**Apache Spark Framework**

Apache Spark is a next generation batch processing framework with stream processing capabilities. Built using many of the same principles of Hadoop's MapReduce engine, Spark focuses primarily on speeding up batch processing workloads by offering full in-memory computation and processing optimization.

Spark can be deployed as a standalone cluster (if paired with a capable storage layer) or can hook into Hadoop as an alternative to the MapReduce engine.

Batch Processing Model

Unlike MapReduce, Spark processes all data in-memory, only interacting with the storage layer to initially load the data into memory and at the end to persist the final results. All intermediate results are managed in memory.

While in-memory processing contributes substantially to speed, Spark is also faster on disk-related tasks because of holistic optimization that can be achieved by analyzing the complete set of tasks ahead of time. It achieves this by creating Directed Acyclic Graphs, or **DAGs** which represent all of the operations that must be performed, the data to be operated on, as well as the relationships between them, giving the processor a greater ability to intelligently coordinate work.

To implement in-memory batch computation, Spark uses a model called called Resilient Distributed Datasets, or **RDDs**, to work with data. These are immutable structures that exist within memory that represent collections of data. Operations on RDDs produce new RDDs. Each RDD can trace its lineage back through its parent RDDs and ultimately to the data on disk. Essentially, RDDs are a way for Spark to maintain fault tolerance without needing to write back to disk after each operation.

Stream Processing Model

Stream processing capabilities are supplied by Spark Streaming. Spark itself is designed with batch-oriented workloads in mind. To deal with the disparity between the engine design and the characteristics of streaming workloads, Spark implements a concept called *micro-batches*\*. This strategy is designed to treat streams of data as a series of very small batches that can be handled using the native semantics of the batch engine.

Spark Streaming works by buffering the stream in sub-second increments. These are sent as small fixed datasets for batch processing. In practice, this works fairly well, but it does lead to a different performance profile than true stream processing frameworks.

Advantages and Limitations

The obvious reason to use Spark over Hadoop MapReduce is speed. Spark can process the same datasets significantly faster due to its in-memory computation strategy and its advanced DAG scheduling.

Another of Spark's major advantages is its versatility. It can be deployed as a standalone cluster or integrated with an existing Hadoop cluster. It can perform both batch and stream processing, letting you operate a single cluster to handle multiple processing styles.

Beyond the capabilities of the engine itself, Spark also has an ecosystem of libraries that can be used for machine learning, interactive queries, etc. Spark tasks are almost universally acknowledged to be easier to write than MapReduce, which can have significant implications for productivity.

Adapting the batch methodology for stream processing involves buffering the data as it enters the system. The buffer allows it to handle a high volume of incoming data, increasing overall throughput, but waiting to flush the buffer also leads to a significant increase in latency. This means that Spark Streaming might not be appropriate for processing where low latency is imperative.

Since RAM is generally more expensive than disk space, Spark can cost more to run than disk-based systems. However, the increased processing speed means that tasks can complete much faster, which may completely offset the costs when operating in an environment where you pay for resources hourly.

One other consequence of the in-memory design of Spark is that resource scarcity can be an issue when deployed on shared clusters. In comparison to Hadoop's MapReduce, Spark uses significantly more resources, which can interfere with other tasks that might be trying to use the cluster at the time. In essence, Spark might be a less considerate neighbor than other components that can operate on the Hadoop stack.