

# Project Final Report

*(DPSort : Distributed, Parallel External MergeSort)*

Jinho Ko

[jinho.ko@postech.ac.kr](mailto:jinho.ko@postech.ac.kr)

# Result Summary

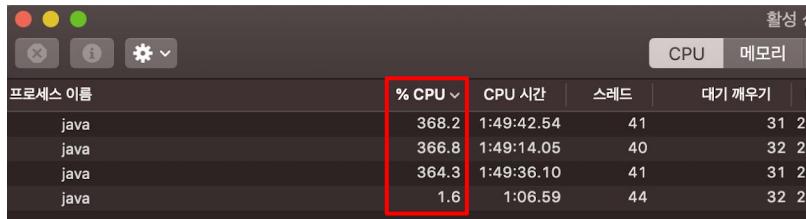
- Overall pipeline works very well with enough robustness
  - Implemented ~4,000 lines across ~50 code files
- Succeeded to sort :
  - **36GB** of unordered files (*couldn't expand more due to environment issue, but it will work for bigger sizes for sure*)
  - 3 workers, 4 threads/worker (tot. 12 threads)

```
jinho@jinho-PC:/media/jinho/DATA/cs434/cs434-output$ ./valsort -s all_36gb.com
Records: 360000000
Checksum: aba91754a81b86b
Duplicate keys: 0
SUCCESS - all records are in order
jinho@jinho-PC:/media/jinho/DATA/cs434/cs434-output$ 
```

- Demo video : <https://www.youtube.com/watch?v=Kq2B0tbqGiE>

# Profiling Analysis

- (+) Good exploitation of multi-threads ( 4 threads / worker )
- (+) Limited use of memory ( total block size / worker = 400MB )
  - Likely to scale enough for larger inputs
- (-) Highly dependent to disk throughput



프로세스 이름	% CPU	CPU 시간	스레드	대기 깨우기	P
java	368.2	1:49:42.54	41	31	27
java	366.8	1:49:14.05	40	32	27
java	364.3	1:49:36.10	41	31	28
java	1.6	1:06.59	44	32	27



프로세스 이름	메모리	스레드	포트	PID	사용자
java	2.53GB	45	129	22327	jinho
java	1.33GB	58	155	22615	jinho
java	1.32GB	58	157	22422	jinho
java	1.32GB	56	151	22518	jinho

# Core Design Concept

- Master is in charge of everything!
  - Even for the partition list of all workers!
- Worker is just a machine that receives a tasks and reports the result
  - Goal : to make worker stateless as much as possible
    - Input : Task message ( task type, input partition )
    - Output : Task result message ( Success/Failure, sampled output)
  - Should be robust to task failure

# Terminology

**Stage** := An execution unit in the program workflow

**Task** := An execution unit in each worker thread to be executed. (*A minimal unit of input!*) Typically, a task is assigned a single partition as its input.

# Overall Program Pipeline

Stage	Task Used	What task does
GenBlockStage	GenBlockTask	<ul style="list-style-type: none"><li>- given a data file, split into a set of smaller partitions</li></ul>
LocalSortStage	LocalSortTask	<ul style="list-style-type: none"><li>- given a partition, perform sorting</li></ul>
SampleKeysStage	SampleKeysTask	<ul style="list-style-type: none"><li>- given a partition and sample ration, scan and sample</li><li>- return the sampled data*</li></ul>
PartitionAndShuffleStage	PartitionAndShuffleTask	<ul style="list-style-type: none"><li>- given a partition and partitioning function, split partition into blocks</li><li>- write 1 partition locally and shuffle out the rest</li></ul>
MergeStage**	MergeTask	<ul style="list-style-type: none"><li>- given 2 partitions, merge them linearly and save</li></ul>
TerminateStage	TerminateTask	<ul style="list-style-type: none"><li>- if whole pipeline succeeds, move output partitions to output directory. otherwise, delete work directory</li></ul>



\* After the sampled data is aggregated by master, it will generate partition function and pass it to the next task

\*\* Note that MergeStage will repeatedly executed until single partition remains to each worker

# Considerations for Parallelism / Limited Resource

- Controllable performance parameters
- Resource-aware implementations / algorithms
  - A Must-follow rule => **Each worker thread should not occupy more than the size of one block!**
- Concurrency control
  - Thread-safe singleton objects

# Performance Parameters

Controllable performance parameters :

- `dpsort.master.blockSizeInLines`
  - `dpsort.master.maxSampleSize`
  - `dpsort.master.maxSampleRatio`
- 
- `dpsort.worker.threads`
  - `dpsort.worker.maxShuffleConnections`
  - `dpsort.worker.maxShuffleMsgBytes`

