Project Proposal

Lee Boyd

Corey Crosser

Sean Soderman

March 1, 2017

1 Background

Big data is comprised of three Vs, volume, velocity and variety. As more and more information is collected each day, the demand for optimized, elastic scaled, and generalized processing platforms continues to grow. This is why MapReduce style platforms (MapReduce, Hadoop, Spark) continue to be a high demand research area, with corporate monetary interests, scientific breakthroughs, and innovative technologies closely aligned with their efficiency. Some optimizations include intelligent partitioning for the shuffle phase to prevent data skew or efficient scheduling to minimize network communication overhead.

2 Problem

MapReduce and its sibling, Hadoop, optimize big data processing by trying to use the closest processors to the data blocks in the distributed file system. Although “taking the processing to the data” is a mantra commonly heard in big data processing, IaaS providers tend to keep their cloud storage and cloud processing nodes in different clusters. This means before processing is taken to the data, data is taken to the clusters.

Often the entire cluster manifests through the scaling up of virtual machines in the cloud processing racks. Moving from cloud storage to the computation cluster is an opportunity for more intelligent placement of local data by maximizing data similarity based on the lambda functions of the job to be performed. Using a combiner with highly homogeneous local data compresses the output of the mapper and reduces network traffic in the shuffle. The problem then is to show that given a sufficiently large job, we can determine a way to place data in HDFS, such that the loss of time from the placement is outweighed by the reduction in network overhead from enhanced data placement.

3 Method

We assume that we are performing big data processing in the cloud. As such, the cluster needs to be created with the distributed file system on the processing side being filled with data from the storage side. Our goal is to customize the placement of files as they are moved, such that similar files are placed in the same HDFS file.

We assume that we have slave-nodes running a kernel in Spark that counts the number of words in a large text corpus stored in simulated storage cloud (linux FS on VM). We choose this task because word counts are required for term-frequency indexing, a tool that is useful for dealing with large unstructured data.

Our example will use articles from CNN.com. Online news is readily available, of high public importance, and intuitively clustered i.e. politics, sports, and entertainment. Using a script to sample the corpus allows us to train an unsupervised learner to efficiently partition the news data into HDFS files. To enhance the quality of our sampling, our script will exploit the hierarchical semi-structured nature of the website data.

Before combining, we augment the lambda function by adding the location of each output key with the location on disk (or in the RDD) of its elements. In our example, the key would just be a single word, but in other cases it may be a combination of words. Various weighting schemes could be chosen for combinations, but this is not our focus in this project. Before combining, we make another transformation that adds a composite key and a value of one for each time a pair of keys are within a maximum distance D from each other on disk. D is a hyperparameter that can be adjusted with experimentation. Dropping the location augment from the original key value output, we will run the combiner, giving us a set of tuples and the number of times they occur within a range of D and our output data. We devote a portion of the partitions in the action phase to calculate the partitions.

The problem of choosing future K clusters as a partition of the disk space in the next round is equivalent to a graph coloring problem, which is NP-hard. We use vertices as keys from our original output, and weighted edges being the number of times key pairs are found close together. Using the algorithm in [Clustering on k-edge-colored graphs, E. Angel1] we can efficiently estimate a clustering with K colors, trying to maximize the weights of edges within clusters.

After the first round, we will create K files in HDFS, each one representing a color from the clustering. Each round, we will double the number of blocks as input, but we will still sample each block before deciding which of the K files to append it to in the cluster. Each time we finish a round we will add this information to the RDD of the previous round. We expect to double the size of input on each round, but will stop clustering data after so many rounds. This is subject to experimentation.