Resilient Distributed Datasets – Review

Yixing Chen, yc3094

The paper *Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing* introduced RDD – a fault-tolerant abstraction for in-memory cluster computing. It's a distributed memory abstraction that lets programmers perform in-memory computations on large clusters in a fault-tolerant manner. RDDs are motivated by two types of applications that current computing frameworks handle inefficiently: iterative algorithms and interactive data mining tools.

Cluster computing frameworks like MapReduce and Dryad have been widely adopted for large-scale data analytics. These systems let users write parallel computations using a set of high-level operators, without worrying about work distribution and fault tolerance. In this paper, RDDs are presented as a fault-tolerant, parallel data structures that let users explicitly persist intermediate results in memory, control their partitioning to optimize data placement, and manipulate them using a rich set of operators.

Contrary to Hadoop's two-stage disk-based MapReduce paradigm, Spark's multi-stage in-memory primitives provide performance up to 100 times faster for certain applications. Although some frameworks provided numerous abstractions for accessing a cluster's computational resources, they lack abstractions for leveraging distributed memory. It's very inefficient for some applications that reuse intermediate results. By allowing user programs to load data into a cluster's memory and query it repeatedly, Spark is well-suited to machine learning algorithms.

Formally, an RDD is a read-only, partitioned collection of records. RDDs can only be created through deterministic operations on either (1) data in stable storage or (2) other RDDs. They do not need to be materialized at all times. Instead, an RDD has enough information about how it was derived from other datasets to compute its partitions from data in stable storage. Programmers start by defining one or more RDDs through transformations on data in stable storage and then use these RDDs in actions, which are operations that return a value to the application or export data to a storage system. They can also call a persist method to indicate which RDDs they want to reuse in future operations

Overall, the scheduler is similar to Dryad's, but it additionally takes into account which partitions of persistent RDDs are available in memory. Two changes have been made in Spark: Class shipping and modified code generation. The authors evaluate RDDs and Spark through a series of experiments on Amazon EC2, the result shows that Spark has better performance than Hadoop in Logistic Regression and K-Means.

In conclusion, RDDs can express a wide range of parallel applications, including many specialized programming models that have been proposed for iterative computation, and new applications that these models do not capture. Different with some existing storage abstractions for clusters which require data replication for fault tolerance, RDDs offer API based on coarse-grained transformations that lets them recover efficiently using lineage.