

YARN and PIG

Introduction

- ▣ Yet Another Resource Navigator
- ▣ Component that handles all the tasks running on the cluster.
- ▣ In 2013 when Hadoop 2.0 was released a fundamental change to Hadoop architecture was made.
- ▣ The MapReduce framework was divided into two
 - δ MapReduce
 - ∞ Responsible for defining what operation should be performed on data
 - δ Yarn
 - ∞ Responsible for determining how those processes need to be run and scheduled across the cluster.
- ▣ YARN is responsible for
 - δ Co-ordinating all tasks on all machines running on the cluster
 - δ Keeps track of all the resources across the cluster in terms of disk space, memory, CPU, etc.
 - δ Assigns new tasks to nodes based on the existing capacity.
 - ∞ In case of failure where node has failed and all the processes on that node has stopped, it will assign new nodes for that task, so processes can continue running.
- ▣ Internally, YARN is made up of two components which are daemons that run on different machines in the cluster.

Resource Manager	Node Manager
Master processor	Slave processes
Runs on a single master node	Run on all other nodes
Schedules tasks across nodes	Manages tasks on the individual node
Manages resources such as disk space, memory, CPU for all nodes in the cluster	Only looks at the node that it's running on.

Block Diagram


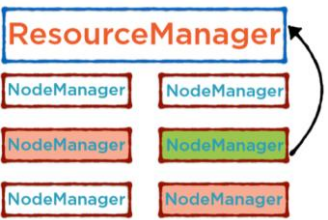
<ul style="list-style-type: none"> One resource manager that runs on the master node. 	
<ul style="list-style-type: none"> Multiple Node Managers that run on the slave nodes. 	

Submitting a Job

<ul style="list-style-type: none"> In this example the highlighted nodes already have jobs running on them. The resources on the highlighted nodes have been occupied by some long running process. 	
<ul style="list-style-type: none"> When a job is submitted to the cluster, it goes to the Resource Manager. The Resource Manager has a big-picture understanding of what resources are available on what nodes in the cluster. On the basis of this it will find a Node Manager on a node that has some capacity free and can accept this job. 	<p>Job</p> <p>Submitting a Job</p>

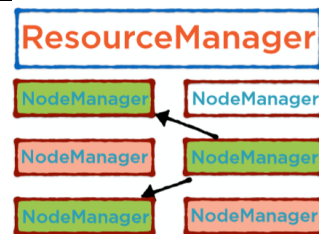
Application Master Process

<ul style="list-style-type: none"> This node manager runs a process or a job within something called a container. Container is not an actual container but is a logical component inside which the process runs. 	
<ul style="list-style-type: none"> All processes on a node are run within a container. A container is defined by resources. 	

<ul style="list-style-type: none"> How much CPU, how much memory, how much disk a particular job needs together forms a container inside which a process runs. 	
<ul style="list-style-type: none"> A new process is required to be spun off on a node, the resource request for that process is made in the form of containers. The task or the process has been assigned to this container. It's the responsibility of the container to run the task. A container executes that specific application 	
<ul style="list-style-type: none"> One Node Manager can have more than one container. <ul style="list-style-type: none"> It can have multiple processes running on the node. After a container has been assigned on a Node Manager. The Resource Manager starts off the Application Master within the Container. The application master process is responsible for performing the computation required for the task and actually processing the data. 	
<ul style="list-style-type: none"> In case of a MapReduce, the application master process will be a mapper process or the reduce logic. The application master process is also responsible for determining whether additional processes are required to complete the task. It also checks whether there are other mapper or reducer processes which need to be run so that the entire objective can be accomplished. 	
<ul style="list-style-type: none"> If additional resources are required, the Application Master makes the request to the Resource Manager running on the master node for additional resources. These additional resources are in the form of containers. The node manager running the application master process requests <ul style="list-style-type: none"> Containers for new mappers and reducers that are required to run to accomplish the task. The request includes CPU requirements, memory requirements and disk space requirements. 	

- ▣ The Resource Manager scans the entire cluster and determines which nodes are available, which nodes have capacity free so that these additional processes can be run.
- ▣ An individual Node Manager will never have this information.
- ▣ A Node Manager knows only about itself.
- ▣ It is the resource manager that knows about all nodes in the cluster.
- ▣ The same job is now given to additional resources in the form of additional nodes.

- ▣ The original application master process which made the request for the additional nodes, then starts off new application master processes on these new nodes that have been assigned by the Resource Manager.
- ▣ The Node Managers and the Resource Manager communicate with each other and work in tandem to accomplish parallel processing.



The Location Constraint

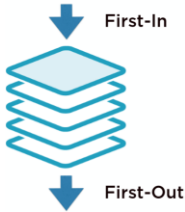
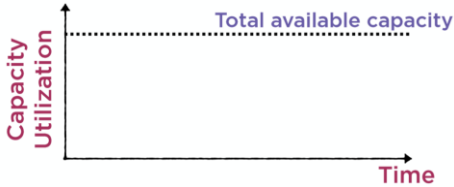
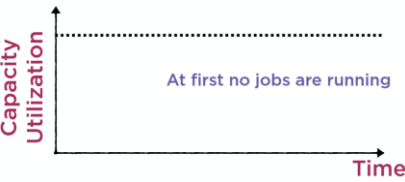
- ▣ Example
 - δ YARN, the resource negotiator receives a MapReduce job that it has to schedule on the Hadoop cluster.
 - δ It has to run a bunch of mapper processes.
- ▣ How does it determine where to run the mapper, on what nodes should the mapper be run.
- ▣ It uses the location constraint to determine where the mapper process should be located.
- ▣ An efficient use of the cluster resources is when there is a
 - δ Minimization of write bandwidth i.e. assign a process to the same node where're the data to be processed lives
 - ∞ There is no need for a process on one node to write to another node on the other side of the cluster.
 - ⇒ This chokes up the interconnectivity that exists between the machines.
 - ∞ The partition on which the mapper process has to be run should be co-located.
- ▣ The overhead of having to copy or read data across machines in the cluster should be avoided.
- ▣ Example
 - δ There is a node which has the data on which a mapper process needs to run.

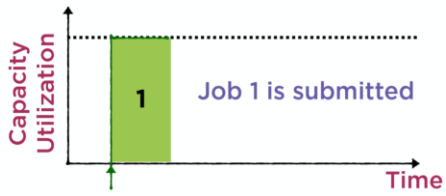
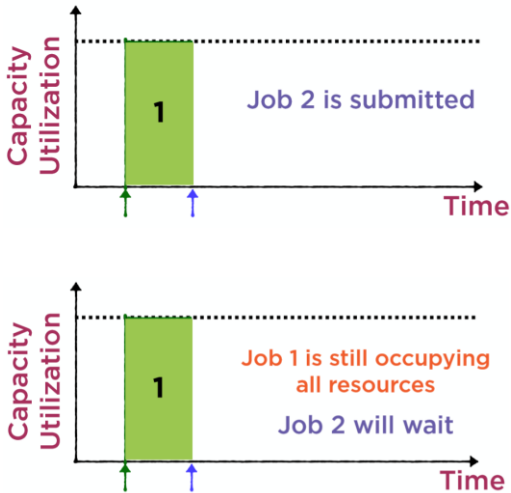
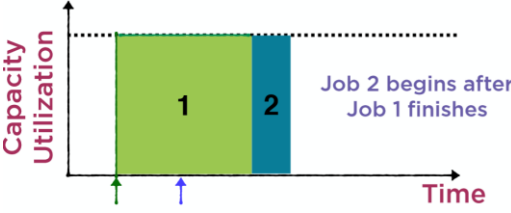
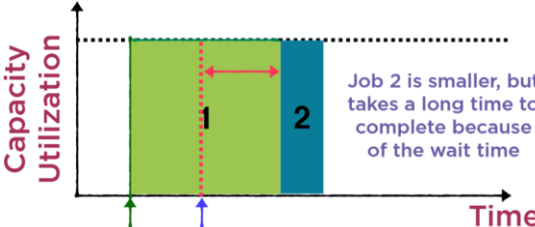
- δ There may other processes running on that node and there is no excess or spare capacity for these new mappers to be started up.
- ▢ If CPU and memory are not available, typically the process will wait.

