

# Project Design Document

*(DPsort : Distributed, Parallel External MergeSort)*

*- Milestone 2 -*

20180274 Jinho Ko

# Progress Report Summary

- Fixed design concepts and brief details
- Done following tasks
  - Write Entrypoint shell files (master / worker (.sh) )
  - Write Configuration files format ( MASTER\_CONF.xml, ~)
  - Adapt and set logging(log4j2) / communication protocol (protobuf + gRPC)
- Implement following functionalities
  - Loading configurations
  - Worker registry
  - Utils
  - Overall Master / Worker context
  - Defining structures for Task / TaskSet / Stage

# TODOs for final phase

- Implement contexts of each Task / TaskSet / Stage
- Write test suites
- Implement HeartBeat(er,manager) threads
- Implement Log aggregation
- Make build.sbt to produce assemblies (jar)
- Code revision and submission
  - use professional tools to draw all architectures

# Design Documents

- *refer to the following slides.*

# Design Concept

- Master is in charge of everything
- Worker is just a machine that receives a task and returns the result.
  - Goal : To be stateless as much as possible
    - Input : TaskMsg
    - Output : TaskStatusMsg
    - How about the local data files? (partitions)
      - Even for the data files, master owns all the partition list.

# Overall Sort Pipeline

Each worker conducts each task type in the following order (in the normal case).  
Note that multiples of task of each type can be generated => to easily achieve multithreading by assigning 1 core per task.

1. GenBlockStage
2. LocalSortStage
3. SampleKeysStage
4. PartitionAndShuffleStage
5. MergeStage
6. TerminateStage

# Core : Class definition

- dpsort.core/
  - execution/
    - Stage { class Stage }
    - Task { trait Task, abstract class BaseTask, class ~~ , ... }
    - TaskSet { class TaskSet }
    - RoleContext { trait RoleContext }
  - heartbeat/
  - network/
    - protobuf-generated classes
    - ServerContext { class ServerContext }
  - storage/
  - utils/
    - FileUtils, IdUtils, SerializationUtils
  - ConfContext { trait Conf, object ConfContext }
  - trait Params
  - class Registry

# Master : Class definition

- dpsort.master/
  - Main { object Main }
  - MasterContext { object MasterContext }
  - MasterConf { object MasterConf }
  - MasterParams { object MasterParams }
  - WorkerMetaStore { object WorkerMetaStore }
  - PartitionMetaStore { object PartitionMetaStore }
  - TaskRunner { object TaskRunner }
  - HeartBeatManager { object HeartBeatManager }
  - HeartBeatServer { object HeartBeatServer }
  - MasterTaskServer { object MasterTaskServer }



# Worker : Class definition

- dpsort.worker/
  - Main { object Main }
  - WorkerContext { object WorkerContext }
  - WorkerConf { object WorkerConf }
  - WorkerParams { object WorkerParams }
  - TaskManager { object TaskManager }
  - HeartBeater { object HeartBeater }
  - ShuffleServer { object ShuffleServer }

# utils

- ID generator
  - taskId, partitionID, workerID
- Object(generic) serializer / deserializer
- File related (absPath, homePath, load / save partitions)