*\*These values increases as data / #workers grows  
=> Must be controlled to some level*

# Memory Constraint

Each worker thread should not occupy more than the size of single block!

```
def sortLines( lines: Array[Array[Byte]] ): Array[Array[Byte]] = {
  /* The algorithm is a in-place sorting algorithm
   * Thus no additional memory that exceeds
   * the partition size will be required
   */
  logger.debug(s"in-place quicksort ${lines.size} lines")
  Sorting.quickSort(lines)( KeyOrdering )
```

```
def mergePartitions(inp1path: String, inp2path: String, outPath: String, lineSizeInBytes: Int): Unit = {

  val inp1Stream = new BufferedInputStream( new FileInputStream(inp1path) )
  val inp2Stream = new BufferedInputStream( new FileInputStream(inp2path) )
  val outStream = new BufferedOutputStream( new FileOutputStream(outPath) )
  try {
    val line1 = Array.fill[Byte]( lineSizeInBytes )( elem = 0 )
    val line2 = Array.fill[Byte]( lineSizeInBytes )( elem = 0 )
    var inp1Eof = false
    var inp2Eof = false
    inp1Stream.read( line1, off = 0, lineSizeInBytes )
    inp2Stream.read( line2, off = 0, lineSizeInBytes )
```

# Concurrency Control

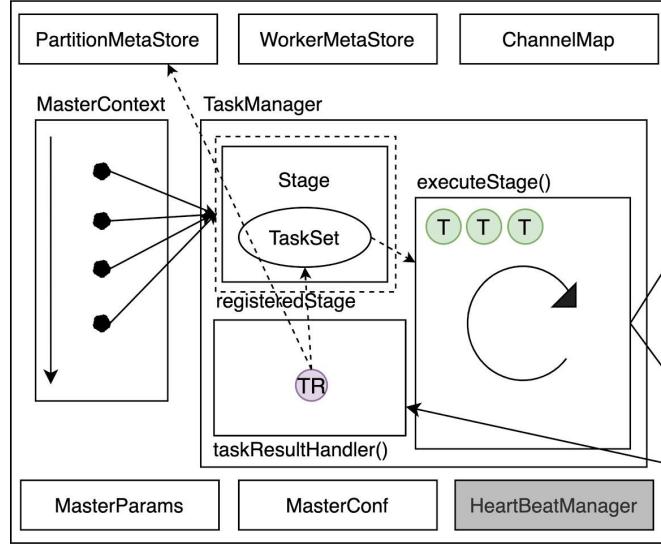
Most singleton objects (usually holding globally-accessible data) should be **thread safe**. We apply locks to resolve synchronization problem.

```
object PartitionMetaStore {  
  
    /*  
     * PartitionMetaStore requires strict concurrency control  
     */  
  
    private val partitionMetaStore  
        = mutable.SortedMap[Int, ListBuffer[PartitionMeta]] ()  
  
    val pmsLock: ReentrantReadWriteLock = new ReentrantReadWriteLock()  
    def readLock: ReentrantReadWriteLock.ReadLock = pmsLock.readLock()  
    def writeLock: ReentrantReadWriteLock.WriteLock = pmsLock.writeLock()  
  
    def addPartitionMeta( workerID:Int, pmeta: PartitionMeta ): Unit = {  
        writeLock.lock()  
        try {  
            partitionMetaStore(workerID) += pmeta  
        } finally {  
            writeLock.unlock()  
        }  
    }  
}
```

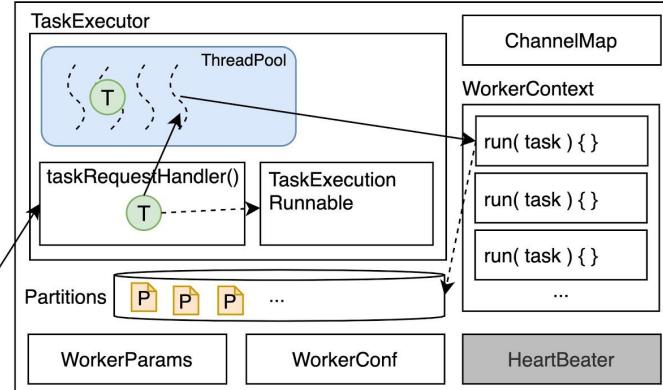
```
object ShuffleManager extends Logging {  
  
    ...  
  
    var numOngoingSendShuffles = 0  
    private def numShuffleSendConnectionsLeft: Int = maxShuffleSendConnections  
    val shuffleSendLock: ReentrantLock = new ReentrantLock() /  
    private def isShuffleSendAvailable: Boolean = numShuffleSendConnectionsLeft > 0  
  
    var numOngoingReceiveShuffles = 0  
    private def numShuffleReceiveConnectionsLeft: Int = maxShuffleReceiveConnections  
    val shuffleReceiveLock: ReentrantLock = new ReentrantLock() /  
    private def isShuffleReceiveAvailable: Boolean = numShuffleReceiveConnectionsLeft > 0  
}
```