Scheduling Policies

- ▢ YARN has very specific scheduling policies for all the jobs that it runs.
- ▢ These scheduling policies are strategies or algorithms which determine how a task is assigned to the cluster.
 - δ FIFO Scheduler
 - δ Capacity Scheduler
 - δ Fair Scheduler

FIFO Scheduler


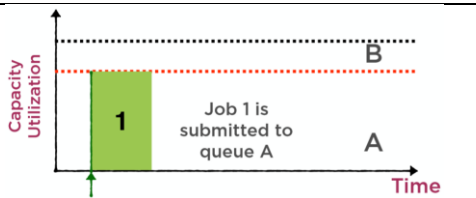
<ul style="list-style-type: none"> ▢ First in First Out ▢ A job that is first submitted to the cluster gets priority and is run on the cluster. ▢ It takes up all available resources. ▢ If any new job comes in, it has to wait until the first job runs through to completion. ▢ Once the first job is complete, only then is the second job scheduled. 	
<ul style="list-style-type: none"> ▢ The X axis represents the time that has passes. ▢ The Y axis represents the capacity of the cluster that has been utilized by any job. ▢ The total available capacity of the cluster is the dotted line in the graph. 	
<ul style="list-style-type: none"> ▢ The cluster is started with no jobs running till time T. 	

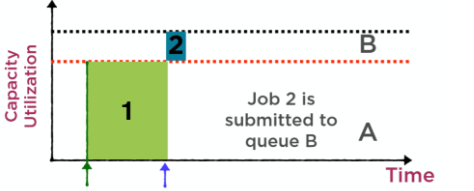
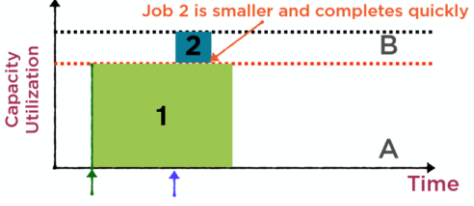
<ul style="list-style-type: none"> At some time T, a job is submitted on the cluster. It takes up all the resources in the cluster. This job occupies all the resources in the cluster and continues running and processing data. 	
<ul style="list-style-type: none"> It may happen that Job 2 is then submitted at some time T₁. This Job2 requires resources of the cluster but there are no free resources available as Job1 is still occupying all the resources. Job2 then has to wait until the cluster becomes free and it can start running. 	
<ul style="list-style-type: none"> Job1 will complete running. Once it releases the resources Job2 will start. It's totally possible that Job2 was actually a very short job. It was processing only a few megabytes of data, not billions and billions of records. 	
<ul style="list-style-type: none"> Even though Job2 was much smaller, it takes a very long time to complete. The additional time was just wait time waiting for the longer Job1 to complete and release the resources of the cluster. 	
<ul style="list-style-type: none"> FIFO scheduler is not a great scheduling technique. FIFO scheduler is rarely used It results in huge waiting time. 	

- δ Even smaller processes end up waiting in queue behind long running processes that may be less important.

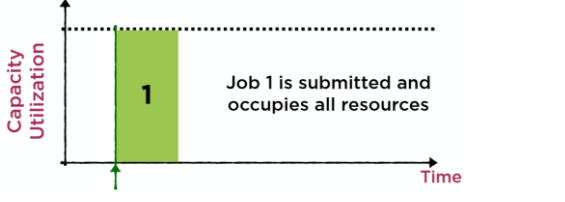
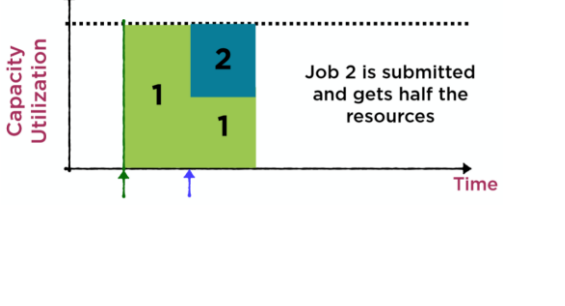
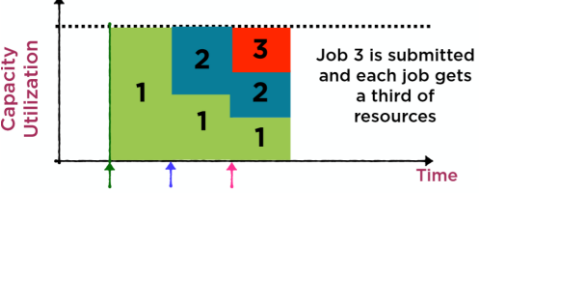
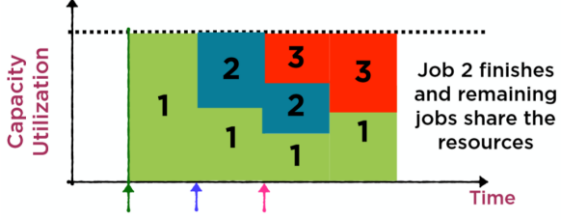
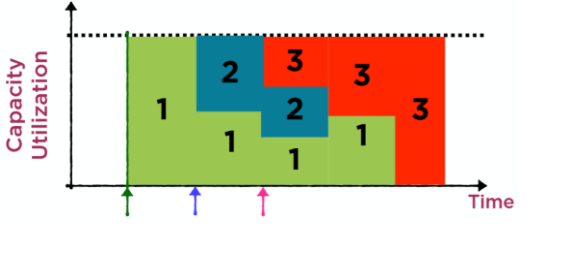
Capacity Scheduler

- ▢ A better alternative to the FIFO scheduling policy which has very long wait time is the Capacity Scheduler.
- ▢ The Capacity scheduler splits up the resources of the entire cluster into different queues.
 - δ The capacity of the cluster is distributed among these queues.
 - ∞ Each queue is containing some memory, some disk space, some CPU processing time.
 - ∞ Each queue is allocated a share of the cluster resources.
- ▢ It can also be thought of as some data nodes will be allocated exclusively to each queue.
 - δ Example
 - ∞ The queue used for all email marketing job should have 30% of the resources.
 - ∞ The queue used for searched jobs should have 70% of the resources.
 - δ Jobs can be submitted to a specific queue.
 - δ Within a queue, FIFO scheduling is used
 - δ One email mail marketing job is not more than another, but searched jobs have higher capacity therefore they are important than email marketing jobs.

<ul style="list-style-type: none"> ▢ Assume that the total capacity of the cluster is actually 100% and represented by dotted lines. ▢ Capacity is going to be divided into two queues, one queue with more resources and another queue with fewer resources. ▢ Queue A which is more important and has more resources. ▢ Queue B is less important and has less number of resources. 	
<ul style="list-style-type: none"> ▢ These resources can only be used for jobs just submitted to that queue. ▢ So queue A cannot poach from queue B's resources and vice versa. ▢ Job1 comes in at some time T_1 and is submitted to queue A. 	

<ul style="list-style-type: none"> Job1 finds all the capacity of queue A available to it and takes up the entire capacity and starts processing data. 	
<ul style="list-style-type: none"> At some point in time T_2, Job2 comes in but it is submitted to Queue B. Queue B has no other jobs running, its capacity is completely free and available for Job2. Job2 starts its processing immediately alongside with Job1. 	
<ul style="list-style-type: none"> Job2 and Job1 progress in parallel. No job is held up because of the other because they're on separate queues. At some point, Job2 being a shorter job will complete much faster than Job1. Job2 does not wait for Job1. <ul style="list-style-type: none"> This is the advantage of capacity scheduling. 	
<ul style="list-style-type: none"> When there are queues that are allocated with different kinds of jobs, Jobs of particular category will not be held up because of jobs in another category. <ul style="list-style-type: none"> If there are smaller jobs in a particular category, they won't get stuck because of long-running ones. The disadvantage is that the cluster might be underutilized if some queues are not occupied. <ul style="list-style-type: none"> If only one job was submitted on one queue it won't use all the resources of the queue. The drawback of the Capacity Scheduler – underutilization of cluster resources where there isn't a job in every queue on the cluster. Resources are always proportionally allocated to all jobs. Any job that is submitted to the cluster is immediately started and the resources available are split across all the jobs currently running. <ul style="list-style-type: none"> This implies that there is absolutely zero wait time for any job 	

Fair Scheduler

<ul style="list-style-type: none"> When the first job, Job1 is submitted, it occupies all resources. There is complete utilization of the cluster. 	
<ul style="list-style-type: none"> At some point in time, Job2 might be submitted to the cluster. However long the job is or whatever it is, it gets half the resources. At this point Job1 and Job2 share the resources 50-50. 	
<ul style="list-style-type: none"> It might be that Job3 is submitted to the cluster. There are three jobs running. Each get an equal share of the cluster resources. No job submitted to the cluster has to wait. 	
<ul style="list-style-type: none"> If one of the job completes its processing, then its resources are taken over in equal shares by the remaining jobs. In this case Job1, Job3 will continue. 	
<ul style="list-style-type: none"> If Job1 gets completed, Job3 occupies all the resources in the cluster. Based on the requirements and the nature of the jobs run on the cluster any of the three schedulers can be configured. 	

