Big data platform (Spark) performance acceleration

*Mentors: Tony Tan, Ning Wu, Yong Wang and **Theo Gkountouvas***

By:

Grishma Atul Thakkar

Virat Goradia

Nipun Midha

Baoshu Brady Qi

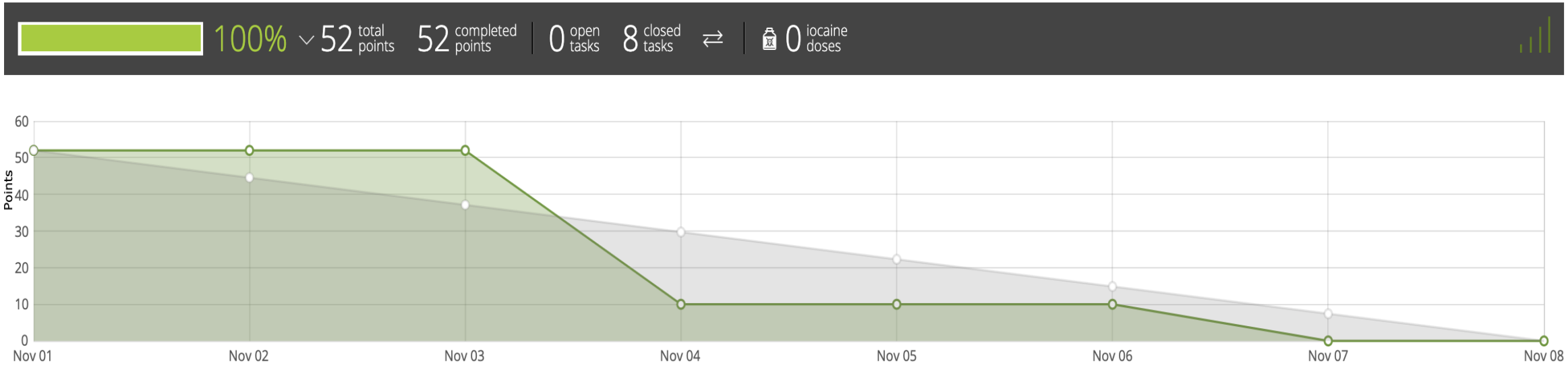
Recap

1. Spark's Shuffle Writer
2. Problem with original Shuffle writer
3. How riffle optimizes Spark's performance
4. Logical plan flow of the Spark
5. And we went in detail to see the flow of Spark's internal.

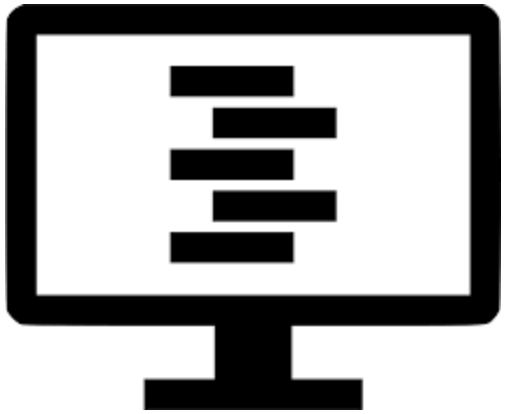
Sprint Goals

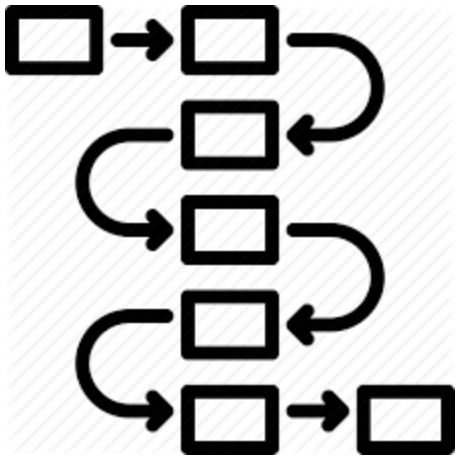
- Design strategies to implement N-Way merge algorithm
- Start implementing the N-Way merge algorithm.