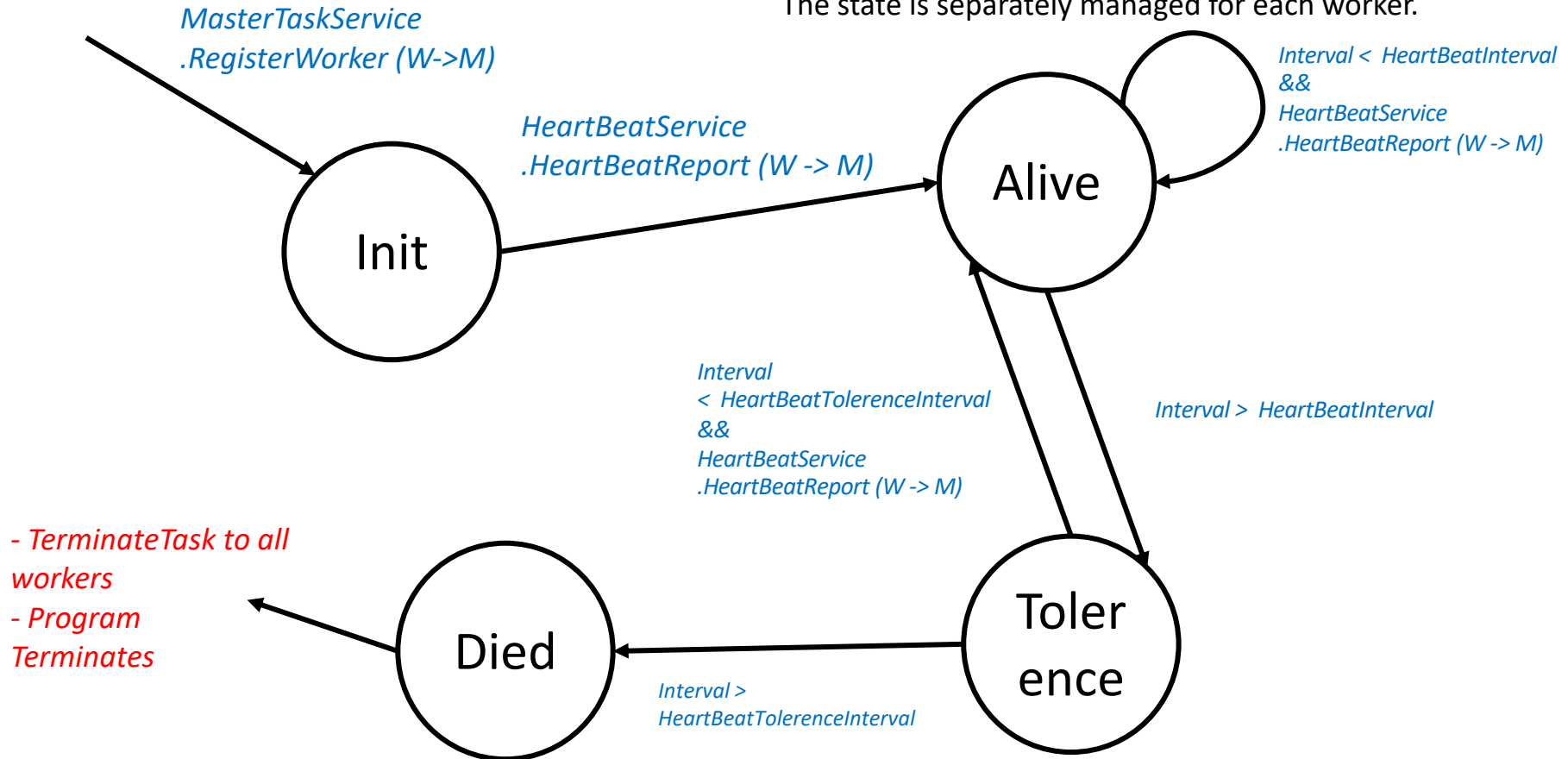
# Communication Protocol

- Use protobuf + gRPC
- in **Master**
  - service MasterTaskService
    - rpc RegisterWorker ( RegistryMsg => ResponseMsg )
    - rpc ReportTaskResult ( TaskReportMsg => ResponseMsg )
  - service HeartBeatService
    - rpc HeartBeatService ( HeartBeatMsg => ResponseMsg )
- in **Worker**
  - service WorkerTaskService
    - rpc RequestTask ( TaskMsg => ResponseMsg )
  - service ShuffleService
    - rpc RequestShuffle ( ShuffleRequestMsg => ResponseMsg )
    - rpc SendShuffleData ( ShuffleDataMsg => ResponseMsg )

# Heartbeat State Diagram

My Heartbeat design is a one-way implementation, worker simply sends and master receives and decides. (*no explicit checking from master*)

The state is separately managed for each worker.

