Adaptive Resource Reservation for Cloud Container Clusters

Muhammet Erket, Travis Le, and Jake Ren

Acknowledgements

We would like to express our gratitude to Dr. Ming-Hwa Wang for giving us this opportunity and encouraging our research on the topics we learned throughout our cloud computing course. Furthermore, we would like to extend our thanks to Amazon, and Amazon Web Services, for giving us the inspiration to work upon this topic.

Table of Contents

List of Tables/Figures

Abstract

In today's society, the amount of data that we process has been steadily increasing. From transportation, to social media, to all facets of life, data is being stored and processed in high amounts. To accommodate these data necessities, cloud cluster containers were developed as one of the methods to process huge amounts of data. Each container acts as a lightweight alternative to virtual machines, and the cluster of containers work collaboratively to process larger amounts of data while avoiding system overloads. One major provider for cloud cluster containers is Amazon, with their AWS, or Amazon Web Services. AWS allows users to purchase containers for use, and users would pay for the amount of data resources they use, including memory, CPU time, etc.

However, each container has the potential to allocate their resources incorrectly. As a default, containers overallocated resources to prevent failures to handle requests. As a result, companies and users that purchase containers are paying excess for the overallocated resources. In this project, we are trying to propose a solution to this issue. By creating an algorithm and code that allocates resources adaptively, we would be able to save costs and power needed to run clusters. By using an adaptive model, we preallocate the required resources based on a container's previous resource demand, creating an optimized and personalized resource allocation for the said container. Our algorithm would employ the use of an integer linear programming model and run multiple simulations to test and compare our results with other methods, such as static and randomized allocation methods. Our solution aims to assist companies in reducing their costs in our heavy, data-centric environment.

1. Introduction

There has been a sustained amount of increase to the data we process today. This ranges from airplane pilot data to data stored on social media sites. Still, the data we have to process is a huge amount and can go even more than 200Tb per day.

However, cloud cluster containers are one of the methods we use to process such a huge amount of data. Containers act as a lightweight alternative to virtual machines. According to the paper *Highly Available Cloud-Based Cluster Management*, "clusters are a collection of computers… ... which collaboratively work on computational problems"(1201). By combining all three terms together, there is a way to process huge amounts of data (possibly in parallel) such that there is not much of a system overload. Cloud cluster containers also use many resources to process data.

Problem

While cloud cluster containers can do a bunch of processes at the same time, there are times where resources are allocated incorrectly. These resources can range from memory to CPU time. The cluster either does not allocate enough resources to handle requests or allocate too many resources that a company wastes money. As it currently stands, Cloud cluster containers currently do not allocate efficiently a correct amount of resources.

By coming up with an algorithm or type of code that allocates resources correctly, there will be a good amount of money saved. In addition, the amount of power used to run clusters will decrease, saving the overall toll on the environment.

As such, we would like to optimize containers to be able to adaptively allocate resources through a better algorithm and linear programming.

Relation to Class

From the syllabus, the course description states in the course description: cluster/grid computing and economics of cloud computing. Our topic revolves around the use of clusters,

which is a topic in the cloud computing class. Clusters are used to increase the amount of resources the company can assign programs to. Plus, clusters are often the cloud because it either stores the data or does computing work that may not be in-house. This topic also plays a role in reducing power consumption.

In the lecture, “Parallel Processing and Distributed Computing,” it is stated that 15% out of 44 million servers are idling, increasing the amount of power unnecessarily used. By decreasing the overall power used in generating clusters, the cost of using cloud services may decrease. This will overall waste less money for companies using the cloud.

In addition, cloud containers have to work directly with the cloud. By using the cloud, we will be exposed to cloud architecture and service models and service oriented architecture. We might have to know how cloud structure works before implementing this project.

Containers have been talked about regularly in class that it merits a topic in cloud computers, especially when used on cloud servers. Our project involves deploying instances on the cloud which each contain containers.

Approaches

To solve this issue, one approach is to set a static value that can be applied to all containers. In other words, all containers will get the same amount of resources regardless of load. While this is the easiest way of assigning resources, it does not solve the process of using resources efficiently. Another approach to this problem is to monitor every container constantly to assign the correct resources to a container. While it does guarantee the optimal resource assignment, it may take up more processing power to monitor everything.

Why our approach is better

Our approach will attempt to be adaptive rather than monitoring everything. By being adaptive, the program will assign resources to a container whenever necessary. They will pull data from what happened currently. Our approach will end up using less power than dynamically allocating resources as the program will only be called whenever a container has to be assigned resources. So, the program is not wasting power running in the background. In

addition, it is better than statically assigning data. Adaptively allocating resources may likely assign the correct amount of resources to containers. By trying to statically assign resources, a scheduler has to assume that one size of the container fits all the containers. Containers may need differing amounts of resources depending on the circumstances.

