SERVER SIMULATION

WEC Programming 2024: Nuthanan and Jennifer

TABLE OF CONTENTS

OUR GOAL

To design and implement a simulation that processes tasks efficiently using available server resources, minimizing...

- Time taken
- Task failures
- Power consumption

OUR IDEA pt. 1

1. “Pre-Processing”

- Read from Servers.csv and Tasks.csv
- Sort servers based on power efficiency

```
# Read servers from csv files
with open('Servers.csv', 'r') as serverFile:
    server_reader = csv.reader(serverFile)

    # Skip the header
    counter = 0
    for row in server_reader:
        if counter > 0:
            serverList.append(row)
        counter += 1

#Read tasks from csv files
with open('Tasks.csv', 'r') as tasksFile:
    tasks_reader = csv.reader(tasksFile)
    # Skip the header
    counter = 0
    for row in tasks_reader:
        if counter > 0:
            taskList.append(row)
        counter += 1

#Sort servers from smallest to biggest power per core
serverList = sorted(serverList, key=itemgetter(2))
```

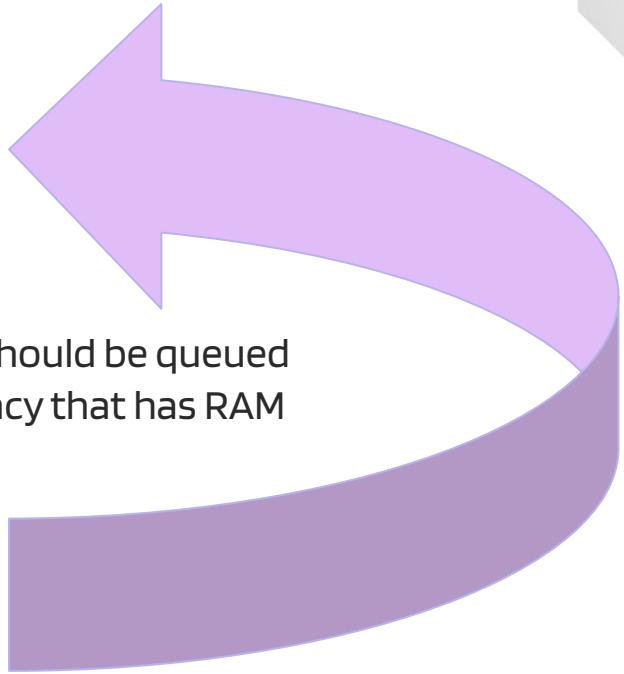
OUR IDEA pt. 2

2. Iterating and reading...

- Check if any of the current tasks are completed
  - If so, output and update
- Check if any of the queued tasks can be run/will fail
  - Output and update as required

- "Read" a task and determine if it can be run now or if it should be queued
  - Assign it to the server with the best power efficiency that has RAM

- Update the turns left in the current and queued tasks



Complete task

```
#Main loop
constantServerList = []
for x in serverList:
    constantServerList.append(x.copy())
turn = 1
currentTasks = []
queuedTasks = []
negativeTasks = []
for task in taskList:
    #Remove completed/failed tasks from the servers and put them in Output.csv. Update server resources available
    for currTask in currentTasks:
        if int(currTask[2]) == 0:
            #write to output.csv
            taskCores = int(currTask[1])
            serverPower = int(constantServerList[int(currTask[5])-1][2])
            taskTurns = int(taskList[int(currTask[0])-1][2])
            outputFile.writerow([turn, int(currTask[0]), 1, taskCores * serverPower * taskTurns, currTask[5]])
            #write to simulation.csv
            timeRecorded = time.time()
            simulationFile.writerow(['Task', timeRecorded - timeStamp, turn, int(currTask[0]), 'Complete'])

            #update server resources
            serverList[int(currTask[5])-1][1] = int(serverList[int(currTask[5])-1][1]) + taskCores
            serverList[int(currTask[5])-1][3] = int(serverList[int(currTask[5])-1][3]) + int(currTask[3])

            #remove from current task
            currentTasks.remove(currTask)
```

