Session 16

Assignment 2

|  |  |
| --- | --- |
| **Prepared For:** | AcadGild |
|  |  |
| **Document Approval:** | **AcadGild** |
|  |  |
|  |  |
|  |  |
|  |  |
| **Project Title:** | Session 16 – Assignment 2 |
|  |  |
| **Prepared By:** | Duncan Burgess |
|  |  |
|  | dburgess@duncb.com |
|  |  |
| **Primary Engineer:** | Duncan Burgess |
|  |  |
| **Document Reference:** | **Session 16 – Assignment 2** |
|  |  |
| **Start Date:** | 13/10/2017 |
|  |  |
|  |  |



# 

# Contents

[Contents 2](#_Toc495599714)

[Change History 3](#_Toc495599715)

[1. Problem Statement 4](#_Toc495599716)

[2. Solutions 4](#_Toc495599717)

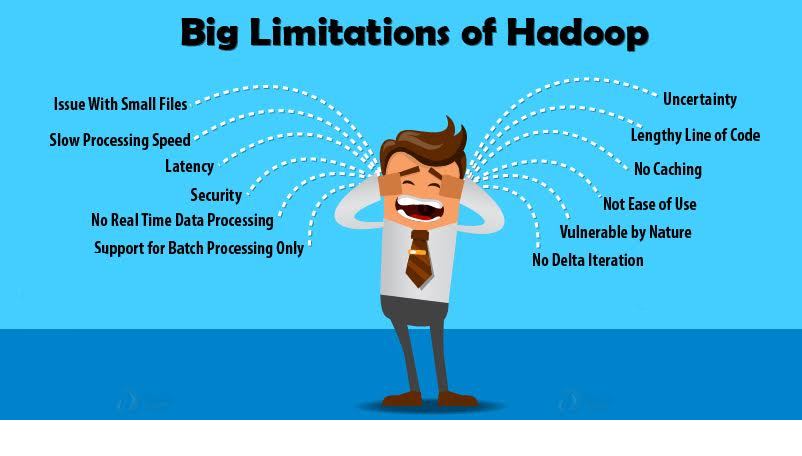
# Change History

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Document Revision** | **Date** | **Authored By** | **Authorised By** | **Sections Affected** | **Reason for Change** |
| Rev 01 | 14/10/2017 | Duncan Burgess |  | All | Initial release. |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

# Problem Statement

* Pen down the limitations of MapReduce.
* What is RDD? Explain few features of RDD?
* List down few Spark RDD operations and explain each of them.

# Solutions



Although[Hadoop](http://data-flair.training/blogs/hadoop-introduction-tutorial-quick-guide/) is the most powerful tool of [big data](http://data-flair.training/blogs/why-learn-big-data-use-cases/), there are various limitations of Hadoop like Hadoop is not suited for small files, it cannot handle firmly live data, slow processing speed, not efficient for iterative processing, not efficient for caching etc.

In this assignment on limitations of Hadoop, firstly we will learn about what is Hadoop and what are the pros and cons of Hadoop. We will see features of Hadoop due to which it is so popular. We will also see some Big Disadvantages of Hadoop

These have been improved significantly with[Apache Spark](http://data-flair.training/blogs/apache-spark-tutorial/).

## Hadoop – Introduction & Features

Let us start with what is Hadoop and what are Hadoop features that make it so popular.

[**Hadoop**](http://data-flair.training/blogs/hadoop-tutorial-for-beginners/)is an open-source software framework for distributed storage and distributed processing of extremely large data sets. Important features of Hadoop are:

* Apache Hadoop is an open source project. It means one can modify its code to business requirements.
* In Hadoop, data is highly available and accessible despite hardware failure due to multiple copies of data. If a machine or any hardware crashes, then data will be accessed from another path.
* Hadoop is highly scalable, as the new hardware can be easily added to the node. Hadoop also provides horizontal scalability which means nodes can be added on the fly without any downtime.
* Hadoop isfault tolerant, as by default 3 replicas of each block is stored across the cluster. So if any node goes down, data on that node can be recovered from the other node easily.
* In Hadoop, data is reliably stored on the cluster despite machine failure due to replication of data on the cluster.
* Hadoop runs on a cluster of commodity hardware which is not very expensive.
* Hadoop is very easy to use, as there is no need of client to deal with distributed computing; the framework takes care of all the things.

But as all technologies have pros and cons, similarly there are many Disadvantages of Hadoop as well. As we have already seen features and advantages of Hadoop above, now let us see the limitations of Hadoop.

## Big Limitations of Hadoop for Big Data Analytics

Various limitations of Hadoop are discussed below in this section along with their solution-

**Issue with Small Files**

**Hadoop** is not suited for small data. [(HDFS) Hadoop distributed file system](http://data-flair.training/blogs/comprehensive-hdfs-guide-introduction-architecture-data-read-write-tutorial/) lacks the ability to efficiently support the random reading of small files because of its high capacity design.

Small files are the major problem in HDFS. A small file is significantly smaller than the **HDFS block** size (default 128MB). If we are storing these huge numbers of small files, HDFS can’t handle these lots of files, as HDFS was designed to work properly with a small number of large files for storing large data sets rather than a large number of small files. If there are too many small files, then the **NameNode** will be overloaded since it stores the namespace of HDFS.

**No Real-time Data Processing**

Apache Hadoop is designed for batch processing, that means it take a huge amount of data in input, process it and produce the result. Although batch processing is very efficient for processing a high volume of data, but depending on the size of the data being processed and computational power of the system, an output can be delayed significantly. Hadoop is not suitable for Real-time data processing.

