A Two-Step Policy Approach used in Cloud Web Service Scheduling Problems

New Mexico

Supercomputing Challenge

Final Report

April 3rd, 2013

Team 59

Los Alamos Middle School

Team Members

Steven Chen

Teacher(s)

Pauline Stephens

Project Mentor

Hsing-Bung Chen

Table of Content

1 Executive Summary

Inspired by problems in data center scheduling, I explore scheduling policies which work best to handle the requests for cloud web data centers. Cloud Computing is a technology that utilizes the internet and remote servers to maintain data and applications. Cloud computing allows consumers and businesses to use applications without installation and access their personal files at any computer with access to the internet. This technology allows for much more efficient computing by centralizing data storage, processing and bandwidth. A data center, known as a server farm or a computing room, is created to provide web services. The data center consists of homogeneous computing server farms and very large scale storage systems. Many large data centers are built to support global scale web-services such as: Google, Facebook, Microsoft Windows Azure cloud server, and Amazon Simple Storage Service (S3). I intend to model the cloud web service as a two-step operation. In step-one, each web based cloud service request is treated as a task that needs computing and storage access. In step-two, I then find an available computing server to handle that selected task. The concept of employing a two-step scheduling policy is to combine the intelligent of task selection and server selection and find an optimal solution to support an efficient cloud service model. For task selection, we adapt the combination First-Come-First-Serve and Shortest-Job-First policies. For server selection, I use the combination of Round-Robin and Shortest-Server-Queue-First policies. I then create five two-step policies consisting of task and server policies. NetLogo [1][2] is used to develop these five proposed two-step scheduling policies. I simulate various cloud service request workloads on those five two step policies and conduct lengthy tests. I collect performance data in terms of waiting time, queuing time, turnaround time, and server utilization. I use the results of numerous tests to judge the qualities of each scheduling policy. Testing results show that Shortest-Job-First and Shortest-Server-Queue-First had decent performance and produced better results in terms of less queuing time and a quick turnaround time.

2 Introduction and Problem Statement

A server farm is a group of computing servers and data storage servers that are connected through interconnected network systems such as Gigabit Ethernet network or 10 Gigabit Ethernet network. Normally they are co-located in the same location. A server farm streamlines internal processes by distributing the workload between the individual components of the farm and expedites computing processes by harnessing the power of multiple servers. Typical server farm infrastructure is shown in Figure 1.

The server farms rely heavily on load-balancing software that accomplishes tasks such as; tracking demand for processing power from different machines, prioritizing the tasks, and scheduling and rescheduling tasks depending on priority and demand that users put on the network. When one server in the farm fails, another can step in as a backup. Combining servers and processing power into a single entity has been relatively common for many years in research and academic institutions. Today, more and more companies are utilizing server farms to handle the enormous amount of tasks and services.

The problem with web server farms is that load imbalances may occur, some servers may be idle with nothing to execute, while other serves are busy and with overloading assigned jobs.

A load balancer plays a key role in server farm architecture. It serves as a computing and data traffic cop to direct the incoming requests to suitable servers. Load balancing is especially important in cases where it is difficult to anticipate the potential number of requests that a server farm will receive. The ultimate goal of the load balancer is to distribute jobs to the servers so that no particular server is overwhelmed while others are relatively idle [3][4][5][6].

Many static and dynamic job scheduling approaches have been proposed and studied in the past. Researchers are focusing on either the job-shop scheduling problem [7][8][9] or the server workload scheduling problem [10][11][12].

2.1 Problem Approach

In this project, I apply a two-step policy approach that combines the job selection and server selection policies. In the first step, two different job selection policies are applied to pick a candidate arriving. In the second step, two different server selection policies are used to locate an available server. Finally the two-step policy then assigns a job to a selected server. I use the NetLogo agent based programming language to implement my proposed two-step policies. I test my NetLogo program on several common benchmarks, and the results of the experiments show that this two-step approach is quite effective and efficient.

2.2 Details the proposed two-step policies

2.2.1 Step-One: Job Selection Policy

Policy 1A FCFS: First Come and First Served

The dispatcher selects job(s) based on the job's arriving order. Jobs may or may not arrive at the same time. Jobs that arrive at the same time will be dispatched at the same time in random order.

Policy 1B SJF: Shortest Job First

The dispatcher selects job(s) based on the estimate processing time of jobs. Jobs with shortest processing time are dispatched first.

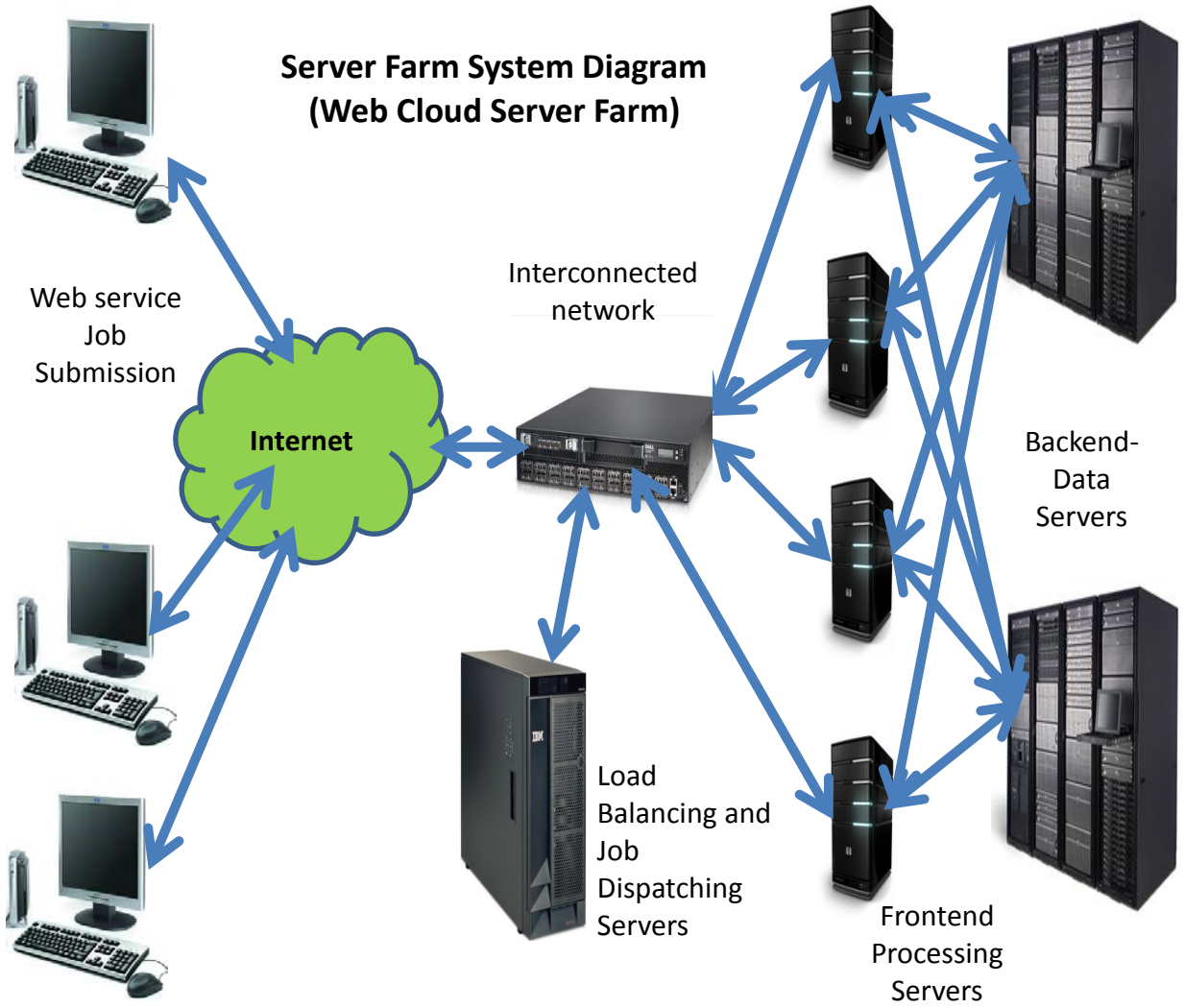

Figure-1: Typical Server Farm Infrastructure.

2.2.2 Step-Two: Server Selection Policy

Purpose: Selecting An Available Server

The dispatcher selects an available server and assigns a selected job from Step-One policy to this server. If this selected server is idle and available to execute a job, it then starts to execute the job. If this selected server is currently executing a job, the job is put in the server waiting queue.

Policy 2A: RR: Round Robin

Select servers in chronological order and looping until all jobs are distributed.

Policy 2B: SSQ: Shortest Server Queue using number of jobs in queue

Select a server with the minimum number of jobs is its job queue.

Policy 2C: SSQ2: Shortest Server Queue using total processing time of jobs in queue

Select a server with the minimum total processing time of jobs in queue

Conceptually, Policy 2C can present a more precise measure of the queuing workload in a server.

2.3 Five "Two-Step Policies"

2.3.1 FCFS-RR: First-Come-First-Served and Round-Robin Server Selection

2.3.2 FCFS-SSQ: First-Come-First-Served and Shortest Server Queue Server Selection

2.3.3 SJF-RR: Shortest Job First and Round-Robin Server Selection

2.3.4 SJF-SSQ: Shortest Job First and Shortest Server Queue Server Selection (using number of job in a server queue)

2.3.5 SJF-SSQ2: Shortest Job First and Shortest Server Queue Server Selection (using total sum of job lengths in a server queue)

3 NetLogo Simulation Program Design and Implementation

3.1 Simulation System – Agent Design and Implementation

In this Netlogo simulation I create three different agents: Job Agent, Dispatcher agent, and Server agent. The interaction between agents is handled through a state machine system shown in Figure 3.

Job Agent:

jobID, jobLength, arrivingTime, expectedStartingTime, dispatchedTime, waitingTime, queuingTime, actualStartingTime, finishedTime, jobStatus, turnaroundTime, JobServerID.

I create up to 1,000 job agents in my simulation program.