# Overall View of the System

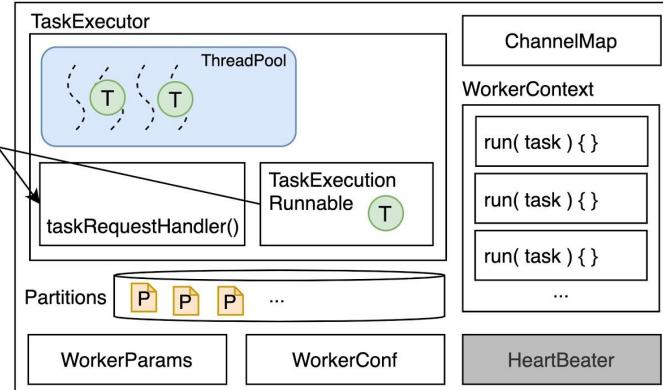
Master



Worker 1



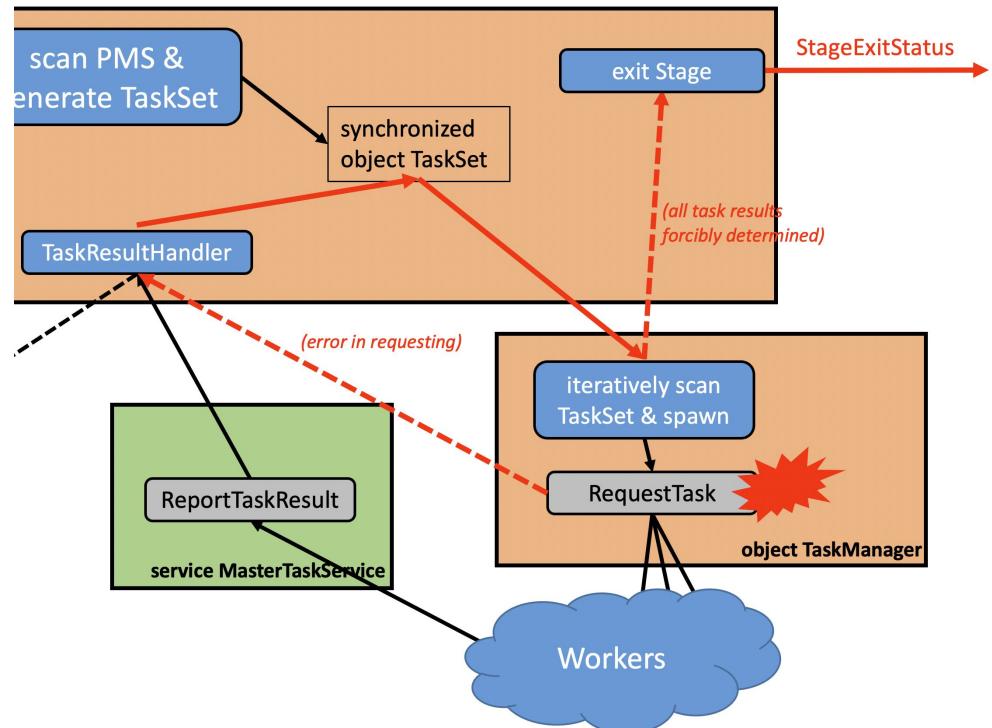
Worker 2



# Robustness to Failure

For some reason TaskRunner cannot request tasks, it can directly call TaskResultHandler to mark failure

This is a very natural approach to cope with worker's error, without creating additional invariants.



# Demonstration

- Will mainly be done via Youtube video

<https://www.youtube.com/watch?v=Kq2B0tbqGiE>

# Running Example - Worker Registration

Each worker registers itself with its IP, port, and input partitions.