Configuring the Scheduling Policy

- Default Policy that Hadoop comes installed with is Capacity Scheduling.
- To set the actual scheduling policy put it in yarn-site.xml.
- If there is nothing specified, it's Capacity scheduling.

```
<configuration>
<property>
  <name>yarn.resourcemanager.scheduler.class</name>
  <value>
    org.apache.hadoop.yarn.server.resourcemanager.scheduler.fair.FairScheduler
  </value>
</property>
</configuration>
```

- ▢ If it needs to be switched over to the Fair Scheduler, the yarn.resourcemanager.scheduler.class has to be set.
- ▢ The value of this property is the scheduling algorithm.

```
<configuration>
<property>
  <name>yarn.resourcemanager.scheduler.class</name>
  <value>
    org.apache.hadoop.yarn.server.resourcemanager.scheduler.fair.FairScheduler
  </value>
</property>
</configuration>
```

```
<configuration>
<property>
  <name>yarn.resourcemanager.scheduler.class</name>
  <value>
    org.apache.hadoop.yarn.server.resourcemanager.scheduler.fair.FairScheduler
  </value>
</property>
</configuration>
```

DEMO - Capacity Scheduler

- ▢ Configuration
- ▢ Open capacity-scheduler.xml in hadoop-2.9.1/etc/hadoop.

```
<property>
  <name>yarn.scheduler.capacity.root.queues</name>
  <value>default</value>
  <description>
    The queues at the this level (root is the root queue).
  </description>
</property>
```

- ▢ This property is used to define a list of queues.

- ▢ Here there is only one queue in the list, with the queue named “default”.
- ▢ This one queue is the default configuration that Hadoop has when it’s installed.
- ▢ To specify the capacity of resources that is allocated to this queue, use the property.

```
<property>
  <name>yarn.scheduler.capacity.root.default.capacity</name>
  <value>100</value>
  <description>Default queue target capacity.</description>
</property>
```

- δ It is the percentage capacity assigned to the default queue.
 - ∞ 100% of the cluster capacity is given to this queue.
- ▢ The word ‘default’ is highlighted in the property name.
- δ This shows that it is for the default queue of the cluster.
- ▢ While specifying the capacity for a different queue, change that word to be the new queue name.
- ▢ Configure two queues on YARN
 - δ Dev queue – for development jobs that do not require too many resources
 - δ Prod queue – for production jobs that require large amount of resources
- ▢ Specify the queue names as comma separated list in the queues.

```
<property>
  <name>yarn.scheduler.capacity.root.queues</name>
  <value>dev, prod</value>
  <description>
    The queues at the this level (root is the root queue)
  </description>
</property>
```

- ▢ To assign capacities for the queues.
- δ The root.dev capacity will be 30%.


```
<property>
  <name>yarn.scheduler.capacity.root.dev.capacity</name>
  <value>30</value>
  <description>Development Queue target capacity.</description>
</property>
```
 - δ Copy and paste same property in order to configure the capacity for the production.
 - δ The root.prod capacity will be 70%.

```
<property>
  <name>yarn.scheduler.capacity.root.prod.capacity</name>
  <value>70</value>
  <description>Production queue target capacity.</description>
</property>
```

▢ Ensure that the sum total of the queue capacities is a 100%.

▢ Save the file.

```
C:\BigData>hadoop jar WC.jar PackageDemo.WordCount -D mapreduce.job.queueName=prod /input/Sample.txt /output2/WCountt
```

▢ Click on Job Id.

The screenshot shows the Hadoop Distributed File System (HDFS) web interface. The browser address bar shows 'localhost:8088/cluster'. The page displays various metrics and a table of running jobs.

Cluster Metrics

Apps Submitted	Apps Pending	Apps Running	Apps Completed	Containers Running
1	0	0	1	0

Cluster Nodes Metrics

Active Nodes	Decommissioning Nodes	Decommissioned Nodes
1	0	0

Scheduler Metrics

Scheduler Type	Scheduling Resource Type	Minimum Resource
Capacity Scheduler	[MEMORY]	<memory:1024, vCore

Show 20 entries

ID	User	Name	Application Type	Queue	Application Priority	StartTime	FinishTime
application_1569308241284_0001	raoal	word count	MAPREDUCE	prod	0	Tue Sep 24 13:24:37 +0550 2019	Tue Sep 24 13:25:42 +0550 2019

▢ This page shows that the job is running on the 'prod' queue.

▢ Clicking on the queue name will take it to another page.

User: raoal
Name: word count
Application Type: MAPREDUCE
Application Tags:
Application Priority: 0 (Higher Integer value indicates higher priority)
YarnApplicationState: FINISHED
Queue: prod
FinalStatus Reported by AM: SUCCEEDED
Started: Tue Sep 24 13:24:37 +0530 2019
Elapsed: 1mins, 5sec
Tracking URL: History
Log Aggregation Status: DISABLED
Application Timeout (Remaining Time): Unlimited
Diagnostics:
Unmanaged Application: false
Application Node Label expression: <Not set>
AM container Node Label expression: <DEFAULT_PARTITION>

▢ This shows all the jobs scheduled for the queue.



NEW,NEW_SAVING,SUBMITTED,ACCEPTED,RUNNING Applications

Cluster Metrics

Apps Submitted	Apps Pending	Apps Running	Apps Completed	Containers Running	Memory Used	Memory Total	Memory Reserved	VCores Used	VCores Total	VCores Available
1	0	0	1	0	0 B	8 GB	0 B	0	8	0

Cluster Nodes Metrics

Active Nodes	Decommissioning Nodes	Decommissioned Nodes	Lost Nodes	Unhealthy Nodes	Rebooted Nodes	Shut Down Nodes
1	0	0	0	0	0	0

Scheduler Metrics

Scheduler Type	Scheduling Resource Type	Minimum Allocation	Maximum Allocation	Maximum Cluster Application
Capacity Scheduler	[MEMORY]	<memory:1024, vCores:1>	<memory:8192, vCores:4>	0

Dump scheduler logs1 min

Application Queues

Legend:

CapacityUsedUsed (over capacity)Max CapacityUsers Requesting Resources

Queue: root0.0% used

Queue: dev0.0% used

Queue: prod0.0% used

Show20entries

Search:

ID	User	Name	Application Type	Queue	Application Priority	StartTime	FinishTime	State	FinalStatus	Running Containers	Allocated CPU VCoers	Allocated Memory MB	Reserved CPU VCoers	Reserved Memory MB	% of Queue	% of Cluster	Progress	Track UI
No data available in table																		

Showing 0 to 0 of 0 entries

Aggregate scheduler counts

PIG

Introduction

- ▣ Pig is a powerful tool that allows developers to write MapReduce jobs.
- ▣ Pig Latin is Pig's language that allows developers to express data flows and describe how data will be transformed.
- ▣ Pig Latin is the language that interacts with the Pig Tool.
- ▣ Pig is an open source engine, that is a part of the Hadoop ecosystem.
- ▣ It is great at working with data that are beyond traditional databases or data warehouses.
- ▣ It can deal well with missing, incomplete or inconsistent data that has no schema.
- ▣ It has its own language for expressing data manipulations.
 - δ It is called as Pig Latin.
- ▣ It is an application environment used to run Pig Latin and convert Pig Latin scripts into MapReduce Jobs.
- ▣ It allows exploration of large datasets without having to write a MapReduce job in Java or Python or Ruby.
- ▣ Pig and Hive sits on top of MapReduce.
- ▣ Each use their own language to write the jobs.
 - δ Pig uses Pig Latin, Hive uses HiveQL.
- ▣ Pig is a technology that allows the developer to write high level scripts that allows to work with data with unknown or inconsistent schema.
- ▣ It is open source technology, a part of the Hadoop eco-system and runs on top of Hadoop
- ▣ It works well with unstructured, incomplete data.
- ▣ Pig can work directly on files in HDFS.
- ▣ Pig is used to get data into a data warehouse.
- ▣ When a Pig Latin script is written, Pig converts it into a MapReduce job without the developer having to write a MapReduce job.
- ▣ The Pig Latin script can then be run over the entire Hadoop cluster
- ▣ A procedural, data flow language to extract, transform and load data