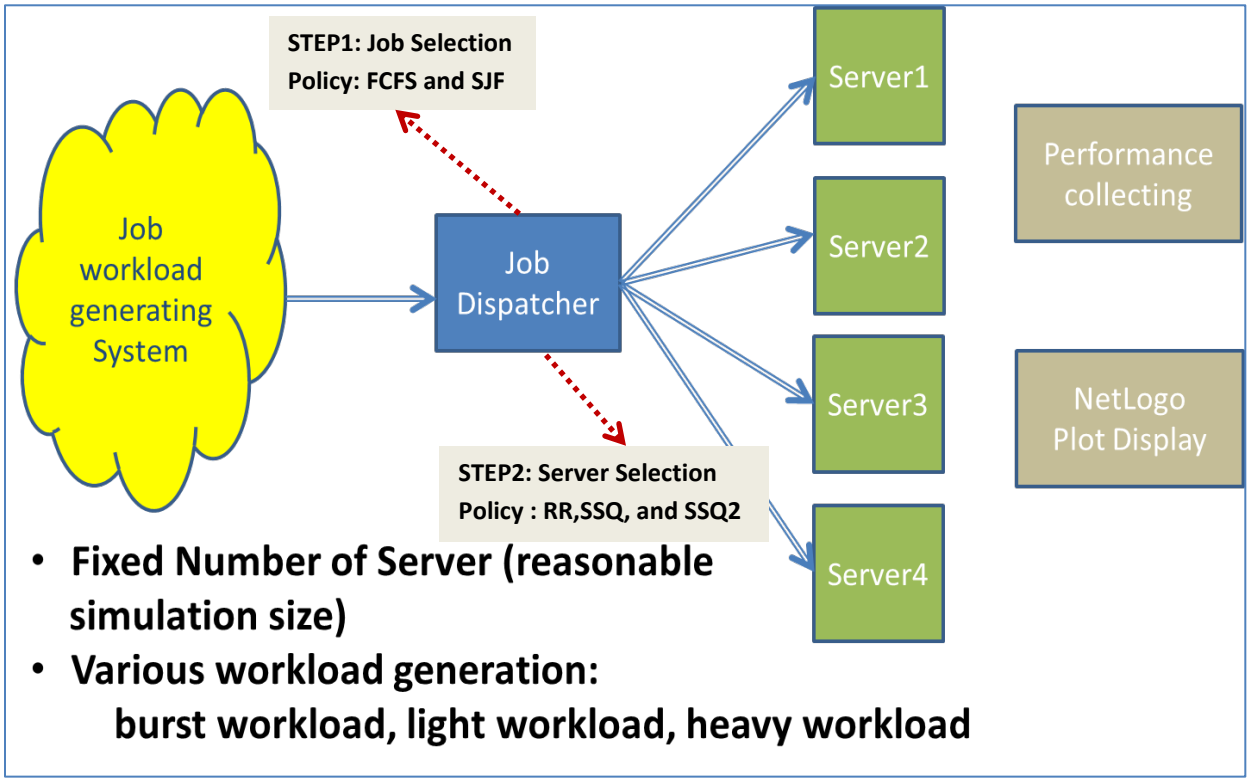

Figure 2: Two-Step Policy System Diagram of Multi-Agent Simulation Program

Dispatcher agent:

A dispatcher is used to pick an arriving job and select an available server to run this job. Only one central Dispatcher is created in my simulation.

Server agent:

A server agent is used to process assigned jobs and record server performance information during the simulation. A server agent record information such as currentStartingTime, nextStartingTime, numberJobAssigned, numberJobProcessed, numberJobFinished , and numberJobInQueue

Eight server agents are created in simulation.

3.2 Simulation System State

Seven states are designed to represent the various cycles of a web server job as shown in Figure 3. Run time actions of various state transitions are described as followed:

- **State JOB-NOT-DISPATCHED**: A job is set to this state when created
- **State JOB-ASSIGNING**: A job is moved to this state when its arriving time is equal to the simulation wall-clock (Netlogo's tick counter).
- **State JOB-DISPATCHED**: A job is moved to this state when it is dispatched to a selected server.
- **State JOB-WAITING-IN-QUEUE**: A job is transferred to this state when a selected server cannot execute this assigning job right away. This job is put into a server's queue and waits to be executed.
- **State JOB-PROCESSING**: A job is transferred to this state when a selected server can execute the assigned job. This job's previous state can be JOB-DISPATCHED or JOB-WAITING-IN-QUEUE.
- **State JOB-FINISHED**: A job moved to this state when its scheduled finishing-time is equal to the current wall-clock (Netlogo's tick counter). This state also increases the finished-job counter by one.
- **State JOB-DONE**: This state is reached when all created job are finished. This occurs when the number of created jobs is equal to the number of finished jobs. Final performance data is generated such as Average-Serverfarm-Utilization, Average-TurnaroundTime, Average-Throughput, Average-WaitingTime, Average-QueuingTime, Average-QueuingJob, Average-ProcessingTime

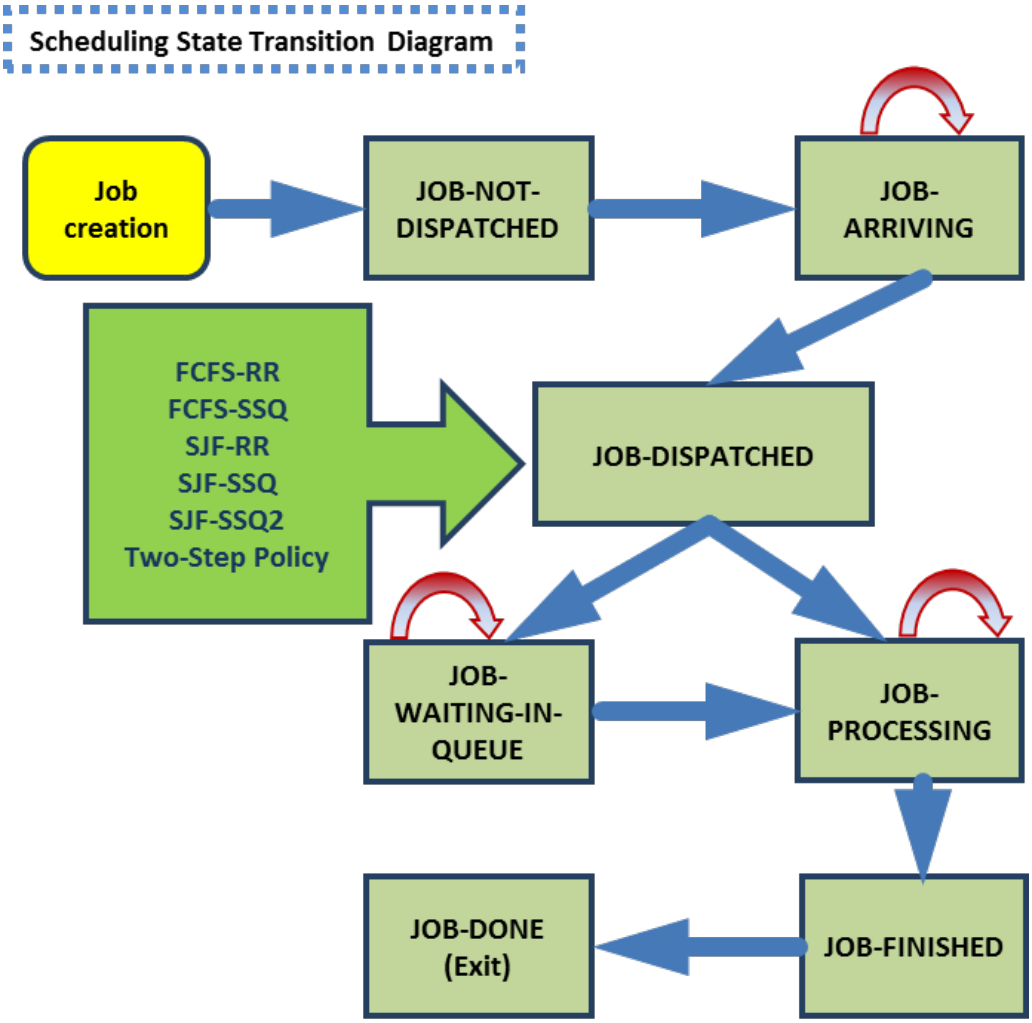

Figure 3: State transition Diagram

4 Performance results

To analyze performance of these five two-step scheduling policies I have conducted a sequence of testing on this simulation program. I used performance metrics defined as followed.

AverageQueuingTime =

($\sum$(JobStartingTime - JobArrivingTime) / NumberOfJobCreated

This metric shows the ability of a scheduling policy to reduce job queuing time and improve server efficiency. A lower value indicates a better response time for a job. It shows how fast a job can be processed by a server after it has arrived. The system queuing time is the interval between the arrival of a job and the start of processing this

job. It can be also referenced as the “response time”. A quick response time in web streaming service has a strong effect on user satisfaction and usability.

AverageTurnAroundTime =

($\sum$(JobTurnaroundTime) / NumberOfJobCreated

Turnaround time is the delay between submission of a job for a processing system and its completion. This metric indicates the degree of satisfactory for a scheduling policy. Most web service users expect to have a shorter turnaround time. Shorter turnaround time means better service quality.

AverageServerUtilization =

($\sum$(TotalProcessingTime / TotalServerUpTime) / NumberofServer

This metric represents how a scheduling policy contributes to overall system power consumption and cost-effectiveness. Higher values obtained from server utilization hints that more jobs can be handled during the same amount of system up time. It is a quite important success factor in today’s data center market. High server utilization can also help to reduce infrastructure costs.

Figure 4, Figure 5, and Figure 6 show that SJF-SSQ2 can out-perform the other four policies with its “Shortest Job First” and “Shortest Server Queue Length” approach.

4.1 Discussions

FCFS-RR is easy to implement but it is lacking compared any intelligent policy in job and server selection.

FCFS-SSQ and SJF-RR have similar performance because they both apply only one intelligent approach when selecting a job or a server.

SJF-SSQ and SJF-SSQ2 have close performance results and both obtain better results than FCFS-RR, FCFS-SJQ, and SJF-RR policies. Both two-step policies utilize two smart selection methods.

