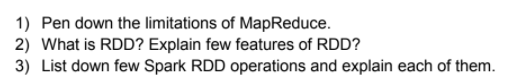


The problem statement



The problem 1:



Some use cases where MapReduce does not work very well :

1. When you need a response fast. e.g. say < few seconds (Use stream processing, CEP etc instead)
2. Processing graphs
3. Complex algorithms - some machine learning algorithms like SVM, and also see 13 drawfs ([The Landscape of Parallel Computing Research: A View From Berkeley](http://www.eecs.berkeley.edu/Pubs/TechRpts/2006/EECS-2006-183.pdf))
4. Iterations - when you need to process data again and again. e.g. KMeans - use Spark
5. When map phase generate too many keys. Then sorting takes for ever
6. Joining two large data sets with complex conditions (equal case can be handled via hashing etc)
7. Stateful operations - e.g. evaluate a state machine
8. Cascading tasks one after the other - using Hive, Pig might help, but lot of overhead rereading and parsing data.

The problem 2:



An RDD(Resilient Distributed Datasets) is, essentially, the Spark representation of a set of data, spread across multiple machines, with APIs to let you act on it. An RDD could come from any datasource, e.g. text files, a database via JDBC, etc.

**Features :**

-fault-tolerant

-parallel data structures that let users explicitly persist intermediate results in memory

-control their partitioning to optimize data placement

-manipulate them using a rich set of operators.

The problem 3:



#### **++(other:** [**RDD**](https://spark.apache.org/docs/0.6.1/api/core/spark/RDD.html)**[T]):** [**RDD**](https://spark.apache.org/docs/0.6.1/api/core/spark/RDD.html)**[T]**

Return the union of this RDD and another one. Any identical elements will appear multiple times (use .distinct() to eliminate them).

#### **aggregate[U](zeroValue: U)(seqOp: (U, T) ⇒ U, combOp: (U, U) ⇒ U)(*implicit* arg0: ClassManifest[U]): U**

Aggregate the elements of each partition, and then the results for all the partitions, using given combine functions and a neutral "zero value".

#### **cache():** [**RDD**](https://spark.apache.org/docs/0.6.1/api/core/spark/RDD.html)**[T]**

Persist this RDD with the default storage level (MEMORY\_ONLY).

#### **cartesian[U](other:** [**RDD**](https://spark.apache.org/docs/0.6.1/api/core/spark/RDD.html)**[U])(*implicit* arg0: ClassManifest[U]):** [**RDD**](https://spark.apache.org/docs/0.6.1/api/core/spark/RDD.html)**[(T, U)]**

Return the Cartesian product of this RDD and another one, that is, the RDD of all pairs of elements (a, b) where a is in this and b is in other.

#### **collect(): Array[T]**

Return an array that contains all of the elements in this RDD.

#### **context: [SparkContext](https://spark.apache.org/docs/0.6.1/api/core/spark/SparkContext.html)**

The [SparkContext](https://spark.apache.org/docs/0.6.1/api/core/spark/SparkContext.html) that this RDD was created on.

#### **count(): Long**

Return the number of elements in the RDD.

#### **countApprox(timeout: Long, confidence: Double = 0.95):** [**PartialResult**](https://spark.apache.org/docs/0.6.1/api/core/spark/partial/PartialResult.html)**[**[**BoundedDouble**](https://spark.apache.org/docs/0.6.1/api/core/spark/partial/BoundedDouble.html)**]**

(Experimental) Approximate version of count() that returns a potentially incomplete result within a timeout, even if not all tasks have finished.

#### **countByValue(): Map[T, Long]**

Return the count of each unique value in this RDD as a map of (value, count) pairs.

#### **countByValueApprox(timeout: Long, confidence: Double = 0.95):** [**PartialResult**](https://spark.apache.org/docs/0.6.1/api/core/spark/partial/PartialResult.html)**[Map[T,** [**BoundedDouble**](https://spark.apache.org/docs/0.6.1/api/core/spark/partial/BoundedDouble.html)**]]**

(Experimental) Approximate version of countByValue().

#### **distinct(numSplits: Int =** [**splits.size**](https://spark.apache.org/docs/0.6.1/api/core/spark/RDD.html#splits:Array%5BSplit%5D)**):** [**RDD**](https://spark.apache.org/docs/0.6.1/api/core/spark/RDD.html)**[T]**

Return a new RDD containing the distinct elements in this RDD.

#### **filter(f: (T) ⇒ Boolean):** [**RDD**](https://spark.apache.org/docs/0.6.1/api/core/spark/RDD.html)**[T]**

Return a new RDD containing only the elements that satisfy a predicate.

#### **first(): T**

Return the first element in this RDD.

#### **flatMap[U](f: (T) ⇒ TraversableOnce[U])(*implicit* arg0: ClassManifest[U]):** [**RDD**](https://spark.apache.org/docs/0.6.1/api/core/spark/RDD.html)**[U]**

Return a new RDD by first applying a function to all elements of this RDD, and then flattening the results.

#### **fold(zeroValue: T)(op: (T, T) ⇒ T): T**

Aggregate the elements of each partition, and then the results for all the partitions, using a given associative function and a neutral "zero value".