Why pig over MapReduce

- ▣ Pig was developed because writing a MapReduce job takes a long time.
- ▣ Pig has a smaller number of lines of codes.
- ▣ Developers quickly test their queries.
 - δ Pigs have a command line tool called grunt shell.
- ▣ There is no need for Java while writing a MapReduce job.
- ▣ In Java a minimum of five different MapReduce libraries have to be imported.
- ▣ It is about 135 lines of codes.
- ▣ The same in Pig Latin would be around 7 lines of code.

```

input_lines = LOAD '/tmp/word.txt' AS (line:chararray);
words = FOREACH input_lines GENERATE FLATTEN(TOKENIZE(line)) AS word;
filtered_words = FILTER words BY word MATCHES '\\w+';
word_groups = GROUP filtered_words BY word;
word_count = FOREACH word_groups GENERATE COUNT(filtered_words) AS count, group AS word;
ordered_word_count = ORDER word_count BY count DESC;
STORE ordered_word_count INTO '/tmp/results.txt';

```

- ▣ There is no worry of importing different MapReduce libraries.
- ▣ The syntax of Pig Latin is very similar to SQL.

	foreach	select	sum(revenue)
(group	revenues	by dept)	
	generate	from	revenues
sum(revenue)		group by	dept

Pig vs. SQL

PIG	SQL
A data flow language, transforms data to store in a warehouse	A query language, is used for retrieving results
Specifies exactly how data is to be modified at every step	Abstracts away how queries are executed
Purpose of processing is to store in a queryable format	Purpose of data extraction is analysis
Used to clean data with inconsistent or incomplete schema	Extract insights, generate reports, drive decisions

History of Pig

- ▣ Pig started out at Yahoo.
 - δ They needed a technology that helped to process terabytes of data as quickly as possible.
 - δ It was developed in 2006.
 - δ It was adopted by Apache in 2010.

Pig Latin

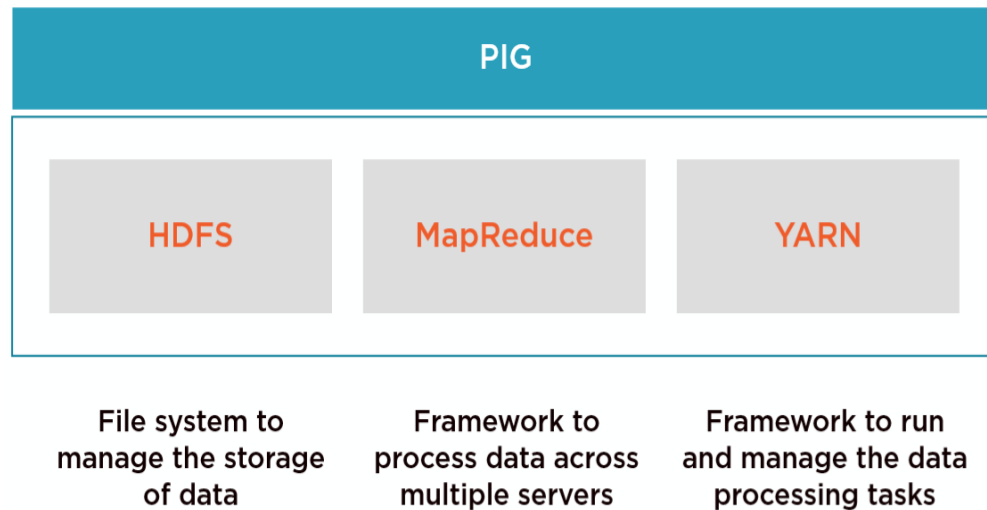
- ▣ Every line in the script can be thought as a part of transformation of an existing dataset to a form which is cleaner and in a better state.
- ▣ Series of well-defined steps to perform operations to get data in a useful form.
- ▣ There is no if statements or for loops in Pig.
- ▣ Data in its raw form is fed into the script.
- ▣ The data goes through a series of transformations applied to the data, aggregations, cleaning up, filling in missing fields, leaving out fields that are not important and then gets its final form.
 - δ Pig should be written with these data operations in mind.
- ▣ Data from one or more sources can be read, processed and stored in parallel since it works on top of Hadoop.
- ▣ Cleans data, precomputes common aggregates before storing in a data warehouse

PIG Installation

- ▣ Download the latest release from <https://pig.apache.org/>
 - δ The latest release is 0.17.0
- ▣ Extract the pig-0.17.0 tar.gz in a folder.
- ▣ Configure the path and the environment variable
 - δ Variable name – PIG_HOME
 - δ Variable value - <Path of Pig directory>
 - δ Add the path of pig to the system variable 'Path'
- ▣ To check if Pig has been installed or not
 - δ Go to command prompt and type pig -version.
 - ∞ It should give the version number of Pig installed.

PIG - Execution

- ▣ Pig runs on top of the Hadoop distributed computing framework and takes advantage of HDFS, MapReduce and YARN in order to run its processing tasks.



- ▣ The raw data that is fed into Pig lies in the form of files on HDFS.
- ▣ Any intermediate files that Pig generates is stored in HDFS
- ▣ Pig writes its final output to HDFS
- ▣ Pig uses MapReduce to decompose operations into MapReduce jobs which run in parallel
 - δ Pig scripts are converted to MapReduce jobs by Pig, that are distributed across the Hadoop cluster.
- ▣ Pig has a huge library that provides non-trivial, built-in implementations of standard data operations, which are very efficient.
- ▣ Developers using Pig do not have to reinvent the wheel for common operations.
- ▣ Pig optimizes operations before MapReduce jobs are run, to speed operations up
- ▣ Pig can be configured not only to run on top of Hadoop, but can also be used with other technologies.
 - δ Apache Tez and Apache Spark
- ▣ **Tez**
 - δ It is a framework that extends the MapReduce framework as provided by Hadoop by making the MapReduce operations faster and more
- ▣ **Spark**
 - δ Spark is another distributed computing technology which is scalable, flexible and fast.
 - ∞ It is fast because it performs most of its operations in memory, rather than use intermediate files on disk.

Running Pig

- ▣ Pig has six execution modes or exectypes:

δ **Local Mode**

- ∞ Runs on a single machine
 - ⇒ It doesn't require cluster of machines or a distributed
- ∞ Files are set up and run using the local file system
 - ⇒ Input files, intermediate results and final result will all be stored on the local file system.
- ∞ This mode is very limited in its capacity to crunch the data.
 - ⇒ It is not used in production system
- ∞ It is used for rapid prototyping and general debugging.
- ∞ Specify local mode using the -x flag
 - ⇒ pig -x local

δ **Tez Local Mode**

- ∞ To run Pig in tez local mode.
- ∞ It is similar to local mode, except internally Pig will invoke Tez runtime engine.
- ∞ Runs on a single machine
- ∞ Files use the local file system
- ∞ Uses the Tez runtime environment rather than MapReduce
- ∞ This is used for rapid prototyping, development and debugging.
- ∞ Specify Tez local mode using the -x flag
 - ⇒ pig -x tez_local

δ **Spark Local Mode**

- ∞ To run Pig in spark local mode.
- ∞ It is similar to local mode, except internally Pig will invoke spark runtime engine.
- ∞ Specify Spark local mode using the -x flag
 - ⇒ pig -x spark_local

δ **Mapreduce Mode**

- ∞ To run Pig in mapreduce mode, access to a Hadoop cluster and HDFS installation is needed.
- ∞ Runs on a Hadoop cluster with HDFS installed
- ∞ Pig can be installed on a single machine which connects to the cluster, this is the gateway machine.
- ∞ Used in a production environment.
- ∞ Mapreduce mode is the default mode
 - ⇒ pig
 - ⇒ pig -x mapreduce

