Beyond Hadoop MapReduce
Apache Tez and Apache Spark

Prakasam Kannan
Computer Science Department
San Jose State University
San Jose, CA 95192
408-924-1000

Prakasam.kannan@sjsu.edu

ABSTRACT
Hadoop MapReduce has become the de facto standard for processing voluminous data on large cluster of machines, however this requires any problem to be formulated into strict three-stage process composed of Map, Shuffle/Sort and Reduce. Lack of choice in inter-stage process communication facilities makes Hadoop MapReduce unfit for iterative (Graph Processing, Machine Learning) and interactive workloads (Streaming data processing, Interactive Data mining and Analysis). This paper delves into Hadoop and MapReduce architecture and its shortcomings and examines alternatives such as Apache Tez and Apache Spark for their suitability in iterative and interactive workloads.

1. HADOOP
Traditionally large-scale computation has been processor bound and handled relatively small amount of data (typically several GBs) once the data is copied into memory, processor can perform complex computations, usually taking more time than the time it took the copy the data into memory.

This kind of computation often performed by scaling the Hardware vertically while vertical scaling of hardware was very expensive it reduced the complexity of the application code greatly. Nevertheless expensive nature of such systems limited their adaptation to areas where cost was not a major concern.

Distributed computing was invented to reduce the cost by scaling the hardware horizontally. However programming such systems is very complex and failure of nodes is the defining difference of such systems. Also these systems stored data centrally and had to suffer from limited bandwidth of the network.

Arrival of Internet age has spawned large websites and petabyte scale databases, storing such massive amount of data centrally and shuttling them across network for processing has become impractical.

Hadoop was created to handle processing of such massive amount of data using large cluster of desktop class hardware. Hadoop design is based on Google's GFS (Google File System) and MapReduce framework thus Hadoop is also made up of these two primary components namely HDFS (Hadoop Distributed File System) and MapReduce.

Hadoop distributes the data across the nodes in the cluster in advance and computation on this data is performed locally on those nodes, Hadoop preserves the reliability by replicating this data across 2 or more nodes.

MapReduce applications that process this distributed data are written using higher-level abstractions to free the application developers from the concerns like scheduling, node failure, network access and temporal dependencies.

1.1 HDFS
HDFS designates one or more nodes in the cluster as Name Node and the rest as Data Nodes. Name Nodes maintain the metadata about the files on the HDFS and the actual data itself reside on one or more Data Nodes according to the replication settings.

Data Nodes also double as Task Executors where actual processing of the data is performed. When a file is copied into HDFS it is split into blocks of 64MBs (or as specified in the configuration) and distributed across the cluster

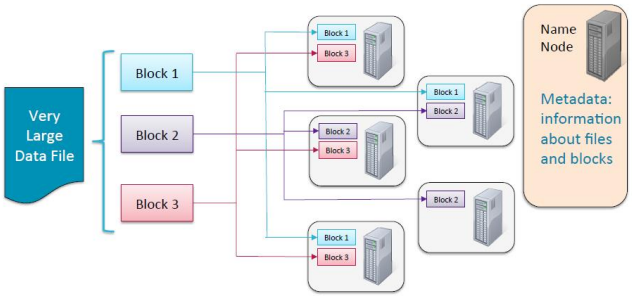


Figure 1. HDFS sharding and replication

1.2 MapReduce
Applications that process the data stored on the Hadoop cluster are expressed in terms of Map and Reduce functions. Data in the file is presented to the Map function by the framework as a pair of key and value and the Map function maps this key and value into another pair of key and value.

These key and value pairs can be any user defined object representation of the underlying data expressible in languages like Java, C, Python, etc. All values produced by a mapper for a given key is collected into list and sorted by the frame work and such sorted list of values and the key are presented to the Reduce function as value and key respectively.

When a MapReduce application is submitted for execution to Hadoop cluster first mapping function is scheduled to run one or

more nodes in the cluster based on number of splits estimated for a given data file.

Once mapping is complete output of the one or more instances of mapping function sorted by key are merged and all values of given key is given to single instance of reduce function which in turn produces one or more key and value pair.