# Running Example - Stage Generation

```
: Stage GenBlockStage finally marked success.  
[PartitionMetaStore:  
Worker 1: part-000001:0, part-000002:0, part-000003:0  
Worker 2: part-000005:0, part-000004:0, part-000006:0  
Worker 3: part-000007:0, part-000009:0, part-000008:0  
]  
2020-12-14 17:14:57.107 INFO --- [run-main-0] d.m.e.LocalSortStage  
: taskset generated with 9 task(s)  
2020-12-14 17:14:57.107 INFO --- [run-main-0] d.m.e.LocalSortStage  
: Register stage LocalSortStage.  
2020-12-14 17:14:57.107 INFO --- [run-main-0] d.m.e.LocalSortStage  
: Executing stage LocalSortStage...  
2020-12-14 17:14:57.114 INFO --- [run-main-0] d.m.TaskManager$  
: task : 18 submitted to worker 3  
2020-12-14 17:14:57.120 INFO --- [run-main-0] d.m.TaskManager$  
: task : 14 submitted to worker 2  
2020-12-14 17:14:57.126 INFO --- [run-main-0] d.m.TaskManager$  
: task : 17 submitted to worker 3  
2020-12-14 17:14:57.133 INFO --- [run-main-0] d.m.TaskManager$  
: task : 11 submitted to worker 1  
2020-12-14 17:14:57.138 INFO --- [run-main-0] d.m.TaskManager$  
: task : 10 submitted to worker 1  
2020-12-14 17:14:57.144 INFO --- [run-main-0] d.m.TaskManager$  
: task : 16 submitted to worker 3  
2020-12-14 17:14:57.150 INFO --- [run-main-0] d.m.TaskManager$  
: task : 13 submitted to worker 2  
2020-12-14 17:14:57.156 INFO --- [run-main-0] d.m.TaskManager$  
: task : 15 submitted to worker 2
```

```
: task request arrived  
2020-12-14 17:14:53.945 DEBUG --- [ault-executor-0] d.w.TaskExecutor$  
: task execution available  
2020-12-14 17:14:53.947 DEBUG --- [ool-15-thread-1] d.w.TaskExecutionRunnable  
: task 1 execution started.  
2020-12-14 17:14:54.068 DEBUG --- [ault-executor-0] d.w.TaskExecutor$  
: task request arrived  
2020-12-14 17:14:54.068 DEBUG --- [ault-executor-0] d.w.TaskExecutor$  
: task execution available  
2020-12-14 17:14:54.069 DEBUG --- [ool-15-thread-2] d.w.TaskExecutionRunnable  
: task 2 execution started.  
2020-12-14 17:14:54.090 DEBUG --- [ault-executor-0] d.w.TaskExecutor$  
: task request arrived  
2020-12-14 17:14:54.090 DEBUG --- [ault-executor-0] d.w.TaskExecutor$  
: task execution available  
2020-12-14 17:14:54.090 DEBUG --- [ool-15-thread-3] d.w.TaskExecutionRunnable  
: task 3 execution started.  
2020-12-14 17:14:54.952 DEBUG --- [ool-15-thread-1] d.c.u.FileUtils$  
: reading 10000 lines, where each line takes 100 bytes  
2020-12-14 17:14:54.975 INFO --- [ool-15-thread-1] d.w.TaskExecutionRunnable  
: task 1 execution finished. now reporting result  
2020-12-14 17:14:55.070 DEBUG --- [ool-15-thread-2] d.c.u.FileUtils$  
: reading 10000 lines, where each line takes 100 bytes  
2020-12-14 17:14:55.091 DEBUG --- [ool-15-thread-3] d.c.u.FileUtils$  
: reading 10000 lines, where each line takes 100 bytes  
2020-12-14 17:14:55.109 INFO --- [ool-15-thread-2] d.w.TaskExecutionRunnable  
: task 2 execution finished. now reporting result  
2020-12-14 17:14:55.150 INFO --- [ool-15-thread-3] d.w.TaskExecutionRunnable
```

After one stage has finished, master then generates next stage with its taskset in it. It then registers the stage to TaskManager and triggers the execution. The worker will soon get task request and accept/decline upon thread pool availability.

# Running Example - Sampling

```
2020-12-14 17:15:01.208 DEBUG --- [ool-15-thread-3] d.c.u.SortUtils$  
: sample from [[B@3e7302cd lines, with sample ratio 0.05  
2020-12-14 17:15:01.216 DEBUG --- [ool-15-thread-3] d.c.u.SortUtils$  
: sampled 523 keys, actual sample ratio: 0.0523  
2020-12-14 17:15:01.218 DEBUG --- [ool-15-thread-2] d.c.u.FileUtils$  
: reading 10000 lines, where each line takes 100 bytes  
2020-12-14 17:15:01.221 INFO --- [ool-15-thread-3] d.w.TaskExecutionRunnable  
: task 19 execution finished. now reporting result  
2020-12-14 17:15:01.230 DEBUG --- [ool-15-thread-2] d.c.u.SortUtils$  
: sample from [[B@274a4bda lines, with sample ratio 0.05  
2020-12-14 17:15:01.230 DEBUG --- [ool-15-thread-4] d.c.u.FileUtils$  
: reading 10000 lines, where each line takes 100 bytes  
2020-12-14 17:15:01.231 DEBUG --- [ool-15-thread-2] d.c.u.SortUtils$  
: sampled 490 keys, actual sample ratio: 0.049  
2020-12-14 17:15:01.233 INFO --- [ool-15-thread-2] d.w.TaskExecutionRunnable  
: task 20 execution finished. now reporting result  
2020-12-14 17:15:01.242 DEBUG --- [ool-15-thread-4] d.c.u.SortUtils$  
: sample from [[B@38056b45 lines, with sample ratio 0.05  
2020-12-14 17:15:01.243 DEBUG --- [ool-15-thread-4] d.c.u.SortUtils$  
: sampled 502 keys, actual sample ratio: 0.0502
```

The worker is given a target sample ratio, and returns the sampled keys.