δ Tez Mode

- ∞ To run Pig in Tez mode, access to a Hadoop cluster and HDFS installation is needed.
- ∞ Runs on a Hadoop cluster with HDFS installed
- ∞ Uses the Tez runtime environment rather than MapReduce
- ∞ Used in a production environment.
- ∞ Specify Tez mode using the -x flag
 - ⇒ -x tez

δ Spark Mode

- ∞ To run Pig in Spark mode, access to a Spark, Yarn or Mesos cluster and HDFS installation is needed.
- ∞ Specify Spark mode using the -x flag
 - ⇒ -x spark

Interactive and Batch Modes

▢ REPL Environment

- δ Read, Evaluate, Print, Loop Environment
- δ It is an interactive environment
- δ Using the command line to type out individual commands and get immediate feedback as a result of this command

▢ Pig Scripts

- δ Create a script file with commands and run them in one go

Running PIG

▢ Open a cmd prompt and type

- δ pig -x local

```
C:\WINDOWS\system32>pig -x local
19/09/30 20:54:02 INFO pig.ExecTypeProvider: Trying ExecType : LOCAL
19/09/30 20:54:02 INFO pig.ExecTypeProvider: Picked LOCAL as the ExecType
2019-09-30 20:54:03,494 [main] INFO org.apache.pig.Main - Apache Pig version 0.17.0 (r1797386) compiled Jun 02 2017, 15:41:58
2019-09-30 20:54:03,494 [main] INFO org.apache.pig.Main - Logging error messages to: C:\BigData\hadoop-2.9.1\logs\pig_1569857043490.log
2019-09-30 20:54:04,085 [main] INFO org.apache.pig.impl.util.Utils - Default bootup file C:\Users\raoal/.pigbootup not found
2019-09-30 20:54:06,119 [main] INFO org.apache.hadoop.conf.Configuration.deprecation - mapred.job.tracker is deprecated. Instead, use mapreduce.jobtracker.address
2019-09-30 20:54:06,124 [main] INFO org.apache.pig.backend.hadoop.executionengine.HExecutionEngine - Connecting to hadoop file system at: file:///
2019-09-30 20:54:07,413 [main] INFO org.apache.hadoop.conf.Configuration.deprecation - io.bytes.per.checksum is deprecated. Instead, use dfs.bytes-per-checksum
2019-09-30 20:54:07,500 [main] INFO org.apache.pig.PigServer - Pig Script ID for the session: PIG-default-e95db997-2882-448e-88f1-9e5b13b81958
2019-09-30 20:54:07,502 [main] WARN org.apache.pig.PigServer - ATS is disabled since yarn.timeline-service.enabled set to false
grunt>
```

▢ This is Pig's REPL environment.

Commands

▢ clear

 δ To clear the screen.

▢ load

 δ To load data

```
grunt> A = load 'build.xml' using PigStorage();
2019-09-30 21:11:27,984 [main] INFO org.apache.hadoop.conf.Configuration.deprecation - io.bytes.per.checksum is deprecated. Instead, use dfs.bytes-per-checksum
```

▢ dump

 δ To see the file that is loaded using the previous command

 ∞ dump <var>

▢ To remove the warnings and messages.

 δ Go to the conf directory of the Pig folder.

```
C:\Pig\pig-0.17.0\conf>dir
Volume in drive C is OS
Volume Serial Number is C44E-6526

Directory of C:\Pig\pig-0.17.0\conf

25-09-2019  04.24 PM    <DIR>          .
25-09-2019  04.24 PM    <DIR>          ..
02-06-2017  07.12 PM              1,136 log4j.properties.template
02-06-2017  07.12 PM              27,539 pig.properties
02-06-2017  07.12 PM              1,110 test-log4j.properties
                3 File(s)              29,785 bytes
                2 Dir(s)  302,975,361,024 bytes free
```

 δ The log4j.properties.template is the default configuration for the Pig logs.

▢ Copy the contents of log4j.properties.template to another file called log4j.properties in order to update the log.

```

C:\Pig\pig-0.17.0\conf>dir
Volume in drive C is OS
Volume Serial Number is C44E-6526

Directory of C:\Pig\pig-0.17.0\conf

30-09-2019  09.31 PM    <DIR>          .
30-09-2019  09.31 PM    <DIR>          ..
02-06-2017  07.12 PM                1,136 log4j.properties
02-06-2017  07.12 PM                1,136 log4j.properties.template
02-06-2017  07.12 PM               27,539 pig.properties
02-06-2017  07.12 PM                1,110 test-log4j.properties
               4 File(s)                30,921 bytes
               2 Dir(s)  302,970,929,152 bytes free

```

▢ Open the file to edit it.

```

# ***** Set root logger level to DEBUG and its only appender to A.
log4j.logger.org.apache.pig=info, A

```

δ A refers to the appender that is used along with the logs.

```

# ***** A is set to be a ConsoleAppender.
log4j.appender.A=org.apache.log4j.ConsoleAppender

```

δ A is an alias for the log4j.ConsoleAppender.

∞ An appender is a component that appends log messages to system.error or system.out.

▢ Edit the files so that only error messages are sent to A.

```

# ***** Set root logger level to DEBUG and its only appender to A.
log4j.logger.org.apache.pig=error, A
log4j.logger.org.apache.hadoop = error, A

```

▢ Restart pig with

δ pig -x local -4 conf/log4j.properties to get rid of the underlying messages.

```
C:\Pig\pig-0.17.0>pig -x local -4 conf/log4j.properties
19/09/30 21:54:35 INFO pig.ExecTypeProvider: Trying ExecType : LOCAL
19/09/30 21:54:35 INFO pig.ExecTypeProvider: Picked LOCAL as the ExecType
19/09/30 21:54:35 INFO pig.Main: Loaded log4j properties from file: conf\log4j.properties
```

δ Use the load function

```
grunt> A = load 'build.xml' using PigStorage();
```

δ There are no log messages now.

▢ Use dump A;

δ The content of this file will be seen without the info messages.

Pig – Batch Mode

▢ Keep the extension of a pig file as **.pig**.

▢ Open an editor and save the following content in a file named as test.pig

```
/*
This is the first Pig Script.
Using Multiline comments
*/
--Single line comment
A = load 'build.xml' using PigStorage(); -- loading data
dump A; -- printing results to screen
```

▢ Go to the command prompt and run the following command

```
C:\Pig\pig-0.17.0>pig -x local -4 conf/log4j.properties test.pig
19/10/01 21:00:18 INFO pig.ExecTypeProvider: Trying ExecType : LOCAL
19/10/01 21:00:18 INFO pig.ExecTypeProvider: Picked LOCAL as the ExecType
19/10/01 21:00:18 INFO pig.Main: Loaded log4j properties from file: conf\log4j.properties
(<!--)
( Licensed to the Apache Software Foundation (ASF) under one or more)
( contributor license agreements. See the NOTICE file distributed with)
( this work for additional information regarding copyright ownership.)
( The ASF licenses this file to You under the Apache License, Version 2.0)
( (the "License"); you may not use this file except in compliance with)
```

▢ When this command is run, Pig will invoke the Grunt shell, run the commands in the script on that Grunt shell and exit the Grunt shell at the end of the script.

The Grunt Shell

▢ The Pig shell which runs commands in both interactive and batch modes

- ▣ Is invoked when the pig command is used in any mode of operation
- ▣ Can also be used as a shell to run HDFS commands

Executing Pig Commands

- ▣ Write commands in Pig Latin on the Grunt shell or a script file
 - δ This is taken by Pig and converted behind the scenes to MapReduce jobs
- ▣ Pig will translate the commands to MapReduce tasks and run them on Hadoop
- ▣ MapReduce will process files on HDFS and return results to Pig

Expression

- ▣ Structure
 - δ All statements end with semicolon
 - δ All Pig Latin scripts are expressed as variables
 - δ It is a procedural language.
 - ∞ Instructions are written step by step.

DATA Types

- ▣ Numeric Types
 - δ Int 32-bit
 - δ Long 64-bit
 - δ Float 32-bit
 - ∞ Can hold 754 decimal places.
 - δ Double 64-bit
 - ∞ These are inherited from Java
- ▣ Text
 - δ Chararray Character string
- ▣ Date
 - δ datetime

Loading Data into Pig

- ▣ General structure of Pig Program
 - δ Load data from an external file into Pig.
 - ∞ This data is stored in a relation.
 - ⇒ Relations are variables in Pig designed to hold datasets on which operations are performed.