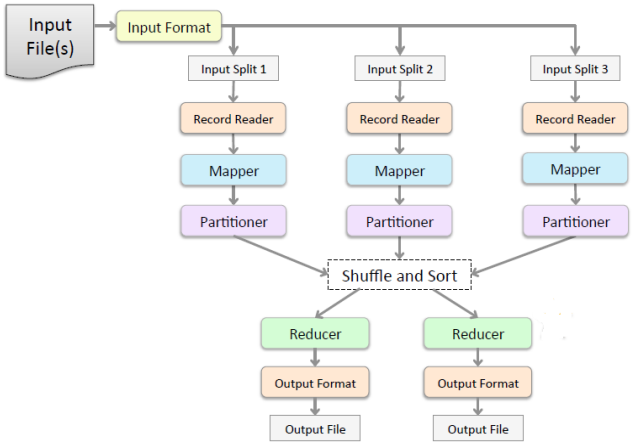


Figure 2. MapReduce application runtime model

1.3 Limitations

While MapReduce ameliorates the complexity of writing a distributed application greatly it achieves that goal through severe restrictions on input/output model and problem composition.

MapReduce requires that all solutions be composed into just two phases however it allows chaining of several such MR phases for execution one after another. While this requirement simplifies the application development it also makes iterative, interactive and graph processing workloads highly inefficient.

Requirement that the nodes in the cluster keep the interactions to the minimum and intermediate output are stored in the hard drive and almost all communication happen through transfer of files makes the turnaround times undesirable for such workloads.

Such inherent limitations of MapReduce coupled with the ever-cheaper DRAM prices and highly reliable datacenter LAN has engendered new models of processing data on Hadoop clusters most notably Apache Tez and Apache Spark.

2. APACHE TEZ

Apache Tez is a new application development model on Hadoop cluster based on the work done by Microsoft on Dryad. While MapReduce can be though of a DAG (Directed Acyclic Graph) of 3 vertices, Dryad allows application to composed into DAG of as many number of vertices of as desired and the interactions among them can be performed using shared memory, network pipes or files.

2.1 Application Model

Tez models an application into dataflow graph where vertices represent the application logic and dataflow among these vertices of application logic as edges. Query plans of declarative languages like SQL maps naturally into such DAGs and improves the fidelity of the API.

Vertices of the application vertices can interact using shared memory, TCP/IP or files stored on HDFS or local file systems these communications can be one-to-one or broadcast to group of vertices or producer vertices scatter data into shards across the cluster and consumer vertices can gather shards.

Scatter-Gather model is used to execute legacy MapReduce applications on DAG execution instead of MapReduce framework.

Group of vertices can be scheduled on single node or single when the vertices on the down stream depend only on the data from the upstream vertices on the same node or process. These vertices can be schedule to execute sequentially or concurrently as well.

Output of the vertices can be persisted to a local file system or HDFS with or with out replication based on the reliability requirements. Tez also provide for ephemeral storage of the output where the output is destroyed after consumption downstream vertices.

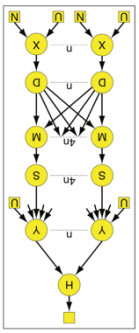


Figure 3. SQL query as DAG

2.1.1 DAG API

```
// Define DAG
DAG dag = new DAG();

// Define Vertex
Vertex Map1 = new Vertex(Processor.class);

// Define Edge
Edge edge = new Edge(Map1, Reduce1,
                     SCATTER_GATHER
                     PRESISTED,
                     SEQUNTIAL,
                     Output.class,
                     Input.class);

// Connect them
dag.addVertex(Map1).addEdge(edge)...
```

2.2 Dynamic Graph Configuration

Most of the time different vertices of the a DAG will have different level of complexity and data characteristics so DAG execution engine of Tez support pluggable vertex management modules to gather runtime information like data samples and sizes.

The execution engine uses the data collected by such modules to optimize and reconfigure execution plan downstream. The diagram below shows how data gathered by such modules are used to determine the number of reducers.

