Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing

I. Summary

RDDs is a distributed memory abstraction that lets programmers perform in-memory computations on large clusters in a fault-tolerant manner. It's motivated by iterative algorithm and interactive data mining tools, which both requires keeping data in memory to improve performance.

For the sake of fault tolerance, RDDs is based on coarse-grained transformation. In particular, RDDs do not need to incur the overhead of checkpointing, as they can be recovered using lineage. Only the lost partitions of an RDD need to be recomputed upon failure, and they can be recomputed in parallel on different nodes, without roll back the whole program.

For operating RDD in Spark, there are two types of operations: transformations and actions. Transformations are lazy operations that define a new RDD, while actions launch a computation to return a value to the program or write data to external storage.

For representing RDDs in Spark, the most interesting question in designing the interface is how to represent dependencies between RDDs. In Spark, dependencies are classified into two types: narrow dependencies and wide dependencies.

For job scheduling, whenever a user runs an action on an RDD, the scheduler examines that RDD's lineage graph to build a DAG of stages to execute. Each stage contains as many pipelined transformations with narrow dependencies as possible.The boundaries of the stages are the shuffle operations required for wide dependencies, or any already computed partitions that can short-circuit the computation of a parent RDD.

For memory management, Spark provides three options for storage of persistent RDDs: in-memory storage as deserialized Java objects, in-memory storage as serialized data, and on-disk storage. LRU eviction policy at the level of RDDs is used to manage the limited memory available, but users have further control via a "persistence priority" for each RDD.