Burndown Chart

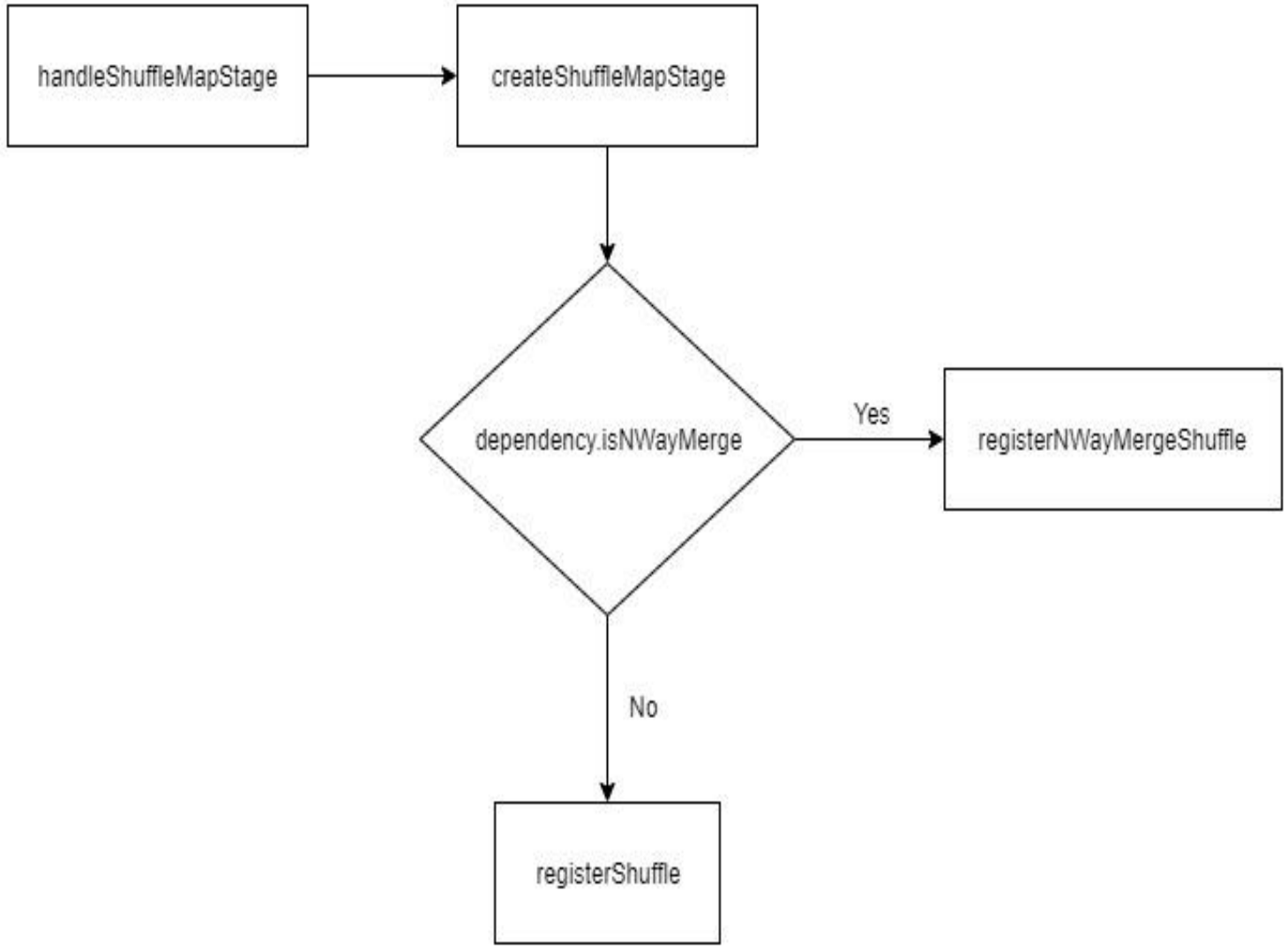


Challenges

