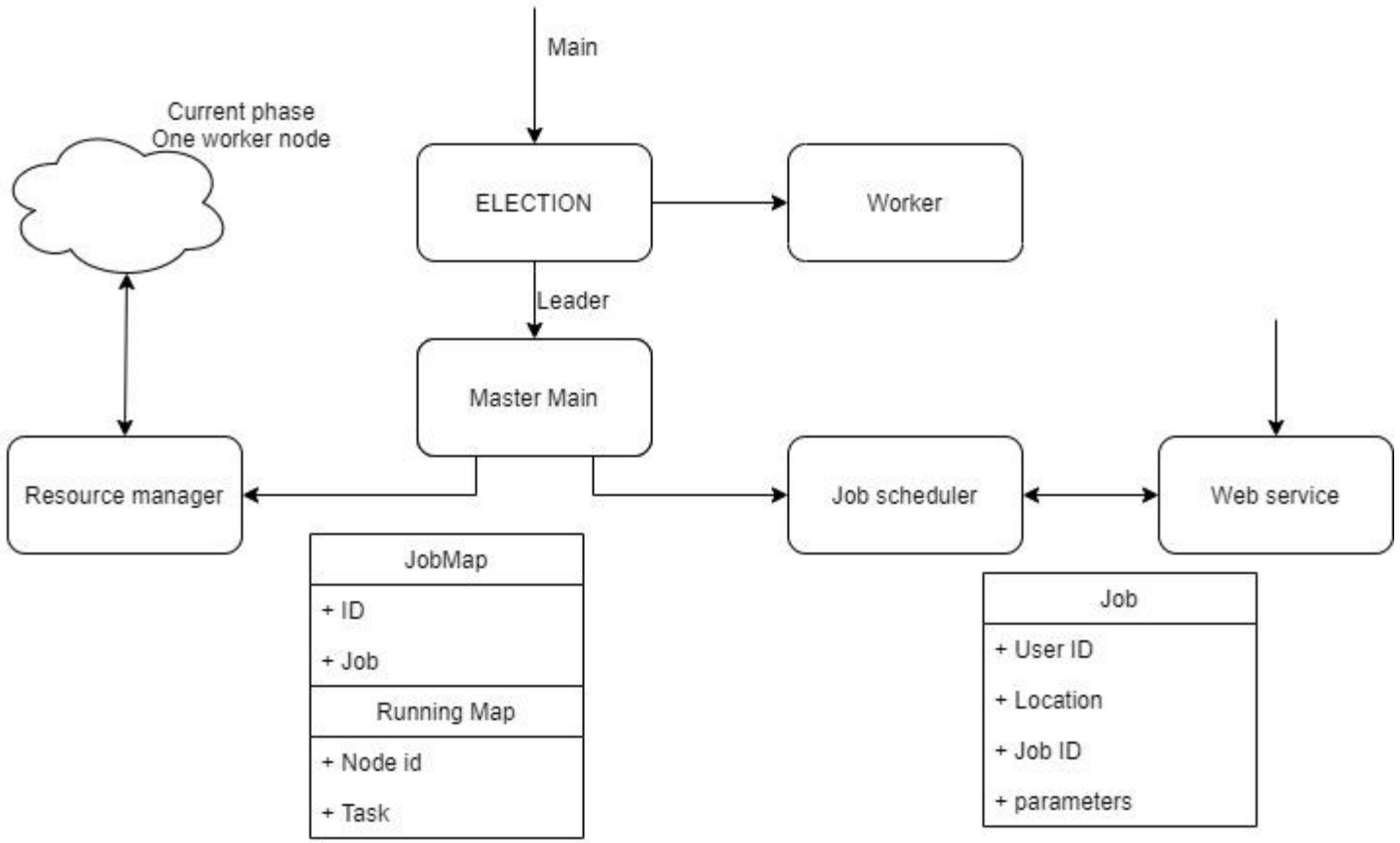
Self adjusted auto provision system at resource level

