

# Self adjusted auto provision system at resource level

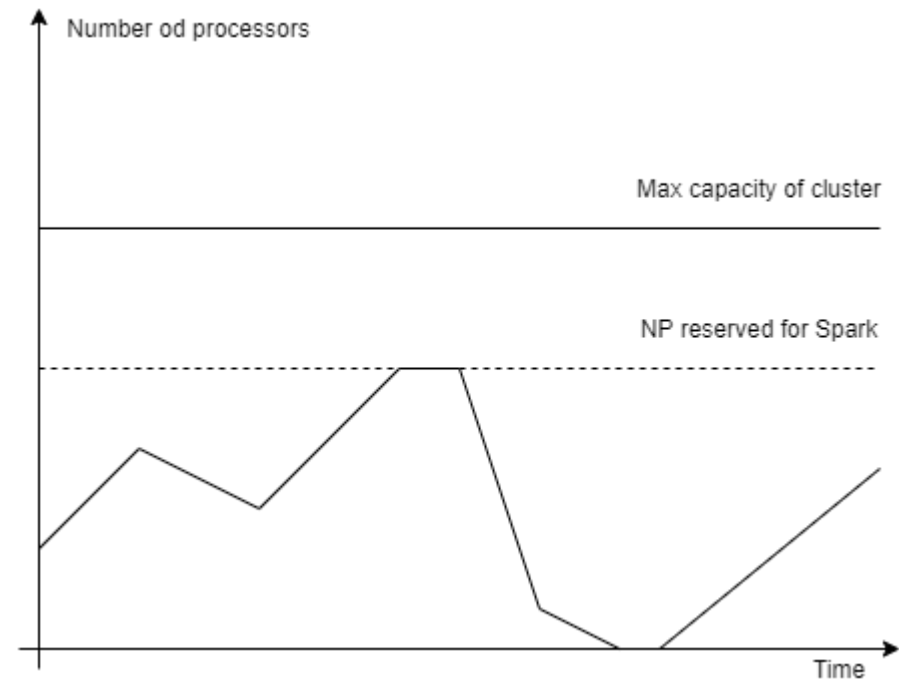
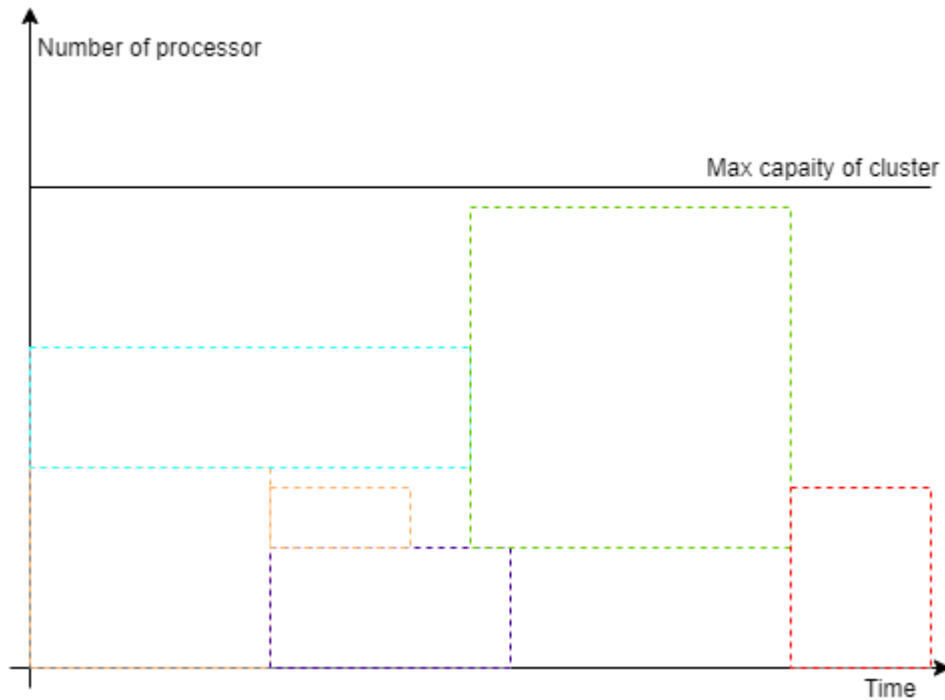
Progress report 28<sup>th</sup> May