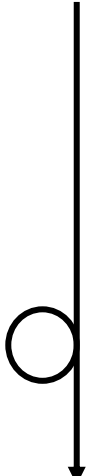


Note : Constant (*HeartBeatInterval*, *HeartBeatToleranceInterval*) pre-configured from *MASTER\_CONF*

# Stage / Task Specification

Stage := An execution unit of pipeline workflow

Task := An execution unit in each worker thread, that assigns physical data partition



Stage	Task Used	What task does
GenBlockStage	GenBlockTask	- given a partition, split partition into a set of smaller partitions
LocalSortStage	LocalSortTask	- given a a partition, perform local sorting
SampleKeysStage	SampleKeysTask	- given a partition, scan and sample - report the sampled data
PartitionAndShuffleStage	PartitionAndShuffleTask	- given a partition and partitioning function, spilt partition into blocks - perform shuffle write of each block
*MergeStage	MergeTask	- given 2 partitions, merge files and save
TerminateStage	TerminateTask	- if term. due to failure, cancel all tasks and report, - if term. with execution done. - copy output file to output dir (if success) / delete workdir.

*Whenever failure reported or worker dies, the running stage will be aborted and TerminateStage will be executed ASAP.*

*\*MergeStage can execute repeatedly until all partitions are reduced for each worker. Master will scan PartitionMetaStore and generate repeated stages until merge-end-condition is satisfied.*

# Message Type Definition

```
/*
 * Message Definition
 */
message RegistryMsg {
    // We send bytes-serialized object directly
    // rather than elaborate message definition
    // due to its complexity
    bytes serializedRegistryObject = 1;
}

message TaskMsg {
    // We send bytes-serialized object directly
    // rather than elaborate message definition
    // due to its complexity
    enum TaskObjectType {
        GenBlockTask = 0;
        TerminateTask = 1;
        // and another 4 tasks
    }
    bytes serializedTaskObject = 1;
}
```

```
message TaskReportMsg {
    enum TaskResultType {
        SUCCESS = 0;
        FAILED = 1;
    }
    int32 taskId = 1;
    TaskResultType taskResult = 2;
    bytes serializedTaskResultData = 3;
}

message ResponseMsg {
    enum ResponseType {
        NORMAL = 0;
        HANDLE_ERROR = 1;
    }
    ResponseType response = 1;
}

message HeartBeatMsg {
    enum HeartBeatType {
        ALIVE = 0;
    }
    HeartBeatType heartBeat = 1;
}
```

# Diagram Definition



Class / Object



Service Protocol



Data



Task



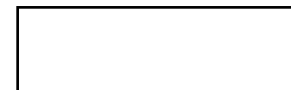
Thread



Function (procedure)



Service Call (networking)



