RDD stands for Resilient Distributed Dataset. If this data is loaded into memory, it can significantly boast performance since the data is reused and hence data eviction will be considerably less as compared to other data. RDD is integrated into Sparks and is a core component of it. RDD provides a programming interface to perform parallel computation on large-scale data in distributed manner using in-memory. RDDs are designed keeping specific use case in mind such as iterative and interactive data mining tool where keeping the data in memory reduces the round trip from the disk significantly and boast the performance as recomputing from the disk is costly. RDD also supports shared memory but provides only coarse grained transformation(such as map, filter, reduce) for fault tolerance unlike DSM that can be used for fined grained transformation. Spark uses the Scala interpreter(consist of 14000 lines of code) and has shown significant performance improvement over Hadoop for given use case.
The RDD abstraction is a read-only partitioned collection of records that can be parallel processed to large number of cluster. Since the RDD are read-only, it helps with concurrency control and makes caching significantly easy(no dirty eviction). RDD contains the lineage information in the form of DAG that aids to the fault tolerance mechanism. Using the lineage information, lost partitions can be recovered using the lineage graph. It also supports checkpointing since recovery may be time consuming for RDD with long lineage chain. Currently the checkpointing is provided via the API but it also plans to add auto checkpointing feature.
RDD has a language integrated api and can perform the following operations:
Transformation such as map, filter, flatMap, reduceBykey and groupByKey to create new RDD from existing.
Actions such as count, collect, reduce, lookup and save that returns a value.
In sparks, there are two kinds of dependencies that define the relationship between RDD partitions:
Narrow dependencies where the partition is depend on one or few other partition and can the job can be performed in parallel fashion such as map, filter and reduce.
In wide dependencies, there's a dependency between partitions and can cause typical overhead. Typical operations involves groupByKey and joins(unless hash-partitioned.
If a particular task fails, rdd re-runs it on another node. Scheduler failures are not supported yet(they claim that its easier to implement though).
RDD is a core component of Spark. It is a powerful framework for particular use cases mentioned and outperforms Hadoop by a significant margin. It exposes a strong api to the user and performs features such as partitioning, in memory caching using the persist flag, lineage etc. Hence it is a reliable and fault tolerant robust framework

Comments:
Apart from the general comparison from RDD and DSM discussed in the paper, I feel security aspect was also one of the leveraged RDD has over DSM that should have been discussed.
One of the key reason why Sparks perform better than Hadoop is because the data is loaded in-memory. Lets assume that if a technology like intel optane had worked out, would Hadoop be more relevant as it could possibly perform similar to Sparks in benchmarks? Rdd does have other benefits and features but I am curious if something like this could have killed sparks.
Since RDDS are serialized and deserialized over the network, can that overhead be reduced if an open source format like parquet is used.