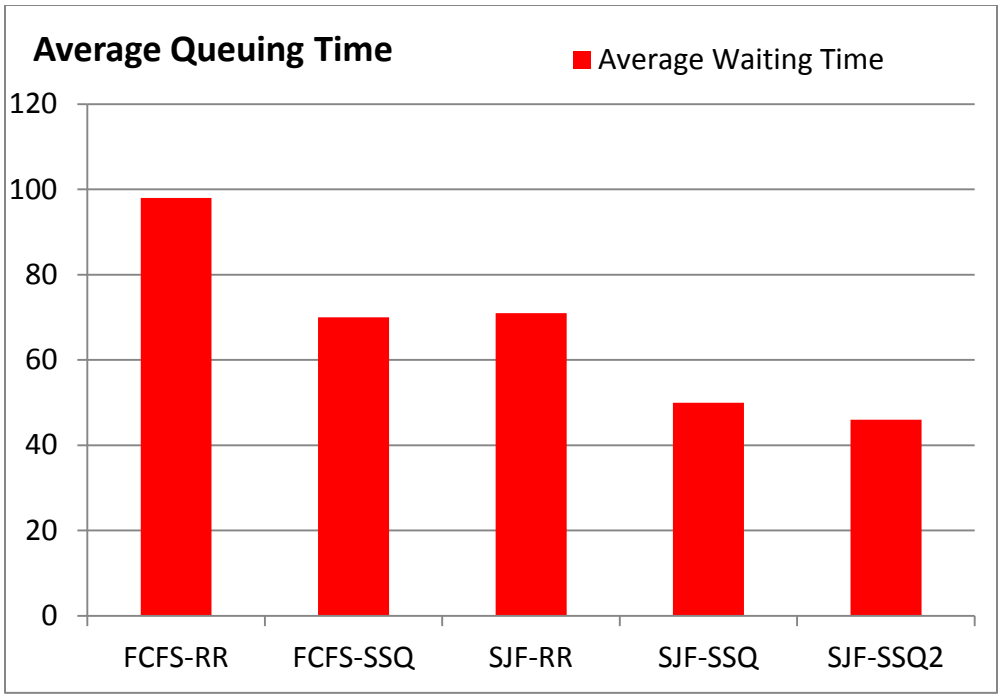

Figure 4: Average Queuing Time Comparison

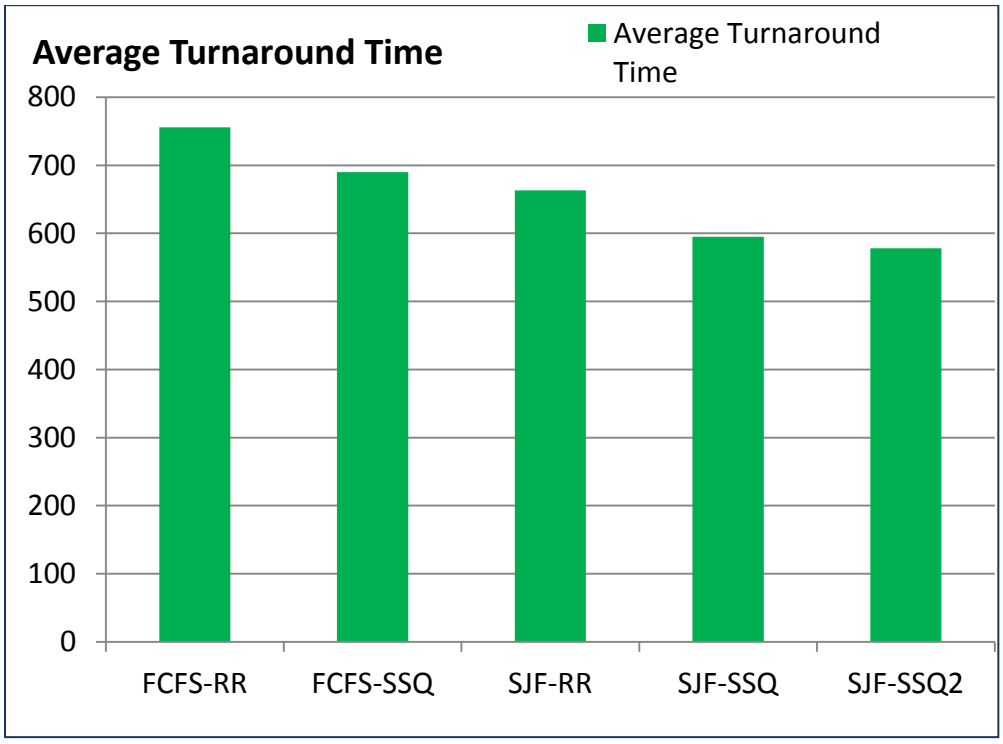

Figure 5: Average Turnaround Time Comparison.

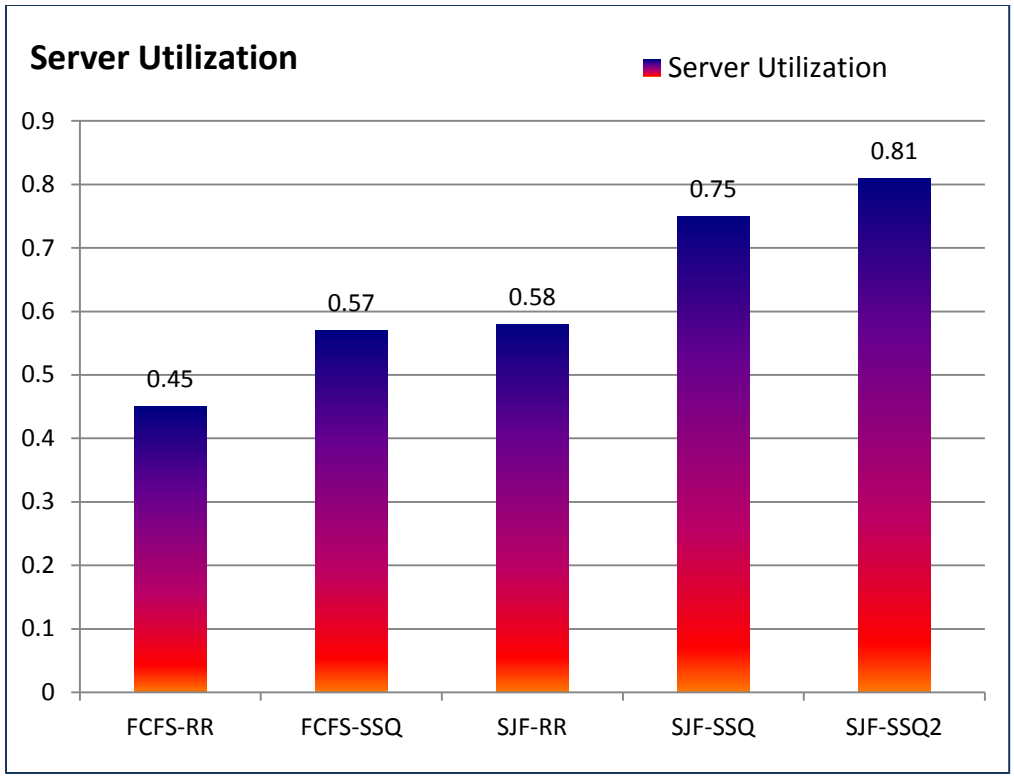

Figure 6: Average Server Utilization Comparison

SJF-SSQ2 can obtain a little better results comparing to SJF-SSQ because it uses actual queuing workload as the server selection criteria. Simulation results justify this.

In Figure-7, the variance of number of jobs in each server queue is small because SSQ uses the number of jobs in the server queue to decide which server should be picked to handle a selected job from the Step-One policy. The variance total queuing job length in each server queue is fairly large.

In Figure-8, SSQ2 uses the total amount of a job's queuing length to locate a server to processes a job. Figure-8 clearly shows a different scenario from Figure-7.

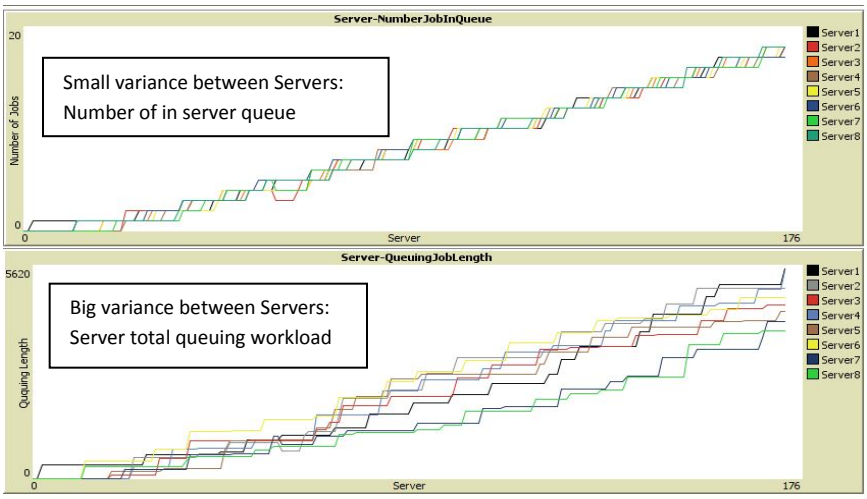

Figure-7: SJF-SSQ , Number of Jobs In Queue vs. Server Workload Queuing

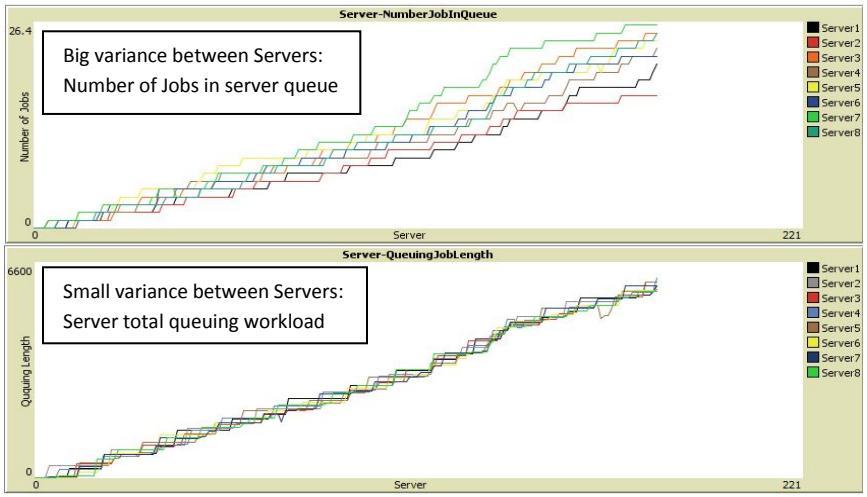

Figure-8: SJF-SSQ2, Number of Jobs In Queue vs. Server Workload Queuing

Figure-9 Illustrates a completed simulation of the SJF-SSQ2 method.