- δ Data that is stored in the relation is subject to a series of transformations that update this data to get the result.
- δ Store the data to file or display it to screen

Relations

- ▣ A dataset with a name is like variables in Java, Python.
- ▣ A relation holds data which is not a single entity but has one or more fields and values associated with these fields.
- ▣ Relations may or may not have a schema associated with them in Pig.
- ▣ Once relations are created cannot be edited.
 - δ Since they are immutable, updates to a relation creates a new relation.
 - δ The original relation remains unchanged.
- ▣ Relations exist for the duration of a single Pig session.
 - δ They are ephemeral and held in memory until the Pig script is completed or the resultset is stored in a file.

Transformations

- ▣ Operations transform data and create new relations.
- ▣ Once data is loaded from a file, the relation will have all the records that exist in the file loaded in memory.
- ▣ Any operation that is performed to transform data on this relation creates a brand-new relation.
- ▣ The original relation remains unchanged.
- ▣ Imagine the transformations that are performed on relations to be chained together.
- ▣ Each transformation emits a new relation in this chain.
- ▣ In Pig none of the relations and the records within them are actually evaluated until the final results are displayed onscreen or stored in to a file.
- ▣ Each of these transformations are stored in memory.
- ▣ Final evaluation is done only when a store or dump command is encountered.
- ▣ This is called lazy evaluation and is an important feature that allows Pig to process huge datasets efficiently.
- ▣ As the records in each individual relation do not need to be evaluated until they are actually needed.

- δ This allows Pig to perform all kinds of optimizations behind the scenes before processing individual datasets.
- ▢ When Pig commands are written, remember that relations are immutable.
- ▢ Each time they are modified there is a need to store the result in a new relation.
- ▢ Every Pig script looks like this
 - δ `relation_1 = load data from file into Pig`
 - ∞ load the data from the file into Pig and store it in some relation `relation_1`
 - δ `relation_2 = pig latin commands to transform relation_1`
 - ∞ A bunch of commands are performed which act on `relation_1` and produce `relation_2`.
 - δ `relation_3 = pig latin commands to transform relation_2`
 - ∞ The previous relation gives rise to `relation_3` which gives rise to `relation_4` and so on.
 - δ `relation_4 = pig latin commands to transform relation_3`
 - ∞ store `relation_4` to file or display results to screen
- ▢ One relation feeds into another, feeds into the third and so on until the final result is got.

Case Sensitivity in Pig

- ▢ Case sensitive
 - δ Relation names
 - δ Field names within relations
 - δ Function names such as `PigStorage()`, `SUM()`, `COUNT()`
- ▢ Case insensitive
 - δ Keywords in Pig such as `load`, `store`, `foreach`, `generate`, `group by`, `order by`, `dump`

DEMO

- ▢ Load data into Pig using the `PigStorage()` function.
- ▢ Load from files as well as directories

```
grunt> groceries = load 'groceries.csv' using PigStorage(',');
2019-10-07 22:14:26,107 [main] INFO org.apache.hadoop.conf.Configuration.deprecation - io.bytes.per.checksum is deprecated. Instead, use dfs.bytes-per-checksum
```

```
(01, Vijayanagar, Bananas, 2019-01-01,7)
(02, Rajajinagar, Apples, 2019-01-02,20)
(03, Jayanagar, Flowers, 2019-01-02,10)
(04, Govindarajanagar, Meat, 2019-01-03,40)
(05, Vijayanagar, Potatoes, 2019-01-04,9)
(06, Jayanagar, Bread, 2019-01-04,5)
(07, Govindarajanagar, Bread, 2019-01-05,5)
(08, Banashankari, Onion, 2019-01-05,4)
(09, Govindarajanagar, Cheese, 2019-01-05,15)
(10, Banashankari, Onion, 2019-01-06,4)
(11, Malleshwaram, Bread, 2019-01-05,5)
(12, Banashankari, Onion, 2019-01-07,4)
(13, Chamarajpet, Bread, 2019-01-07,5)
(14, Banashankari, Tomato, 2019-01-07,6)
```

▢ Load data into Pig using the PigStorage() function.

```
grunt> groceries = load 'groceries.csv' using PigStorage(',');
2019-10-07 22:14:26,107 [main] INFO org.apache.hadoop.conf.Configuration.deprecation - io.bytes.per.checksum is deprecated. Instead, use dfs.bytes-per-checksum
```

▢ PigStorage is a built-in function Hive offers to pass files on disk.

▢ The default delimiter that PigStorage expects in a file is a 'tab'.

▢ Here comma is used as a delimiter since the file loaded is a comma separated file.

▢ The load statement is followed by the keyword 'as' and then the schema.

```
grunt> groceries = load 'groceries.csv' using PigStorage(',')
>> as_
>> (
>> order_id: chararray,
>> location: chararray,
>> product: chararray,
>> day: datetime,
>> revenue: double
>> );
```

δ The order_id, location and product are all simple strings.

δ The date on which the order was made is of the type datetime.

δ The revenue is of the type double.

- Applying the dump command will give the details as given below

```
(01, Vijayanagar, Bananas,,7.0)
(02, Rajajinagar, Apples,,20.0)
(03, Jayanagar, Flowers,,10.0)
(04, Govindarajanagar, Meat,,40.0)
(05, Vijayanagar, Potatoes,,9.0)
(06, Jayanagar, Bread,,5.0)
(07, Govindarajanagar, Bread,,5.0)
(08, Banashankari, Onion,,4.0)
(09, Govindarajanagar, Cheese,,15.0)
(10, Banashankari, Onion,,4.0)
(11, Malleshwaram, Bread,,5.0)
(12, Banashankari, Onion,,4.0)
(13, Chamarajpet, Bread,,5.0)
(14, Banashankari, Tomato,,6.0)
(,,, )
(,,, )
(,,, )
```

- Use describe command

```
grunt> describe groceries;
groceries: {order_id: chararray,location: chararray,product: chararray,day: datetime,revenue: double}
```

- Foreach iterates over every record in the relation and the generate statement chooses certain fields of the relation that the programmer is interested in.
- Here in the code the field accessed is at index 1 i.e. the second field.
- The result is stored in locations relation.

```
grunt> locations = foreach groceries generate $1;
```

- Each record in this relation is a tuple with one field.

```
grunt> dump locations;
( Vijayanagar)
( Rajajinagar)
( Jayanagar)
( Govindarajanagar)
( Vijayanagar)
( Jayanagar)
( Govindarajanagar)
( Banashankari)
( Govindarajanagar)
( Banashankari)
( Malleshwaram)
( Banashankari)
( Chamarajpet)
( Banashankari)
```

- Use two indices to generate a record.

```
grunt> prod_revenue = foreach groceries generate $2, $4;
```

```
grunt> dump prod_revenue;
( Bananas,7.0)
( Apples,20.0)
( Flowers,10.0)
( Meat,40.0)
( Potatoes,9.0)
( Bread,5.0)
( Bread,5.0)
( Onion,4.0)
( Cheese,15.0)
( Onion,4.0)
( Bread,5.0)
( Onion,4.0)
( Bread,5.0)
( Tomato,6.0)
```

- To create a subset of the groceries data, use limit

```
grunt> groc_sub = limit groceries 5;
```

```
grunt> dump groc_sub;
(01, Vijayanagar, Bananas,,7.0)
(02, Rajajinagar, Apples,,20.0)
(03, Jayanagar, Flowers,,10.0)
(04, Govindarajanagar, Meat,,40.0)
(05, Vijayanagar, Potatoes,,9.0)
```

▢ To store these relations onto the local machines in a directory use 'store' command.

```
grunt> store groc_sub into 'sub_dir' using PigStorage();
```

- δ Here **sub_dir** is a directory inside the pig installation directory.
- δ PigStorage() is used to serialize to disk as well as deserialize and read from file.
- δ The resulting file will be tab-delimited since the delimiter has not been specified.

▢ Go to the command prompt of the installation directory and type dir command.