On a side note, this approach can potentially be improved by pulling previous monitoring data from the last time the scheduling program had to allocate resources to a container. This will allow the scheduling algorithm to receive more accurate data.

We originally developed this project to optimize container assignments to cluster instances but later we noticed this solution could also be used for VM type selection during cluster creation. In order to do that we will create a virtual cluster which contains several instances of each VM type and our actual containers which will run in the cluster. Then scheduling results will give us the cheapest cluster we can construct with a good utilization.

Scope

Our scope includes the following:

- Calculating minimum amount of resources for each container on the cloud.
- Use simulated data to simulate values that could be found on real containers
- Implementing a scheduling program to apply our the knowledge gained from the researched papers and data collected
- Implementing a simulation to verify our scheduling program.
- Testing our program with other methods of scheduling programs, such as static and random assignment scheduling

2. Theoretical Basis

Theoretical background

Linear programming is a method to model the relationships between our containers and their resource demands as a linear function to calculate an optimal standard which will promote

resource efficiency. By using linear programming, each container would be assigned an optimal amount of resources to run their applications. To describe how linear programming works, constraint equations and maximal/minimal equations are generated from a supposed model. Then, every constraint equation is graphed. Afterwards, all the points where any line intersections with another are recorded. So, all those values are compared against the maximal/minimal equation to determine which values produce the maximal/minimal result.

Related Research

In the paper, *Scheduling Frameworks for Cloud Container Service*, it was suggested to dynamically allocate resources on runtime in containers. Although running a subprocess that is scheduling containers may be more effective, having a process that is always running in the background may not be the most efficient method of allocating resources. In addition, the way the paper allocates resources seems complicated which may be difficult to implement in a timely fashion.

On a different paper, *A Service Performance based Dynamic Provisioning Approach in Containerized Cloud Environments*, provides a method for allocating resources. However, only states high level requirements. It does not get into detail about how to schedule resources. It only provides a different method to monitoring and how to organize instances and containers better.

Furthermore, in another paper, *Adaptive Resource Views for Containers*, the authors created adaptive environments to accommodate CPU run time as well as memory allocation. Their methods will be similar to our approach, with the use of adaptive containers to increase efficiency. However, for their memory storage, their elastic containers did not achieve any benefit when the containers did not surpass their memory allocation. In particular, we want to be able to minimize the memory allocation for our containers especially in these situations in order to help mitigate costs for the users paying for the container services.

In this project we are primarily focusing on solving resource allocation problems for individual containers deployed on a Container Clusters. Resource allocation is necessary to protect

containers from starvation in case of instant high demanding services. But if resource allocation goes beyond the necessary limits then we will end up in an inefficient resource utilization.

Our Solution

We will use previously recorded resource consumption statistics as the basis for our solution. We will construct prediction models for demands of each container as a poisson process. Then we will pick a minimum confidence satisfaction value α to calculate the minimum resource requirement for each container. This value α will represent the allocated resources that will be able to address α amount of incoming requests.

After minimum we will select a β confidence value to calculate the amount of pool resources which will satisfy the β amount of overall requests.

Then we will try to deploy containers to instances as total amount of resources is as close as it can be to $\sum C_{min}$ + PoolResources. This deployment scenarios can be expressed as following integer linear programming model:

Definitions

$x_{i,j}$ is the binary decision variable showing if container i is deployed on the instance j or not

r_i is the minimum resource required for container i which is calculated with confidence value α.

C_j is the cost of instance number j (constant).

y_j is the decision variable which indicates if instance j is used by any container or not

P_j is the minimum resources required for the pool

R_j is the total available resources for instance j (constant).

λ_i is the average resource usage of container i.

Integer Linear Programming Model

minimize $\sum_j C_j y_j$ (Objective is to use as minimum instances as possible to minimize total cost)

Subject to:

$y_j \geq x_{i,j} \ \forall i,j$ (If any container is deployed on instance j the y_j will be 1)

$\sum_j x_{i,j} = 1 \ \forall i$ (Each container should be assigned only once)

$\sum_i x_{i,j} r_i + P_j \leq R_j \ \forall j$ (Allocated resources should be less than available resource for each instance)

P_j = PoissonDistributionCoefficient(β) * $\sum_i x_{i,j} \lambda_i$(where P_j should be calculated based on the deployed containers and PoissonDistributionCoefficient is calculated based on the β from the chi-square table.)