Pseudocode block



Execution flow that always proceeds

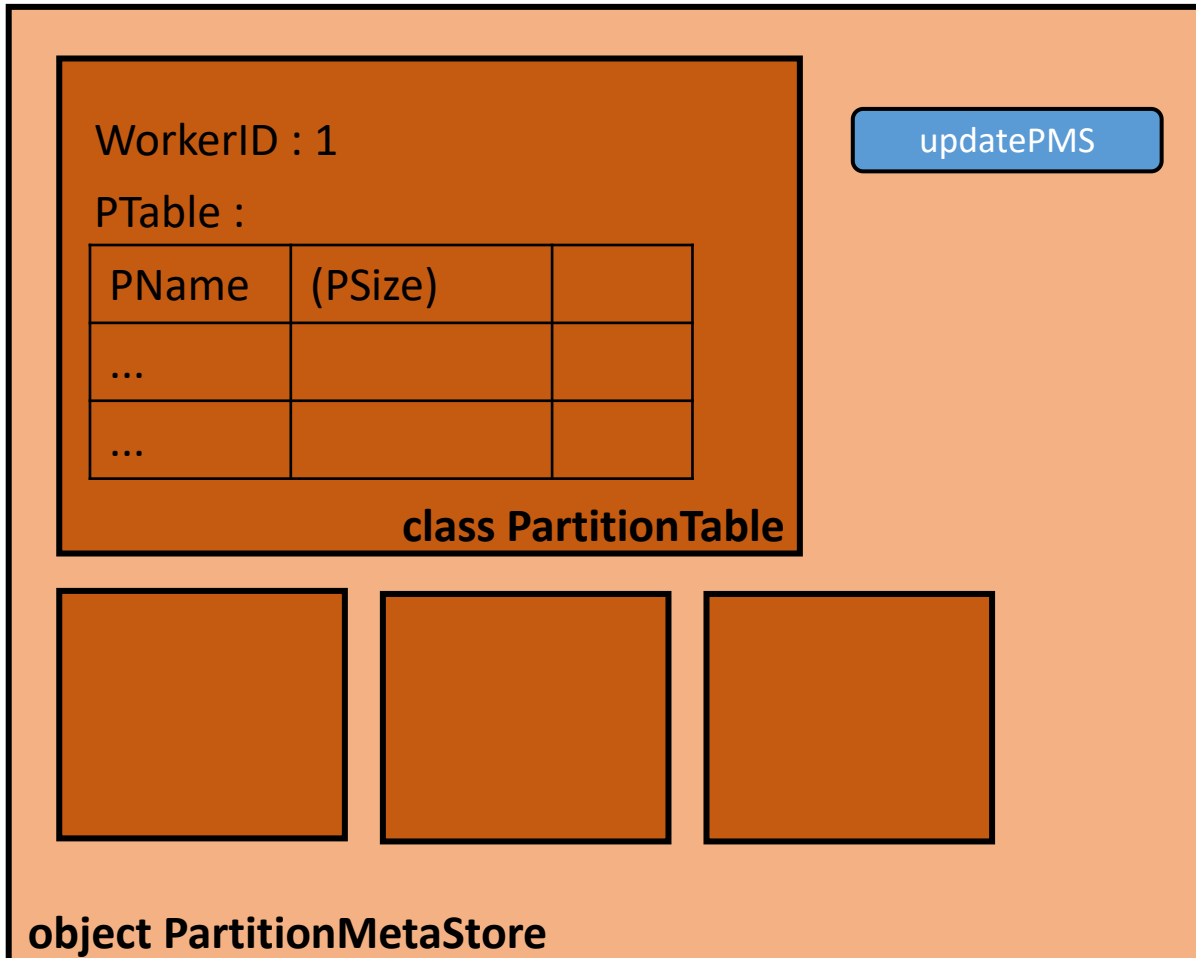


Conditional execution flow



Abnormal execution flow

# Master : PartitionMetaStore



shared access :  
synchronization required.

PID must obey one of the  
following formats :

- filename (normal case)
- filename with AbsPath  
(when start/exiting  
program)