#### **foreach(f: (T) ⇒ Unit): Unit**

Applies a function f to all elements of this RDD.

#### **getStorageLevel: [StorageLevel](https://spark.apache.org/docs/0.6.1/api/core/spark/storage/StorageLevel.html)**

Get the RDD's current storage level, or StorageLevel.NONE if none is set.

#### **glom():** [**RDD**](https://spark.apache.org/docs/0.6.1/api/core/spark/RDD.html)**[Array[T]]**

Return an RDD created by coalescing all elements within each partition into an array.

#### **groupBy[K](f: (T) ⇒ K)(*implicit* arg0: ClassManifest[K]):** [**RDD**](https://spark.apache.org/docs/0.6.1/api/core/spark/RDD.html)**[(K, Seq[T])]**

Return an RDD of grouped items.

#### **groupBy[K](f: (T) ⇒ K, numSplits: Int)(*implicit* arg0: ClassManifest[K]):** [**RDD**](https://spark.apache.org/docs/0.6.1/api/core/spark/RDD.html)**[(K, Seq[T])]**

Return an RDD of grouped elements.

#### **Val id: Int**

A unique ID for this RDD (within its SparkContext).

#### **final def iterator(split:** [**Split**](https://spark.apache.org/docs/0.6.1/api/core/spark/Split.html)**): Iterator[T]**

Internal method to this RDD; will read from cache if applicable, or otherwise compute it.

#### **map[U](f: (T) ⇒ U)(*implicit* arg0: ClassManifest[U]):** [**RDD**](https://spark.apache.org/docs/0.6.1/api/core/spark/RDD.html)**[U]**

Return a new RDD by applying a function to all elements of this RDD.

#### **mapPartitions[U](f: (Iterator[T]) ⇒ Iterator[U])(*implicit* arg0: ClassManifest[U]):** [**RDD**](https://spark.apache.org/docs/0.6.1/api/core/spark/RDD.html)**[U]**

Return a new RDD by applying a function to each partition of this RDD.

#### **mapPartitionsWithSplit[U](f: (Int, Iterator[T]) ⇒ Iterator[U])(*implicit* arg0: ClassManifest[U]):** [**RDD**](https://spark.apache.org/docs/0.6.1/api/core/spark/RDD.html)**[U]**

Return a new RDD by applying a function to each partition of this RDD, while tracking the index of the original partition.

#### **partitioner: Option[**[**Partitioner**](https://spark.apache.org/docs/0.6.1/api/core/spark/Partitioner.html)**]**

Optionally overridden by subclasses to specify how they are partitioned.

#### **persist():** [**RDD**](https://spark.apache.org/docs/0.6.1/api/core/spark/RDD.html)**[T]**

Persist this RDD with the default storage level (MEMORY\_ONLY).

#### **persist(newLevel:** [**StorageLevel**](https://spark.apache.org/docs/0.6.1/api/core/spark/storage/StorageLevel.html)**):** [**RDD**](https://spark.apache.org/docs/0.6.1/api/core/spark/RDD.html)**[T]**

Set this RDD's storage level to persist its values across operations after the first time it is computed.

#### **pipe(command: Seq[String], env: Map[String, String]):** [**RDD**](https://spark.apache.org/docs/0.6.1/api/core/spark/RDD.html)**[String]**

Return an RDD created by piping elements to a forked external process.

#### **pipe(command: Seq[String]):** [**RDD**](https://spark.apache.org/docs/0.6.1/api/core/spark/RDD.html)**[String]**

Return an RDD created by piping elements to a forked external process.

#### **pipe(command: String):** [**RDD**](https://spark.apache.org/docs/0.6.1/api/core/spark/RDD.html)**[String]**

Return an RDD created by piping elements to a forked external process.

#### **preferredLocations(split:** [**Split**](https://spark.apache.org/docs/0.6.1/api/core/spark/Split.html)**): Seq[String]**

Optionally overridden by subclasses to specify placement preferences.

#### **reduce(f: (T, T) ⇒ T): T**

Reduces the elements of this RDD using the specified associative binary operator.

#### **sample(withReplacement: Boolean, fraction: Double, seed: Int):** [**RDD**](https://spark.apache.org/docs/0.6.1/api/core/spark/RDD.html)**[T]**

Return a sampled subset of this RDD.

#### **saveAsObjectFile(path: String): Unit**

Save this RDD as a SequenceFile of serialized objects.

#### **saveAsTextFile(path: String): Unit**

Save this RDD as a text file, using string representations of elements.

#### **take(num: Int): Array[T]**

Take the first num elements of the RDD.

#### **takeSample(withReplacement: Boolean, num: Int, seed: Int): Array[T]**

#### **toArray(): Array[T]**

Return an array that contains all of the elements in this RDD.

#### **union(other:** [**RDD**](https://spark.apache.org/docs/0.6.1/api/core/spark/RDD.html)**[T]):** [**RDD**](https://spark.apache.org/docs/0.6.1/api/core/spark/RDD.html)**[T]**

Return the union of this RDD and another one.