We will solve the ILP model to assign containers to instances then we will run performance and utilization experiments with different **α** and βs to find optimum values for those.

How our solution differs from others

Our solution primarily focuses on the pre deployment scheduling of containers in the cluster whereas other solutions were focusing on the run time task scheduling. In addition, we are making use of pre-existing data to optimize our scheduling of containers.

Why our solution is better

Our solution is focusing on reducing the total cost in cloud clusters by optimizing container deployments based on their resource requirements. We are also using a stochastic model to calculate future demands. The advantage of using our pre-deployment scheduling makes our solution more elastic and we can apply our solution more universally to any container system without changing the container's internal infrastructure.

3. Hypothesis/Goal

Our hypothesis is that, with the use of a better algorithm and linear programming, we can optimize cloud cluster containers to adaptively allocate resources and improve overall efficiency. This will in turn allow users to save costs when using the cloud container services.

4. Methodology

How to generate/collect input data

We will be generating our own input data through creating a .json file where we input probable sets of data that could be found in a real life container cluster. We will be estimating these values through comparison of actual AWS ECS container resource demands. We will be estimating both container values as well as instance values and test different sets of data to ensure our algorithm is successful for various situations.

How to solve the problem

Algorithm Design

We will input various testing α and β values together with analytics data to our program. We will calculate poisson mean and standard deviation from the analytics. We will then calculate minimum allocation, our upper confidence bounds, and pool requirements based on Poisson values. We will then apply these values to solve for our assignment problem with the use of the previously defined equations.

Language

For this project, we will be primarily using one language: Python 3**.**

Tools

We will be using multiple tools to implement this project. These tools are

- matplotlib Python Module to generate graphs of data
- Linear Programming module of Python
- Python scipy module
- Simplex algorithm to calculate assignment results
- Pulp python library to solve linear integer programming model

How to generate output

Our program will output container assignments to the instances of the cluster together with the minimum allocation requirements for each container and also the pool.

How to test against hypotheses

We will have to build a simulation of the work space to see what happens if we scheduled randomly. This method will be repeated with the input data we received from AWS along with some worst case scenarios. The output of this simulation will indicate the CPU utilization rate, memory utilization rate, and time spent scheduling. This will later be compared to other tests we will generate. Afterwards, we will evenly distribute the containers to each instance and the resources to each container. We will repeat the method done with the random scheduling algorithm. Finally, we will use our custom scheduling algorithm to schedule containers and repeat the simulation for the last time. We will likely run this simulation multiple times to get an accurate result.

With all the output data, we will compare and contrast the utilization rate, the amount of resources that were not fulfilled. There will also be a data point if the containers need more resources to run a supposed task. We will consider containers that are close to 100% but not at 100% to be using the data effectively. If the data is below a certain threshold, we would consider the container to be not optimized. In addition, if there is a high percentage of requests that were not fulfilled, it is not considered a great algorithm.

By adding all the containers that are optimized in their respective scheduling algorithm, we could determine which scheduling algorithm has better utilization. Thus, we can achieve a better utilization than the vanilla containers.

5. Implementation

5.1 Data Analysis

For the data analysis, import the json, statistics, math, and scipy Python modules.

5.1.1 Parse Equations

```
def findContainer(containerName, containers):

      if containerName not in containers.keys():
            return

      usages = containers.get(containerName).get("Usages")
      resourceList = []
      for usage in usages:
            resourceList.append(int(usage.get("ResourceDemand")))

      return resourceList

def findInstanceAvaR(instanceName, instances):
      if instanceName not in instances.keys():
            return

      instanceValues = instances.get(instanceName)
      avaR = int(instanceValues.get("AvaR"))
      return avaR

def findInstanceCost(instanceName, instances):
      if instanceName not in instances.keys():
            return

      instanceValues = instances.get(instanceName)
      cost = int(instanceValues.get("Cost"))
      return cost

def findContainersSubset(containerNameList, containers):

      subsetList = []
      for container in containerNameList:
            subsetList.extend(findContainer(container, containers))

      return subsetList
```

5.1.2 Mathematical Functions

```
def containerMean(resourceList):
      return statistics.mean(resourceList)

def containerStdDev(resourceList):
      return statistics.stdev(resourceList)

def normalUpperBound(resourceList, confidence):
      eventTotal = len(resourceList)
      mean = containerMean(resourceList)
      upperBound = 0
      if confidence == 95:
            upperBound = mean + 1.96 * math.sqrt(mean/eventTotal)
      elif confidence == 99:
            upperBound = mean + 2.60 * math.sqrt(mean/eventTotal)

      return upperBound

def poissonUpperBound(resourceList, confidence):
      eventTotal = len(resourceList)
      mean = containerMean(resourceList)
      total = mean * eventTotal
      confInv = 100 - confidence
      totalUpperBound = scipy.stats.chi2.ppf((100 - confInv/2) *
0.01, 2*(total+1))/2
      upperBound = totalUpperBound/eventTotal

      return upperBound
```

