Tag: coarse-grained transformations

Resilient Distributed Datasets

Posted on

Resilient Distributed Dataset (RDD) that was originally developed at the University of California, Berkeley is a partitioned collection of records that can be operated on in parallel. It provides the basic abstraction to Spark. Its **coarse-grained transformations** provide efficient fault tolerance to cluster computing frameworks.

Many machine-learning algorithms apply the same operation repeatedly on the intermediate data sets. MapReduce provides fault tolerance by replication of each operation result across different computers connected with network. It is called fine-grained updates. Since data replication, serialization, and transmission are involved, the fine-grained updates decrease computation speed.

RDD resolved this problem with coarse-grained updates where coarse-grained transformations are logged rather than their results. If a result is lost, it is re-computed by the logged transformation without rolling back the whole program. Information about how a dataset was derived from other datasets is called “lineage.”

The figure 1 below shows a lineage-based fault tolerance mechanism of RDD. RDD has two kinds of operation, transformations and actions. While transformations perform only on memory and do

not access to a storage, actions return a computation result to a program or write data to a storage. Thus, transformations enable less storage access while fault tolerance is ensured by their logs.

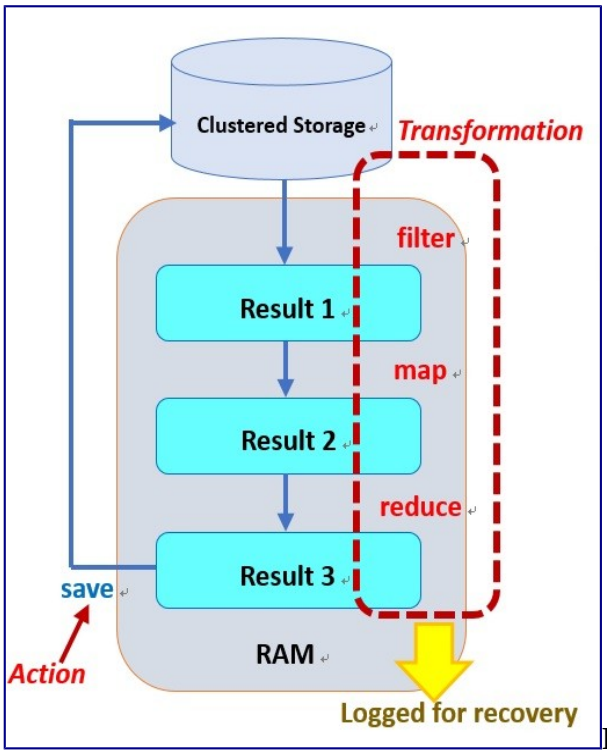

Figure 1: Lineage-based Fault Tolerance of RDD

Figure 2 shows fine-grained updates implemented in MapReduce, where intermediate results are saved in a storage for data recovery. It requires costly data replication and computation speed is effected by network bandwidth. Also it consumes a large amount of storage. Therefore, MapReduce is inefficient for iterative operations.

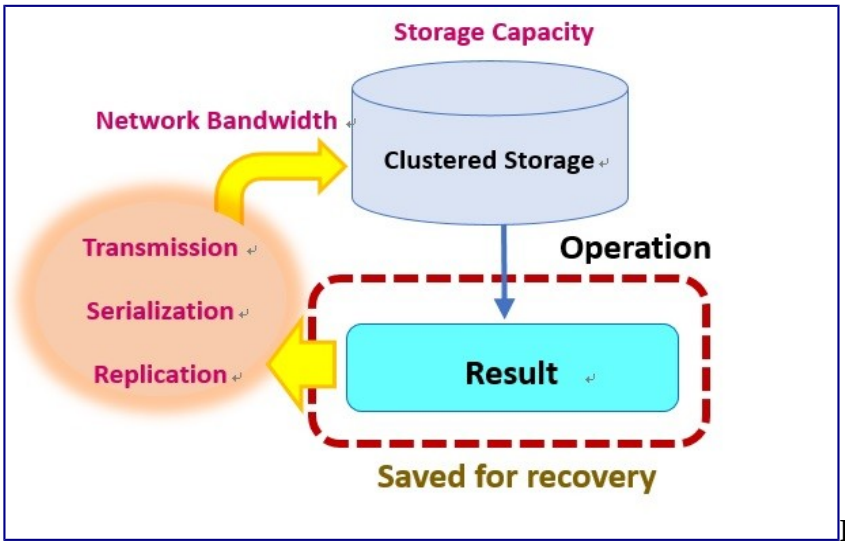

Figure 2: Fine-grained Updates

In this way, RDD provides efficient fault tolerance to cluster computing frameworks.

RDD is implemented in Spark and provides high computation for iterative machine learning algorithms and interactive data mining where intermediate transformation results are cached in memory and reused repeatedly without disk access.

However, Spark extremely consumes its memory as it holds all intermediate results across iterations. Required memory size varies depending on algorithms and size of data to be handled.