Weekly report 20th May

You Hu

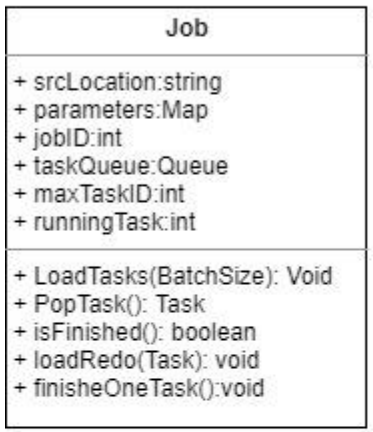
Simple Design

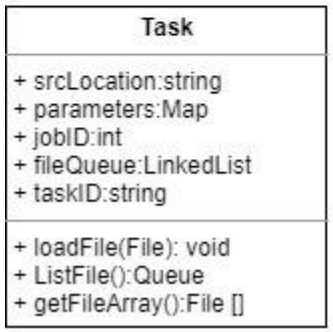


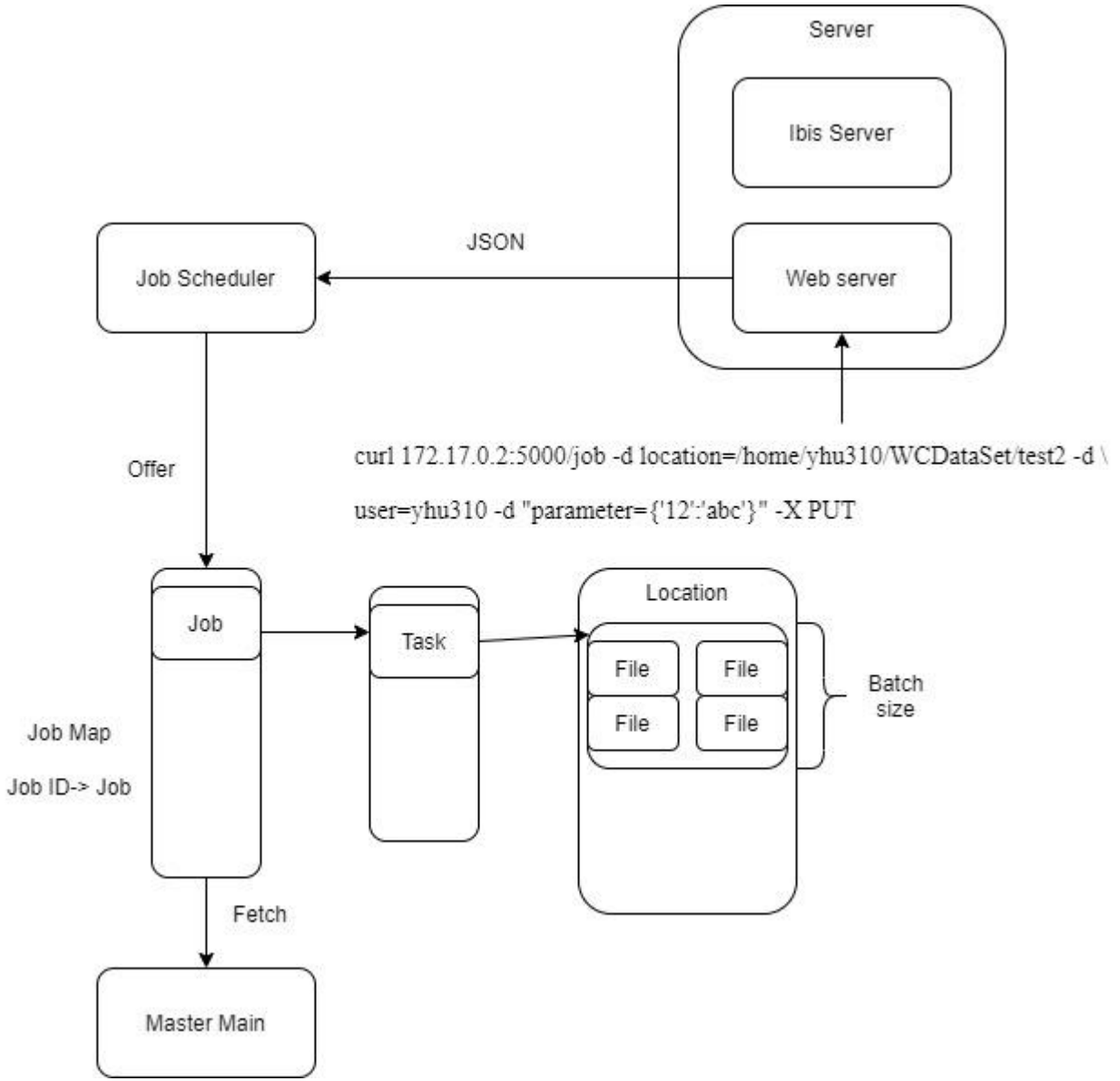
Same code, Different action

Job submission

RestFul API
Job loads files to tasks

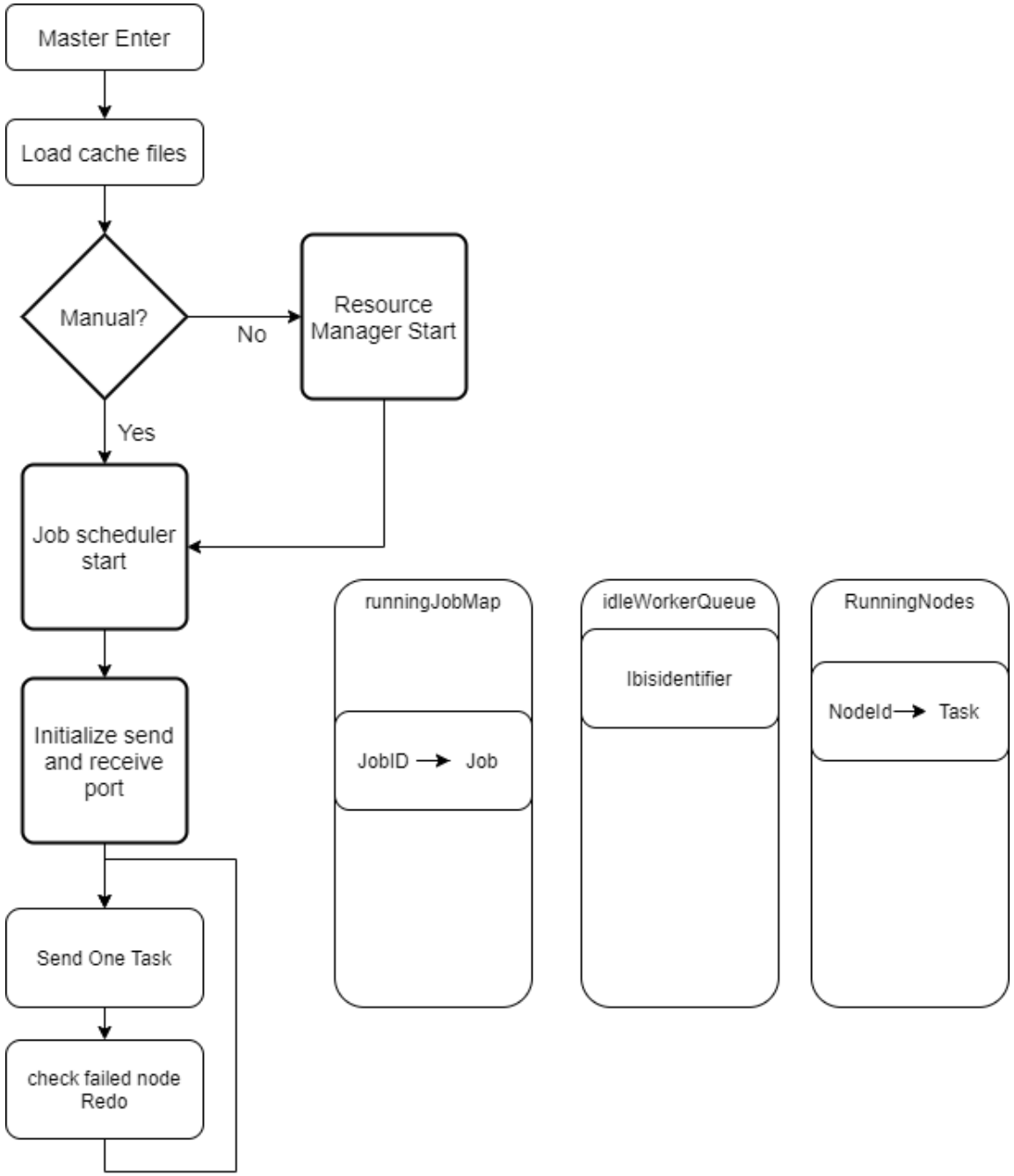






Master main

First initial settings
Then loops for task deliver and fault task redo

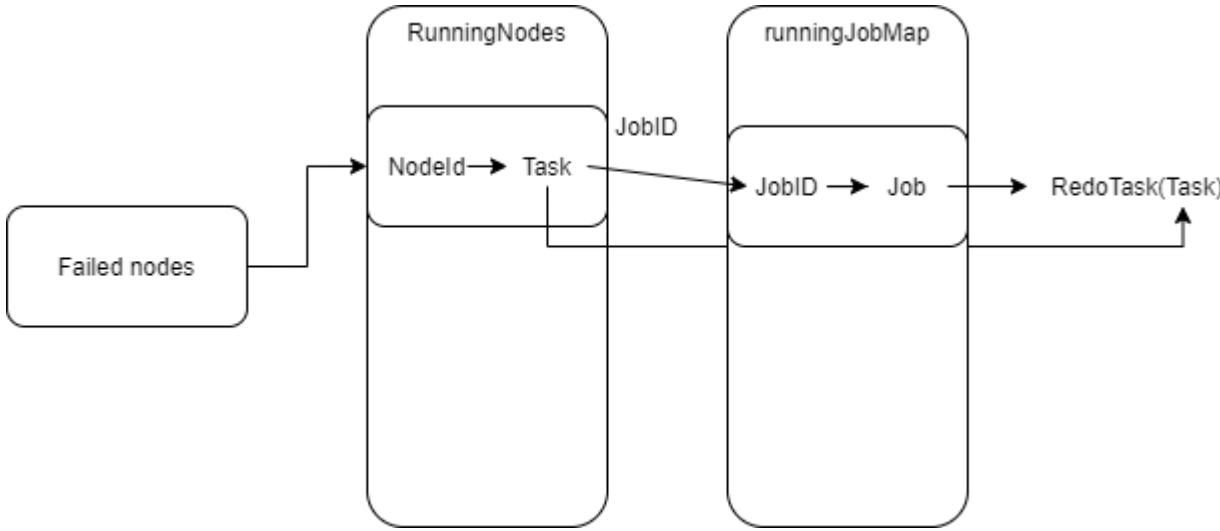


Master send one task

- If idleWorkerQueue not empty && runningJobMap not empty
  - Fetch a *Task* from runningJobMap and a *ibisIdentifier* from idleWorkerQueue
  - Send
    - If success: Add pair <ibisIdentifier, Task> to RunningNodes
    - If failed(connection failed): pop a new *ibisIdentifier*, Send again
      - If idleWorkerQueue is empty: redo Task, and end this section
- FIFO Job/Task Queue
  - Tree map: sorted via the key(Job Id)
  - Pop task: iterate, if taskQueue is not empty, pop a task, else next job(with ascending order by the JobId)

