Biography
My name is Jianneng Li, and I am a EECS Master of Engineering student, with concentration in Data Science and Systems. I am primarily interested in building distributed processing systems, including many of the open source software such as Hadoop MapReduce, Apache Spark, and Apache Storm. I am also interested in developing high-performance applications to run on top of those frameworks.

I am taking CS267 to learn about the basic and fundamental concepts that form the building blocks of distributed and parallel computing. The knowledge will help me better understand the designs and rationales behind modern distributed computing frameworks. As for the project, I would love to work on something along the line of what I just described. I am also taking CS 289A Introduction to Machine Learning this semester, so I am open to the possibilities of join class project on an application of parallel computing in machine learning.

GraySort on Apache Spark
Sorting has always been a popular research area in computer science. As computers become more powerful, people are constantly trying to sort data faster and faster using a network of machines. The website sortingbenchmarks.org [4] is one such website dedicated to tracking the records in sorting. It has many metrics, such as GraySort, which asks for the fastest sort rate in terms of TBs per minute, CloudSort, which is the same as GraySort but on a public cloud, and JouleSort, which wants data to be sorted using as little energy as possible. Each benchmark also has two categories: Daytona requires the code to be general purpose, while Indy needs the software to only sort 100-byte records with 10-byte keys.

Apache Spark is a general purpose large-scale data processing engine that came out of the Berkeley AMPLab. Dubbed as a better version of Hadoop MapReduce, Spark has gained much attention since it was opened sourced in 2010. The core abstraction behind Spark is called Resilient DistributedDatasets (RDDs), which represent read-only collection of objects partitioned across a set of machines. RDDs contain lineage, making them easy to recover upon machine failures. More importantly, RDDs can be cached in memory, allowing fast data access that is crucial for iterative algorithms.

In November 2014, Databricks, the startup company developing Spark and related software, announced [2] that they tied first place for the 2014 Daytona GraySort contest. With 206 machines on Amazon’s public cloud, Spark sorted 100 TB of data in 23 minutes, averaging 4.27 TB/minute. The record beat the previous one held by Hadoop MapReduce by being 3 times faster while using 10 times fewer machines. The other winner had a rate of 4.35 TB/minute.
There is a good reason for Spark to invest in sorting as well. As explained by Databricks in a separate blog post [3], at the core of sorting is the shuffle operation, or the process of moving relevant data to the relevant machine. Shuffling in general is required for any data processing that perform aggregation. Since accessing data across the network is slower than in memory or even on disk, it is a good idea to minimize such movements with clever techniques.

In a blogpost as well as for the contest submission [2], Databricks explained the technical details behind the achievement. For example, a new shuffle algorithm was implemented to reduce the memory overhead, and the network module is decoupled from the execution module to minimize the effect on memory usage and stalls due to garbage collection. These changes allowed Spark to saturate the 10Gbps link between nodes, improving the system performance. In addition, to reduce cache misses, Databricks redesigned the layout of records in memory.

I believe that the application in this case achieved its objective. Judging from the network traffic shown above, the performance is very close to optimal. Sorting with Spark should scale to larger problem as well, as it is designed to be parallel and fault-tolerant. In fact, the Databricks blogpost showcased Spark sorting 1 PB of data using 190 machines in 234 minutes. The rate is 4.27 TB/s — same as the one for 100 TB.

References