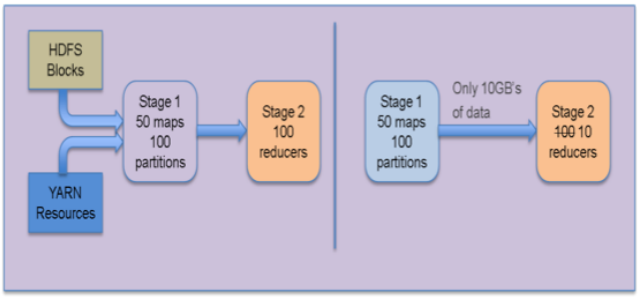


Figure 4. Dynamic graph configuration

Most of this flexibility and efficiency is achieved in declarative fashion with out bogging down the application developer with the nitty-gritty of scheduling and networking APIs. This level of flexibility and efficiency makes Tez a great candidate for interactive data mining applications.

3. APACHE SPARK

Apache Spark was developed in 2009 at UC Berkeley AMP Lab and then open sourced in 2010. While MapReduce and DAG Execution engines abstract distributed processing on a cluster Spark provides abstraction over distributed memory on such clusters.

Spark is quite different from distributed memory abstractions like MemCached in that the later is provides for fine grained read and modify operations on mutable objects that is suitable for OLTP systems. In such systems fault tolerance is provided through replication of data, which may be impractical at data volumes of OLAP systems.

Spark provides an abstraction based on coarse-grained transformations that apply same operation to many data items. Spark also keeps track of enough information about the lineage of such transformations, so such they can be recomputed in the event of failures.

3.1 Resilient Distributed Datasets

RDD is a read-only, partitioned collection of records that can be created though deterministic operations on data in a stable storage or other RDD [3]. These deterministic operations are generally referred as transformations.

As mentioned early these RDDs have enough information embedded in them describing how they are derived so they can be recomputed going all the back from the stable storage if needed.

Spark allows application programmers to control how these RDD's are partitioned and persisted based on use case. For an example a RDD that is needed by different application or rerun of the same application can choose to save it on disk.

On the other hand transient datasets or lookup tables are kept entirely in memory similarly datasets that are to be joined can be partitioned using same hash function so they are collocated.

Spark takes liberty to spill over partitions of RDDs that cannot fit in memory or completely destroy on heuristics that it can be trivially recomputed or not needed anymore.

3.1.1 Spark API

```
Lines = spark.textFile("hdfs://...")

errors = lines.filter(_.startsWith("ERROR")) errors.persist()

errors.count()

// Count errors mentioning MySQL:
errors.filter(_.contains("MySQL")).count()

// Return the time fields of errors mentioning
// HDFS as an array (assuming time is field
// number 3 in a tab-separated format):
errors.filter(_.contains("HDFS")).map(_.split('\t')(3)).collect()
```

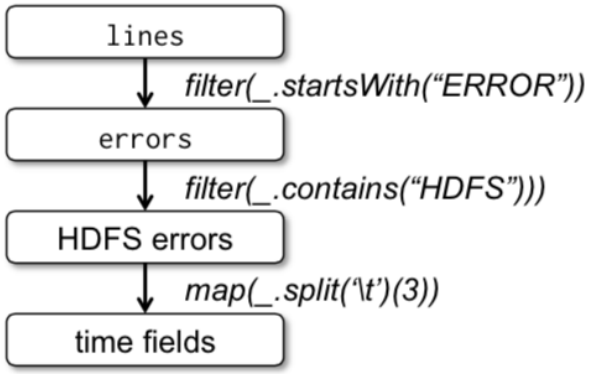


Figure 5. Lineage graph for the above query

3.2 Spark Application Model

Spark uses DSL like API similar to DryadLINQ [4] written in Scala, it also has binding for other popular languages like Java, Python, etc.,

Spark distributions come with Scala or Python based shells that can be used to run short scripts or execute interactive queries. Full-fledged applications can also be built using the one of the language bindings.

Transformations and Actions are two different sets of operations provided by Spark. Transformations are used to define RDDs these operations are lazy they are not realized until one of the terminal action is executed on such RDDs.