You Hu

# Agenda

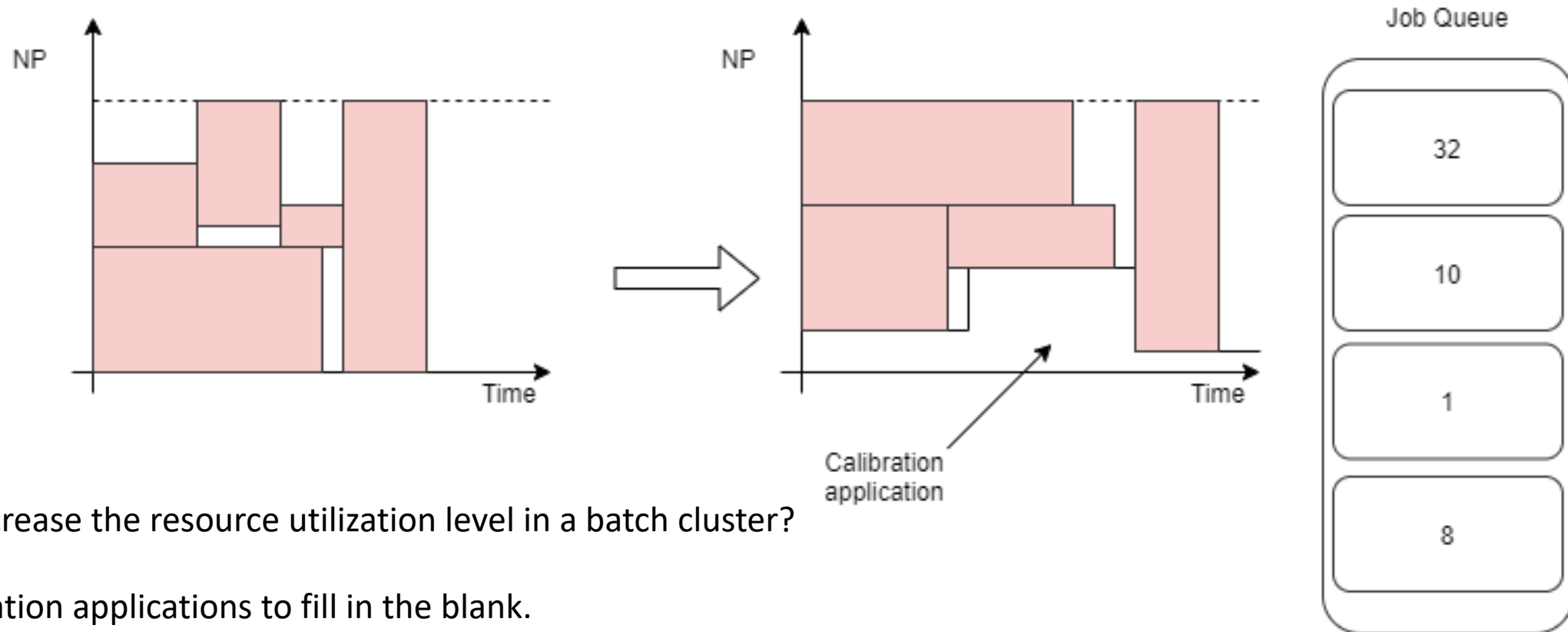
- Recap
- Plan
- Details of current implementation
  - Master-worker Arch
  - Fault tolerance
- Issues not completely solved
- TODO and Problems

# Recap – issue and research question



Existing solutions resulting resource waste

# Recap – issue and research question

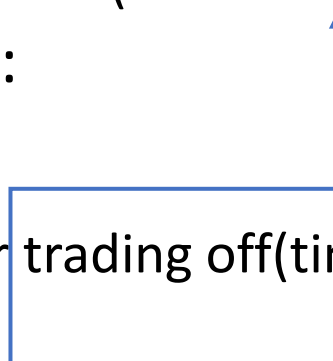


How to increase the resource utilization level in a batch cluster?

Use calibration applications to fill in the blank.

