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 and run them through the kernel function should give us the data we need to cluster the news data into HDFS files that are reflective of the lambda function. To enhance the quality of our sampling, our script will exploit the hierarchical semi-structured nature of the website data to try to obtain the most representative sample set for the learner.

We will start by finding the clusters or optimal files to create with HDFS. We will use a small but representative sample of the data, running the function that we want to use, in this case reduceByKey on a split of each article. We will augment the output with the RDD address from which the key was produced.

Our next transform will look for those keys that are some distance D from each other. We can use a flatmap to expand the range of D in each direction from the keys' location in memory. Next, we will need to then find some way to flip the keys and their data. Once flatmapped addresses are keys, we can use an inner join and again flip the data and the keys to create a composite key of tuples, combinations of keys that have come with a range D of each other in memory. Using this data, one final transform reduceByKey will create a relative frequency for which keys come come with a range of D from each other in the corpus. In our example, the key would just be a single word, but in other cases it may be a combination of words.

The problem of choosing amax K clusters as a partition of the disk space is equivalent to a graph coloring problem, which is NP-hard. Composite keys from the output as vertices and weirghted edges as their frequencies, we can use the algorithm in [Clustering on k-edge-colored graphs, E. Angel1], which estimates at most K clusters based on the sum of internal edge weights.

4 Testing

Once we have decided on partitions, will create at most K files in HDFS, each one representing a color from the clustering algorithm. We can now begin the process of moving the data from the storage area to HDFS. We can do so by taking a random sample of the words for each article and voting on which cluster they belong to. We may normalize by the frequency with which we found those words in our original sample. After placement in HDFS we can run the transformations and actions as normal.

We will compare the speed of these results against the speed of running without the partitions. The reason we will not factor the clustering time, is because we assume this operation is done off-line. The big data task is assumed to be something that is done frequently on data that is high velocity, such as social media or news. We expect once we establish reasonable partitions for a domain, there will not be that much change to warrant redefining the partitions every time we query the data.

5. Future work

For future work, we would like to integrate the partitioner with the actual tasks. Each time we run more tasks, the partitioning scheme would become better, until a suitable convergence. We would also like to do more work on generalizing the process to more complex tasks beyond just a single reduceByKey.