Eli Sorey
CS11 Hadoop
5/21/16

Week Six: Understanding Spark

Hadoop vs Spark

One of the key differences between Hadoop and Spark is the way in which intermediate results are stored. In the case of Hadoop, results of each map and reduce process are written to disk (within HDFS). Spark, on the other hand, keeps these results in memory. This makes Spark much better suited for use cases involving iterative computing, because it avoids incurring costs for reading and writing to disk. This allows Spark to be used as a tool for interactive tool for big data analysis, whereas Hadoop is too slow to be used interactively. The under-the-hood technology that allows Spark to do this is its use of Resilient Distributed Datasets (RDDs). RDDs are the atom unit of data in Spark, representing the finest level of control the user has over her data. The Spark framework provides high efficiency and fault tolerance through *lineage* of its RDDs. In a nutshell, Spark keeps track of the transformations (maps, filters, etc.) needed to create an RDD, rather than explicitly keeping the RDD in memory. Once the user calls for an *action* (any command that requires the RDD explicitly, such as counting entries), Spark spins up the RDD into memory based on its lineage. In the event of a fault, RDDs are easily reconstructed from their lineages. This is all in contrast to the Distributed Shared Memory (DSM) architecture employed by Hadoop and other distributed frameworks. Under DSM, the user operates in a global address space that spans the entire cluster of machines, and has fine-level (ie, individual byte) access to the memory.

In spites of its advantages, Spark does not replace Hadoop. For starters, Spark relies on an underlying distributed data storage system in order to operate; the most common choice here is HDFS. Even more than this though, Hadoop is still the right tool for certain jobs requiring fine grained access to memory. Examples include managing state in web applications or orchestrating an incremental web crawler. Additionally, Hadoop is a much more mature project, and has a stronger ecosystem of libraries surrounding it. If a user has no need for the interactive capabilities offered by Spark, he is most likely better off simply using Hadoop.

Transformations vs Actions in Spark

Transformations are commands that create new RDDs from existing ones. Spark evaluates transformations lazily, not actually computing their results until they are needed. When are they needed? Actions are commands that return results from RDDs and therefore require the RDDs to be loaded into RAM. Examples of transformations include maps and filters, and examples of actions include counts and saving data to a file.