[A] Initially, the Job-Dispatcher keeps dispatching arriving jobs until all jobs are dispatched. I assume in this case that each server has unlimited server queue so it can accept as many jobs as possible.

[B] After all jobs are dispatched, servers continue to process jobs queued in each server. We can see the queuing length of each server decreasing over time. It means that jobs are removed from a server queue and are being executed.

[C] Also we can see that number of processing jobs and finished jobs are increasing over time.

[D] Eventually we see (1) each server's queue is empty, (2) and all jobs are finished.

The visualized patch area is used to animate the state transition procedure and workload distribution

(Job Dispatch→Job Queuing→Job Processing→Job Finished)

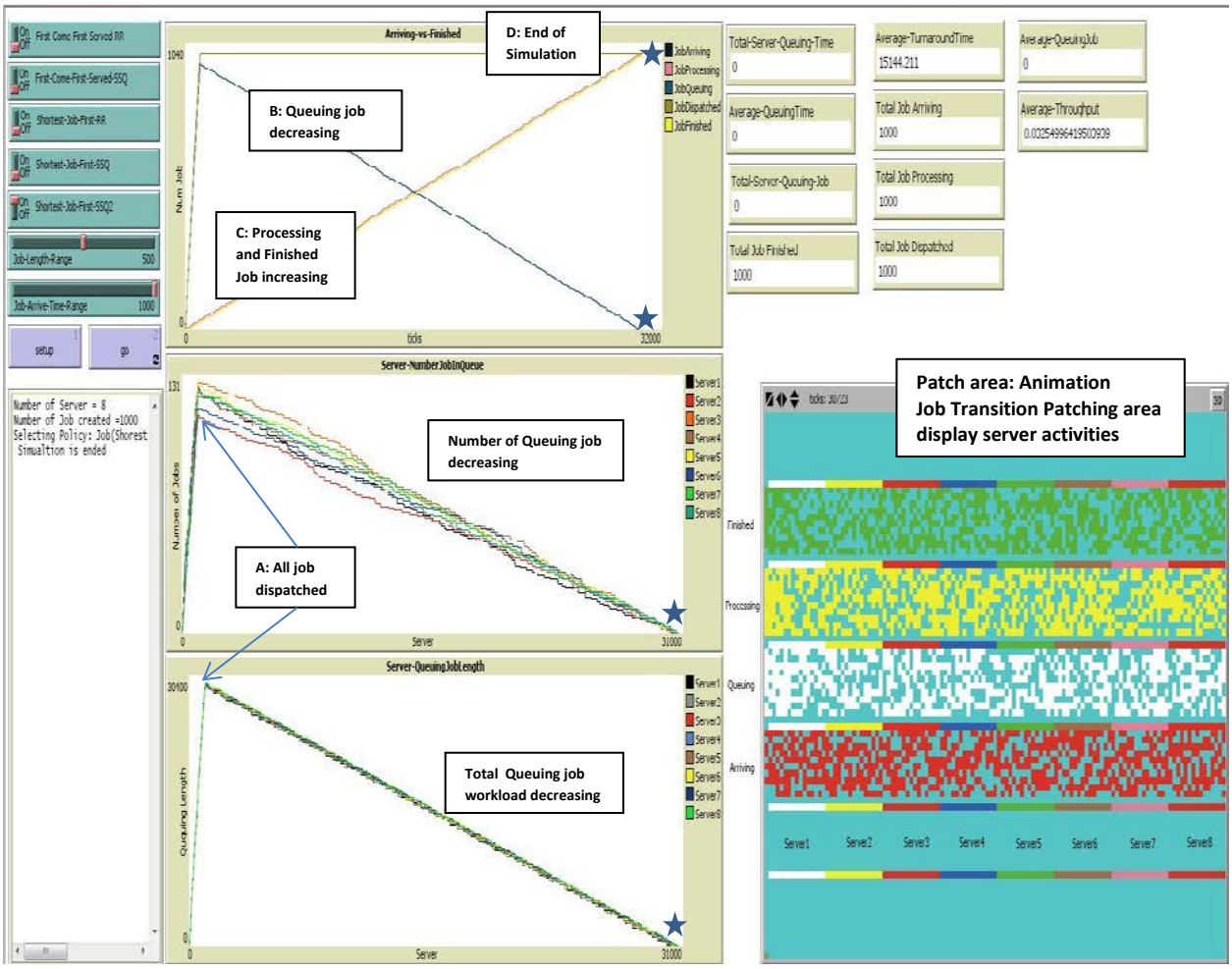

Figure-9: A complete visualized SJF-SSQ2 simulation process

5 Major Achievement

Server farms are heavily deployed in today large data centers. The performance bottleneck of a server farm is limited to the overall power resource and the power consumption. Computing servers run 24/7 in server farms and consume vast amounts of electricity. I believe that better server utilization from a well-designed job scheduling program can help to improve system service quality and further reduce power consumption for handling the same amount of web service requests. When a computing server can quickly handle service and reduce

unnecessary idle time, it indicates that a server can move to deep sleep state and reduce power consumption.

With the combination of job selection and sever selection, the SJF-SSQ and SJF-SSQ2 have clearly proved that they can reduce a job's queuing-time, quickly return results to web service jobs, and sustain better server utilization. My simulation has shown evidence supporting this. My evidence further suggests that an efficient job dispatcher can maintain a balanced workload among servers, decrease job turnaround time, and eventually receive a high degree of customer satisfaction. Furthermore it shows a positive sign of using those methods to reduce power consumption in large scale data centers.

6 Acknowledgments

First, I would like to thank the Super-Computing Challenge program and the committee members. The kickoff program at New Mexico Tech was especially helpful. Through this project, I gained knowledge, useful information, and a variety of idea to implement in my simulation program. Furthermore, we would like to thank the comments and suggestions from my intern reviewing judges. I would also like thank our Supercomputing Challenge Project teacher Mrs. Pauline Stephens for sponsoring my project and her constant encouragement throughout these in the past five months. Her unwavering support for the past few months has been amazing. In addition, I also would like to thank Ariel Chen for reviewing my report and Mr. HB Chen for mentoring my project.

Finally I would like to thank my parents for helping me prepare posters, editing the final project report, and setting up different testing environments.

References

[1] NetLogo: http://ccl.northwestern.edu/netlogo/

[2] NetLogo 5.0.3 User Manual: Programming Guide,

[3] Mor Harchol-Balter Computer Science Dept, Carnegie Mellon University, "Scheduling in Server Farms", presentation 2007

[4] Varun Gupta Mor Harchol-Balter Karl Sigman Ward Whitt, "Analysis of Join-the-Shortest-Queue Routing for Web Server Farms", Performance 2007, Performance Evaluation, Vol 64, Issues 9-12

[5] Anshul Gandhi, Mor Harchol-Balter,Rajarshi Das, Charles Lefurgy, "Optimal Power Allocation in Server Farms", CMU Technical Report March 2009, CMU-CS-09-113

[6] CPU scheduling, Duke Class note , http://www.cs.duke.edu/~chase/cps210-archive/slides/cpu.pdf

[7] Walsh P., “A high-throughput computing environment for job shop scheduling (JSP) genetic algorithms”, Evolutionary Computation, 2004. CEC2004. Congress on
[8] Albert Jones and Luis C. Rabelo, “Survey of Job Shop Scheduling Techniques”, Published Online: 27 DEC 1999
[9] Anant Singh Jain and Sheik Meeran, “A State-of-art Review of Job-Shop Scheduling Techniques”, Department of Applied Physics, electronic and Mechanical Engineering, University of Dundee, Dundee, Scotland, UK DDI 4HN
[10] Richard Olejnik, Iyad Alshabani, Bernard Toursel, Eryk Laskowski, Marek Tudruj, “Load balancing in the SOAJA Web Service Platform”, IEEE 2008 Proceedings of the International Multiconference on Computer Science and Information Technology
[11] George Porter and Randy H. Katz, “Effective Web Service Load Balancing through Statistical Monitoring”, Communications of ACM, March 2006
[12] Digvijay Singh Lamba , Pankaj Jalote , and Dheeraj Sanghi , “A Web Service for evaluation of load balancing strategies for distributed web server systems”, Dept. of Computer Sc. and Engg., IIT Kanpur