Table 1. Transformations and Actions

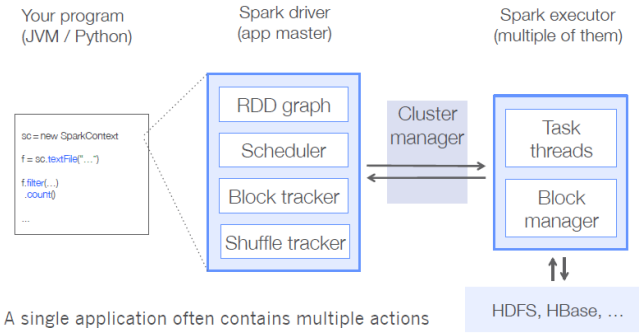
Transformations	Actions
map(*f* :T⇒U) *filter*(*f*:T⇒Bool) *flatMap*(*f* : T ⇒ Seq[U]) *sample*(*fraction* : Float) *groupByKey*() *reduceByKey*(*f* : (V, V) ⇒ V) *union*() *join*() *cogroup*() *crossProduct*() *mapValues*(*f* : V ⇒ W) *sort*(*c* : Comparator[K]) *partitionBy*(*p* : Partitioner[K])	*count*() *collect*() *reduce*(*f*:(T,T)⇒T): *lookup*(*k* : K) *save*(*path* : String)

3.3 Spark Runtime

Spark converts all transformations and terminal actions into a DAG and executes it using a DAG execution engine similar to that of Dryad.

Spark uses master-slave architecture similar to that of MapReduce to execute such DAG on the cluster. Spark application runtime consist of 2 major components one is called the Driver and the other is Executer. Driver coordinates a large number of Executers running on the cluster.

Spark Application is launched on the cluster using an external service called cluster manger these cluster managers are pluggable components. Currently Spark supports Hadoop YARN and Apache Mesas as cluster mangers.



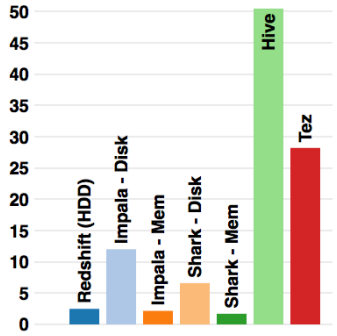
4. PERFORMANCE BENCHMARK

Following benchmarks measure the response time for relational queries such as scans, aggregation and joins on the various clusters similar to Hadoop. These benchmarks were performed by AMP Labs on 3 different datasets of varying sizes. Further information about the benchmark can be found at: https://amplab.cs.berkeley.edu/benchmark

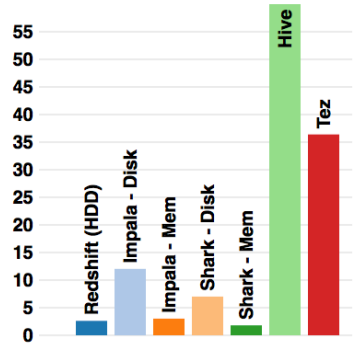
4.1 Scan Query

```
SELECT
        pageURL, pageRank
  FROM
        rankings
WHERE
        pageRank > X
```

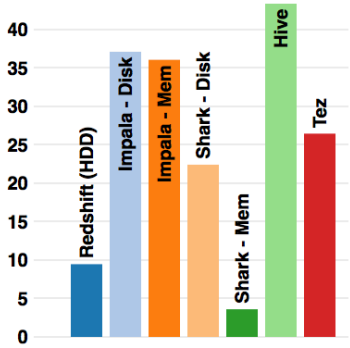
Query 1A
32,888 results



Query 1B
3,331,851 results



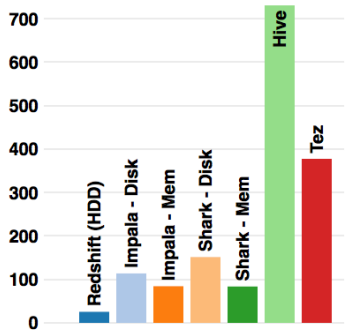
Query 1C
89,974,976 results



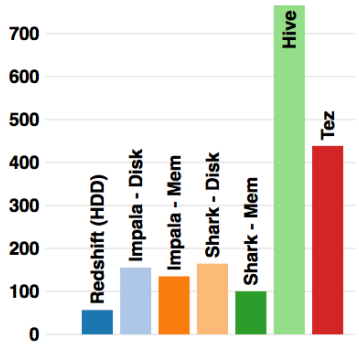
4.2 Aggregation Query

```
SELECT  SUBSTR(sourceIP, 1, X),  SUM(adRevenue)
  FROM uservisits
GROUP BY SUBSTR(sourceIP, 1, X)
```