Run queued task or fail it

```
#See if any queued tasks can be run
for qTask in queuedTasks:
    x = 0
    if int(qTask[4]) != -1 and int(qTask[2]) + turn > int(qTask[4]):

        #update server ram
        serverList[int(qTask[5])-1][3] = int(serverList[int(qTask[5])-1][3]) + int(qTask[3])

        #delete task from queue
        queuedTasks.remove(qTask)

        #output to output.csv
        outputFile.writerow([turn, qTask[0], 0, 0, 0])
        #output to simulation.csv
        timeRecorded = time.time()
        simulationFile.writerow(['Task', timeRecorded - timeStamp, turn, qTask[0], 'Failed'])


        continue
    while (x < len(serverList)):
        if int(qTask[1]) <= int(serverList[x][1]) and int(qTask[3]) <= int(serverList[x][3]): # if taskCores < serverCores and taskRam < serverRA
            serverList[x][1] = int(serverList[x][1]) - int(qTask[1]) #cores
            serverList[x][3] = int(serverList[x][3]) - int(qTask[3]) #RAM
            qTask.append(x+1) #server number
            currentTasks.append(qTask.copy())
            queuedTasks.remove(qTask)
            break
        x+=1
```

Run task now vs. queue it

```
#Read the next row in Tasks.csv for a new task to send to a server & send it to a server
x = 0
timeRecorded = time.time()
simulationFile.writerow(['Task', timeRecorded - timeStamp, turn, task[0], 'Read'])
if int(task[4]) != -1 and int(task[4]) < int(task[2]): # if turn completion requirement is less than turns required
    x = len(serverList) + 1
while (x < len(serverList)):
    if int(task[1]) <= int(serverList[x][1]) and int(task[3]) <= int(serverList[x][3]): # if taskCores < serverCores and taskRam < serverRAm
        serverList[x][1] = int(serverList[x][1]) - int(task[1]) #cores
        serverList[x][3] = int(serverList[x][3]) - int(task[3]) #RAM
        task.append(x+1) #server number
        currentTasks.append(task.copy())
        break
    elif int(task[1]) > int(serverList[x][1]) and int(task[1]) <= int(constantServerList[x][1]) and int(task[3]) <= int(serverList[x][3]):
        serverList[x][3]  = int(serverList[x][3]) - int(task[3]) #RAM
        task.append(x+1) #server number
        queuedTasks.append(task.copy())
        break
    x += 1

#if x >= len(serverList): output to Output.csv that it failed
if x >= len(serverList):
    outputFile.writerow([turn, task[0], 0, 0, 0])
    timeRecorded = time.time()
    simulationFile.writerow(['Task', timeRecorded - timeStamp, turn, task[0], 'Failed'])
```

Update current and queued tasks

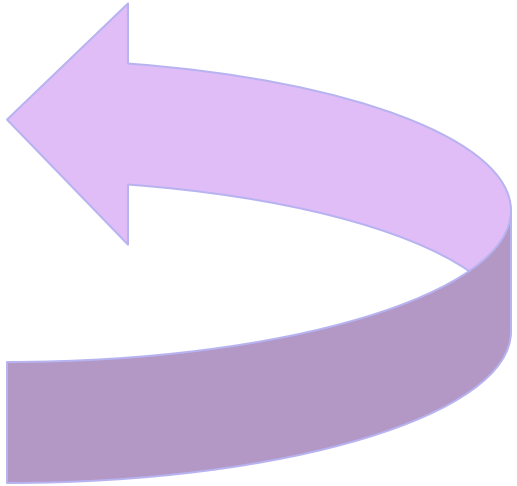
```
#Update the current tasks
for currTask in currentTasks:
    currTask[2] = int(currTask[2]) -  1
for qTask in queuedTasks:
    qTask[4] = int(qTask[4]) - 1

#Sort servers from smallest to biggest number
serverList = sorted(serverList, key=itemgetter(0))
timeRecorded = time.time()
for x in serverList:
    simulationFile.writerow(['Server', timeRecorded - timeStamp, turn, x[0], x[1], x[3]])
#Sort servers from smallest to biggest power per core
serverList = sorted(serverList, key=itemgetter(2))
turn += 1
```

OUR IDEA pt. 3

3. Finishing up the current and queued tasks

- Check if any of the current tasks are completed
  - If so, output and update
- Check if any of the queued tasks can be run/will fail
  - Output and update as required

- Update the turns left in the current and queued tasks



OTHER THINGS WE TRIED

-1s

One of the problems with regards to **negative numbers** is that we pass it in chronological order from how the tasks are layed out.

However, it would make more logical sense, if we push the -1s down to the bottom of the queue, so that the bigger tasks that have a **finite number of turns** will be run and more tasks will be completed.

EXTENSIONS

At 3:25 pm, we noticed that the simulation should display the RAM being used instead of the RAM available

In theory, however, there is a quick fix! We must simply subtract the available RAM from the "constantRAM"