# Running Example - PartitionAndShuffle

```
2020-12-14 17:15:04.414 DEBUG --- [ool-15-thread-1] d.w.e.PartitionAndShuffleContext$ : partitioning done :  
2020-12-14 17:15:04.415 DEBUG --- [ool-15-thread-1] d.w.e.PartitionAndShuffleContext$ : (127.0.0.1, 10002) : 3448 lines  
2020-12-14 17:15:04.415 DEBUG --- [ool-15-thread-1] d.w.e.PartitionAndShuffleContext$ : (127.0.0.1, 20002) : 3324 lines  
2020-12-14 17:15:04.415 DEBUG --- [ool-15-thread-1] d.w.e.PartitionAndShuffleContext$ : (127.0.0.1, 30002) : 3228 lines  
2020-12-14 17:15:04.421 DEBUG --- [ool-15-thread-3] d.w.e.PartitionAndShuffleContext$ : partitioning done :  
2020-12-14 17:15:04.421 DEBUG --- [ool-15-thread-3] d.w.e.PartitionAndShuffleContext$ : (127.0.0.1, 10002) : 3357 lines  
2020-12-14 17:15:04.421 DEBUG --- [ool-15-thread-3] d.w.e.PartitionAndShuffleContext$ : (127.0.0.1, 20002) : 3304 lines  
2020-12-14 17:15:04.421 DEBUG --- [ool-15-thread-3] d.w.e.PartitionAndShuffleContext$ : (127.0.0.1, 30002) : 3339 lines  
2020-12-14 17:15:04.430 DEBUG --- [ool-15-thread-1] d.w.ShuffleManager$ : shuffle local write done  
2020-12-14 17:15:04.431 DEBUG --- [ool-15-thread-3] d.w.ShuffleManager$ : shuffle local write done  
2020-12-14 17:15:04.435 INFO --- [ool-15-thread-1] d.c.n.ChannelMap$ : channel connection to 127.0.0.1:20002 established with type dpsort.worker.ShuffleReqChann  
el@1ec01b  
2020-12-14 17:15:04.435 INFO --- [ool-15-thread-3] d.c.n.ChannelMap$ : channel connection to 127.0.0.1:20002 established with type dpsort.worker.ShuffleReqChann  
el@4b575c75  
2020-12-14 17:15:04.436 DEBUG --- [ool-15-thread-2] d.w.e.PartitionAndShuffleContext$ : partitioning done :  
2020-12-14 17:15:04.437 DEBUG --- [ool-15-thread-2] d.w.e.PartitionAndShuffleContext$ : (127.0.0.1, 10002) : 3424 lines  
2020-12-14 17:15:04.437 DEBUG --- [ool-15-thread-2] d.w.e.PartitionAndShuffleContext$ : (127.0.0.1, 20002) : 3404 lines  
2020-12-14 17:15:04.437 DEBUG --- [ool-15-thread-2] d.w.e.PartitionAndShuffleContext$ : (127.0.0.1, 30002) : 3172 lines  
2020-12-14 17:15:04.437 DEBUG --- [ool-15-thread-1] d.w.ShuffleReqChannel : requesting shuffle  
2020-12-14 17:15:04.438 DEBUG --- [ool-15-thread-2] d.w.ShuffleManager$ : shuffle local write done
```

```
2020-12-14 17:15:04.458 DEBUG --- [ool-15-thread-1] d.w.ShuffleManager$ : shuffle request granted : now transmitting partition part-000035 to 127.0.0.1:20002  
2020-12-14 17:15:04.458 DEBUG --- [ool-15-thread-1] d.w.ShuffleReqChannel : requesting shuffle  
2020-12-14 17:15:04.462 DEBUG --- [ool-15-thread-3] d.w.ShuffleManager$ : shuffle request granted : now transmitting partition part-000032 to 127.0.0.1:20002  
2020-12-14 17:15:04.462 DEBUG --- [ool-15-thread-2] d.w.ShuffleReqChannel : requesting shuffle  
2020-12-14 17:15:04.467 DEBUG --- [ool-15-thread-2] d.w.ShuffleManager$ : shuffle request granted : now transmitting partition part-000029 to 127.0.0.1:20002  
2020-12-14 17:15:04.471 DEBUG --- [ool-15-thread-1] d.w.ShuffleReqChannel : sending shuffle data  
2020-12-14 17:15:04.475 DEBUG --- [ool-15-thread-2] d.w.ShuffleReqChannel : sending shuffle data  
2020-12-14 17:15:04.475 DEBUG --- [ool-15-thread-3] d.w.ShuffleReqChannel : sending shuffle data  
2020-12-14 17:15:04.494 DEBUG --- [ault-executor-0] d.w.ShuffleServiceImpl : shuffle request arrived  
2020-12-14 17:15:04.494 DEBUG --- [ault-executor-0] d.w.ShuffleManager$ : accepting shuffle request  
2020-12-14 17:15:04.500 DEBUG --- [ault-executor-0] d.w.ShuffleServiceImpl : shuffle request arrived  
2020-12-14 17:15:04.500 DEBUG --- [ault-executor-0] d.w.ShuffleManager$ : accepting shuffle request  
2020-12-14 17:15:04.507 DEBUG --- [ault-executor-0] d.w.ShuffleServiceImpl : shuffle request arrived  
2020-12-14 17:15:04.507 DEBUG --- [ault-executor-0] d.w.ShuffleManager$ : accepting shuffle request  
2020-12-14 17:15:04.515 DEBUG --- [ool-15-thread-1] d.w.ShuffleReqChannel : sending shuffle termination  
2020-12-14 17:15:04.515 DEBUG --- [ool-15-thread-3] d.w.ShuffleReqChannel : sending shuffle termination  
2020-12-14 17:15:04.515 DEBUG --- [ool-15-thread-2] d.w.ShuffleReqChannel : sending shuffle termination  
2020-12-14 17:15:04.521 DEBUG --- [ault-executor-1] d.w.ShuffleServiceImpl : shuffle request arrived  
2020-12-14 17:15:04.521 DEBUG --- [ault-executor-1] d.w.ShuffleManager$ : accepting shuffle request  
2020-12-14 17:15:04.524 INFO --- [ool-15-thread-2] d.c.n.ChannelMap$ : channel connection to 127.0.0.1:30002 established with type dpsort.worker.ShuffleReqChann  
el@7f02d083  
2020-12-14 17:15:04.528 DEBUG --- [ool-15-thread-2] d.w.ShuffleReqChannel : requesting shuffle
```

