My plan:

to change its scheduler, to implement function, finally solve its load-imbalance problem, reduce it’s overhead and improve the performance.

**Default scheduler:** loosely round -robin scheduler (RRS)

Emitter try to send the task in a round-robin fashion,

but in case one of the workers' input queue is full, the Emitter does not wait

till it can insert the task in the queue, but jumps to the next worker until the

task can be inserted in one of the queues. This is a very simple policy but it

doesn't work well if the tasks have very different execution costs.

**Improved scheduler**: adaptive round-robin scheduler (ARRS)

Focusing on the computing capability of workers,faster workers will receive more

tasks than slower workers. By keeping the height of the input queues roughly at the same level.

**The way to change scheduler:**

FastFlow provides a suitable way to define a task-farm skeleton with the

"auto-scheduling" policy. This scheduling behavior may be simply implemented by using the method set\_scheduling\_ondemand() of the ff\_farm class, So that I can use set\_scheduling\_ondemand() function to replace the default Fastflow round-robin scheduler by my new scheduler and fully transparent to the user.

**Schedule algorithm:**

# define CHUNK SIZE 300

# define NEAR END 4\*NUMBER OF WORKERS

int n\_cheduled = 0

def arrsScheduler ( the task )

if ( n\_scheduled<CHUNK SIZE)

/ / hand􀀀out tasks

if (queuesUnbalanced ( ) )

scheduleLowestQueue (the\_task )

else

scheduleRoundRobin (the\_task )

if (CHUNK\_SIZE – n\_scheduled==NEAR\_END)

sendFakeTaskToAllWorkers ( )

n\_scheduled++

return true

else

/ / wait for queues to run empty

waitForFakeTask ( )

recordQueueLevels ( )

n\_scheduled = 0

return false