It requires: auto scaling/provisioning;

# Recap - Questions statement (One month ago)

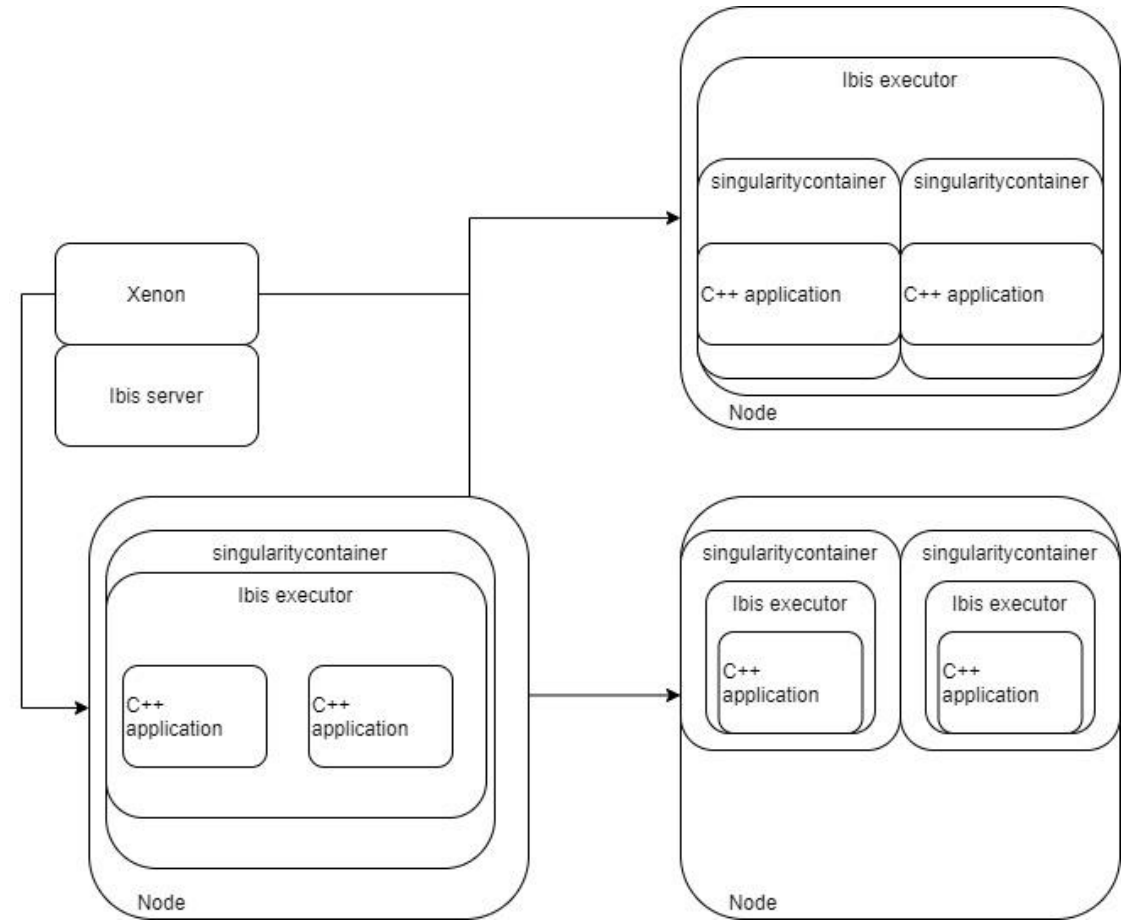
- Distributed algorithm/architecture: Master – worker
    - Communication: IPL -> Ibis server(s-for backup)
  - Resource management algo:
    - Estimate job execution time
    - Schedule->SLURM+Xenon
    - Dynamic adjust: a metrics for trading off(time to start and shut down nodes)
  - Fault tolerance
    - Node manage: master->re-election; worker->task persistence
    - Tasks(NO snapshot): Zookeeper/Redis/any lightweight distributed database
  - Data locality: move comp rather than move data
    - Remote sites
    - Shared storage
    - Data aware(?)
- 

