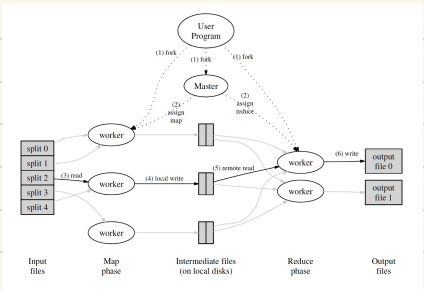
SPARK

task parallelism - different computations on same data

data parallelism - same computation on dataset partitions

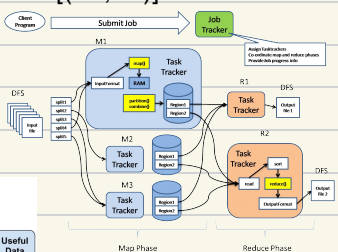
latency: 1000 (disk), 1000000 (network) x slower operation than access



Map/Reduce

map((K1,V1), f: (K1,V1) -> (K2,V2)): List[(K2,V2)]

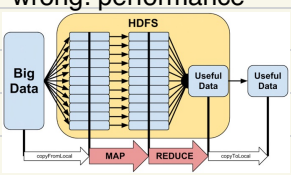
reduce((K2,List[V2])): List[(K3,V3)]



Hadoop

good: fault tolerance

wrong: performance



Spark

- automates distribution of data and computations on cluster of computers
- fault-tolerant abstraction to distributed datasets
- based on FP primitives
- 2 abstractions to data: list-like (RDDs) and table-like (datasets)

Resilient Distributed Datasets (RDDs)

- immutable
- reside in memory
- transparently distributed
- feature all FP primitives
- Transformation: apply function returning RDD (lazy)
- Action: request computation of result (eager)

RDD[A]

Transformations:

map - apply f on all items

RDD[A].map(f: A -> B): RDD[B]

flatMap - apply f on all RDD contents

RDD[A].flatMap(f: A -> Iterable[B]): RDD[B]

filter - apply predicate p

RDD[A].filter(p: A -> Boolean): RDD[A]

Actions:

collect - return all elements

RDD[A].collect(): Array[A]

take - return first n elements

RDD[A].take(n): Array[A]

reduce, fold - combine elements to a single result

RDD[A].reduce(f: (A,A) -> A): A

aggregate - aggregate elements of each partition

RDD[A].aggr(init: B)(seqOp: (B, A) -> B, combOp: (B, B) -> B): B

RDD[(K, V)]

Transformations:

reduceByKey - merge values for each key

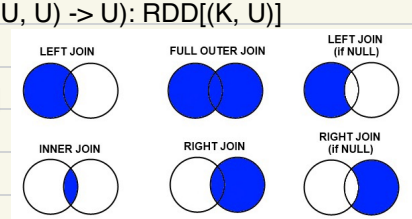
reduceByKey(f: (V, V) -> V): RDD[(K, V)]

aggregateByKey - aggregate the values for each key

aggrByKey(zero: U)(f: (U, V) -> U, g: (U, U) -> U): RDD[(K, U)]

join - pair elements with matching keys

join(b: RDD[(K, W)]): RDD[(K, (V, W))]



Join Types

left: RDD[(K, A)] right: RDD[(K, B)]

inner join (join) - only records that have keys in both RDDs

outer joins (left/rightOuterJoin) - records that have keys in left/right RDD [loj/roj]

full outer join - records that have keys in either left or right RDD [foj]