Appendix – Netlogo program

```
;; A Two Step Policy Approach used in Cloud Web Service Scheduling Problems
;;
;;
;; Implemented scheduling policy
;;
;; Two Step Policy:  Job Selection aand Server Selection
;; 1. First-Come-First-Served and Round-Robin Server Selection: FCFS-RR
;; 2. First-Come-First-Served and Shortest Server Quque Server Selection : FCFS-SSQ
;; 3. Shortest Job First and Round-Robin Server Selection: SJF-RR
;; 4. Shortest Job First and Shortest Server Queue Server Selection: SJF-SSQ
;;
;; Performance Studies
;;
;; 1. Average Job Throughput
;; 2. Average Server- Farm Utilization
;; 3. Average Job Turnaround Time
;; 4. Average Job Waiting Time
;; 5. Job Scheduling Markspan

breed [dispatchers dispatcher]  ;;  0

;; Job
breed [jobs job] ;; 1 - 1000

;; Server Farm
;; Fixed number of servers used in simulation
breed [ServerFarm  Server]  ;; 1001  to   1008


;; global variables
globals
[
   Number-Server-Used
   Number-of-jobs

   ;; First Come Fist Served Performance counters
   Average-Serverfarm-Utilization
   Average-TurnaroundTime
```

```
Average-Throughput
Average-WaitingTime
Average-QueuingTime
Average-QueuingJob
Average-ProcessingTime

Total-Server-Queuing-Job
Total-Server-Queuing-Time

MaxWaitingTime
MaxQueuingTime
MaxWaitingJob
MaxQueuingJob

Markspan

FinalFinishedTime
TotalProcessingTime
TotalWaitingTime
TotalQueuingTime
TotalTurnaroundTime
Current-Server-ID
Next-Server-ID

selectedServer
TotalJobFinished
TotalJobProcessing
TotalJobWaiting
TotalJobQueuing
TotalJobDispatched
TotalJobArriving
TotalJobStarting
TotalJobCreated

ProcessingJob
currentJobID
terminatingSimulation

numPolicyUsed ;; number of policy used in simulation
numPolicyDone ;; number of policy done with simulation
numJobCreated ;; number of Job created for simulation
numServerUsed ;; number of server used in simulation
numReportDone ;; number of performance report is done

;; Flag of POLICY
POLICY-SELECTED
POLICY-NOT-SELECTED
POLICY-DONE
POLICY-NOT-DONE

;; POLICY Select Status information
FCFS-RR-policy-select-status
FCFS-SSQ-policy-select-status
SJF-RR-policy-select-status
SJF-SSQ-policy-select-status
SJF-SSQ2-policy-select-status
;; POLICY Status information
policy-status

MaxJobLength
MaxQueueLength
GlobalJobID
CurrentJobWaiting
CurrentJobQueuing

selected-Server

;; JOB status related information
JOB-UNKNOWN
JOB-READY
JOB-ARRIVING
```

```
   JOB-WAITING-IN-QUEUE
   JOB-STARTING
   JOB-PROCESSING
   JOB-FINISHED
   JOB-NOT-DISPATCHED
   JOB-DISPATCHED
   JOB-DONE

   Server121-QueueLength
   Server121-NumJobInQueue
   Server121-xcor
   Server122-QueueLength
   Server122-NumJobInQueue
   Server122-xcor
   Server123-QueueLength
   Server123-NumJobInQueue
   Server123-xcor
   Server124-QueueLength
   Server124-NumJobInQueue
   Server124-xcor
   Server125-QueueLength
   Server125-NumJobInQueue
   Server125-xcor
   Server126-QueueLength
   Server126-NumJobInQueue
   Server126-xcor
   Server127-QueueLength
   Server127-NumJobInQueue
   Server127-xcor
   Server128-QueueLength
   Server128-NumJobInQueue
   Server128-xcor

   jobWaiting-xcor
   jobWaiting-ycor
   jobArriving-xcor
   jobArriving-ycor
   jobDiapatched-xcor
   jobDispatched-ycor
   jobQueuing-xcor
   jobQueuing-ycor
   jobProcessing-xcor
   jobProcessing-ycor
   jobFinished-xcor
   jobFinished-ycor
]


;; Job agents own data
jobs-own
[
  jobID
  jobLength
  arrivingTime
  expectedStartingTime
  dispatchedTime
  waitingTime
  queuingTime
  startingTime
  finishedTime
  jobStatus
  lastJobStatus
  jobDispatchStatus
  turnaroundTime

  responseIndex ;; turnaroundTime - jobLength
  responseRatio ;; jobLength / turnaround

  jobServerID
  assignedServerID
```

```
  myjobWaiting-xcor
  myjobWaiting-ycor
  myjobArriving-xcor
  myjobArriving-ycor
  myjobDiapatched-xcor
  myjobDispatched-ycor
  myjobQueuing-xcor
  myjobQueuing-ycor
  myjobProcessing-xcor
  myjobProcessing-ycor
  myjobFinished-xcor
  myjobFinished-ycor
]

;;  server own data
ServerFarm-own
[
    currentStartingTime
    nextStartingTime
    numberJobAssigned
    numberJobProcessed
    numberJobFinished
    numberJobInQueue      ;; server queue length

    ServerWorkloadQueuing ;; sum of workload from current waiting job in queue

    Server-Average-Utilization
    Server-Average-TurnaroundTime
    Server-Average-Throughput
    Server-Average-ProcessingTime
    Server-Average-Queuing-Job

    Server-ResponseIndex
    Server-AverageResponseIndex
    Server-ResponseRatio
    Server-AverageResponseRatio

    ServerMaxQueuingJob

    Server-CurrentJobQueuing
    Server-totalProcessingTime
    Server-totalProcessedJob
    Server-xcoordinate
    Next-serverID
    serverID
]


dispatchers-own
[
  FCFS-RR-status
  FCFS-SSQ-status
  SJF-RR-status
  SJF-SSQ-status
  SJF-SSQ2-status

  TotalDispatchedJob
  TotalJobLength
  TotalArrivingJob
]

;;
;;  setup procedure to initializ all globals data and reset tick counter to zero
;;
to setup
  clear-all

  init-globals  ;; initialize global variables values

  ;;  create one dispatcher
  ;; turtle 0 :  dispatcher
```

```
  create-dispatchers 1  ;; id-0
  create-jobs Number-of-Jobs ;; 1 1000

  setup-Jobs-Info;;
  setup-Jobs-Patch-Info;;

  ;;  create eight servers
  ;; turtle 1-
  create-ServerFarm  Number-Server-Used ;; id 1001-1008

  ask ServerFarm [
      set serverID who
      set Next-serverID serverID + 1
      if serverID = 1009 [  ;; this rounf robin policy
         set Next-serverID 1001
      ]
  ]

  setup-dispatcher
  setup-ServerFarm
  reset-ticks
end


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; setup schedule basic information
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
to setup-dispatcher
   ask dispatcher 0 [
       set FCFS-RR-status   POLICY-NOT-SELECTED
       set FCFS-SSQ-status  POLICY-NOT-SELECTED
       set SJF-RR-status    POLICY-NOT-SELECTED
       set SJF-SSQ-status   POLICY-NOT-SELECTED
       set SJF-SSQ2-status  POLICY-NOT-SELECTED
       set totalDispatchedJob  0
       set totalJobLength      0
       set totalArrivingJob    0
   ]
end

to setup-ServerFarm
   ask  ServerFarm [
      set currentStartingTime           0
      set nextStartingTime              0
      set numberJobAssigned             0
      set numberJobProcessed            0
      set numberJobFinished             0
      set numberJobInQueue              0  ;; server queue length
      set ServerWorkloadQueuing         0  ;; sum of workload from current waiting job in queue
      set Server-Average-Utilization    0
      ;;set Server-Average-ToursounrTime 0
      set Server-Average-Throughput     0
      set Server-Average-ProcessingTime 0
      set Server-Average-Queuing-Job    0
      set maxQueuingJob                 0
      set Server-CurrentJobQueuing      0
      set Server-totalProcessingTime    0
      set Server-totalProcessedJob      0
      set Server-ResponseIndex          0
      set Server-AverageResponseIndex   0
      set Server-ResponseRatio          0
      set Server-AverageResponseRatio   0
      set Next-serverID                 who
      set serverID                      who
      set Server-xcoordinate            (who - 1000) *  16
   ]
end

to setup-Jobs-Patch-Info

     ask patches [
```

```
    set pcolor cyan
  ]


ask jobs [
  ;;set jobID who  ;; who is the turtle number assigned by the NetLogo system

  set jobWaiting-xcor  0
  set jobWaiting-ycor  24
  set jobDiapatched-xcor 0
  set jobDispatched-ycor 24
  set jobQueuing-xcor    0
  set jobQueuing-ycor    36
  set jobProcessing-xcor 0
  set jobProcessing-ycor 48
  set jobFinished-xcor   0
  set jobFinished-ycor   60
  let start-ycor  1
  set jobArriving-xcor  jobID
  set jobArriving-ycor  12

  if jobID <= 15 [
     ;;set jobArriving-ycor  start-ycor
     ask patch jobArriving-xcor jobArriving-ycor [
         set pcolor white
     ]

     ask patch jobArriving-xcor 22 [
         set pcolor white
     ]

     ask patch jobArriving-xcor 34 [
         set pcolor white
     ]

     ask patch jobArriving-xcor 46 [
         set pcolor white
     ]

     ask patch jobArriving-xcor 58 [
         set pcolor white
     ]
     ask patch jobArriving-xcor 70 [
         set pcolor white
     ]
  ]

  set start-ycor  3
  if jobID > 15 [
     if jobID <= 30 [
        ;;set jobArriving-ycor  start-ycor
        ask patch jobArriving-xcor jobArriving-ycor [
            set pcolor yellow
        ]
        ask patch jobArriving-xcor 22 [
            set pcolor yellow
        ]

        ask patch jobArriving-xcor 34 [
            set pcolor yellow
        ]

        ask patch jobArriving-xcor 46 [
            set pcolor yellow
        ]

        ask patch jobArriving-xcor 58 [
            set pcolor yellow
        ]
             ask patch jobArriving-xcor 70 [
            set pcolor yellow
```

```
      ]
   ]
]

set start-ycor  5
if jobID > 30 [
   if jobID <= 45 [
      ;;set jobArriving-ycor  start-ycor
       ask patch jobArriving-xcor jobArriving-ycor [
          set pcolor red
       ]

      ask patch jobArriving-xcor 22 [
          set pcolor red
      ]

      ask patch jobArriving-xcor 34 [
          set pcolor red
      ]

      ask patch jobArriving-xcor 46 [
          set pcolor red
      ]

      ask patch jobArriving-xcor 58 [
          set pcolor red
      ]
      ask patch jobArriving-xcor 70 [
          set pcolor red
      ]

   ]
]

set start-ycor  7
if jobID > 45 [
   if jobID <= 60 [
      ;;set jobArriving-ycor  start-ycor
      ask patch jobArriving-xcor jobArriving-ycor [
          set pcolor blue
      ]

      ask patch jobArriving-xcor 22 [
          set pcolor blue
      ]

      ask patch jobArriving-xcor 34 [
          set pcolor blue
      ]

      ask patch jobArriving-xcor 46 [
          set pcolor blue
      ]
      ask patch jobArriving-xcor 58 [
          set pcolor blue
      ]
      ask patch jobArriving-xcor 70 [
          set pcolor blue
      ]


   ]
]

set start-ycor  9
if jobID > 60 [