# Possible arch solutions

Node-IbisExe-(Multi)Container-Application

Node-Container-IbisExe-(Multi)Application

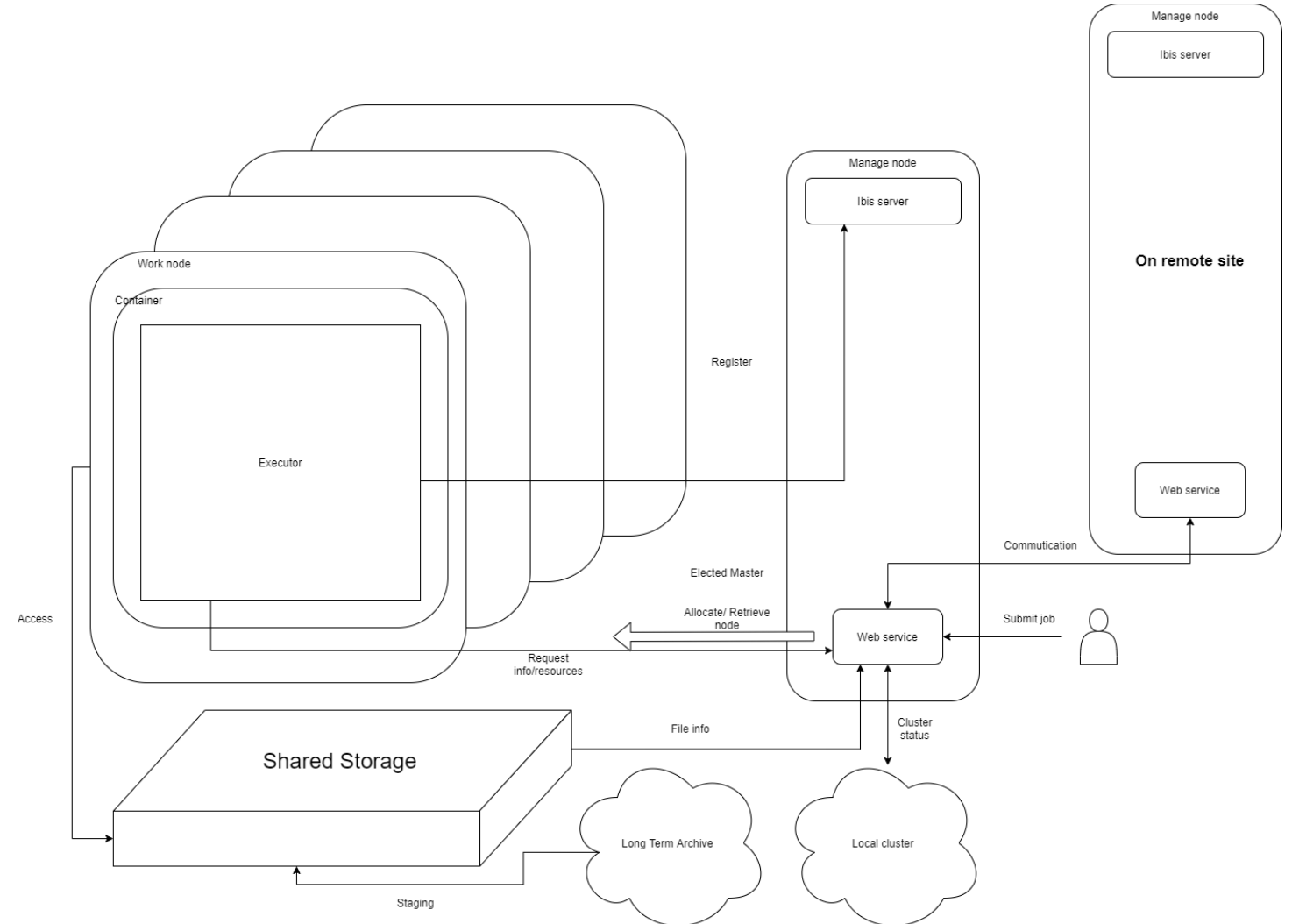
Node-(Multi)Container-IbisExe-Application



# System design 0.2.1

One cluster->multiple nodes  
One node->one container  
One container->one driver