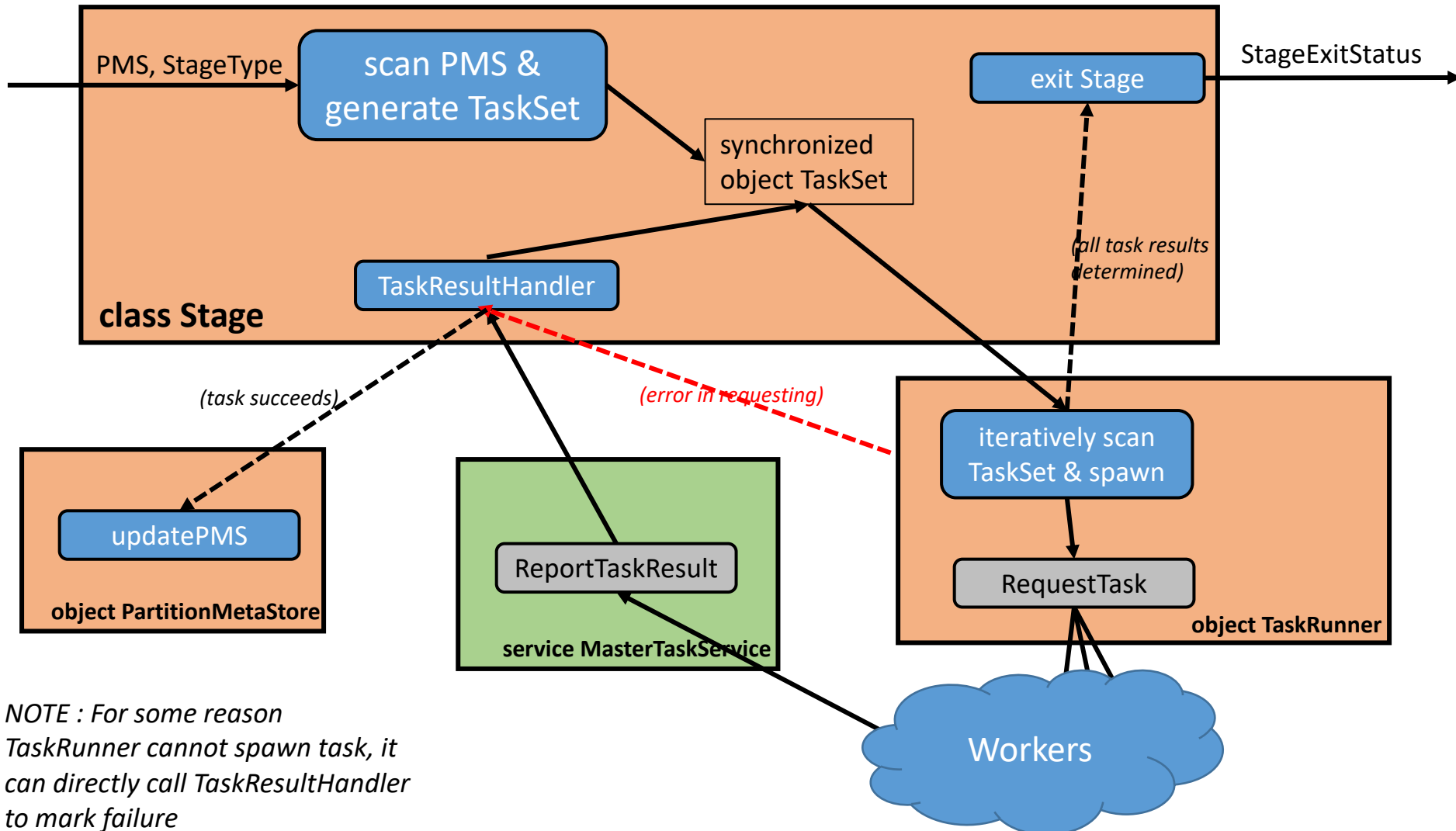
# Master : Overall Execution

- *Defined in MasterContext (similar for WorkerContext)*
  - *@ init)*
    - Start Server : {MasterTaskServer, HeartBeatServer}
  - *@ terminate)*
    - Stop Server : {MasterTaskServer, HeartBeatServer}
  - *@ execute)*
    1. wait for N workers
    2. Execute GenBlockStage
    3. (if prior stage success) Execute LocalSortStage
    4. (if prior stage success) Execute SampleKeysStage
    5. (if prior stage success) Execute PartitionAndShuffleStage
    6. (if prior stage success) Execute MergeStage
    7. Iterate stage 6 until condition ( merge done for all worker ) satisfied
    8. Execute TerminateStage

> *NOTE : We don't maintain task queue in worker, but master keeps checks worker's availability with some interval.*

# Master : Task Generation & Request

in each stage, the following pipeline is executed :