   if jobID <= 75 [
      ;;set jobArriving-ycor  start-ycor
      ask patch jobArriving-xcor jobArriving-ycor [
          set pcolor green
      ]
```

```
      ask patch jobArriving-xcor 22 [
          set pcolor green
      ]

      ask patch jobArriving-xcor 34 [
          set pcolor green
      ]

      ask patch jobArriving-xcor 46 [
          set pcolor green
      ]

      ask patch jobArriving-xcor 58 [
          set pcolor green
      ]

      ask patch jobArriving-xcor 70 [
          set pcolor green
      ]


   ]
]

set start-ycor  11
if jobID > 75 [
   if jobID <= 90 [
      ;;set jobArriving-ycor  start-ycor
      ask patch jobArriving-xcor jobArriving-ycor [
          set pcolor brown
      ]

      ask patch jobArriving-xcor 22 [
          set pcolor brown
      ]

      ask patch jobArriving-xcor 34 [
          set pcolor brown
      ]

      ask patch jobArriving-xcor 46 [
          set pcolor brown
      ]

      ask patch jobArriving-xcor 58 [
          set pcolor brown
      ]
      ask patch jobArriving-xcor 70 [
          set pcolor brown
      ]


   ]
]

set start-ycor  13
if jobID > 90 [
   if jobID <= 105 [
      ;;set jobArriving-ycor  start-ycor
      ask patch jobArriving-xcor jobArriving-ycor [
         set pcolor pink
      ]


      ask patch jobArriving-xcor 22 [
          set pcolor pink
      ]

      ask patch jobArriving-xcor 34 [
          set pcolor pink
```

```
            ]

            ask patch jobArriving-xcor 46 [
                set pcolor pink
            ]

            ask patch jobArriving-xcor 58 [
                set pcolor pink
            ]
            ask patch jobArriving-xcor 70 [
                set pcolor pink
            ]
        ]
     ]

     set start-ycor  15
     if jobID > 105 [
        if jobID <= 120 [
           ;;set jobArriving-ycor  start-ycor
           ask patch jobArriving-xcor jobArriving-ycor [
               set pcolor red
           ]

           ask patch jobArriving-xcor 22 [
               set pcolor red
           ]

           ask patch jobArriving-xcor 34 [
               set pcolor red
           ]

           ask patch jobArriving-xcor 46 [
               set pcolor red
           ]

           ask patch jobArriving-xcor 58 [
               set pcolor red
           ]
           ask patch jobArriving-xcor 70 [
               set pcolor red
           ]



        ]
     ]
   ]


  set jobArriving-ycor  24

end




to setup-Jobs-Info
   let totalNumberJob          Number-of-Jobs + 1
   let selectedJobLength       Job-Length-Range + 1
   let selectedJobArrivingTime Job-Arrive-Time-range + 1

   ask jobs [
       set jobID who  ;; who is the turtle number assigned by the NetLogo system
       set jobLength random selectedJobLength
       if jobLength = 0 [
          set jobLength 1
       ]

       set arrivingTime random  selectedJobArrivingTime
```

```
       if arrivingTime = 0 [
          set arrivingTime  1
       ]

       set expectedStartingTime  arrivingTime

       set dispatchedTime      0
       set waitingTime         0
       set queuingTime         0
       set startingTime        0
       set finishedTime        0
       set ResponseIndex       0
       set ResponseRatio       0
       set turnaroundTime      0
       set jobServerID         1001  ;; default server for each job
       set jobDispatchStatus   JOB-NOT-DISPATCHED
       set jobStatus           JOB-NOT-DISPATCHED
       set lastJobStatus       JOB-NOT-DISPATCHED
       print (word "setup-Jobs-Info jobID=" jobID " jobLength=" jobLength "
arrivingTime="arrivingTime )
   ]
end

;;
;; initialize all global variables used in simulation
;;
to init-globals

   set Number-Server-Used  8
   set Number-of-jobs      1000

   output-print  "Number of Server = 8"
   output-print  "Number of Job created =1000"

   set terminatingSimulation 0
   ;; System related

   set Average-serverfarm-utilization 0
   set Average-TurnaroundTime         0
   set Average-throughput             0
   set Average-WaitingTime            0
   set Average-QueuingTime            0
   set MaxWaitingTime                 0
   set MaxQueuingtime                 0
   set MaxWaitingJob                  0
   set MaxQueuingjob                  0


   set Markspan                       0

   set FinalFinishedTime              1
   set TotalProcessingTime            1
   set TotalWaitingTime               1
   set TotalQueuingTime               1
   set TotalTurnaroundTime            1
   set Current-Server-ID              1001
   set ProcessingJob                  1

   ;; policy related information
   set TotalJobDispatched             0
   set TotalJobFinished               0
   set TotalJobWaiting                0
   set TotalJobQueuing                0
   set TotalJobArriving               0
   set TotalJobStarting               0
   set TotalJobCreated                Number-of-Jobs

   set TotalJobProcessing             0

   ;; one policy at a time
```

```
   set numPolicyUsed     0              ;; number of policy used in simulation
   set numPolicyDone     0              ;; number of policy done with simulation
   set numReportDone     0

   set numJobCreated     Number-of-Jobs ;; number of Job created for simulation

   set Current-Server-ID  1001


   ;; Flag of POLICY
   set POLICY-SELECTED             "POLICY-SELECTED"
   set POLICY-NOT-SELECTED         "POLICY-NOT-SELECTED"
   set POLICY-DONE                 "POLICY-DONE"
   set POLICY-NOT-DONE             "POLICY-NOT-DONE"

   ;; POLICY Status information
   set policy-status       POLICY-NOT-DONE

   ;; POLICY Select  Status information
   set FCFS-RR-policy-select-status  POLICY-NOT-SELECTED
   set FCFS-SSQ-policy-select-status POLICY-NOT-SELECTED
   set SJF-RR-policy-select-status   POLICY-NOT-SELECTED
   set SJF-SSQ-policy-select-status  POLICY-NOT-SELECTED
   set SJF-SSQ2-policy-select-status POLICY-NOT-SELECTED

   ;; JOB State information
   set JOB-UNKNOWN          "JOB-UNKNOWN"
   set JOB-READY            "JOB-READY"
   set JOB-ARRIVING         "JOB-ARRIVING"
   set JOB-WAITING-IN-QUEUE "JOB-WAITING-IN-QUEUE"
   set JOB-STARTING         "JOB-STARTING"
   set JOB-PROCESSING       "JOB-PROCESSING"
   set JOB-FINISHED         "JOB-FINISHED"
   set JOB-NOT-DISPATCHED   "JOB-NOT-DISPATCHED"
   set JOB-DISPATCHED       "JOB-DISPATCHED"
   set JOB-DONE             "JOB-DONE"


   ;; check how many policy are selected in simultion
   if First-Come-First-Served-RR [
      set numPolicyUsed  1
      ask dispatchers  [
          set FCFS-RR-status   POLICY-SELECTED
      ]
      set FCFS-RR-policy-select-status POLICY-SELECTED
      output-print "Selecting Policy: Job(First Come First Serve) and server(Round Robin)"
   ]

   set numPolicyUsed 0

   if First-Come-First-Served-SSQ [
      if numPolicyUsed = 1 [
         output-print "Only one policy can be selected at a time"
         output-print "Simulation was tenerated"
         output-print "Please only choose one scheduling policy"
         stop
      ]

      set numPolicyUsed   1
      ask dispatchers [
          set FCFS-SSQ-status   POLICY-SELECTED
      ]
      set FCFS-SSQ-policy-select-status POLICY-SELECTED
      output-print "Selecting Policy: Job(First Come First Serve) and server(Shortest Server
Quque)"
   ]

   if Shortest-Job-First-RR [
      if numPolicyUsed = 1 [
         output-print "Only one policy can be selected at a time"
         output-print "Simulation was tenerated"
```

```
         output-print "Please only choose one scheduling policy"
         stop
      ]

      set numPolicyUsed  1
      ask dispatchers [
          set SJF-RR-status   POLICY-SELECTED
      ]
      set SJF-RR-policy-select-status POLICY-SELECTED

      output-print "Selecting Policy: Job(Shorest Job First) and server(Round Robin)"
   ]

   if Shortest-Job-First-SSQ [
      if numPolicyUsed = 1 [
         output-print "Only one policy can be selected at a time"
         output-print "Simulation was tenerated"
         output-print "Please only choose one scheduling policy"
         stop
      ]

      set numPolicyUsed  1
      ask dispatchers  [
          set SJF-SSQ-status   POLICY-SELECTED
      ]

      set SJF-SSQ-policy-select-status POLICY-SELECTED
      output-print "Selecting Policy: Job(Shorest Job First) and server(Shortest Server Queue)"
   ]

   if Shortest-Job-First-SSQ2 [
      if numPolicyUsed = 1 [
         output-print "Only one policy can be selected at a time"
         output-print "Simulation was tenerated"
         output-print "Please only choose one scheduling policy"
         stop
      ]

      set numPolicyUsed  1
      ask dispatchers  [
          set SJF-SSQ2-status   POLICY-SELECTED
      ]

      set SJF-SSQ2-policy-select-status POLICY-SELECTED
      output-print "Selecting Policy: Job(Shorest Job First) and server(Shortest Server Queue)"
   ]



   if numPolicyUsed = 0 [
      output-print (word "You didnot select any scheduling policy.  Simulation is terminated
now")
      stop
   ]


     set jobWaiting-xcor  0
     set jobWaiting-ycor  24
     set jobDiapatched-xcor 0
     set jobDispatched-ycor 36
     set jobQueuing-xcor    0
     set jobQueuing-ycor    36
     set jobProcessing-xcor 0
     set jobProcessing-ycor 48
     set jobFinished-xcor   0
     set jobFinished-ycor   60
     set jobArriving-xcor  0
     set jobArriving-ycor  24

end
```

```
;; ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;;
;; "go" is the major control routine
;; this routine is repeated until the "stop" condition is met
;;
;;    jobID who  ;; who is the turtle number assigned by the NetLogo system
;;    jobLength random selectedJobLength
;;    arrivingTime random selectedJobLength
;;    jobStatus JOB-NOT-DISPATCHED
;; ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++


to go

   ;;print (word "enter GO loop ")
   print (word "totalJobFinished " totalJobFinished " Number-of-Jobs created " Number-of-Jobs)