After partitioning, the worker will first write local partition to the work directory and shuffle out the rest. The ShuffleManager uses ChannelMap to retrieve the matching channel and request shuffle. The shuffle request will be granted upon availability. After the request has been granted, the worker will send chunks of partitions and finally the shuffle termination message.

# Running Example - Merge

```
... Stage MergeStage finally marked success
[PartitionMetaStore:
Worker 1: part-000052:0, part-000055:0, part-000056:0, part-000057:0, part-000058:0
Worker 2: part-000053:0, part-000059:0, part-000060:0, part-000062:0, part-000061:0
Worker 3: part-000054:0, part-000064:0, part-000065:0, part-000066:0, part-000063:0
]
2020-12-14 17:15:12.418 INFO --- [run-main-0] d.m.e.MergeStage
: taskset generated with 6 task(s)
2020-12-14 17:15:12.418 INFO --- [run-main-0] d.m.e.MergeStage
: Register stage MergeStage.
2020-12-14 17:15:12.418 INFO --- [run-main-0] d.m.e.MergeStage
: Executing stage MergeStage...
2020-12-14 17:15:12.422 INFO --- [run-main-0] d.m.TaskManager$
: task : 54 submitted to worker 3
2020-12-14 17:15:12.426 INFO --- [run-main-0] d.m.TaskManager$
: task : 53 submitted to worker 3
2020-12-14 17:15:12.430 INFO --- [run-main-0] d.m.TaskManager$
: task : 49 submitted to worker 1
2020-12-14 17:15:12.434 INFO --- [run-main-0] d.m.TaskManager$
: task : 51 submitted to worker 2
2020-12-14 17:15:12.438 INFO --- [run-main-0] d.m.TaskManager$
: task : 50 submitted to worker 1
2020-12-14 17:15:12.443 INFO --- [run-main-0] d.m.TaskManager$
: task : 52 submitted to worker 2
2020-12-14 17:15:15.444 INFO --- [run-main-0] d.m.TaskManager$
: status report : 6 task(s) finished, 0 task(s) remaining
2020-12-14 17:15:15.444 INFO --- [run-main-0] d.m.e.MergeStage
: Stage MergeStage exited with code 0
2020-12-14 17:15:15.444 INFO --- [run-main-0] d.m.e.MergeStage
: Stage MergeStage finally marked success.
[PartitionMetaStore:
Worker 1: part-000058:0, part-000067:0, part-000068:0
Worker 2: part-000061:0, part-000069:0, part-000070:0
Worker 3: part-000063:0, part-000072:0, part-000071:0
]
2020-12-14 17:15:15.445 INFO --- [run-main-0] d.m.e.MergeStage
: taskset generated with 3 task(s)
2020-12-14 17:15:15.445 INFO --- [run-main-0] d.m.e.MergeStage
: Register stage MergeStage.
2020-12-14 17:15:15.445 INFO --- [run-main-0] d.m.e.MergeStage
: Executing stage MergeStage...
2020-12-14 17:15:15.448 INFO --- [run-main-0] d.m.TaskManager$
: task : 57 submitted to worker 3
2020-12-14 17:15:15.451 INFO --- [run-main-0] d.m.TaskManager$
: task : 55 submitted to worker 1
2020-12-14 17:15:15.455 INFO --- [run-main-0] d.m.TaskManager$
: task : 56 submitted to worker 2
2020-12-14 17:15:18.456 INFO --- [run-main-0] d.m.TaskManager$
: status report : 3 task(s) finished, 0 task(s) remaining
2020-12-14 17:15:18.456 INFO --- [run-main-0] d.m.e.MergeStage
: Stage MergeStage exited with code 0
2020-12-14 17:15:18.456 INFO --- [run-main-0] d.m.e.MergeStage
: Stage MergeStage finally marked success.
[PartitionMetaStore:
Worker 1: part-000068:0, part-000073:0
Worker 2: part-000070:0, part-000074:0
Worker 3: part-000071:0, part-000075:0
]
```

