CS267 Project Proposal:
Performance and Scalability Characteristics of Spark

Mosharaf Chowdhury
mosharaf@cs.berkeley.edu

March 29, 2010

Motivation

In recent years, a new model of performing data-parallel computations on clusters of unreliable machines (e.g., MapReduce [1], Dryad [2]) has become widely popular. These systems achieve their scalability and fault tolerance by providing a programming model where users create acyclic data flow graphs to pass input data through a set of operators. This allows the underlying system to schedule jobs and to react to faults without user intervention.

While this data flow programming model is useful for a large class of applications, applications that reuse a *working set* of data across multiple parallel operations cannot be expressed efficiently as acyclic data flows. Such iterative jobs are extremely prevalent in machine learning algorithms that repeatedly apply a function 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.

What is Spark?

Spark is a new cluster computing framework that supports applications with working sets while providing the same scalability and fault tolerance properties as MapReduce. The main abstraction in Spark is that of 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 can explicitly broadcast and cache an RDD in memory across machines and reuse it in multiple MapReduce-like *parallel operations*. RDDs achieve fault tolerance through a notion of *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 shared memory abstraction, they represent a sweet-spot between expressivity on the one hand and scalability and reliability on the other hand, and have been found well-suited for a variety of applications.

In addition, Spark also supports parallelized reduce and collect operations as well as distributed shared variables such as broadcast variables and accumulators.

Project Overview

Spark is still in its early development stage and require robust performance measurement to pinpoint its strengths as well as its weaknesses. Our primary objective in this project is to measure the performance and scalability characteristics of Spark using suitable workloads, compare it with competing frameworks (e.g., Hadoop MapReduce implementation), identify rooms for improvement, and finally, implement algorithms and mechanisms to mitigate those weaknesses.

Objectives and Deliverables

Initially, we are considering using standard iterative machine learning algorithms (e.g., linear regression, alternating least squares) and synthetic workloads to start our measurement study.

Next, we want to study Spark performance and scalability using collaborative filtering and related Netflix prize dataset (consisting of 100,480,507 ratings that 480,189 users gave to 17,770 movies) as the test workload.

In fact, an initial attempt to do so has pointed out that the existing centralized broadcasting mechanism implemented in Spark does not scale after only eight workers. We want to study MPI communication primitives and design and implement scalable broadcasting mechanisms for Spark to support large-scale distributed parallelized operations.

Once we are done with the aforementioned objectives, we might also get access to large amount of traffic data collected from Northern California through the Mobile Millennium project and get to run and measure performance of their algorithms on those data using Spark. However, this highly depends on our progress on making Spark scalable by completing the aforementioned basic and intermediate objectives as well as on the limited time we will have to complete the project.

All these experiments will be performed using EC2 on tens of nodes (hundreds if we can get access to such facility) so that the measurements represent real world characteristics of Spark and the newly implemented algorithms and mechanisms can survive realistic workloads in production environments.

References

[1] Jeffrey Dean and Sanjay Ghemawat. MapReduce: Simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[2] Michael Isard, Mihai Budiu, Yuan Yu, Andrew Birrell, and Dennis Fetterly. Dryad: Distributed data-parallel programs from sequential building blocks. In *EuroSys*, pages 59–72, 2007.