5.1.3 Classes

```
class Container:
      def __init__(self, name, usage):
            self.Name = name
            self.MeanUsage = usage
```

```
class Instance:
      def __init__(self, name, availableResource, cost):
            self.Name = name
            self.AvailableResource = availableResource
            self.Cost = cost
```

5.1.4 Data Creation

```
data = {}

with open('testdata.json') as f:
      data = json.load(f)

containers = data.get('Containers')
instances = data.get('Instances')
Containers = []
Instances = []

for containerName in containers.keys():
      containerResources = findContainer(containerName, containers)
      meanUsage = containerMean(containerResources)
      Containers.append(Container(containerName, meanUsage))

for instanceName in instances.keys():
      instanceAR = findInstanceAvaR(instanceName, instances)
      instanceCost = findInstanceCost(instanceName, instances)
      Instances.append(Instance(instanceName, instanceAR,
instanceCost))
```

5.2 Integer Linear Programming Model

5.2.1 Definitions

```
Alpha = 0.20 #constant
Beta = 0.95 #constant

#LP variable definitions
NumberOfContainers = len(ParseEquations.Containers)
NumberOfIInstances = len(ParseEquations.Instances)
XMat = LpVariable.dicts("Matrix", ((i, j) for i in
range(NumberOfContainers) for j in range(NumberOfIInstances)), 0, 1,
cat='Integer')
InstanceUsage_var = LpVariable.dicts("InstanceUsage", (j for j in
range(NumberOfIInstances)), 0 , 1, cat='Integer')
```

5.2.2 Linear Program

```
prob = LpProblem("CloudOpt", LpMinimize)
prob += lpSum([ParseEquations.Instances[j].Cost *
InstanceUsage_var[j] for j in range(NumberOfIInstances)]), "Total
Cost "


for j in range(NumberOfIInstances):
  for i in range(NumberOfContainers):
    prob += InstanceUsage_var[j] >= XMat[(i,j)]

for i in range(NumberOfContainers):
  prob += lpSum([XMat[(i, j)] for j in range(NumberOfIInstances)]) ==
1

for j in range(NumberOfIInstances):
  prob += lpSum([XMat[(i,
j)]*ParseEquations.Containers[i].MeanUsage*Alpha for i in
range(NumberOfContainers)]) + lpSum([XMat[(i,
j)]*ParseEquations.Containers[i].MeanUsage*Beta for i in
range(NumberOfContainers)]) <=
ParseEquations.Instances[j].AvailableResource
```

```
prob.writeLP('eqation.lg')
prob.solve()
print("Status:", LpStatus[prob.status])
print("All variables:")
for v in prob.variables():
    print(v.name, "=", v.varValue)

data = {}
data['Instances'] = {}
j = 0
for instance in ParseEquations.Instances:
    if(InstanceUsage_var[j].varValue == 0):
        j += 1
        continue
    data['Instances'][instance.Name] = {
        'AvailRes' : instance.AvailableResource,
        'deployedCon': {}
    }
    for i in range(NumberOfContainers):
        if XMat[(i,j)].varValue == 1:

data['Instances'][instance.Name]['deployedCon'][ParseEquations.Contai
ners[i].Name] = ParseEquations.Containers[i].MeanUsage * Alpha
    j += 1
```

5.2.3 Output Generation

```
with open("Schedule.json", "wt") as out:
    json.dump(data, out, indent = 4)

containersData = {}
containersData["Containers"] = {}
for c in ParseEquations.Containers:
    containersData["Containers"][c.Name] = {}
    containersData["Containers"][c.Name]['Mean'] = c.MeanUsage

with open("Containers.json", "wt") as containerJson:
```

```
    json.dump(containersData, containerJson, indent = 4)


#prob += lpSum([XMat[(i,'Instance1')] for i in ContainerSets]) == 1
```

5.3 Model Simulations

For the simulation, import the os, sys, json, matplotlib module, and helper function generateRandOne to generate and visualize the simulation data.

