Summary Of Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing

Venkatesh Subramaniam

Resilient Distributed Dataset:

Resilient Distributed Datasets (RDD) is a fundamental data structure of Spark. It is an immutable distributed collection of objects. RDDs are motivated by two types of applications that current computing frameworks handle inefficiently: iterative algorithms and interactive data mining tools.

Iterative Algorithm : An iterative algorithm executes steps in iterations. It aims to find successive approximation in sequence to reach a solution.

Interactive Algorithm : Querying the data and applying a algorithm based on the user input methods.

Why RDD’s:

RDDs provide an interface based on coarse-grained transformations (e.g., map, filter and join) that apply the same operation to many data items. This allows them to efficiently provide fault tolerance by logging the transformations used to build a dataset (its lineage) rather than the actual data. Spark is up to 20× faster than Hadoop for iterative applications, speeds up a real-world data analytics report by 40×, and can be used interactively to scan a 1 TB dataset with 5–7s latency because of the presence of RDD’s

Spark Programming Interface:

Spark provides the RDD abstraction through a language integrated API similar to DryadLINQ in Scala , a statically typed functional programming language for the Java VM. To use Spark, developers write a driver program that connects to a cluster of workers.

Implementation :

The system runs over the Mesos cluster manager , allowing it to share resources with Hadoop, MPI and other applications. Spark can read data from any Hadoop input source (e.g., HDFS or HBase) using Hadoop’s existing input plugin APIs, and runs on an unmodified version of Scala.

The major parts of the system are

Job Scheduling:

The Job Scheduling assigns the jobs to the nodes and creates job stages. Whenever a user runs an action on an RDD, the scheduler examines that RDD’s lineage graph to build a DAG of stages to execute.

Interpreter Integration :

Helps run Spark interactively from the interpreter to query big datasets. The Scala interpreter normally operates by compiling a class for each line typed by the user, loading it into the JVM, and invoking a function on it. The two major changes to the interpreter in Spark is

1. Class shipping - worker nodes fetch the bytecode for the classes created on each line, we made the interpreter serve these classes over HTTP
2. Modified code generation: modified the code generation logic to reference the instance of each line object directly

Memory Management :

The three parts of memory managements in spark

Java Objects – for improving the speed of the process

In Memory Managements – for efficient in memory storage

On the Disk Storage – for storage of large data that is not possible to store in in memory

Support for Check-pointing:

Lineage can always be used to recover RDDs after a failure, such recovery may be time-consuming for RDDs with long lineage chains. Thus, it can be helpful to checkpoint some RDDs to stable storage.

Industry Use Case :

1. Twitter Spam Detection
2. In Memory Analytics
3. Traffic Predictions