   if totalJobFinished = Number-of-Jobs [
      print (word "GO - processing all created job ")
      print (word "Simualtion is ended")
      output-print " Simualtion is ended "
      stop
   ]


   Scheduling-Policy
   tick
end

;; main processing program to handle two-step policy scheduling state transition
to Scheduling-Policy
     print (word "First-Come-First-Served-RR  go!  tick=" ticks)

     ask jobs with [jobStatus = JOB-NOT-DISPATCHED and arrivingTime = ticks] [
        set currentJobID who
        ;;
        print (word "*****find a not-dispacted job " currentJobID  " current tick= " ticks  "
myJoBID " jobID " arrivingTime " arrivingTime " length " jobLength )
        if arrivingTime = ticks [  ;; am I ready to be scheduled
            set lastJobStatus    jobStatus
            set jobStatus        JOB-ARRIVING
            set TotalJobArriving TotalJobArriving + 1
            set TotalJobWaiting  TotalJobWaiting + 1
            print (word "change jobID=" who "from JOB-NOT-DISPATCHED status to JOB-ARRIVING")
            ask server jobServerID [
                set numberJobAssigned numberJobAssigned + 1
            ]

            set jobArriving-xcor  who
            let localRange  random 10
            let local-ycor jobArriving-ycor  + localRange

            ask patch jobArriving-xcor local-ycor [
               set pcolor red
            ]

        ]
     ]

     ;; if there is arriving job ready to be dispatched here
     ;; if FCFS-RR policy is used
     if First-Come-First-Served-RR [
        print(word "FCFS-RR-POLICY")
        ask jobs with [jobStatus = JOB-ARRIVING ] [
            set currentJobID who
            print (word "find an arriving job " currentJobID  " current tick= " ticks  " myJoBID
" jobID " arrivingTime " arrivingTime " length " jobLength )
            policy-processing
         ]
     ]
```

```
     ;; if FCFS-SSQ policy is used
     if First-Come-First-Served-SSQ [
        print(word "FCFS-SSQ-POLICY")
        ask jobs with [jobStatus = JOB-ARRIVING ] [
            set currentJobID who
            print (word "find an arriving job " currentJobID  " current tick= " ticks  " myJoBID
" jobID " arrivingTime " arrivingTime " length " jobLength )
            policy-processing
         ]
     ]


     ;; if SJF-RR policy is used
     if Shortest-Job-First-RR [
        print(word "SJF-RR-POLICY")
        ;; select a job with shortest job length
        ask jobs with[jobStatus = JOB-ARRIVING] [
            set currentJobID who
            print (word "SJF-RR find a shortest job + JOB-ARRIVING status SJF-RR-JOBID="
currentJobId " tick=" ticks "  call policy-processing")
            policy-processing

        ]
     ]

     ;; if SJF-SSQ policy is used
     if Shortest-Job-First-SSQ [
        print(word "SJF-SSQ-POLICY")
        ask jobs with [jobStatus = JOB-ARRIVING] [
            set currentJobID who
            print(word "SJF-SSQ-POLICY   myJobID " currentJobID  " JOB-ARRIVING   JobLength "
jobLength)
            print (word "SJF-SSQ: find a shortest job + JOB-ARRIVING status SJF-RR-JOBID="
currentJobId " tick=" ticks "  call policy-processing")
            policy-processing

        ]
     ]


     ;; if SJF-SSQ policy is used
     if Shortest-Job-First-SSQ2 [
        print(word "SJF-SSQ2-POLICY")
        ask jobs with [jobStatus = JOB-ARRIVING] [
            set currentJobID who
            print(word "SJF-SSQ2-POLICY   myJobID " currentJobID  " JOB-ARRIVING   JobLength "
jobLength)
            print (word "SJF-SSQ2: find a shortest job + JOB-ARRIVING status SJF-SSQ-JOBID="
currentJobId " tick=" ticks "  call policy-processing")
            policy-processing

        ]
     ]



     print (word "Checkpoint 0001 ")

     let jobCurrentLength 0
     ask jobs with [jobStatus = JOB-DISPATCHED ] [
         print (word "Checkpoint 0002");
         set jobCurrentLength jobLength
         ;; inside jobs turtle contend
         ;; update job current status
         set currentJobID who
         print (word "This job=" who " was dispatched" )
         print (word "This job=" who " tick=" ticks " startingTime=" startingTime "
finishedTime=" finishedTime)
         if startingTime > ticks [
```

```
            set lastJobStatus    jobStatus
            set jobStatus        JOB-WAITING-IN-QUEUE
            ask server assignedServerID [
                set numberJobInQueue  numberJobInQueue + 1
                set ServerWorkloadQueuing ServerWorkloadQueuing + jobCurrentLength
                print (word "Server " jobServerID "  increasing numberJobInQueue to "
numberJobInQueue)
            ]

            set jobQueuing-xcor  ( assignedServerID - 1001 ) *  16 +  random 16
            print (word "HHHHHHHHHHHHHHHHHHHHH   queuing xcor "  jobQueuing-xcor  "  ycor  "
jobQueuing-ycor)
            let localRange  random 10

            let local-ycor jobQueuing-ycor  + localRange
            ask patch jobQueuing-xcor local-ycor [
               set pcolor white
            ]

        ]

        ;; it is time to start processing this job
        if startingTime = ticks [
           set queuingTime    ticks -  dispatchedTime
           print (word "this job=" who " tick=" ticks " startingTime=" startingTime)
           print (word "this job=" who " tick=" ticks " finishedTime=" finishedTime)

           set lastJobStatus jobStatus
           set jobStatus     JOB-PROCESSING
           set ProcessingJob ProcessingJob + 1
           set TotalJobProcessing TotalJobProcessing + 1

           set jobProcessing-xcor  ( assignedServerID - 1001 ) *  16 +  random 16
           let localRange  random 10
           let local-ycor jobProcessing-ycor  + localRange
           ask patch jobProcessing-xcor local-ycor [
              set pcolor yellow
           ]


           if lastJobStatus = JOB-WAITING-IN-QUEUE [
               ask server jobServerID [
                   set ServerWorkloadQueuing ServerWorkloadQueuing - jobCurrentLength
                   set numberJobInQueue  numberJobInQueue - 1
               ]
           ]

           ;; update server job processing information
           ask Server assignedServerID [
               set numberJobProcessed    numberJobProcessed + 1
           ]
        ]
    ]

    ask jobs with [jobStatus = JOB-WAITING-IN-QUEUE ] [
        set jobCurrentLength jobLength
        if startingTime = ticks [
           print (word "this job=" who " tick=" ticks " startingTime=" startingTime)
           set lastJobStatus   jobStatus
           set jobStatus       JOB-PROCESSING
           let localRange  random 10
           set jobProcessing-xcor  ( assignedServerID - 1001 ) *  16 +  random 16
           let local-ycor jobProcessing-ycor  + localRange


           ask patch jobProcessing-xcor local-ycor [
              set pcolor yellow
           ]

           ask server  assignedServerID [
               set ServerWorkloadQueuing ServerWorkloadQueuing - jobCurrentLength
```

```
            set numberJobInQueue    numberJobInQueue - 1
            set numberJobProcessed  numberJobProcessed + 1
         ]

         set ProcessingJob      ProcessingJob + 1
         set TotalJobProcessing TotalJobProcessing + 1

      ]
   ]

   ask jobs with [jobStatus = JOB-PROCESSING ] [
      let next-local-tick ticks + 1
      ;;if FCFS-RR-finishedTime = next-local-tick [
      if finishedTime = ticks [
         print (word "this job=" who " tick=" ticks " FCFS-RR-startingTime=" startingTime  )
         print (word "this job=" who " tick=" ticks " FCFS-RR-finishedTime=" finishedTime  )
         set lastJobStatus              jobStatus
         set jobStatus                  JOB-FINISHED

         let localRange  random 10
         set jobFinished-xcor  ( assignedServerID - 1001 ) *  16 +  random 16


          let local-ycor jobFinished-ycor  + localRange
         ask patch jobFinished-xcor local-ycor [
            set pcolor green
         ]

         set ProcessingJob ProcessingJob - 1
         ask Server assignedServerID [
            set numberJobProcessed  numberJobProcessed - 1
            set numberJobFinished      numberJobFinished + 1
         ]

         print (word "this job=" who " was JOB-FINISHED" )
         set totalJobFinished totalJobFinished + 1
      ]
   ]

   ask jobs with [jobStatus = JOB-FINISHED ] [
      set finishedTime       startingTime + jobLength
      set turnaroundTime     finishedTime - arrivingTime
      set lastJobStatus jobStatus
      set jobStatus JOB-DONE
      set finalFinishedTime finishedTime
      set responseIndex turnaroundTime - jobLength
      set responseRatio jobLength / turnaroundTime

      let localResponseIndex  responseIndex

      ask server assignedServerID [
          set Server-ResponseIndex  Server-ResponseIndex + localResponseIndex
      ]


      if totalJobFinished = totalJobCreated [
         print (word "all created jobs are finished processing ")
         set policy-status   POLICY-DONE
      ]
   ]

   update-policy-Status
   update-Performance-Chart-Data

end

to RR-get-nextServerID

  set selectedServer Current-Server-ID
  set Current-Server-ID Current-Server-ID + 1
```

```
  if Current-Server-ID = 1009 [
     set Current-Server-ID  1001
  ]
end

;;
;; Policy processing routines
;;
to policy-processing

   let jobStartingTime     99999
   let jobCurrentStatus    0
   let jobCurrentLength    0
   print (word "policy-processing   currentJobID " currentJobID);
   print (word "Checkpoint 0003");

   if First-Come-First-Served-RR [
      print(word "FCFS-RR-POLICY: call RR-get-nextServerID")
      RR-get-nextServerID ;; get my server ID
      print (word "selectedServer " selectedServer)
   ]

   ;; if FCFS-SSQ policy is used
   if First-Come-First-Served-SSQ [
      find-shortest-server-queue-server
      print(word "call FCFS-SSQ-POLICY: call find-shortest-server-queue-server selectedServer"
selectedServer)
   ]

   ;; if SJF-RR policy is used