**No Delta Iteration**

Hadoop is not so efficient for iterative processing, as Hadoop does not support cyclic data flow(i.e. a chain of stages in which each output of the previous stage is the input to the next stage).

**Latency**

In Hadoop, MapReduce framework is comparatively slower, since it is designed to support different format, structure and huge volume of data. In **MapReduce,** Map takes a set of data and converts it into another set of data, where individual element are broken down into [key value pair](http://data-flair.training/blogs/key-value-pairs-hadoop-mapreduce/) and Reduce takes the output from the map as input and process further and MapReduce requires a lot of time to perform these tasks thereby increasing latency.

**Not Easy to Use**

In Hadoop, MapReduce developers need to hand code for each and every operation which makes it very difficult to work. MapReduce has no interactive mode, but adding one such as[**hive**](http://data-flair.training/blogs/apache-hive-tutorial-introductory-guide/)and[**pig**](http://data-flair.training/blogs/apache-pig-tutorial-introduction-guide/)makes working with MapReduce a little easier for adopters.

**Security**

Hadoop can be challenging in managing the complex application. If the user doesn’t know how to enable platform who is managing the platform, your data could be at huge risk. At storage and network levels, Hadoop is missing encryption, which is a major point of concern. Hadoop supports **Kerberos authentication**, which is hard to manage.

HDFS **supports access control lists (ACLs)** and a traditional file permissions model. However, third party vendors have enabled an organization to leverage**Active Directory Kerberos and LDAP** for authentication.

**No Abstraction**

Hadoop does not have any type of abstraction so MapReduce developers need to hand code for each and every operation which makes it very difficult to work.

**Vulnerable by Nature**

Hadoop is entirely written in **java**, a language most widely used, hence java been most heavily exploited by cyber criminals and as a result, implicated in numerous security breaches.

**No Caching**

Hadoop is not efficient for caching. In Hadoop, MapReduce cannot cache the intermediate data in memory for a further requirement which diminishes the performance of Hadoop.

**Lengthy Line of Code**

Hadoop has 1,20,000 line of code, the number of lines produces the number of bugs and it will take more time to execute the program.

**Uncertainty**

Hadoop only ensures that data job is complete, but it’s unable to guarantee when the job will be complete.

**Conclusion**

As a result of Hadoop’s limitation, the need for Spark arose. This made the system friendlier to use with a huge amount of data. Spark provides in-memory processing of data thus improves the processing speed.

# Features of Spark RDD



**In-memory Computation**

SparkRDDs have a provision of [**in-memory computation**](http://data-flair.training/blogs/apache-spark-in-memory-computing/)**.** It stores intermediate results in distributed memory (RAM) instead of stable storage(disk).

**Lazy Evaluations**

All transformations in Apache Spark are lazy, in that they do not compute their results right away. Instead, they just remember the transformations applied to some base data set.

Spark computes transformations when an action requires a result for the driver program. **5.3. Fault Tolerance**

Spark RDDs are fault tolerant as they track data lineage information to rebuild lost data automatically on failure. They rebuild lost data on failure using lineage, each RDD remembers how it was created from other datasets (by transformations like a map, join or groupBy) to recreate itself.

**Immutability**

Data is safe to share across processes. It can also be created or retrieved anytime which makes caching, sharing & replication easy. Thus, it is a way to reach consistency in computations.

**Partitioning**

Partitioning is the fundamental unit of parallelism in Spark RDD. Each partition is one logical division of data which is mutable. One can create a partition through some transformations on existing partitions.

**Persistence**

Users can state which RDDs they will reuse and choose a storage strategy for them (e.g., in-memory storage or on Disk).

**Coarse-grained Operations**

It applies to all elements in datasets through maps or filter or group by operation.

**Location-Stickiness**

RDDs are capable of defining placement preference to compute partitions. Placement preference refers to information about the location of RDD. The **DAGScheduler** places the partitions in such a way that task is close to data as much as possible. Thus, speed up computation.

# Spark RDD Operations

RDD in Apache Spark supports two types of operations:

* Transformation
* Actions

## Transformations

Spark RDD Transformations are functions that take an RDD as the input and produce one or many RDDs as the output. They do not change the input RDD (since RDDs are immutable and hence one cannot change it), but always produce one or more new RDDs by applying the computations they represent e.g. Map(), filter(), reduceByKey() etc.

Transformations are **lazy** operations on an RDD in Apache Spark. It creates one or many new RDDs, which executes when an Action occurs. Hence, Transformation creates a new dataset from an existing one.

Certain transformations can be pipelined which is an optimization method, that Spark uses to improve the performance of computations. There are two kinds of transformations: **narrow transformation**, **wide transformation**.

## Actions

An**Action** in Spark returns final result of RDD computations. It triggers execution using lineage graph to load the data into original RDD, carry out all intermediate transformations and return final results to Driver program or write it out to file system. Lineage graph is dependency graph of all parallel RDDs of RDD.

**Actions** are RDD operations that produce non-RDD values. They materialize a value in a Spark program. An Action is one of the ways to send result from executors to the driver. First(), take(), reduce(), collect(), the count() is some of the Actions in spark.

Using transformations, one can create RDD from the existing one. But when we want to work with the actual dataset, at that point we use Action. When the Action occurs it does not create the new RDD, unlike transformation. Thus, actions are RDD operations that give no RDD values. Action stores its value either to drivers or to the external storage system. It brings laziness of RDD into motion.