Cluster computing frameworks like MapReduce and Dryad have been widely adopted for large-scale data analytics. These systems let users write parallel computations using a set of high-level operators, without having to worry about work distribution and fault tolerance. But these systems are inefficient for an important class of emerging applications: those that reuse intermediate results across multiple computations. In these frameworks the only way to reuse data between computations (e.g., between two MapReduce jobs) is to write it to an external stable storage. This is a substantial overhead due to data replication, disk I/O, and serialization, which can dominate application execution times. In order to address this problem some specialized frameworks for some applications were designed like Pregel and HaLoop. However, these frameworks only support specific computation patterns and do not provide abstractions for more general reuse of intermediate data. resilient distributed datasets (RDDs) is a new abstraction that enables efficient data reuse in a broad range of applications. RDDs are fault-tolerant, parallel data structures that let users explicitly persist intermediate results in memory, control their partitioning to optimize data placement, and manipulate them using a rich set of operators.

(1) “...individual RDDs are immutable...” What does it mean by being “immutable”? What benefits does this property of RDD bring?
   It means after they are created, they cannot be modified anymore.
   It makes it easier to describe the lineage graph and as a result makes the recovery process simpler and in turn the cost for fault tolerance characteristic would be reduced.

(2) When an RDD is being created (new data are being written into it), can the data in the RDD be read for computing before the RDD is completed created?
   No, because in that point of time the data set is not completely settled and cannot be used in other operations.
“This allows them to efficiently provide fault tolerance by logging the transformations used to build a dataset (its lineage) rather than the actual data.” “To achieve fault tolerance efficiently, RDDs provide a restricted form of shared memory, based on coarse-grained transformations rather than fine-grained updates to shared state.” Why does using RDD help to provide efficient fault tolerance? Or why does coarse-grained transformation help with the efficiency?

By using coarse-grained transformation each data set can be represented by the input data and a set of transformations which is going to apply to many data records. This makes the rebuilding process for a lost data set fairly simple and low cost because there is no need to store the actual data set contents for recovery.

What is difference between transformation and action?

Actions are operations that return a value to the application or export data to a storage system while transformations apply the same operation to many data items.

“In addition, programmers can call a persist method to indicate which RDDs they want to reuse in future operations.” What’s the consequence if a user does not explicitly request persistence of an RDD?

In this case the RDD would be discarded from the memory which means if it was needed in future calculation, it should be reproduced again.

Explain Figure 1 about a lineage graph.

At the top, the ‘lines’ RDD is positioning, then by using ‘filter(_.startsWith("ERROR"))’ transformation, the ‘errors’ RDD is created. Then
by using a second ‘filter(_.contains("HDFS"))’ transformation, the ‘HDFS errors’ RDD is produced and at the end by using the final ‘map(_.split("\t")(3))’ transformation, the final ‘time fields’ RDD is created.

It should be noted that any RDD from this graph can be recovered and reproduced using only the input RDD and the series of transformation operations.