executes the task) can be customized.

To reiterate, every vertex of the dataflow graph can be modeled as a combination of input, processing, and output modules. The input module specifies the set of inbound edges required for this task while the processing module (labeled the *processor*) depicts the data transformation aspects as part of the vertex. The output module references the output data that is generated by the transformation and is passed to the outbound edges of the vertex (node). The vertex is basically equivalent to a task in Tez. The parallelism of the task can either be specified when the DAG is setup or via the user plug-ins that are running at the Application Master (AM) level. The AM is basically a component of Hadoop YARN. The context/configuration information (environment or application specific) is provided to the 3 Tez modules via an initialization routine that is invoked first. Subsequently, the *run* method for the processor is initiated for every task instance (governed by the parallelism factor). Once this method finishes, the task is completed (from a logical perspective).

The output framework (controlled via the *LogicalOutput* class) dumps the data that is received from the processor and may also present information to potentially ensuing downstream input modules (depending on the DAG). Tez does provide sophisticated error handling features for fatal (task has to be terminated) as well as non-fatal (such as re-reading the input) error scenarios. One unique feature is the ability to dynamically optimize the Tez DAG execution via information that is available at runtime. To illustrate, actual data samples and data sizes can be used to further optimize the DAG execution plan on-the-fly. As discussed in this primer, Tez is tightly integrated with Hadoop 2.0 YARN and hence is capable of efficiently negotiating any resource management scenarios and is further capable of accepting YARN containers for execution.

Some (very) Simple Hive/MR & Hive/Tez Benchmark Runs

For this primer, some simple Hive/MR and Hive/Tez benchmarks were executed in a sandbox environment. The setup consisted of a HDP 2.1 setup on an Intel 4-core system (16GB memory) that was equipped with 4 Seagate Cheetah 15K.7 600GB drives. The simple benchmark dataset consisted of a file containing 7 columns and 120,000 rows (similar to the *HVAC.csv* file described in [8] - just more rows). The benchmarks were executed first with no optimization, second with ORC, and third with vectorization.

The Optimized Row Columnar (ORC) file format provides a highly efficient way to store Hive data. ORC was designed to overcome limitations embedded into other Hive file formats. In general, the ORC format improves read/write/processing performance with Hive. With vectorization, 1,000 rows are fetched simultaneously (compared to only 1 without vectorization). This feature normally reduces the application CPU time at an improved cluster utilization level. Vectorization further lowers the application latency as the container usage is optimized. Vectorization only works on ORC Hive tables.

Table 1: Benchmark Results (Response Time in Seconds)

Optimization	MR 1st Run	MR 2nd Run	Tez 1st Run	Tez 2nd Run
None	468.75	467.92	315.36	138.75
ORC	446.17	445.52	311.41	136.45
ORC/Vectorization	NA	NA	309.22	136.22

These very simple benchmark runs nicely highlight some of the features discussed in this primer. With no optimization, the 1st and 2nd Tez run improve the response time behavior (compared to the 1st MR run) by ~32.7% and ~70.6%, respectively. The major improvement between the 1st and 2nd Tez run is due to the dynamic query optimization and process re-usage feature embedded into the Tez framework. As expected, the response time delta between the 1st and 2nd MR run is minuscule.

As depicted in Table 1, the ORC and vectorization features did not have a major impact on the response time behavior of these simple benchmark scenarios. The theory is that this is due to the rather small dataset being used for these benchmarks and hence the statement being made is that in order to really see the advantages of ORC and vectorization, much larger datasets have to be processed.

References

1. *Marz, Big Data: Principles and Best Practices of Scalable Real-time Data Systems, Manning Publications, October 2014*
2. *Srirama et al., Adapting scientific computing problems to clouds using MapReduce. Future Generation Computer System 28, 2012*
3. *Bunch et al., Mapscale: A cloud environment for scientific computing, Technical Report, University of California, Computer Science Department, 2009*
4. *Agneeswaran, TEZ Introduction, Analytics Magazine, 2014*
5. *Murthy et al., Apache Tez: Accelerating Hadoop Query Processing, Hadoop Summit, 2013*
6. *Hortonworks, Apache Tez : Accelerating Hadoop Query Processing, 2013*
7. *Tez Blogs and Tez Apache Website, 2015*
8. *Hortonworks, Interactive Query for Hadoop with Apache Hive on Apache Tez, Benefits of the Stinger Initiative delivered, http://hortonworks.com/hadoop-tutorial/supercharging-interactive-queries-hive-tez/*