Spark: Cluster Computing with Working Sets

Matei Zaharia, Mosharaf Chowdhury, Justin Ma, Michael J. Franklin, Scott Shenker, Ion Stoica
University of California, Berkeley

MapReduce and its variants have been highly successful in implementing large-scale data-intensive cluster applications. However, most of these systems are built around an acyclic data flow model that does not capture many important use cases. We present Spark, a new cluster computing framework motivated by one such class of use cases: applications that reuse a *working set* of data across multiple parallel operations. This includes many iterative machine learning algorithms, as well as interactive data analysis tools. Spark provides a set of distributed memory abstractions that allow it to efficiently support applications with working sets, while retaining the scalability and fault tolerance of MapReduce. We show that Spark can outperform Hadoop by 10x in iterative machine learning jobs, and can be used to interactively query a 40 GB dataset with sub-second response time. We propose to present both a poster about Spark and a demo of the system being used interactively to search Wikipedia.

The main abstraction in Spark is a *resilient distributed dataset* (RDD), which represents a read-only collection of objects partitioned across a set of machines that can be rebuilt if a partition is lost. Users define RDDs by starting with an on-disk dataset (e.g., a file in the Hadoop Distributed Filesystem) and transforming it. Users can explicitly control the *persistence* of each RDD, e.g., ask for it to be cached in memory across the nodes in the cluster. They can then reuse RDDs across MapReduce-like *parallel operations*. RDDs achieve fault tolerance by maintaining information about *lineage*: if a partition of an RDD is lost, the RDD has enough information about how it was derived from other RDDs to be able to rebuild just that partition. Although RDDs are not a general distributed memory abstraction, they represent a sweet-spot between expressivity on the one hand and scalability and reliability on the other hand, and we have found them well-suited for a variety of applications.

By providing RDDs, Spark is a better fit than MapReduce and Dryad for two classes of datacenter applications:

- **Iterative jobs**: Many common machine learning algorithms apply a function repeatedly to the same dataset to optimize a parameter (e.g., through gradient descent). While each iteration can be expressed as a MapReduce/Dryad job, each job must reload the data from disk, incurring a significant performance penalty. In contrast, in Spark, the application can represent the data as an RDD and ask that it be cached in memory across the nodes of the cluster.
- **Interactive analytics**: Hadoop is often used to run ad-hoc exploratory queries on large datasets, through SQL interfaces such as Pig and Hive. With Hadoop-based tools, each query incurs significant latency (tens of seconds) because it runs as a separate MapReduce job and reads data from disk. With Spark, a user can load a dataset of interest into memory across a number of machines and then query it repeatedly.

Spark is implemented in Scala, a statically typed high-level programming language for the Java VM, and exposes a functional programming interface similar to DryadLINQ. In addition, Spark can be used interactively from a modified version of the Scala interpreter, which allows the user to define RDDs, functions, variables and classes and use them in parallel operations on a cluster. We believe that Spark is the first system to allow an efficient, general-purpose programming language to be used interactively to process large datasets on a cluster.

Although our implementation of Spark is at an early stage, experience with the system is encouraging. Spark can outperform Hadoop by 10x in iterative machine learning workloads and can be used interactively to scan a 40 GB dataset with sub-second latency. Several machine learning researchers in our lab have used Spark implement a number of algorithms, including logistic regression, expectation maximization, and bootstrap sampling.

At OSDI, we propose to present both a poster and a demo of Spark being used interactively to search through Wikipedia. In the demo, we show that searching through Wikipedia (a 40 GB dataset) by reading it from disk (as in a standard MapReduce environment) takes about 20 seconds per query on a 20-node cluster. We then load Wikipedia as a cached RDD, where each machine in the cluster holds a portion of the data in memory, and show that the query time falls to 0.5-1 seconds, even for full scans of the data. This leads to a nearly realtime experience where users can enter new queries at the Scala interpreter interactively and explore the data as if they were working with a small dataset on a single machine. We also plan to present future directions for Spark, such as other distributed memory abstractions for cluster computing.