SAGECAL employs multi-thread

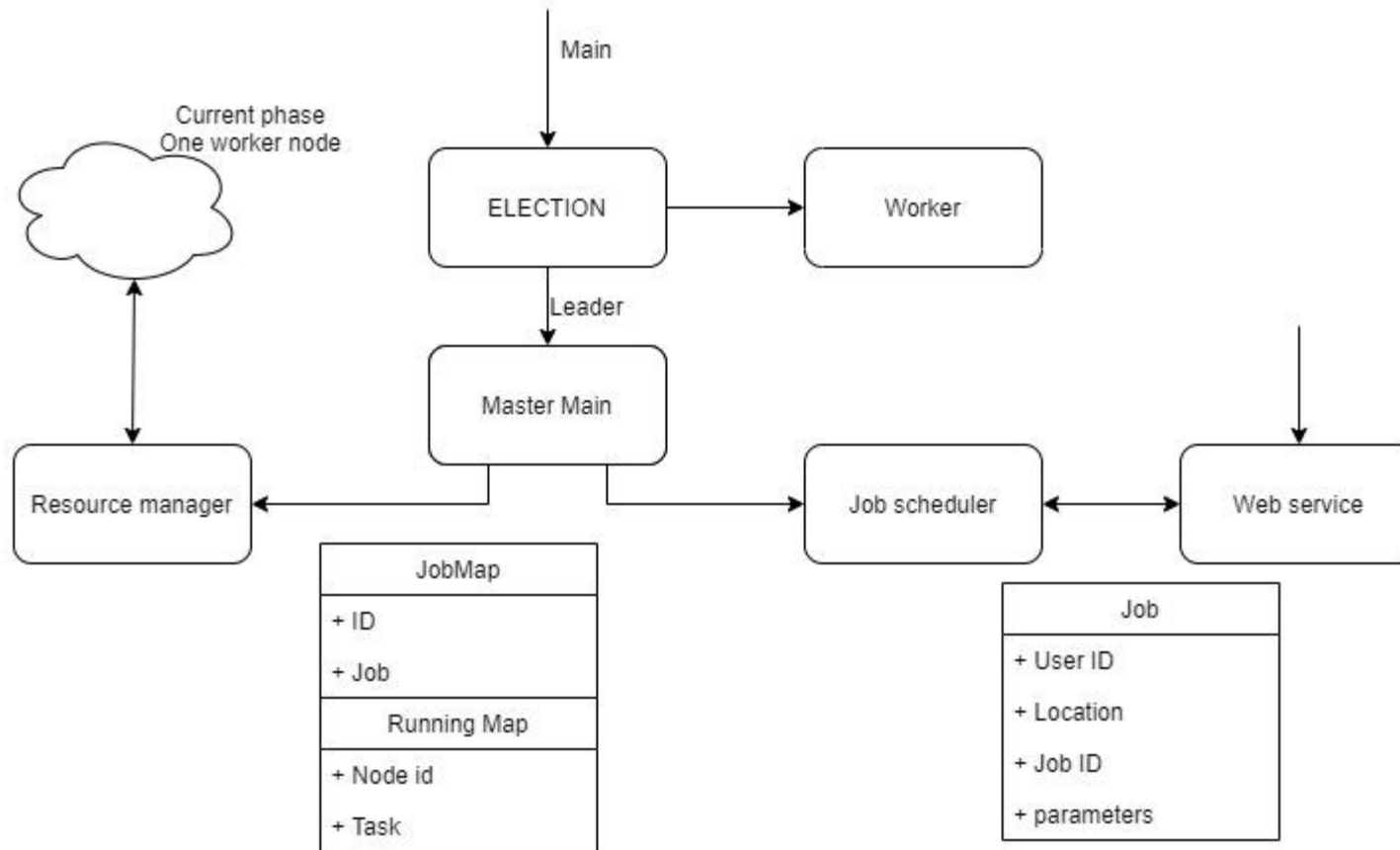


# Plan

- Master-worker on DAS5 (Finished) using Word Count as debug case
  - Simple JNI is available, but integrate Sagecal is a problem, system call as a replacement for now
- Fault tolerance (Finished); issue: Too Slow for failure detection
- Auto-scaling: monitoring and scaling policy (Est. 2-3 weeks)
- Cross region: web server handle it (Est. 1-2 weeks)
- Plus: handling failure of Ibis server(remote site backup)
- Literature study