Master check failed

- Fetch the dead/left nodes since last call
- Find the Intersection between dead/left set and running Nodes
- According to the id of failed node, redo the task
- Caching the RunningJobMap and the tasks on RunningNodes



Master Upcall

```
ControlMessage m = (ControlMessage) readMessage.readObject();
readMessage.finish();
synchronized (idleWorkerQueue) {
    synchronized (runningNodes) {
        runningNodes.remove(readMessage.origin().ibisIdentifier());// No matter whether there is, it can be removed
        idleWorkerQueue.offer(readMessage.origin().ibisIdentifier());// add this active node to idleQueue
    }



}

if (m.isEmptyRequest() == false) { // false-> this controlMessage carrys result of task; true-> this is an init message
    System.out.println("Job:" + m.getJobID() + " Task:" + m.getTaskID() + " Finished with code:" + m.getStatusCode());
    synchronized (runningJobMap) {
        runningJobMap.get(m.getJobID()).finishOneTask();
        if (runningJobMap.get(m.getJobID()).isFinished()) {
            runningJobMap.remove(m.getJobID()); // If all task of this job finished, it can be removed from running Job Map
        }
    }
}
```

Worker main

- Init ports
- Send a init controlMessage to master notify an active node join
- Loop (finished==false)
  - Fetch a *Task* from workerTaskQueue
  - Process the Task with given: location, paramenters
  - Send back the result to master

- TODO: any time detect fail of master, forcedly terminate the task and reelection

Worker Upcall

```
Task t = (Task)
readMessage.readObject();
readMessage.finish();
synchronized (workerTaskQueue) {
    workerTaskQueue.offer(t);
}
```

Fault tolerance

- Worker failed
  - Master detects the failed nodes
  - Redo the task
- Master failed
  - Pre-processing: periodically snapshots
  - Worker detects master failing
  - Reelection
  - New master load the snapshot
- Server(TODO)
  - Ideally using backup site; and switch back when server is avalible