   if Shortest-Job-First-RR [
      print(word "SJF-RR-POLICY: call RR-get-nextServerID")
      RR-get-nextServerID ;; get my server ID
   ]

   ;; if SJF-SSQ policy is used
   if Shortest-Job-First-SSQ [
      find-shortest-server-queue-server
      print(word "SJF-SSQ-POLICY: call find-shortest-server-queue-server  selectedServer "
selectedServer)
   ]

   ;; if SJF-SSQ policy is used
   if Shortest-Job-First-SSQ2 [
      find-shortest-server-queue-server2
      print(word "SJF-SSQ-POLICY: call find-shortest-server-queue-server  selectedServer "
selectedServer)
   ]


   ask job currentJobID  [
       set jobCurrentLength jobLength
       let localJobStatus jobStatus
       let myServerID assignedServerID


       ask Server selectedServer [
           ;; decide the job starting time

           if nextStartingTime > ticks [
              print (word "process-policy: nextStartingTime=" nextStartingTime "> ticks= " ticks
" put job in Queue")
              ;; this job cannot not start now, put it in waiting queue

              ask job currentJobID [
                  set lastJobStatus  localJobStatus
                  set jobStatus      JOB-WAITING-IN-QUEUE
              ]
              let localRange  random 10
              set jobQueuing-xcor  ( selectedServer - 1001 ) *  16 +  random 16
```

```
            let local-ycor jobQueuing-ycor  + localRange

            ask patch jobQueuing-xcor local-ycor [
                set pcolor white
            ]



            set jobStartingTime       nextStartingTime
            set nextStartingTime      nextStartingTime + jobCurrentLength
            set numberJobInQueue      numberJobInQueue + 1
            set ServerWorkloadQueuing ServerWorkloadQueuing + jobCurrentLength
            set numberJobAssigned     numberJobAssigned + 1
        ]

        if nextStartingTime <= ticks [
            print (word "process-policy: nextStartingTime <= ticks, start this job right NOW")
            set jobStartingTime       ticks
            set nextStartingTime      ticks + jobCurrentLength

            ask job currentJobID [
                set lastJobStatus  localJobStatus
                set jobStatus      JOB-PROCESSING
            ]

            let localRange  random 10
            set jobProcessing-xcor  ( myServerID - 1001 ) *  16 +  random 16


            let local-ycor jobProcessing-ycor  + localRange

            ask patch jobProcessing-xcor local-ycor [
                set pcolor yellow
            ]

            set totalProcessingTime   totalProcessingTime + jobCurrentLength
            set numberJobProcessed    numberJobProcessed  + 1
            set ProcessingJob         ProcessingJob + 1
            set TotalJobProcessing    TotalJobProcessing  + 1
        ]

        set numberJobAssigned       numberJobAssigned + 1
        print (word "process-FCFS-RR-policy: current server=" selectedServer)
    ] ;; inside the Server context

    ;; decide the job starting time
    set assignedServerID   selectedServer
    set jobServerID        selectedServer
    set startingTime       jobStartingTime
    set waitingTime        startingTime - arrivingTime
    set finishedTime       startingTime + jobLength
    set turnaroundTime     finishedTime - arrivingTime
    print (word "job=" who " dispatch-status is now JOB-DISPATCHED");
    set totalJobDispatched totalJobDispatched + 1
]

end



to update-policy-Status

   let myJobID 0
   ask jobs  [
       ask job who [
           set waitingTime    dispatchedTime - arrivingTime
           set finishedTime   startingTime   + jobLength
           set turnaroundTime finishedTime   - arrivingTime
           set queuingTime    startingTime   - dispatchedTime
       ]
```

```
   ]

   ;; get the final finished time
   print (word "Final finished Time = " finalFinishedTime)

   set totalProcessingTime  sum [jobLength ]      of jobs
   set totalWaitingTime     sum [waitingTime ]    of jobs
   set totalQueuingTime     sum [queuingTime ]    of jobs
   set totalTurnaroundTime  sum [turnaroundTime ] of jobs

   if totalJobFinished > 0   [
      set Average-WaitingTime    totalWaitingTime    / totalJobFinished
      set Average-QueuingTime    totalqueuingTime    / totaljobFinished
      set Average-TurnaroundTime  totalTurnaroundTime / totaljobFinished
      set Average-ProcessingTime totalProcessingTime / totaljobFinished
   ]

   if  finalFinishedTime >= 1  [
       set Average-throughput  totalJobFinished /  finalFinishedTime
   ]

   if totalJobFinished >= 1 [
      set Average-TurnaroundTime  totalTurnaroundTime / totalJobFinished
      set Average-WaitingTime      totalWaitingTime    /  totalJobFinished
   ]

   set markspan   finalFinishedTime

end



to update-Final-policy-Status

   let totalFinalProcessingTime        totalProcessingTime
   let totalMarkspan                   markspan * numServerUsed
   let Laverage-serverfarm-utilization 0.0

   set Total-Server-Queuing-Job  sum [numberJobInQueue] of ServerFarm
   set Total-Server-Queuing-Time sum [serverWorkloadQueuing] of ServerFarm

   set Average-QueuingTime Total-Server-Queuing-Time / 8
   set Average-QueuingJob  Total-Server-Queuing-Job / 8

   set Average-TurnaroundTime sum [turnaroundTime] of jobs / TotalJobFinished
   set Average-Throughput TotalJobFinished / ticks


   output-print "Average Job Size"
   output-print Average-ProcessingTime
   output-print "Average-TurnaroundTime  "
   output-print Average-TurnaroundTime
   output-print "Average-Throughput   "
   output-print Average-Throughput



   set Laverage-serverfarm-utilization totalFinalProcessingTime / totalMarkspan
   set average-serverfarm-utilization  Laverage-serverfarm-utilization

   output-print "average-serverfarm-utilization"
   output-print average-serverfarm-utilization

   output-print "MARKSPAN  "
   output-print markspan



end
```

```
to update-Performance-Chart-Data

   set totalProcessingTime    0
   set totalWaitingTime       0
   set totalQueuingTime       0
   set totalTurnaroundTime    0

   set totalProcessingTime  sum [jobLength ]      of jobs with [jobStatus = JOB-FINISHED]
   set totalQueuingTime     sum [queuingTime ]    of jobs with [jobStatus = JOB-WAITING-IN-QUEUE]
   set totalTurnaroundTime  sum [turnaroundTime ] of jobs


   set Total-Server-Queuing-Job  sum [numberJobInQueue] of ServerFarm
   set Total-Server-Queuing-Time sum [serverWorkloadQueuing] of ServerFarm

   set Average-QueuingTime Total-Server-Queuing-Time / 8
   set Average-QueuingJob  Total-Server-Queuing-Job  / 8

   if TotalJobFinished  > 0  [
      set Average-TurnaroundTime sum [turnaroundTime] of jobs / TotalJobFinished
      set Average-Throughput TotalJobFinished / ticks
   ]


   if  finalFinishedTime >= 1 [
       set average-throughput  totalJobFinished / finalFinishedTime
   ]

   ;;set numberJobInQueue

   let counter  0
   ask jobs with [jobStatus = JOB-WAITING-IN-QUEUE] [
       set counter counter + 1
   ]

   set totalJobQueuing   counter


   set markspan                  finalFinishedTime

   set-current-plot "Arriving-vs-Finished"
   set-server-plot-data
   ;;clear-plot
   set-current-plot "Server-NumberJobInQueue"
   set-current-plot "Server-QueuingJobLength"

   ask serverFarm [
    print (word who "--" ServerWorkloadQueuing "--" numberJobInQueue)
   ]

end


to set-server-plot-data

   ask server 1001  [
       set Server121-QueueLength   ServerWorkloadQueuing
       set Server121-NumJobInQueue numberJobInQueue
       set Server121-xcor 8
       ;set Server121-Queue-Length  Random 100
       ;set Server121-NumJobInQueue Random 100
       ;set Server121-xcor 8

   ]

   ask server 1002  [
       set Server122-QueueLength   ServerWorkloadQueuing
       set Server122-NumJobInQueue numberJobInQueue
```

```
        set Server122-xcor 16
    ]

    ask server 1003  [
        set Server123-QueueLength   ServerWorkloadQueuing
        set Server123-NumJobInQueue numberJobInQueue
        set Server123-xcor 24
    ]
    ask server 1004  [
        set Server124-QueueLength   ServerWorkloadQueuing
        set Server124-NumJobInQueue numberJobInQueue
        set Server124-xcor 32
    ]
    ask server 1005  [
        set Server125-QueueLength   ServerWorkloadQueuing
        set Server125-NumJobInQueue numberJobInQueue
        set Server125-xcor 40
    ]
    ask server 1006  [
        set Server126-QueueLength   ServerWorkloadQueuing
        set Server126-NumJobInQueue numberJobInQueue
        set Server126-xcor 48
    ]
    ask server 1007  [
        set Server127-QueueLength   ServerWorkloadQueuing
        set Server127-NumJobInQueue numberJobInQueue
        set Server127-xcor 56
    ]
    ask server 1008  [
        set Server128-QueueLength   ServerWorkloadQueuing
        set Server128-NumJobInQueue numberJobInQueue
        set Server128-xcor 64
    ]

end



;; Select a sever with the shortest queue length
;;
to find-shortest-server-queue-server
  let selectedServerID 1001
  let localServerQueueLength  0

  ask ServerFarm with-min [numberJobInQueue]
  [
    set selectedServerID who

  ]

  ask server selectedServerID [
      set localServerQueueLength  numberJobInQueue
  ]

  set selected-Server selectedServerID
  print (word "FCFS-SSQ select serverID=" selected-Server " queueLength=" localServerQueueLength)
  set Current-Server-ID selected-Server
  set selectedServer selected-Server
  ;;
end



;; Select a sever with the shortest queue length
;;
to find-shortest-server-queue-server2
  let selectedServerID 1001
  let localServerWorkloadQueue  0

  ask ServerFarm with-min [ServerWorkloadQueuing]
  [
```

```
    set selectedServerID who

  ]

  ask server selectedServerID [
      set localServerWorkloadQueue  ServerWorkloadQueuing
  ]

  set selected-Server selectedServerID
  print (word "FCFS-SSQ select serverID=" selected-Server " queueTime=" localServerWorkloadQueue)
  set Current-Server-ID selected-Server
  set selectedServer selected-Server
  ;;
end
```