# Simple Design



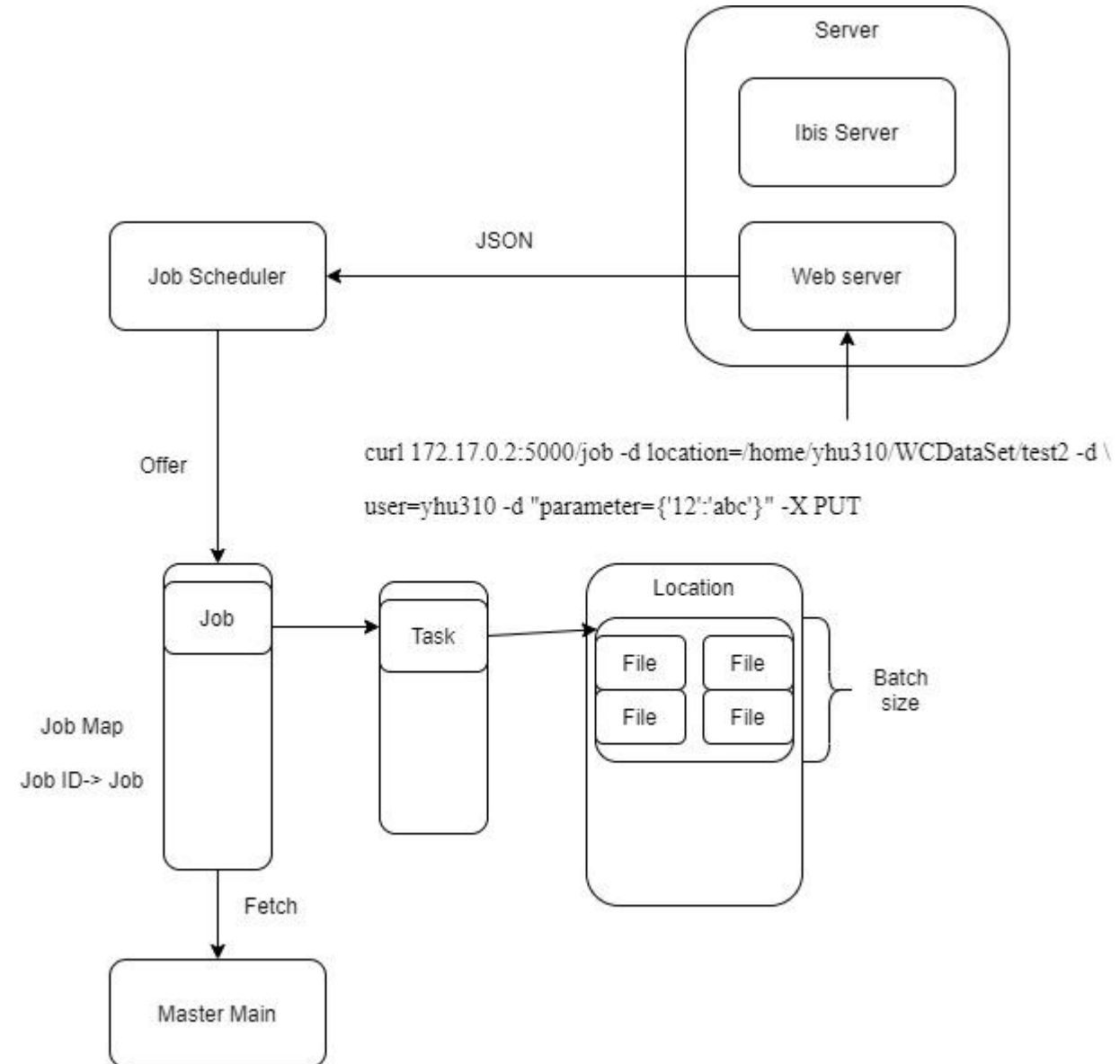
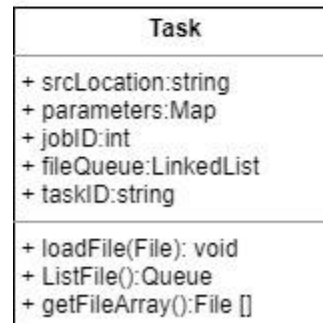
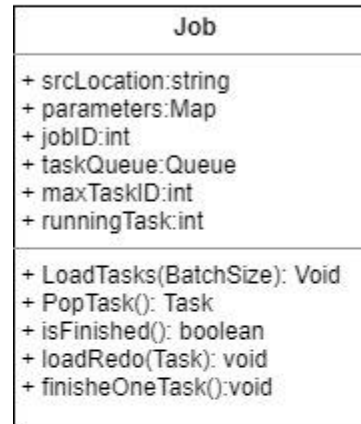
Same code, Different action

