**[Apache Spark: A Unified Engine for Big Data Processing](https://sit.instructure.com/courses/56018/assignments/316750)**

Apache Spark is a distributed processing engine that is open source and used for large data applications. It uses in-memory caching and efficient query execution to provide rapid queries against any quantity of data. Spark, in a nutshell, is a general-purpose engine for large-scale data processing. The capacity of Apache Spark to handle streaming data is its primary application case. With so much data being processed on a daily basis, it is critical for businesses to be able to stream and analyze it all in real-time, and Spark Streaming is capable of handling this additional effort. Many academics have continued to develop an integrated standard library over Spark, which includes capabilities ranging from data import to machine learning. According to literature reviews, the majority of Spark users mix various libraries in their applications. As parallel data processing becomes more frequent, one of the most critical considerations for both usability and speed will be the composability of processing units. A large portion of data analysis is exploratory, with users desiring to quickly integrate library functions into a workable pipeline.

However, in the case of "big data," transferring data across computers has an impact on performance. As a result, users want broad and composable abstractions. It, too, interacts with the system to spread data throughout the cluster and process it in parallel, just like Hadoop MapReduce. The intermediate processing data is saved in memory. It supports a variety of languages and has built-in APIs in Java, Scala, and Python. As a result, it can create applications in a variety of languages. For interactive querying, Spark provides 80 high-level operators. Spark stores intermediate calculations in memory, making it significantly quicker than Hadoop MapReduce.

Spark assesses RDDs slowly, which allows it to discover the most efficient plan for the user's calculation. Transformations, in particular, produce a new RDD object reflecting the result of a calculation but do not instantly calculate it. When an action is invoked, Spark examines the whole graph of transformations that were used to generate an execution plan. Ultimately, RDDs explicitly permit data sharing across calculations. RDDs are "ephemeral" by default, meaning they must be recomputed each time they are utilized in an action (such as count). Users can, however, choose to keep certain RDDs in memory or for quick reuse. RDDs also recover automatically from failures. Traditionally, fault tolerance in distributed computing systems has been achieved by data replication or check pointing. Each RDD keeps note of the transformation graph that was used to generate it and replicates these operations on the underlying data to reconstruct any missing partitions. This article demonstrates how to analyze the compute efficiency of Spark using SQL and other machine learning techniques. It emphasizes the relevance of Apache Spark composability in big data programming libraries and supports the creation of more readily interoperable libraries.