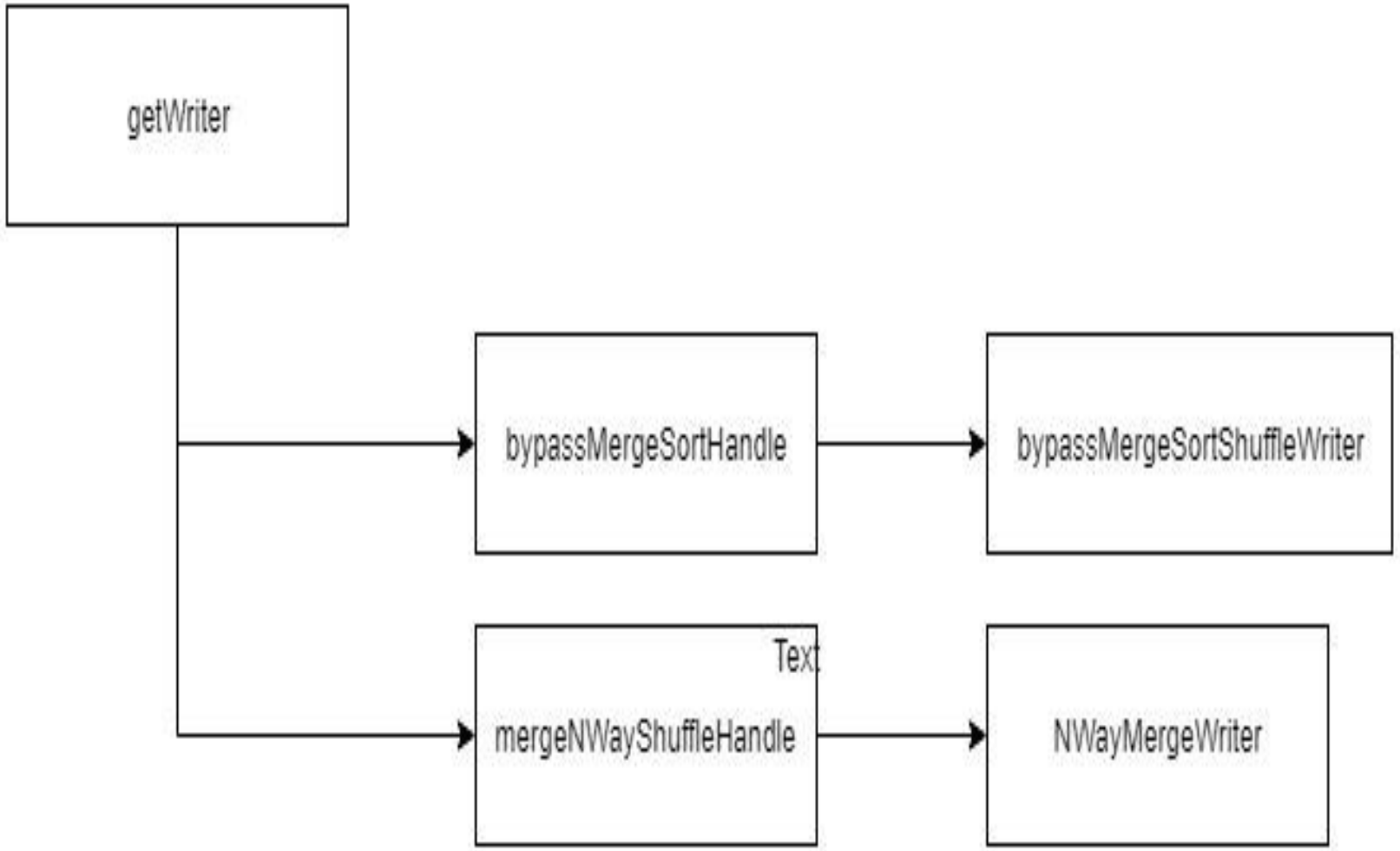




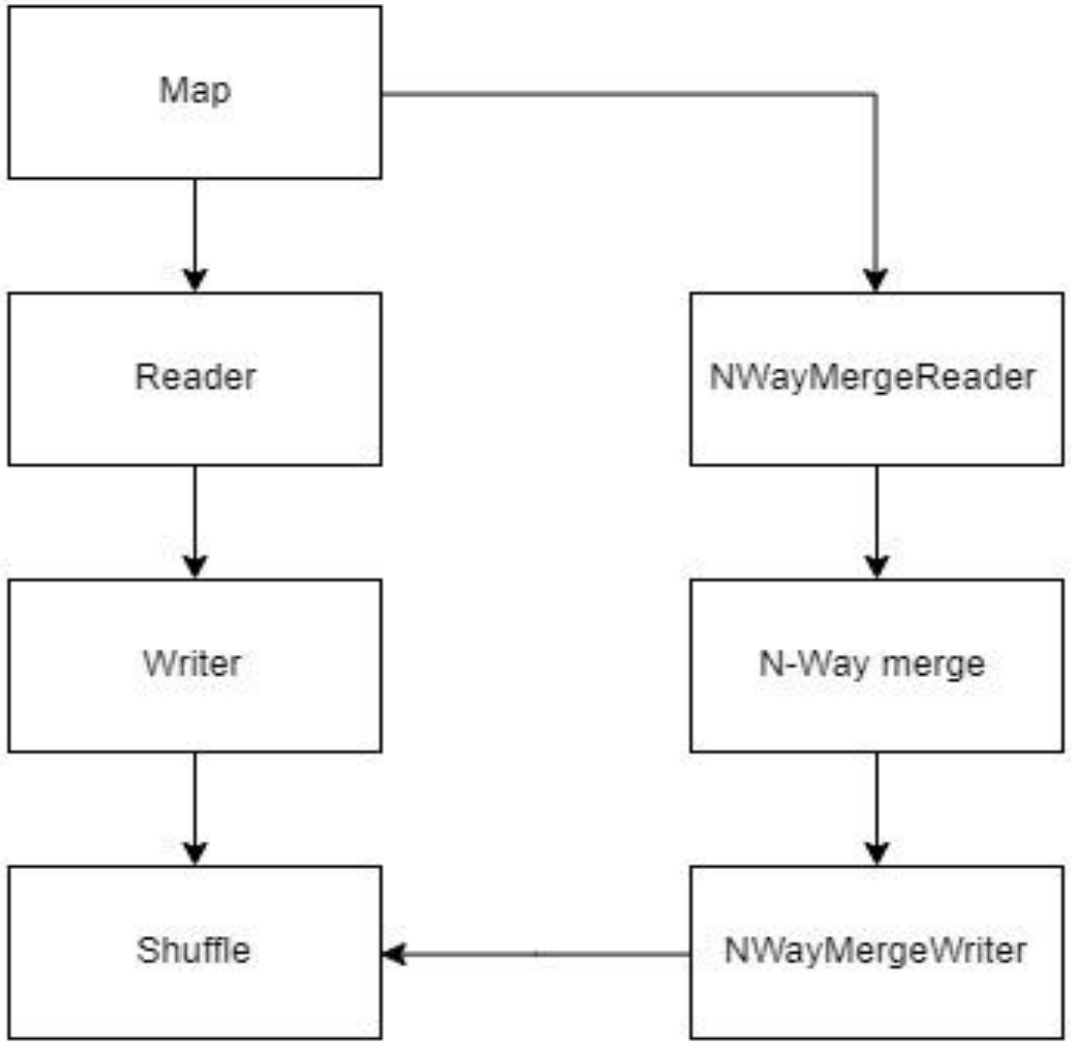
Our design (DAGScheduler)



Our design (SortShuffle Manager)