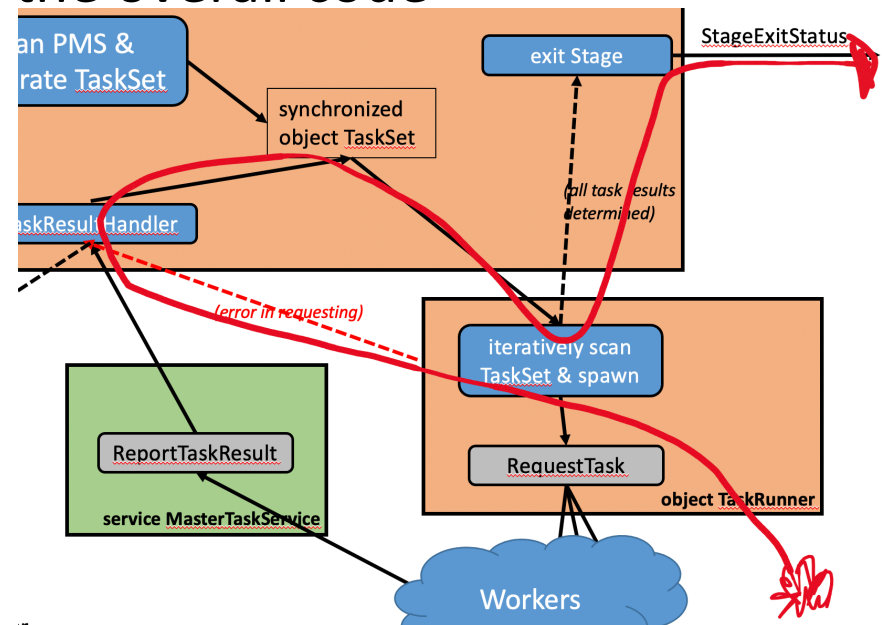
*NOTE : For some reason TaskRunner cannot spawn task, it can directly call TaskResultHandler to mark failure*

# Master : Task Generation

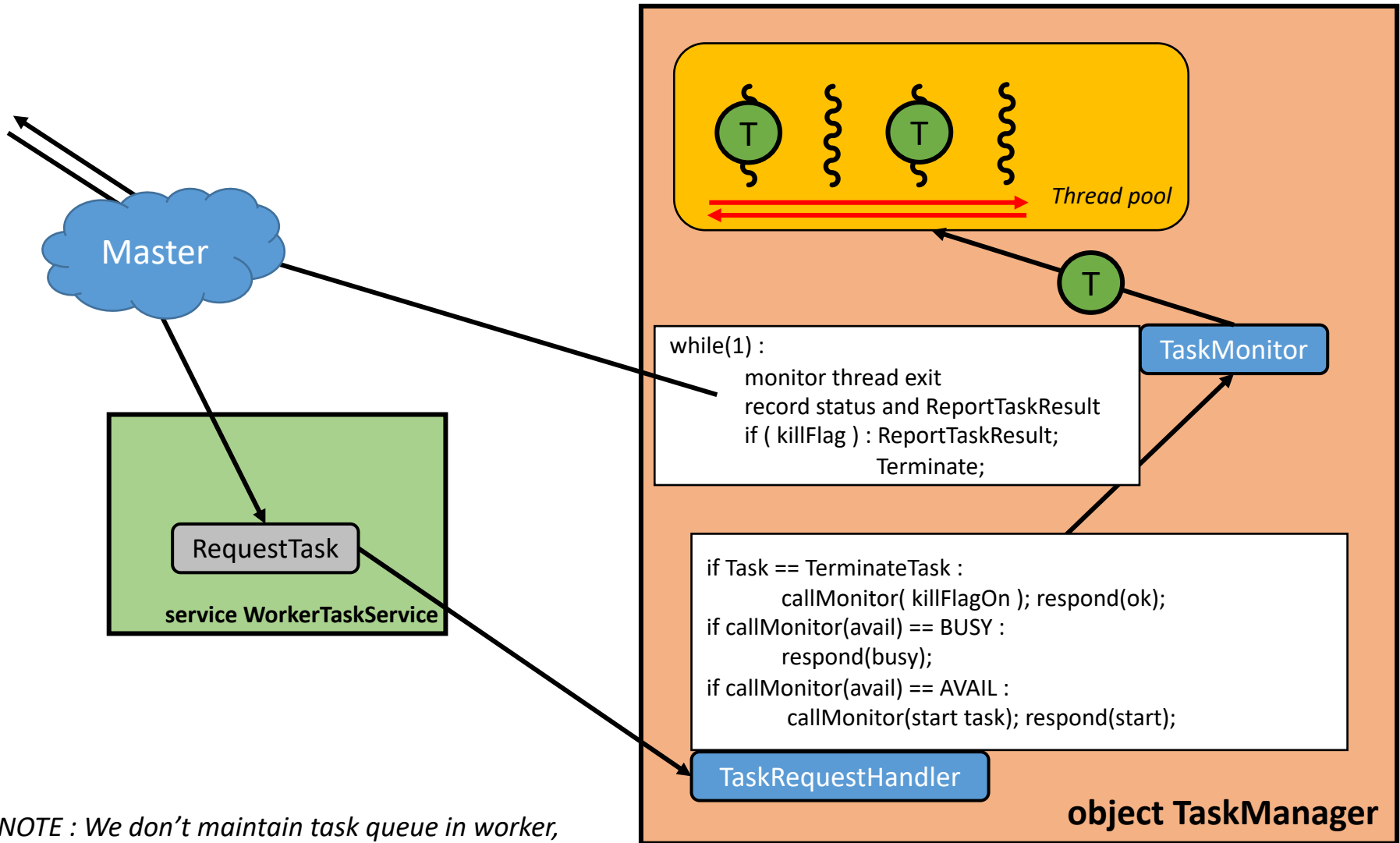
- In each stage definition,
  - each stage will scan all partition informations, and generate Set of Tasks (TaskSet)
  - Example of Task)
    - @ GenBlockTask
      - InputPartitions : [ PART\_001 ]
      - OutputPartitions : [ PART\_002, PART\_003, PART\_004 ]
      - ( GenBlockTask.run() ) :
        - loads an input partition from work directory
        - splits partition equally into 3 pieces, and the intermediate partitions to the work directory
        - delete original partition
        - report success
    - @ PartitionAndShuffleTask
      - InputPartitions : [PART\_001 ]
      - OutputPartitions : [W2:3421:PART\_002, PART\_003, W4:3421:PART\_004 ]
      - additional : partitionfunction
      - ( GenBlockTask.run() ) :
        - loads an input partition from work directory
        - partition the data according to the partitionfunction
        - write back locally / and shuffle each partition as specified
        - delete original partition
        - report success
- Every Task has a well-formed structure, and thus each worker will simply execute specified in AnyTask.run()
  - I chose this design, because although it is tricky to format the task definition at first, it is much easier to improve the performance once the architecture is implemented
- The stage is declared “success” when all tasks in its TaskSet is marked success