▢ The 'sub_dir' has been created.

```
25-09-2019 04.25 PM <DIR> src
14-10-2019 11.36 AM <DIR> sub_dir
25-09-2019 04.25 PM <DIR> test
01-10-2019 09.00 PM 183 test.pig
25-09-2019 04.25 PM <DIR> tutorial
```

▢ Use type * command in windows and cat * command in linux to see the contents of the directory.

```
C:\Pig\pig-0.17.0\sub_dir>type *
part-r-00000.crc
crc 0,j0|
_SUCCESS.crc
crc 0
part-r-00000

01 Vijayanagar Bananas 20.0 7.0
02 Rajajinagar Apples 20.0
03 Jayanagar Flowers 10.0
04 Govindarajanagar Meat 40.0
05 Vijayanagar Potatoes 9.0
```

▢ Every field in the file is delimited by tabs.

▢ Use the same store command but use the delimiter as a ','.

```
grunt> store groc_sub into 'sub_dir' using PigStorage(',');
19/10/14 21:58:54 ERROR grunt.Grunt: ERROR 6000: <line 24, column 0> Output Location Validation Failed for: 'file:///C:/Pig/pig-0.17.0/sub_dir' More info to follow:
Output directory file: C:/Pig/pig-0.17.0/sub_dir already exists
Details at logfile: C:\BigData\hadoop-2.9.1\logs\pig_1571032286344.log
```

▢ This gives an error since the directory specified for storage should not exist.

```
grunt> store groc_sub into 'subset_dir' using PigStorage(',');
```

```
C:\Pig\pig-0.17.0\subset_dir>type *  
  
.part-r-00000.crc  
  
crc  @ |r|  
._SUCCESS.crc  
  
crc  @  
part-r-00000  
  
01, Vijayanagar, Bananas,,7.0  
02, Rajajinagar, Apples,,20.0  
03, Jayanagar, Flowers,,10.0  
04, Govindarajanagar, Meat,,40.0  
05, Vijayanagar, Potatoes,,9.0
```

- ▢ The file SUCCESS indicates that the MapReduce was completed successfully in this folder.
- ▢ The file with part as prefix contains the actual data.

```
C:\Pig\pig-0.17.0\subset_dir>dir  
Volume in drive C is OS  
Volume Serial Number is C44E-6526  
  
Directory of C:\Pig\pig-0.17.0\subset_dir  
  
14-10-2019  10.03 PM  <DIR>      .  
14-10-2019  10.03 PM  <DIR>      ..  
14-10-2019  10.03 PM                12 .part-r-00000.crc  
14-10-2019  10.03 PM                8  ._SUCCESS.crc  
14-10-2019  10.03 PM            153 part-r-00000  
14-10-2019  10.03 PM                0  _SUCCESS  
               4 File(s)            173 bytes  
               2 Dir(s)  329,205,870,592 bytes free
```

Data Types in Pig

- ▢ Scalar
 - δ Primitive types to represent a single entity or field.
- ▢ Complex
 - δ Collection types to represent a group of entities.
 - δ Every entity within this collection will have its own data type.

Scalar Data Type

- ▢ Five categories
 - δ Boolean
 - δ Numeric
 - δ String
 - δ Date/time
 - δ Bytes

Boolean

- ▢ Used to represent

- δ true or false
- δ yes/no
- ▣ Information about any topic.

Numeric

- ▣ Integers
 - δ Int: 4 bytes, range -2^{31} to $2^{31} - 1$
 - δ Long: 8 bytes, range -2^{63} to $2^{63} - 1$
- ▣ Decimals
 - δ Float: 4 bytes
 - δ Double: 8 bytes

String

- ▣ Chararray
 - δ Unbounded, variable length character string

Date/Time

- ▣ Datetime
 - δ Allows to represent time with nanosecond precision.
 - δ Time specified in the date, hour, minute, seconds, milliseconds, nanoseconds format

Bytes

- ▣ Bytearray
 - δ A blob used to represent any kind of data
 - δ Default type when
 - ∞ none of the other types are specified
 - ∞ Schema of the relation is unknown

Complex Data Types

- ▣ Represents the collection of a group of entities
 - δ Tuple
 - δ Bag
 - δ Map
 - ∞ Each of the individual entities can have their own corresponding data type.

Tuple

- ▣ An ordered collection of fields
- ▣ Example - (134, "John", "Smith", "HR", 9)
- ▣ Enclosed within parenthesis.

- ▢ Each field has its own data type.
- ▢ The data type is optional, they default to bytearray.
- ▢ Every record in a relation is a tuple.
- ▢ A tuple can be considered the equivalent of a row in a traditional relational database.

DEMO

- ▢ TOTUPLE() function allows the creation of tuples from individual fields in a relation.
- ▢ Use a file called student.txt which is tab-delimited.

```
a1234 AAA 8 (Bangalore, 9999999999)
a1235 BBB 8 (Mangalore, 8888888888)
a1236 CCC 8 (Mysore, 9999988888)
a1238 DDD 8 (Tumkur, 8888899999)
a1241 EEE 8 (Nellamangala, 8888877777)
```

- ▢ The first three fields are scalar fields.
- ▢ The last two fields are specified as a tuple.
- ▢ Load it using the following command.

```
grunt> stud = load 'student.txt'
>> as
>> (
>> stud_id: chararray,
>> name: chararray,
>> grade: int,
>> contact: tuple(city: chararray, phone: chararray)
>> );
```

- ▢ A tuple is specified using a field name for the entire tuple.
- ▢ It is indicated as a tuple using the tuple keyword and parenthesis.

```
grunt> dump stud;
(a1234,AAA,8,(Bangalore, 9999999999))
(a1235,BBB,8,(Mangalore, 8888888888))
(a1236,CCC,8,(Mysore, 9999988888))
(a1238,DDD,8,(Tumkur, 8888899999))
(a1241,EEE,8,(Nellamangala, 8888877777))
```

- ▢ Every record is enclosed in an outer parenthesis.
 - ∞ This is a tuple that represents a record.
- ▢ Use foreach to generate specific field records.

```
grunt> stud_info = foreach stud generate $1, $3.$0;
```

- ▢ To access individual fields of a tuple, the . operator is used.

<p>δ \$3.\$0</p> <ul style="list-style-type: none"> ∞ represents the first element of the fourth element in the record. ∞ Contact's first field i.e. city. 	<pre>grunt> dump stud_info; (AAA,Bangalore) (BBB,Mangalore) (CCC,Mysore) (DDD,Tumkur) (EEE,Nellamangala)</pre>
<ul style="list-style-type: none"> ▢ Use the describe command. ▢ It shows that all the fields in the relation are scalar types. 	<pre>grunt> describe stud_info; stud_info: {name: chararray,city: chararray}</pre>
<ul style="list-style-type: none"> ▢ Load the same text file without a schema into pig. ▢ The last field which was a tuple in the schema will be read in as a bytearray. 	<pre>grunt> stud_no_schema = load 'student.txt';</pre>
<ul style="list-style-type: none"> ▢ Use dump to see the relation. ▢ The output is similar looking to the output of the previous relation. ▢ The individual elements of the collection cannot be accessed using the dot operator. 	<pre>grunt> dump stud_no_schema; (a1234,AAA,8,(Bangalore, 9999999999)) (a1235,BBB,8,(Mangalore, 8888888888)) (a1236,CCC,8,(Mysore, 9999988888)) (a1238,DDD,8,(Tumkur, 8888899999)) (a1241,EEE,8,(Nellamangala, 8888877777))</pre>
<pre>grunt> stud_no_schema_INFO = foreach stud_no_schema generate \$1, \$3.\$0; grunt> dump stud_no_schema_INFO; 19/10/15 12:37:11 ERROR mapreduce.MRPigStatsUtil: 1 map reduce job(s) failed! 19/10/15 12:37:11 ERROR grunt.Grunt: ERROR 1066: Unable to open iterator for alias stud_no_schema_INFO Details at logfile: C:\BigData\hadoop-2.9.1\logs\pig_1571113139948.log</pre>	
<ul style="list-style-type: none"> ▢ To access the elements of the collection, the entire collection has to be accessed. 	
<pre>grunt> stud_no_schema_info = foreach stud_no_schema generate \$1, \$3; grunt> dump stud_no_schema_info; (AAA,(Bangalore, 9999999999)) (BBB,(Mangalore, 8888888888)) (CCC,(Mysore, 9999988888)) (DDD,(Tumkur, 8888899999)) (EEE,(Nellamangala, 8888877777))</pre>	


```

grunt> groceries = load 'groceries.csv' using PigStorage(',')
>> as
>> (
>> ord_id: chararray,
>> loc: chararray,
>> prod: chararray,
>> day: datetime,
>> amt: double
>> );
grunt> dump groceries;
(01, Vijayanagar, Bananas,,7.0)
(02, Rajajinagar, Apples,,20.0)
(03, Jayanagar, Flowers,,10.0)
(04, Govindarajanagar, Meat,,40.0)
(05, Vijayanagar, Potatoes,,9.0)
(06, Jayanagar, Bread,,5.0)
(07, Govindarajanagar, Bread,,5.0)
(08, Banashankari, Onion,,4.0)
(09, Govindarajanagar, Cheese,,15.0)
(10, Banashankari, Onion,,4.0)
(11, Malleshwaram, Bread,,5.0)
(12, Banashankari, Onion,,4.0)
(13, Chamarajpet, Bread,,5.0)
(14, Banashankari, Tomato,,6.0)

