Paper Review –6

Spark

Title: Spark: Cluster Computing with Working Sets

Summary

Apache Spark is a framework introduced by a team at UC Berkeley to build a distributed computing framework that fits into a generic task model that can be used by various large-scale data applications. The paper focuses primarily on applications that reuse a working set of data across multiple parallel operations. Spark achieves this by using RDD (Resilient distributed datasets). In paper, Spark has outperformed Hadoop by 10X for certain type of applications.

Good Aspects

During the development process of Mesos, a distributed resource manager framework, the creators felt that the Map reduce paradigm was not really making use of vast functionalities and performance improvements on Mesos. This was specially found on iterative programming models where Map reduce jobs loaded data every time to perform computations in an iterative application which suffered penalty due to I/O time. The creators of Mesos explored a new framework that helps these class of Iterative applications to be able to take advantage of the design of Spark to produce highly efficient and throughput based applications. Spark was able to produce a 10X improvement in Iterative Jobs over Hadoop and also has a interactive based query system with sub-second response time.

Review

Following details will cover all aspect of the paper providing crucial and also interesting aspects of the paper that appealed the reader.

Apache Spark is cluster based distributed framework for Iterative Models. It is implemented in Scala and exposes a functional programming interface. It also allows to be used using a modified version of a Scala interpreter and is a first of its kind that uses a modern programming language

Architecture

- **Model**
  - **RDD – (Resilient Distributed Dataset)**
    - RDD is a read only collection of objects partitioned across a set of machines that can be rebuilt if a partition is lost.
    - RDD can be constructed in 4 ways
      1. From a file in Shared file system.
      2. By parallelizing a Scala collection.
      3. By Transforming an existing RDD.
      4. By changing persistence of an existence model.

  - **Persistence in RDD**
    - Cache action – This leaves the dataset lazy, but hints that it should be kept in memory after the first time its computed.
    - Save action – Evaluates the dataset and writes into HDFS.

- **Several Parallel Operations**
  - **Reduce –** Combines dataset elements using an associative function.
  - **Collect –** Sends all elements of the dataset to the driver program.
  - **Foreach –** Passes each element through a user provided function.

- **Shared Variables –** Closures in Scala causes variables to be copied into worker every time its executed. Hence Spark provides two new shared variables for common usage patterns.

  - **Broadcast Variable –** Broadcast variables are created such that Spark ensures it to be copied to all workers once when performing any parallel operation.
  - **Accumulator Variable –** These are variables that workers can only add to using an associative operation and only the driver can read.

- **Sample Text Search code**.

```
val file = spark.textFile("hdfs://...")
val errs = file.filter(_.contains("ERROR"))
val ones = errs.map(_ => 1)
val count = ones.reduce(_+_)
```

  - First a distributed dataset called file is created.
  - Then the dataset is transformed to contain lines that contain ERROR.
  - Each are mapped to 1 and then added using reduce.

- Here errs and ones are lazy RDD's.

- **Sample Logistic Regression Code**

```
// Read points from a text file and cache them
val points = spark.textFile(...)
.map(parsePoint).cache()
// Initialize w to random D-dimensional vector
var w = Vector.random(D)
// Run multiple iterations to update w
for (i <- 1 to ITERATIONS) {
      val grad = spark.accumulator(new
      Vector(D))
      for (p <- points) { // Runs in parallel
            val s = (1/(1+exp(-p.y*(w dot p.x)))-
            1)*p.y
            grad += s * p.x
      }
w -= grad.value
}
```

  - This is an iterative classification algorithm that finds the hyperplane of Euclidian geometry points.
  - The for key-word is invoking the for each on the collection where the body is the closure.
  - Grad here is used as an accumulation operator and has implemented the operator +=

- **Implementation**

  - Spark is built on top of Mesos.
  - The dataset is stored in the form of RDD that can be recovered easily since its implemented using the concept of Lineage.
  - Each RDD implements *getPartitions*, *getIterator*, *getPreferredLocations* operations.
  - Certain changes are made to Scala closures in order to make them efficient like performing static analysis of bytecode generated and removing unused references.

- **Scala Interpreter**
  - **Two changes made**
    - Interpreter outputs the classes it defines to a shared filesystem from which they can be loaded by workers in a custom Class Loader.

- Reuse of singleton objects used previously used instead of *getInstance* methods.

Results

- **Logistic Regression –** With Hadoop each Job takes 127 seconds, but with Spark first job takes 174 seconds and subsequent tasks take 6 seconds.
- **Alternating Least Squares –** Caching Matrix R that contains the ratings was found to have improved performance by 2.8x in an experiment with 5000 movies on a 20node EC2 cluster.
- **Interactive Spark –** Used to dump queries and took 35 seconds for the first time and later only 0.5 to 1 seconds for subsequent queries.

Related Work

- **Twister –** Similar but doesn't support Fault tolerance.
- **Lineage –** A concept where each object keeps track of its parents and can use that information to reconstruct in case of any failure of RDD.
- **DSM (Distributed Shared Memory)** – DSM reconstruct in case of failure by using recover points and Spark uses Lineage to recover.

Conclusion and Summary

The following paper presents a new framework that scales well and produces great efficiency for certain type of applications that utilize an iterative model. From the paper its clear how the techniques used and the mesos 'distributed resource management' framework has helped it Achieve the goal.

It's proved it self to efficient in 2 use cases –Iterative Jobs and Interactive Analysis. Spark is efficient when many operations are performed continuously as it uses the concept of Caching of RDD and are built to tackle Fault tolerance by using concepts such as Lineage to reconstruct RDD when they are lost.

Spark is currently used in production by many big companies and is likely to be used in the future as its framework very generic and scalable.