# Master : Invariants in Task Request

- 2 cases possible
  - 1. Task request is not available (due to network issue)
  - 2. Heartbeat determines worker dead
- All tasks will soon be marked failure, and stage will exit with fail status. The termination process is done very naturally, in order to reduce invariants to the overall code



# Worker : Task Management



> NOTE : We don't maintain task queue in worker, but master keeps checks worker's availability with some interval.

# Worker : Shuffle

- Partition function is given as follows
  - input : key range
  - output : { (destIP,PORT,destpartitionname) FORALL #blocks }
- If destination is not specified, it will save partition locally. Otherwise, it will shuffle
- @Shuffle ; Query IP:PORT to ChannelTable and send shuffle message
  - If channel not exists, open new channel and add to ChannelTable

# Tests

- *Following points are planned to be tested*
  - (utils.Id) ID generators should not generate duplicate indices
  - Config load / set / get works as my intents
  - Serialization / Deserialization is working for several test objects
  - Performance of partitioner
  - Heartbeat execution behavior

# M/W : Config

- each file loads default config file and user defined file (worker and master have the different file)
  - default loc. : master/s/m/resources/default.properties
  - user file loc. : conf/conf.properties
- default is parsed loaded first and then is overridden by user config
  - user config can only “update” the property, not newly “creating” it



