Spark FAQ

Q: Is Spark currently in use in any major applications?

A: Yes: https://databricks.com/customers

Q: How common is it for PhD students to create something on the scale of Spark?

A: Unusual! It is quite an accomplishment. Matei Zaharia won the ACM doctoral thesis award for this work.

Q: Should we view Spark as being similar to MapReduce?

A: There are similarities, but Spark can express computations that are difficult to express in a high-performance way in MapReduce, for example iterative algorithms. You can think of Spark as MapReduce and more. The authors argue that Spark is powerful enough to express a range of prior computation frameworks, including MapReduce (see section 7.1).

There are systems that are better than Spark in incorporating new data that is streaming in, instead of doing batch processing. For example, Naiad (https://dl.acm.org/citation.cfm?id=2522738). Spark also has streaming support, although it implements it in terms of computations on "micro-batches" (see Spark Streaming, SOSP 2013).

There are also systems that allow more fine-grained sharing between different machines (e.g., DSM systems) or a system such as Picolo (http://piccolo.news.cs.nyu.edu/), which targets similar applications as Spark.

Q: Why are RDDs called immutable if they allow for transformations?

A: A transformation produces a new RDD, but it may not be materialized explicitly, because transformations are computed lazily. In the PageRank example, each loop iteration produces a new distrib and rank RDD, as shown in the corresponding lineage graph.

Q: Do distributed systems designers worry about energy efficiency?

A: Energy is a big concern in CS in general! But most of the focus in energy-efficiency in cluster computing goes to the design of data centers and the design of the computers and cooling inside the data center.

Chip designers pay lots of attention to energy; for example, your processor dynamically changes the clock rate to avoid making the processor too hot.

There is less focus on energy in the design of distributed systems, mostly, I think, because that is not where the big wins are. But there is some work, for example see http://www.cs.cmu.edu/~fawnproj/.

Q: How do applications figure out the location of an RDD?

A: The application names RDDs with variable names in Scala. Each RDD has location information associated with it, in its metadata (see Table 3). The scheduler uses all this information to colocate computations with the data. An RDD may be generated by multiple nodes but it is for different partitions of the RDD.

Q: How does Spark achieve fault tolerance?

A: When persisting an RDD, the programmer can specify that it must be
replicated on a few machines. Spark doesn't need complicated protocols
like Raft, however, because RDDs are immutable and can always be
recomputed using the lineage graph.

Q: Why is Spark developed using Scala? What's special about the language?

A: In part because Scala was new and hip when the project started.

One good reason is that Scala provides the ability to serialize and ship
user-defined code ("closures") as discussed in Â§5.2.

This is something that is fairly straightforward in JVM-based languages
(such as Java and Scala), but tricky to do in C, C++ or Go, partly
because of shared memory (pointers, mutexes, etc.) and partly because the
closure needs to capture all variables referred to inside it (which is
difficult unless a language runtime can help with it).

Q: Does anybody still use MapReduce rather than Spark, since Spark seems
to be strictly superior? If so, why do people still use MR?

If the computation one needs fits the MapReduce paradigm well, there is
no advantage to using Spark over using MapReduce.

For example if the computation does a single scan of a large data set
(map phase) followed by an aggregation (reduce phase), the computation
will be dominated by I/O and Spark's in-memory RDD caching will offer
no benefit since no RDD is ever re-used.

Spark does very well on iterative computations with a lot of internal
reuse, but has no architectural edge over MapReduce or Hadoop for simple
jobs that scan a large data set and aggregate (i.e., just map() and
reduceByKey() transformations in Spark speak). On the other hand,
there's also no reason that Spark would be slower for these computations,
so you could really use either here.

Q: Is the RDD concept implemented in any systems other than Spark?

Spark and the specific RDD interface are pretty intimately tied to each
other. However, two key ideas behind RDDs -- deterministic, lineage-based
re-execution and the collections-oriented API -- are certainly widely-used
in many systems. For example, DryadLINQ, FlumeJava, and Cloud Dataflow
offer similar collection-oriented APIs; and the Dryad and Ciel systems
referenced by the paper also keep track of how pieces of data are
computed, and re-execute that computation on failure, similar to
lineage-based fault tolerance.

As a matter of fact, RDDs themselves in Spark are now somewhat
deprecated: Spark has recently moved towards something called "DataFrames"
(https://spark.apache.org/docs/latest/sql-programming-guide.html#datasets-and-dataframes),
which implements a more column-oriented representation while maintaining
the good ideas from RDDs.

Q: What is hash partitioning?

Hash partitioning is an idea from databases to implement database
joins efficiently.  Splitting an RDD into partitions using a hash,
means hashing the primary key of an RDD and storing all the records
with the same hash in the same partition.  If you have two RDDs with
the same primary keys, and you hash-partition them, then the
partitions of the two RRDs with the same keys will end up on the same
machine.  This is nice if you need to compute a join() on these two
RRDs because the join() will not require communication with other
machines, because the partitions with the same key are on the same
machine.

This shows up, for example, in the PageRank example with the links and ranks RRDs. Their keys are URLs and hash partitioning ensure that entry for "mit.edu" for both RRDs end up on the same machine. So, when joining the two RDDs on "mit.edu", no communication is necessary with another machine.