DAS5 provides uniformed namespace

# Job submission

## RestFul API

Job loads files to tasks

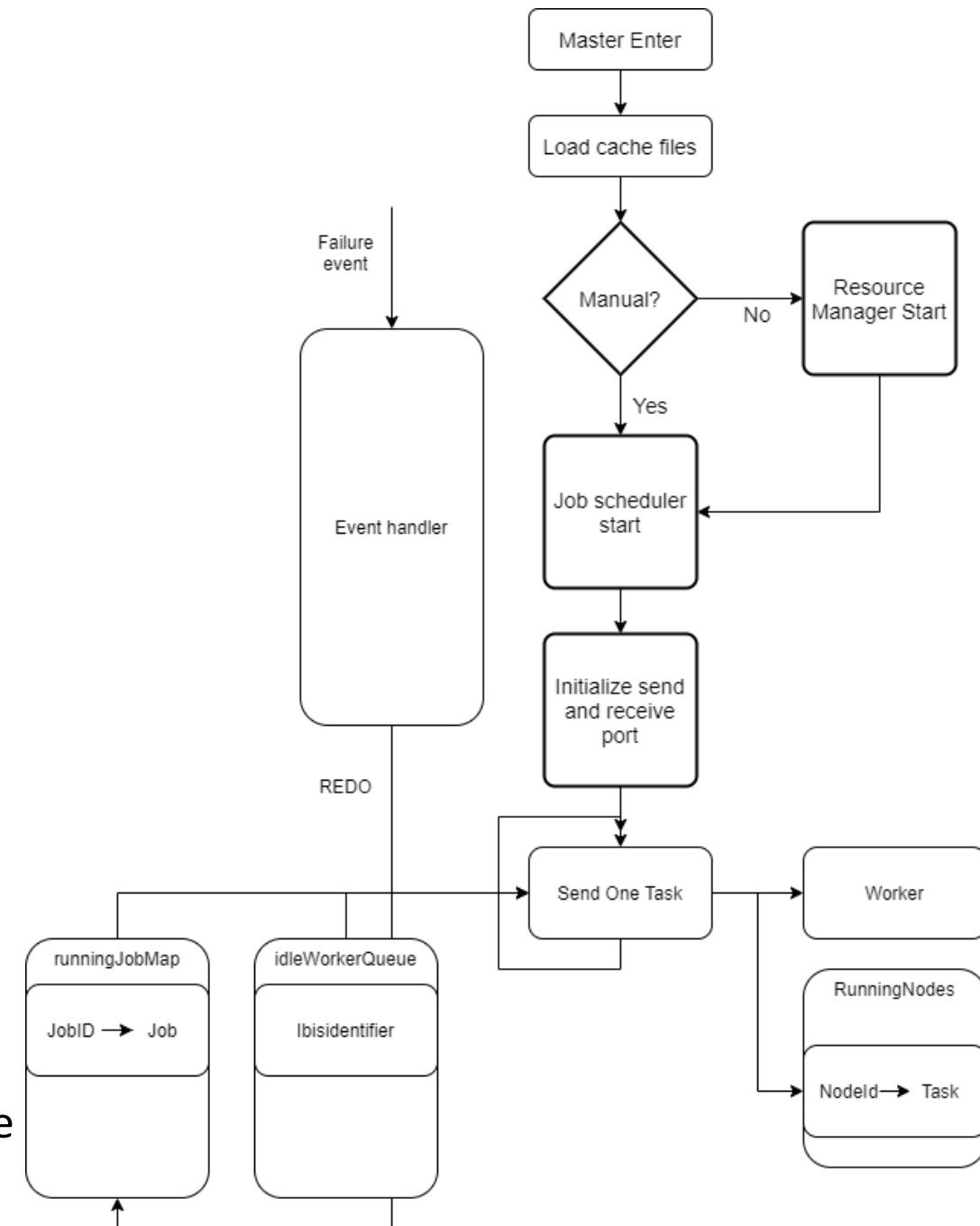


# Master main

First initial settings

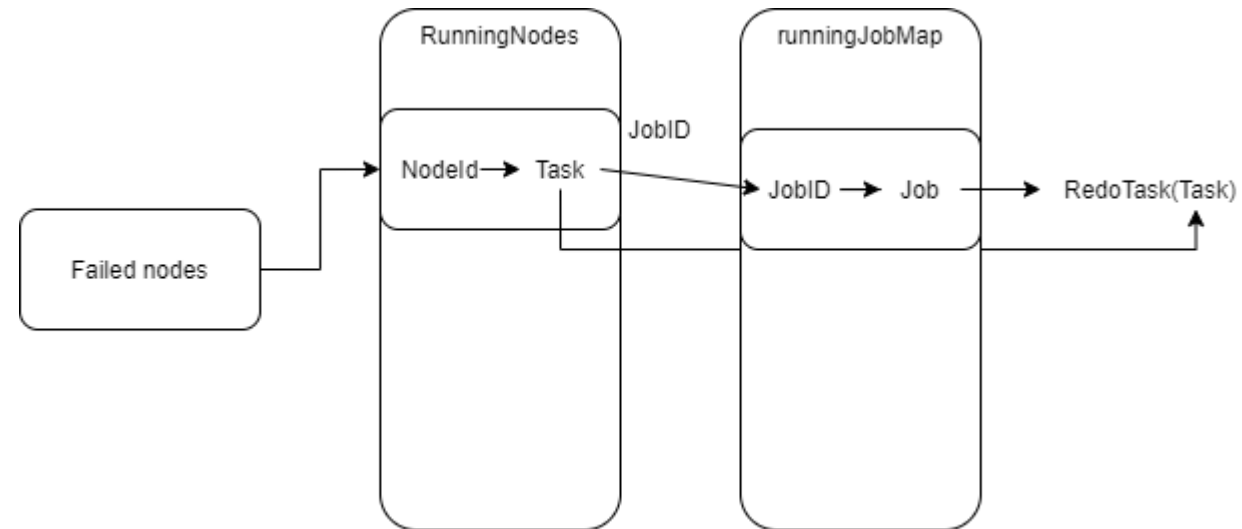
Then loops for task deliver

- 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

- EventHandler receive died events
- According to the id of failed node, redo the task
- Caching the RunningJobMap and the tasks on RunningNodes



# 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, parameters
  - Send back the result to master
- Upcall: receive task from master and push to workerTaskQueue
- Any time detect fail of master, forcedly terminate the task and reelection

# 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 available

# Issues

- The failure detection(event handler) takes 1-2 mins
  - For worker's failure its ok
  - For master failure, it costs too much
  - Is it possible to configure the heartbeat time intervals?
- Snapshot of running jobs
  - Build-in serialization+Input/Output Stream always EOFException
- Integrate SAGECAL With JNI
  - Currently using *ubuntu:bionic* as base container

# TODO – auto-scaling

Parameters:

- $MaxN$ : max number of nodes for use
- $MinN$ : min number of nodes for use
- $T_i$ : total number of tasks in job  $i$
- $L_i$ : number of tasks left in job  $i$

$$\eta_i = \frac{L_i}{T_i}$$

- $\eta_i$ : unfinished rate of job  $i$

$$Goal = MAX(MaxN \times AVG(\eta), MinN)$$

- $Goal$  is the number the system want to keep