# Results

# 1. Master / WorkerEntrypoint

```
~/dev/cs434-project/bin > dev !1 > ls -la
total 24
drwxr-xr-x  4 jinho  staff   136 Nov 16 13:05 .
drwxr-xr-x 22 jinho  staff   748 Nov 14 00:46 ..
-rw-r--r--  1 jinho  staff  2431 Nov 16 12:17 master
-rw-r--r--  1 jinho  staff  4465 Nov 16 13:05 worker
~/dev/cs434-project/bin > dev !1 > bash bin/master 3
~/dev/cs434-project/bin > dev !1 > bash bin/worker localhost:1234 -I conf -O conf
```

Executing shell files in bin/folder will execute assembled jar files.  
*(currently it runs sbt for technical issue, but wil be resolved)*

```
~/dev/cs434-project > dev !1 > bash bin/worker localhost:1234 -I conf -O cone
Error: Directory not found : cone
~/dev/cs434-project > dev !1 > bash bin/worker localhost:1234 -I conf
Error: No output path provided
~/dev/cs434-project > dev !1 > bash bin/worker localhost:1234 -I conf
Error: No output path provided
~/dev/cs434-project > dev !1 > bash bin/worker localhost:1234
Error: Wrong number of input path(s) provided
~/dev/cs434-project > dev !1 > bash bin/worker -h
Usage: worker [option...] <MASTER_IPPORT> -I <dir1> <dir2> .. -O <dir>

-h, --help                display help

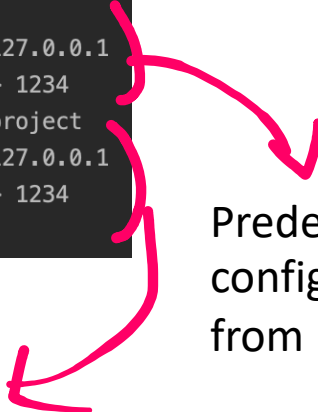
<MASTER_IPPORT>          master IP:PORT
-I, --input <dir1> <dir2> .. input directories
-O, --output <dir>        output directory
```

The implemented shells detects all kinds of initial errors


- *file existence, #args, arg formatting, supported OS,..*
- *and so on.*

## 2. Loading configuration

```
d.m.Main$           : load configurations
d.m.MasterConf$     : setting property dpsort.master.ip => 127.0.0.1
d.m.MasterConf$     : setting property dpsort.master.port => 1234
d.c.u.FileUtils$    : system path : /Users/jinho/dev/cs434-project
d.m.MasterConf$     : setting property dpsort.master.ip => 127.0.0.1
d.m.MasterConf$     : setting property dpsort.master.port => 1234
d.m.Main$           : load arguments
```



Predefined system configurations (load configs from resource file)



Overriding configurations from user-defined conf file.

# 3. Worker(s) Registry

```
INFO --- [run-main-0] d.c.n.ServerContext : MasterTaskServer started, listening on 1234
INFO --- [run-main-0] d.c.n.ServerContext : HeartBeatServer started, listening on 0
INFO --- [run-main-0] d.m.MasterContext$ : waiting for workers
334 INFO --- [ault-executor-0] d.m.MasterTaskServiceImpl : Worker id: 1 registered. 2 remaining.
INFO --- [ault-executor-0] d.m.MasterTaskServiceImpl : Worker id: 2 registered. 1 remaining.
INFO --- [ault-executor-0] d.m.MasterTaskServiceImpl : Worker id: 3 registered. 0 remaining.
INFO --- [run-main-0] d.m.MasterContext$ : all () workers registered
INFO --- [run-main-0] d.c.n.ServerContext : MasterTaskServer shutdown
INFO --- [run-main-0] d.c.n.ServerContext : HeartBeatServer shutdown
INFO --- [run-main-0] d.m.Main$ : dpsort master finished
```

# 4. Serialization

```
DEBUG --- [ault-executor-0] d.c.u.SerializationUtils$ : dserialization of object type: dpsort.core.Registry success
```

```
2020-11-16 22:41:33.775 DEBUG --- [ run-main-0] d.c.u.SerializationUtils$ : serialization of object type:  
dpsort.core.Registry success
```

Some messages are not defined via protobuf, but serialized/deserialized via java library  
(due to their complexity and variability during implementation)