At each merge iteration, each task will merge a pair of input and output a merged partition. The stage repeats until each worker has a single partition.

# Running Example - Output Validation

# Sort success with equi-sized partitions

# Applied Design Patterns

- Dependency Injection
  - when registering each stage to TaskManager
- Command Pattern
  - Receiver : TaskContext implementation
  - Commander : Task Object
  - Invoker : Each worker's task executor

# Dependency Injection

```
/* Execute Stages */

val stage0 = new GenBlockStage
lastStageExitStatus = stage0.executeAndwaitForTermination()
println(s"${PartitionMetaStore.toString}")

if( lastStageExitStatus == StageExitStatus.SUCCESS ) {
  val stage1 = new LocalSortStage
  lastStageExitStatus = stage1.executeAndwaitForTermination()
  println(s"${PartitionMetaStore.toString}")
}

if( lastStageExitStatus == StageExitStatus.SUCCESS ) {
  val stage2 = new SampleKeyStage
  lastStageExitStatus = stage2.executeAndwaitForTermination()
  genPartitionFunction
}

if( lastStageExitStatus == StageExitStatus.SUCCESS ) {
  val stage3 = new PartitionAndShuffleStage
  lastStageExitStatus = stage3.executeAndwaitForTermination()
  println(s"${PartitionMetaStore.toString}")
```

In each stage execution, the function executeAndwaitForTermination will register(inject) itself to TaskManager and trigger its execution. *(the function waits for the future!)*

```
trait Stage extends Logging {

  ...

  def executeAndwaitForTermination(): StageExitStatus.Value = {
    logger.info(s"Register stage ${this.toString}...")
    val registerResult = TaskManager.registerStage( stage = this )
    if( !registerResult ) {
      logger.error(s"Cannot register stage to TaskRunner")
      return StageExitStatus.FAILURE
    }
    logger.info(s"Executing stage ${this.toString}...")

    val stageExitCode: Int = Await.result( executeStage, Duration.Inf )
  }
}

object TaskManager extends Logging{

  private var registeredStage: Stage = null

  def taskResultHandler: TaskReportMsg => Unit = registeredStage.taskResultHandler

  def registerStage( stage: Stage ) : Boolean = {
    if ( registeredStage == null ) {
      registeredStage = stage
      true
    } else { false }
  }

  def executeStage: Future[Int] = {
    // submit until terminate condition is satisfied
  }
}
```

# Command Pattern - Commander

```
@SerialVersionUID(1001L)
final class GenBlockTask( i: Int,
                        wi: Int,
                        st: TaskStatus.Value,
                        inputPart: String,
                        outputPart: Array[String],
                        offsets: Array[(Int, Int)])
}
extends BaseTask(i, wi, TaskType.GENBLOCKTASK, st, Array[...])
with Serializable {

}

@SerialVersionUID(1002L)
final class TerminateTask( i: Int,
                           wi: Int,
                           st: TaskStatus.Value,
                           inputPart: String,
                           outputPart: String,
                           )
extends BaseTask(i, wi, TaskType.TERMINATETASK, st, Array[...])
with Serializable {
```

Each task object contains following parameters

- Task ID
- Worker ID which the task is assigned to
- Status
- Input partition(s)
- Output partition(s)
- etc.