Scaling condition

$$Index = Demand - Queue$$

$$\begin{cases} Scale\_up & Index > 0 \\ Scale\_down & Index < 0 \end{cases}$$

$$Demand = Goal - Reserved$$

$$Queue = (Required_{top} - IdleN) \times Time$$

Where  $Reserved$  is the number(minus 1 to exclude the master node) of nodes reserved for our system;  $Required_{top}$  is the required number of nodes that the job on top of the queue needs;  $IdleN$  is the number of Idle nodes in the cluster;  $Time$  is the time from the first time job  $i$  show on the top of the queue till now, this provides a priority to the job in the queue.

What is the scheduling policy on DAS5?

Strict FIFO /priority Queue? Preemptive? Fill back?

How to configure the scheduler arguments?

`description.setSchedulerArguments();`

How to scale down elegantly? Force kill or notify and end by itself?



# TODO – performance evaluation

- Metrics: resource utilization rate; speedup compare to MPI and Spark
  - Resource utilization: probation of idle worker
  - Speedup: from the time submitting job to the time job finished
- Require: reproducible test environment
  - Simulate incoming jobs (with various setting: NP, execute time)
  - Data set with different size (100MB, 1GB, 5GB or more? What is LOFAR daylily case?)

# TODO – cross region

The data set will be partitioned into multiple sub set  
Server split and transfer job to other sites  
The output file should be transferred and collected

Or shared folder among clusters