Our design (Achieved so far)



registerNWayMergeShuffle

```
def registerNWayMergeShuffle(shuffleId: Int, numMaps: Int, N: Int): Unit = {
  if (shuffleStatuses.put(shuffleId, new ShuffleStatus(numMaps/N + numMaps%N)).isDefined) {
    throw new IllegalArgumentException("Shuffle ID " + shuffleId + " registered twice")
  }
}
```

SortShuffleManager

```scala
/**
 * Obtains a [[ShuffleHandle]] to pass to tasks.
 */
override def registerShuffle[K, V, C](
    shuffleId: Int,
    dependency: ShuffleDependency[K, V, C]): ShuffleHandle = {

  if(SortShuffleWriter.shouldNWayMerge(conf, dependency)) {
    //If we want to decrease the number of partitions read by shuffle reader, we do a N-Way merge
    new NWayMergeHandle[K, V](
      shuffleId, dependency.asInstanceOf[ShuffleDependency[K, V, V]])
```

```scala
  case mergeNWayShuffleHandle: NWayMergeHandle[K @unchecked, V @unchecked] =>
    new NWayMergeWriter(
      env.blockManager,
      mergeNWayShuffleHandle,
      mapId,
      env.conf,
      metrics,
      shuffleExecutorComponents)
```

SortShuffleWriter

```scala
private[spark] object SortShuffleWriter {
  def shouldBypassMergeSort(conf: SparkConf, dep: ShuffleDependency[_, _, _]): Boolean = {
    // We cannot bypass sorting if we need to do map-side aggregation.
    if (dep.mapSideCombine) {
      false
    } else {
      val bypassMergeThreshold: Int = conf.get(config.SHUFFLE_SORT_BYPASS_MERGE_THRESHOLD)
      dep.partitioner.numPartitions <= bypassMergeThreshold
    }
  }

  def shouldNWayMerge(conf: SparkConf, dep: ShuffleDependency[_, _, _]): Boolean = {
    // We cannot bypass sorting if we need to do map-side aggregation.
    dep.isNWayMerge
  }
}
```

NWayMergeHandle

```scala
/**
 * Subclass of [[BaseShuffleHandle]], used to identify when we've chosen to use the
 * N-Way merge shuffle path.
 */
private[spark] class NWayMergeHandle[K, V](
    shuffleId: Int,
    dependency: ShuffleDependency[K, V, V])
  extends BaseShuffleHandle(shuffleId, dependency) {
}
```

NWayMergeWriter

Concrete ShuffleWriter that ShuffleMapTask uses to write records into one single shuffle block data file when the task runs for a ShuffleDependency

Created exclusively when SortShuffleManager selects a writer for a NwayMergeHandle

Write – Writing records into One Single Shuffle Block Data File

Write – Concatenating Per-Partition Files Into Single File (and Tracking Write Time)

Why is this flawed?

Reducers will solely depend on the Merger output, final result would be incorrect if some merge tasks failed even if mapper tasks are all correct.

Our intentions:

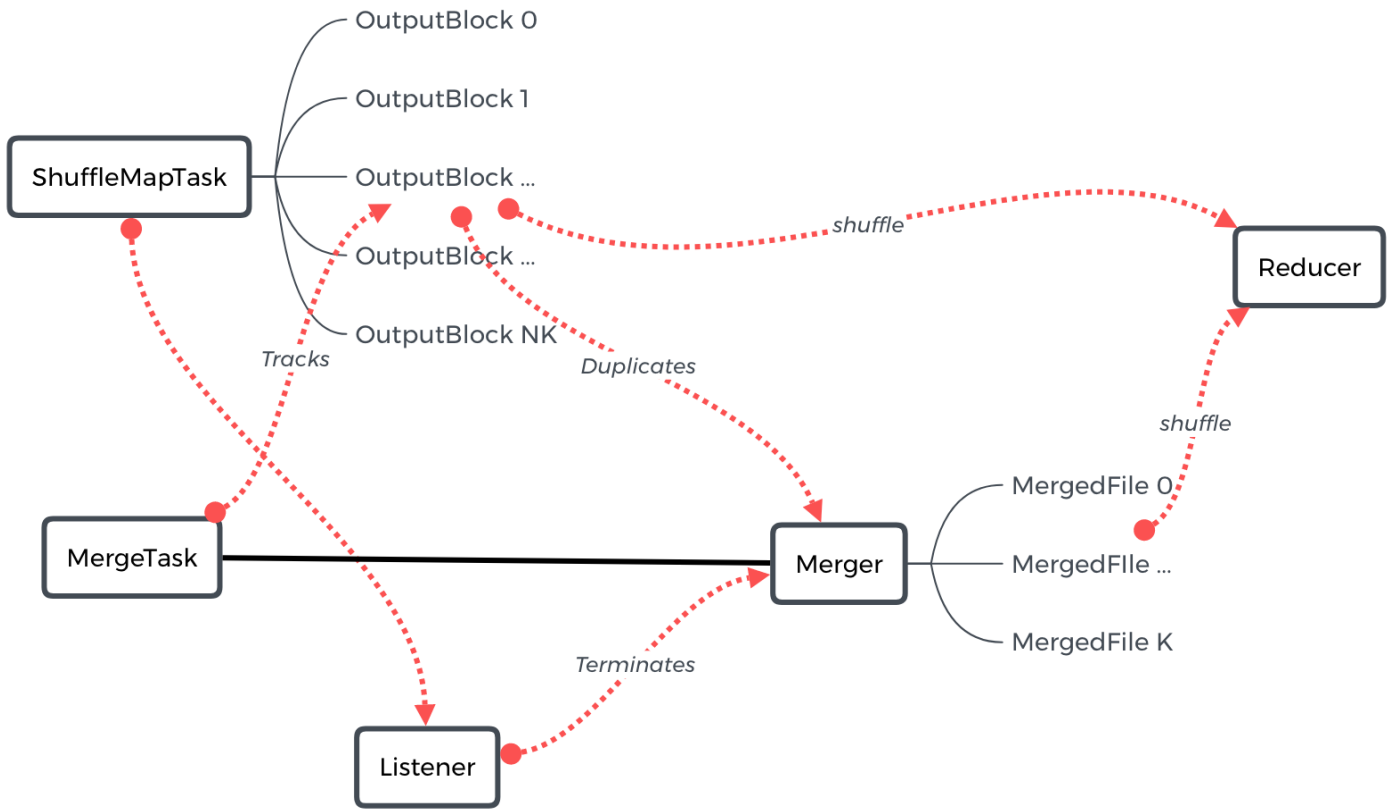
1. Maintain the valid output of mapper.
2. Reduce the number of requests in shuffling phase by combining small files into larger ones.
3. Merge should be terminated almost immediately after all Mappers finish.

Solution

1. Map task and merge task start at the same time

2. Merge Task tracks the mapout put, whenever it meet the certain condition (e.g. N blocks), it starts to merge blocks together

3. Metadata of both merged files and origin files will be send to reducers and reducers will figure out if merged files are available



ShuffleBlockFetcherIterator

Old:

Address of Blocks: blocksByAddress: Iterator[(BlockManagerId, Seq[(BlockId, Long, Int)])]
Merge Continuous Blocks: mergeContinuousShuffleBlockIdsIfNeeded(curBlocks)

Number of fetch request depends on the number of discontinuous blocks

Our Implementation:

Merge blocks from different Mappers, reduce the number of fetch request more effeciently

Address of Blocks: mergedBlockByAddress: Iterator[(BlockManagerId, Seq[((BlockId, Long, Int), Seq[(BlockId, Long, Int)])], Seq[(BlockId, Long, Int)])],
Check Validation: Size check

Next Sprint Goals

- Change our implementation and use observer pattern

Any Questions?

Thank You!