*Note that each task object should be serializable in order to be distributed across the network*

# Command Pattern - Invoker

- When task object is handled in, it invokes the `run()` method of that task's execution context

```
object TaskExecutor extends Logging {  
    ...  
  
    def taskRequestHandler(task: BaseTask): ResponseMsg =  
        logger.debug(s"task request arrived")  
        if( isTaskSubmitAvailable ) {  
            logger.debug(s"task execution available")  
            numRunningThreads += 1  
            val execFuture: Future[_] = executor.submit( new TaskExecutionRunnable(task) )  
        }  
  
    class TaskExecutionRunnable(task: BaseTask) extends Runnable with Logging {  
        override def run(): Unit = {  
            ...  
  
            try {  
                val taskOutput: Either[Unit, ByteString] = ExecCtxtFetcher.getContext(task).run(task)  
            }  
        }  
    }  
}
```

```
private def submitTask( task: BaseTask ): Boolean = {  
    // generate TaskMsg  
    val taskMsg = new TaskMsg( serializeObjectToByteString(task) )  
    // request  
    val reqChannel: TaskReqChannel = ChannelMap  
        .getChannel( WorkerMetaStore.getWorkerIpPort(task.getWorkerId) )  
        .asInstanceOf[TaskReqChannel]  
    logger.debug(s"trying to submit task ${task.getId} via ${reqChannel}")  
    // get respond  
    val submitResponse: ResponseMsg = reqChannel.requestTask( task )  
    ...  
}
```

# Command Pattern - Receiver

```
trait TaskExecutionContext {
  def run( _task: BaseTask ): Either[Unit, ByteString]
}

object EmptyContext extends TaskExecutionContext {

  def run( _task: BaseTask ): Left[Unit, Nothing] = {
    val task = _task.asInstanceOf[EmptyTask]
    val rndTime = new scala.util.Random(task.getId).nextInt(10)
    println(s"This is emptytask : wait for ${rndTime}s and finish");
    Thread.sleep( rndTime * 1000 )

    Left( Unit )
  }
}

object GenBlockContext extends TaskExecutionContext with Logging {

  def run( _task: BaseTask ): Left[Unit, Nothing] = {
    val task = _task.asInstanceOf[GenBlockTask]
    try {
      val filepath = task.inputPartition.head
      for( (outPartName, pIdx) <- task.outputPartition.zipWithIndex ){
        ...
      }
    } catch {
      case e: Exception =>
        log.error(s"Error while processing task $task: ${e.getMessage}")
        Left( e )
    }
  }
}
```

Each execution context receives a task object.

Each context typecasts the input task object before accessing the parameters.

# Further Improvements / Extensions

- Add robustness to worker failure (*they often can fail! => Network issue, GC overhead*)
  - Task retry policy
  - Heartbeat implementation
- Introduce buffers (*advanced partition management*)
  - Current approach is the Hadoop MapReduce approach : to read/write partitions on disk every operation
  - Spark approach can speed up significantly : to maintain in-memory as much as possible, and spill the rest to disk

# What I've Learned

- Precautious design / Documentation matters!
  - Always write notes on 1. Design 2. Implementation scheduling & procedure
  - Nevertheless, it is always never sufficient.
- Assertion(controlling invariants) indeed helps
  - Experienced from other course projects (OS, DB) as well
- Importance of testing
  - As code and data gets bigger, I started to regret not writing unit tests
  - The debugging time grows exponentially as the program gets closer to the production
- Automated environment configuration can reduce tons of unnecessary works
- Although the system is complicated, the code should remain simple
  - Complicacy comes from well-formed design, not from a complicated code

# What I've Learned (*con't*)

- Resource matters
  - Memory-aware programming
- Concurrency matters
  - Multiple threads incur deadlock / data consistency failure
  - Deadlock problem in shuffle request / resource management
- Knowledge is the power
  - An unclear notion on the external library (protobuf, fileIO, ..) will lead to misuse of time

# References

- Github repo : <https://github.com/dgggit/cs434-project>
- Design documents : <https://github.com/dgggit/cs434-doc/tree/master/project>
- Scala Official Docs : <https://docs.scala-lang.org/>
- ScalaPB : <https://scalapb.github.io/>
- Stackoverflow, and other websites
- Slides from POSTECH CS434 2020F lecture videos