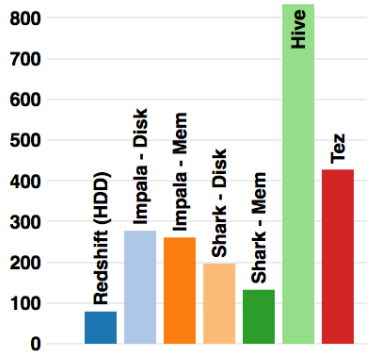
Query 2A
2,067,313 groups



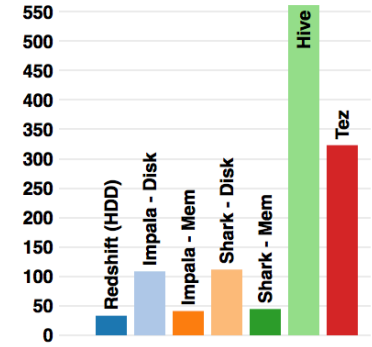
Query 2B
31,348,913 groups



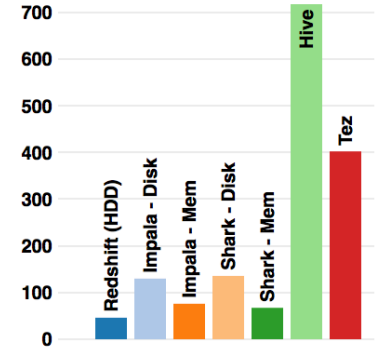
Query 2C
253,890,330 groups



Query 3A
485,312 rows



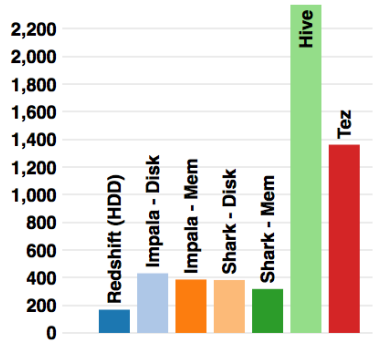
Query 3B
53,332,015 rows



4.3 Join Query

```
SELECT
        sourceIP,
        totalRevenue,
        avgPageRank
FROM
  (SELECT
            sourceIP,
            AVG(pageRank) as avgPageRank,
           SUM(adRevenue) as totalRevenue
    FROM
           Rankings AS R, UserVisits AS UV
  WHERE R.pageURL = UV.destURL
      AND UV.visitDate BETWEEN
              Date(`1980-01-01') AND Date(`X')
    GROUP BY UV.sourceIP)
  ORDER BY totalRevenue DESC LIMIT 1
```

Query 3C
533,287,121 rows



5. CONCLUSION

It is very clear from cited sources and the benchmarks that Apache Tez and Apache Spark provide significant advance over the Hadoop MapReduce framework in terms of performance, data source diversity and expressiveness of the API.

It is also very important to note that these advances were enabled by the advances in hardware as much as in the software that has enabled lower prices of memory modules, solid state drives and fast Ethernet.

6. REFERENCES

[1] T. White. Hadoop: The Definitive Guide, 3rd Edition http://techbus.safaribooksonline.com/book/software-engineering-and-development/9781449328917

[2] M. Isard, M. Budiu, Y. Yu, A. Birrel. Dryad: Distributed Data-Parallel Programs from Sequential Building Blocks.

[3] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M.J. Franklin, S. Shenker, I. Stoica. Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing.

[4] Y.Yu, M. Isard, D. Fetterly, M. Budiu, U´.Erlingsson, P.K. Gunda, and J. Currey. DryadLINQ: A system for general-purpose distributed data-parallel computing using a high-level language. In *OSDI '08*, 2008.