Big Data Assignment

By Poornima Ramachandran

The research paper was based on Resilient Distributed Data Set (RDD) which is a distributed memory abstraction which lets a programmer perform in memory computations on large clusters in fault tolerant way . Further the paper also emphasis on the improving the speed of computing of large data sets using Spark as the platform.

In the following section of the report I will be mentioning in a detailed manner the process in which the results were achieved as mentioned in the various sections of the paper.

**Abstract**

RDDs are the fundamental data structures of the Spark software which is used in this project to arrive at our final conclusion. The research was based on current computing frameworks and based on the things which it handled inefficiently that is the iterative algorithms and interactive data mining tools. An iterative algorithm is a mathematical procedure that generates a sequence of improving approximate solutions for a class of performance. In both cases the keeping data in the memory can improve the performance by a good proportion. To achieve the fault tolerance efficiently RDDs provides restricted form of shared memory based on coarse grained transformation rather than fine grained transformation.

A fine grained transformation will be an update of one record in a database whereas the coarse grain is generally functional operators for example map, reduce, flat map, and join. Spark ‘s model takes advantage of this because once it saves the DAG of operation it can be used to recomputed as long as the original data is still there. The other reason why we prefer the coarse grain approach is because in a fine grain approach we cannot recomputed the data as we can in coarse grain approach. The other important function of coarse grain approach is you can save one function that can update a billion records but it also comes at a cost.

In this paper they also show that RDDs are expensive enough to capture a world class computations including the recent specialized programming model for iterative jobs such as Pregel and new applications that these model don’t capture.

**Introduction**

The entire concept of RDDs in this paper was reviewed on cluster computing frameworks. The cluster computing frameworks like map reduce and drayad have been widely adopted for large scale data analytics. These systems can let the user write parallel computations using a set of high level operators without having to worry about work distribution and fault tolerance.

Current frameworks provide numerous abstraction for accessing a clusters computational resources they lack abstraction for leveraging distributed .This makes them inefficient for an important class of emerging results across multiple computations. Data reuse is common in many iterative machine learning and graph algorithm including Page rank, K means clustering and logistic regression.

Another compelling use case is the interactive data mining where a user runs multiple adhoc queries on the same data set of the data. But in the most current frame work the only way to use the data between computations is to safe it in an external storage device. Recognizing the problem the programmers have developed new frameworks for some application that require reuse. Hence here the abstraction used is RDD as this enables an efficient data reuse in a broad range of applications Existing abstractions for in memory storage. With this inference the only way to provide fault tolerance is are to replicate the data across the machines or to the log updates across machines. Both the approaches are expensive for data intensive workloads as they require coping large amounts of data over the cluster network whose bandwidth is far lower than that of RAM and they incur substantial storage overhead. In contrast to these systems RDD provides an interface based on coarse grained transformations that apply the same operations to many data items. This allows them to efficiently provide fault tolerance by logging the transformation used to build a data set rather than the actual data.

**They evaluate RDDs and spark through both micro benchmarks and measurements of user applications. Which shows that Spark is up to 20 times faster than hadoop for iterative application. And it also speeds up the real world data analytics report by 40 times and can be used interactively to scan a 1 TB data set with 5-7 latency.**

In the second leg of this section the researchers describe the advantages of the RDD model as a distributed memory abstraction where they compare it to distributed shared memory. Where they described the main difference between RDD and distributed shared memory where they state that RDDs can be written or created only be created through coarse grained transformation while distributed shared memory allows to read and write in each memory location. This quality of RDD makes it more fault tolerant than others.

The third segment of the paper introduces us to the Spark programming interface used to arrive at the conclusion. The researchers have allowed RDD to perform various operations such as join, groupbykey, and reducebykey.

Also the researchers have complemented the learning with two iterative applications namely logistic regression and Page rank. As many machine learning algorithm are iterative in nature namely logistic regression because they can run much faster by keeping their data in the memory. Page Rank is an algorithm that updates rank of each document by adding up contributions from the documents that link to it. We can also reliably replicate some of the order of the ranks to reduce the fault recovery times The link data set is much bigger than the rank dataset because each document may have many links . Further this can be optimize the communication in page rank by controlling the partitioning of RDDs in such a way we can ensure that join operation between links and ranks requires no communication.

The representation of RDD done is a simple graph based representation. The question of interpreting the dependencies is between the RDDs is solved in this paper by used two types of representation they are narrow dependencies where each partition of the parent is used by at most one partition of the child and wide dependencies where multi child partitions depend on it . This distinction was necessary for two reasons the narrow dependencies allows pipelined execution on one cluster node which can compute all the parent partitions. In contrast the wide dependencies require data from all parent partitions to be available to shuffle across the nodes using a map reduce like operation.

**Implementation:** In the paper spark was implemented about 14000 lines of Scala. The system runs over the Mesos cluster manager allowing it to share resources with Hadoop, MPI and other applications. This allows them to perform the activities such as job scheduling, memory management and support for check pointing. The job scheduler assigns tasks to machine based on data locality using delay scheduling. If a task needs to process a partition that is available in the memory on a node we can send it to that node. Otherwise if a task processing a partition for which the containing RDD provides preferred locations we send it that node.

Also as all the computations in spark currently run in response to actions called in the driver program they also experimented with letting task on cluster call the look up operation which provides random access to elements of hash portioned RDDs by the keys.

**Interpreter Integration**

The changes which was made in the interpreter in spark are 1. Class shipping and 2. Modified code generation. The class shipping is to let worker nodes fetch the bytecode for the class create on each line, we made the interpreter serve these classes over HTTP and modification code generation the singleton object created for each line of code is accessed through a static method on its corresponding class.

**Evaluation**

The evaluation spark and RDDs through a series of experiments on amazon EC2 as well as benchmarks of user applications. Overall results show the following:

* Spark outperforms Hadoop by up to 20 times in iterative machine learning and graph applications . The speedup comes from avoiding I/o and deserialization costs by storing data in memory as Java objects.
* Applications written by the users perform and scale well. In particular, we used spark to speed up analytics report that was running on hadoop is 40 times faster .
* When the nodes fail spark can recover quickly by rebuilding only the lost RDD partitions.
* Spark can be used to query a 1 TB dataset interactively with the latencies of 5 -7

It was also implemented two iterative machine learning applications logistic regression and K means to compare the performance for the following

Hadoop , hadoop bin mem and Spark

The researchers ran both algorithms for 10 iterations on 100 GB data sets using 25-100 machines . The key difference between the two applications is the amount of computation they perform per byte of data.

In Page Rank the researchers compared the performance of spark with hadoop using a 54 GB Wikipedia dump . The result showed that demonstrated that in memory storage alone provided spark with a speed of 2.4 times the speed over hadoop.

**Fault recovery**: When the evaluation was done for RDD partition using lineage after a node failure in K means applications. Without failure it consisted of 400 tasks working on 100 GB of data. While execution there were some loss of task running on the machine however spark re ran these tasks in parallel on the machines where they reread the dataset again.

**User Applications built with spark**

The user applications build with spark mentioned In the paper are 1.In Memory Analytics 2. Traffic model 3. Twitter spam classification.

**Conclusion**: With the implementation and analysis the researches have focused more on the fault tolerance of the RDDs and the also eventually improving the speed of the analysis of the data in Spark platform.