Assignment 2 Reflection

Lukasz Filanowski 18414616

Each question in Section one is separated into a separate .scala file. They are labelled according to the question they solve in Section 1. Eg: task1.scala corresponds to Section 1 point 1. In the case of point 3 which has parts (a) and (b), separate files are made for each of these. Section 2 is done in one single file called graphCreate.

# Q3

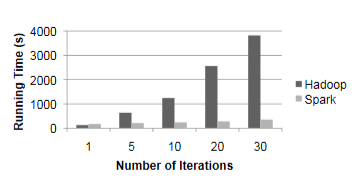
## Spark: Cluster Computing with Working Sets

In this paper, the author focuses on a specific class of application which unlike its predecessors, isn’t built around an acyclic data flow and reuses working data sets across several parallel operations. The Spark framework is applicable to interactive machine learning algorithms as well as data analysis tools. Spark remains at the same time very scalable and has the error tolerance of traditional MapReduce. The Spark framework introduces a new dataset know as the Resilient Distributed Dataset (RDD). This read-only abstraction can be partitioned across several machines and rebuilt easily if a partition is lost. The paper aims to outline the workings of Spark RDD’s as well as its advantages over traditional MapReduce frameworks.

The RDD can be seen as an abstraction of Distributed Shared Memory (DSM). However, there are some differences between the two. RDD’s have a much more restricted programming model. This however allows for efficient dataset recreation and rebuilding if a node fails. This is done through lineage information which is captured in the RDD object. In DSM, this is done through checkpoints. In RDD’s there is no overhead if no nodes fail. Data computation is pushed in the same way as in MapReduce. Another framework that Spark shares similarities with is a MapReduce framework known as Twister. Spark’s parallel operations which persist across operations was recognised by Twister which allows for map tasks to be kept static data in memory. The design for Spark’s language design and design was inspired by DryadLINQ. Both use .NET language support which helps integrate queries in an expression tree that can be run on a cluster. However, Spark expands on this by allowing shared variables. Scala was used for language integration in Spark. The inspiration for this came from SMR which is a Scala interface for Hadoop. SMR uses closures to define map/reduce tasks. Spark once again expands on the concept by implementing more robust closure serialization.

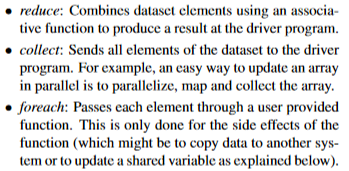
Spark was developed to optimize areas of underperformance of MapReduce. Through research, it was found that 2 main such areas exist. Many machine learning algorithms work on the same dataset and apply a function to it repeatedly in order to optimize a parameter. Even though each iteration can be expressed as a MapReduce job, the data from the disk must be loaded in each time, which can affect performance greatly. Secondly, Hadoop is used to run ad-hoc queries on big datasets. Due to limitations of the MapReduce model, Hadoop must load each query as a separate job and read the data from disk each time. This once again applies significant latency. Spark aims to support applications with large working sets and provide the same scalability and error tolerance as MapReduce.

Spark RDD’s support a model of caching the RDD in memory and distribute it across many machines. This can be reused in several MapReduce-like parallel operations. RDD’s also achieve great fault recovery through lineage. If a partition is lost, the RDD will have enough information to rebuild the partition. Spark has also been shown to be able to outperform Hadoop by 10 times in iterative machine learning jobs. It can also be used effectively to scan a 39GB dataset with sub-second delay. It has also proven to be far more efficient than Hadoop at tasks such as Logistic Regression



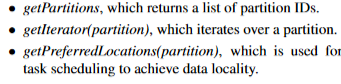
Spark allows users construct RDDs’ in a few ways. They can do so from a file system, such as a Hadoop Distributed File System (HDFS). An array un Scala, also known as a Scala collection, can be parallelized into an RDD. An RDD can also be transformed into another form such as a flatMap or List. A Spark RDD also supports several parallel operations. The data in the RDD can be reduced, collected and have a foreach statement used on it. All of these operations provide powerful tools to handle parallel operations.

List of supported parallel operations, with others such as shuffle and a group reduce operation being added with later iterations:



Shared variables were another implementation added to the Spark framework. These shared variables are made available to the user to support 2 simple use patterns. The first, known as a broadcast variable, ensures that if a piece of data is used in many parallel operations, it is wrapped and only copied to each worker once, instead of being packaged with every closure. Accumulators can be used to implement counters as workers can only add to them and only the driver can read them.

Spark has been demonstrated to be effective in a variety of programming problems such as text search, logistic regression and Alternating Least Squares, which is a filtering problem, just to name a few. Spark is built on top of Mesos which is a cluster operating system. It allows for multiple parallel applications to share a cluster. Building off Mesos also greatly decreased the programming effort required to create Spark. Each RDD is implemented with a simple interface which features 3 operations:



In future, the Spark framework is to be expanded and improved by implementing the shuffle operation that can repartition an RDD according to a key. An interactive interface is also to be added as well as formal characterization of the properties of an RDD and other abstractions.

The paper clearly demonstrates the strengths of the Spark framework and the reasons why it was developed and widely applied throughout. The limiting factor of the paper nowadays is that it is quite outdated as Spark is now much more developed and commonly used. However, it is a good insight into the development and early life of Spark