```

▮ The describe command will give the schema of the relation.

```

grunt> describe groceries;
groceries: {ord_id: chararray,loc: chararray,prod: chararray,day: datetime,amt: double}

```

▮ Create a tuple of product and amount for every record using TOTUPLE().

```

grunt> groceries_order = foreach groceries generate $0, $1, TOTUPLE($2, $4);
grunt> dump groceries_order;
grunt> describe groceries_order;
groceries_order: {ord_id: chararray,loc: chararray,org.apache.pig.builtin.totuple_prod_14: (prod: chararray,amt: double)}

```

▮ The third field is a tuple with a builtin name given by pig as org.apache.builtin.totuple_prod_14.

```
grunt> dump groceries_order;
(01, Vijayanagar,( Bananas,7.0))
(02, Rajajinagar,( Apples,20.0))
(03, Jayanagar,( Flowers,10.0))
(04, Govindarajanagar,( Meat,40.0))
(05, Vijayanagar,( Potatoes,9.0))
(06, Jayanagar,( Bread,5.0))
(07, Govindarajanagar,( Bread,5.0))
(08, Banashankari,( Onion,4.0))
(09, Govindarajanagar,( Cheese,15.0))
(10, Banashankari,( Onion,4.0))
(11, Malleshwaram,( Bread,5.0))
(12, Banashankari,( Onion,4.0))
(13, Chamarajpet,( Bread,5.0))
(14, Banashankari,( Tomato,6.0))
```

```
grunt> stud_info = foreach stud generate TOTUPLE($0, $1, $2), $3;
```

```
grunt> describe stud_info;
stud_info: {org.apache.pig.builtin.totuple Stud_id_31: (stud_id: chararray,name: chararray,grade: int),contact: (city: c
hararray,phone: chararray)}
```

```
grunt> dump stud_info;
((a1234,AAA,8),(Bangalore, 9999999999))
((a1235,BBB,8),(Mangalore, 8888888888))
((a1236,CCC,8),(Mysore, 9999988888))
((a1238,DDD,8),(Tumkur, 8888999999))
((a1241,EEE,8),(Nellamangala, 8888877777))
```

▢ Every record has two tuples – student information and contact information.

Bag

- ▢ An unordered collection of tuples.
- ▢ Each tuple can have a different number and type of fields.
- ▢ Duplicates may be present.
- ▢ Enclosed within curly braces.
- ▢ Example
 - δ {(134, "John", "Smith", "HR", 9)
 - δ (238, "Jill", "Paul", "Engg", 8)
 - δ (561, "Nina", "Tang", "Engg", 9)}
- ▢ Absent fields will be nulls when accessed.
- ▢ A collection of tuples is a relation.
- ▢ Therefore a relation is a bag of tuples also called the outer bag.
- ▢ If there is a bag within the fields of the relation it is referred as an inner bag.

Demo

▢ Use the stud_subj.txt	a1234	AAA	12	Maths	Chemistry	Physics
	a1324	BBB	12	Maths	Physics	
	a1235	CCC	12	Maths	Chemistry	
	a1324	DDD	12	Maths	Physics	English

<ul style="list-style-type: none"> Load it into pig 	<pre>grunt> stud_subj = load 'stud_subj.txt' >> as >> (>> stud_id: chararray, >> name: chararray, >> grade: int, >> sub1: chararray, >> sub2: chararray, >> sub3: chararray >>);</pre>
<ul style="list-style-type: none"> Use dump command The fourth student has enrolled only for two subjects, so the third subject will have a null. 	<pre>grunt> dump stud_subj; (a1234,AAA,12,Maths,Chemistry,Physics) (a1324,BBB,12,Maths,Physics,) (a1235,CCC,12,Maths,Chemistry,) (a1324,DDD,12,Maths,Physics,English)</pre>
<pre>grunt> stud_subj_bag = foreach stud_subj generate \$0, \$1, \$2, TOBAG(\$3, \$4, \$5); grunt> describe stud_subj_bag; stud_subj_bag: {stud_id: chararray,name: chararray,grade: int,{{chararray}}}</pre>	
<ul style="list-style-type: none"> The curly braces denotes that the last field is a bag. 	
<ul style="list-style-type: none"> TOBAG() function that converts fields within a relation to a bag. Each subject in the bag is a tuple. 	<pre>grunt> dump stud_subj_bag; (a1234,AAA,12,{{(Maths),(Chemistry),(Physics)}}) (a1324,BBB,12,{{(Maths),(Physics),()}}) (a1235,CCC,12,{{(Maths),(Chemistry),()}}) (a1324,DDD,12,{{(Maths),(Physics),(English)}})</pre>
<ul style="list-style-type: none"> 	

Map

- It is a collection of key value pairs.
- Key-value pairs, values can be looked up by key
- Keys are unique.
 - Every key can occur only once.
- Example
 - [John#HR
 - Jill#Engg
 - Nina#Engg]
- Map key-values are specified in square brackets
- Keys are always of type chararray
- The values can be of any data type
- The # is the delimiter when specifying maps in files

Demo

<p>Use the data stud_marks.txt</p>	<pre>a1234 AAA 12 [Maths#95, Physics#78, Chemistry#89] a1235 BBB 12 [Maths#45, Physics#48, Chemistry#84] a1324 CCC 12 [Maths#75, Physics#56, Chemistry#49] a1325 DDD 12 [Maths#65, Physics#59, Chemistry#44]</pre>
<p>Load the data into pig</p>	<pre>grunt> stud_marks = load 'stud_marks.txt' >> as >> (>> stud_id: chararray, >> name: chararray, >> grade: int, >> marks: map [int] >>); grunt> describe stud_marks; stud_marks: {stud_id: chararray,name: chararray,grade: int,marks: map[int]} grunt> dump stud_marks; (a1234,AAA,12,[Chemistry#89, Physics#78,Maths#95]) (a1235,BBB,12,[Chemistry#84, Physics#48,Maths#45]) (a1324,CCC,12,[Chemistry#49, Physics#56,Maths#75]) (a1325,DDD,12,[Chemistry#44, Physics#59,Maths#65])</pre>
<p>To access the map element</p>	<pre>grunt> stud_math = foreach stud_marks generate \$1, \$3#'Maths'; grunt> dump stud_math; (AAA ,95) (BBB ,45) (CCC ,75) (DDD ,65)</pre>
<p>Create a map of the name with the grade.</p>	<pre>grunt> stud_info_map = foreach stud_marks generate \$0, TOMAP(\$1, \$2); grunt> dump stud_info_map; (a1234,[AAA #12]) (a1235,[BBB #12]) (a1324,[CCC #12]) (a1325,[DDD #12]) grunt> describe stud_info_map; stud_info_map: {stud_id: chararray,map[int]} grunt> stud_info_map = foreach stud_marks generate \$0, TOMAP(\$1, \$2), \$3; grunt> dump stud_info_map; (a1234,[AAA #12],[Chemistry#89, Physics#78,Maths#95]) (a1235,[BBB #12],[Chemistry#84, Physics#48,Maths#45]) (a1324,[CCC #12],[Chemistry#49, Physics#56,Maths#75]) (a1325,[DDD #12],[Chemistry#44, Physics#59,Maths#65]) grunt> describe stud_info_map; stud_info_map: {stud_id: chararray,map[int],marks: map[int]}</pre>