5.3.0.5 Helper Function

```
import math
import random

def generateRandOne(mean):
      L = math.exp(-mean)
      k = 0
      p = 1.0
      while p > L:
      k = k+1
      p = p * random.random()
      return k
```

5.3.1 Setup

```
RANGE_SET = 100
#arguement 1 arrangement file, arguemnt 2 supposed mean file,
arguement 3 output file
if len(sys.argv) != 4:
      print("Not enough or too many arguements.")
      sys.exit(0)
c={}
g={}
with open("inputFiles/ContatinerDeployment/" + sys.argv[1]) as
inputFile:
      c = json.load(inputFile)
with open("inputFiles/ContatinerStats/" + sys.argv[2]) as inputFile:
      g = json.load(inputFile)
i = 0
```

```
resources = {}
stats = {}
compiled = {'Instances':{},'Average Usage': 0,'Total Unstatified
Demand' :0}
totalPool = 0
availpool = 0
availRes = 0
totalrandomdemand = 0
unstatifieddemand = 0
unusedRes = 0
pool = 0
Utilization = 0
```

5.3.2 Simulation of Data

```
for x in c['Instances']:
      compiled['Instances'][x + ' Unstatified Demand'] = 0
for n in range(1,RANGE_SET + 1):
      i = 0
      stats[n] = {}
      totalPool = 0
      stats[n]['Total Unstatified Demand'] = 0
      stats[n]['Average Usage'] = 0
      #generate random resources
      for x in g['Containers']:
      #print(g['Containers'][x])
      resources[x] = generateRandOne(g['Containers'][x]['Mean'])
      stats[n]['Instances'] = {}
      for x in c['Instances']:
      i +=1
      availRes = c['Instances'][x]['AvailRes']
      availpool = c['Instances'][x]['AvailRes']
      totalPool = 0
      totalrandomdemand = 0
      unstatifieddemand = 0
      unusedRes = 0
      Utilization = 0
      for y in c['Instances'][x]['deployedCon']:
```

```
            pool = resources[y] - c['Instances'][x]['deployedCon'][y]
            totalrandomdemand += resources[y]
            unusedRes += max(-pool,0)
            availpool -= c['Instances'][x]['deployedCon'][y]
            if availpool < 0:
                  availpool = 0
            if pool < 0:
                  pool = 0
            totalPool += pool
      unstatifieddemand = max((totalPool -
availpool)/totalrandomdemand,0)
      unusedRes += max(availpool - totalPool, 0)
      Utilization = 1 - (max(unusedRes,0)/availRes)
      """
      if x =="Instance2":
            print(c['Instances'][x]['deployedCon'][y])
            print(resources['C3'])
            print(unstatifieddemand)
            print(Utilization)
      """
      stats[n]['Instances'][x] = {'UnsatisfiedDemand' :
unstatifieddemand, 'Utilization': Utilization}
      compiled['Instances'][x + ' Unstatified Demand'] +=
unstatifieddemand
      stats[n]['Total Unstatified Demand'] += unstatifieddemand
      stats[n]['Average Usage'] += Utilization

      stats[n]['Average Usage'] = stats[n]['Average Usage'] /
len(c['Instances'])
      stats[n]['Total Unstatified Demand'] = stats[n]['Total
Unstatified Demand'] / len(c['Instances'])
      compiled['Average Usage'] += stats[n]['Average Usage']
      compiled['Total Unstatified Demand'] += stats[n]['Total
Unstatified Demand']
      #print(totalPool)
      #print(overflowed)

compiled['Average Usage'] = compiled['Average Usage']/RANGE_SET
```

```
compiled['Total Unstatified Demand'] = compiled['Total Unstatified
Demand']/RANGE_SET
for x in c['Instances']:
      compiled['Instances'][x +' Unstatified Demand'] =
compiled['Instances'][x + ' Unstatified Demand']/RANGE_SET
stats[0] = compiled
```

5.3.3 Outputting File

```
with open("output/" + sys.argv[3], "wt") as out:
      json.dump(stats, out, indent = 4, sort_keys=True)
```

5.3.4 Graphing File

```
if len(sys.argv) != 2:
      print("Not enough or too many arguements.")
      sys.exit(0)

with open("output/" + sys.argv[1]) as inputFile:
      c = json.load(inputFile)

x=[]
y={}
z = {}
z2 = []
for b in c["1"]["Instances"] :
      y[b] = []
      z[b] = []
for n in range(1,101):
      x.append(n-1)
      z2.append(c[str(n)]['Average Usage'])
      for b in c["1"]["Instances"] :
      y[b].append(c[str(n)]['Instances'][b]['UnsatisfiedDemand'])
      z[b].append(c[str(n)]['Instances'][b]['Utilization'])

plt.figure(1)
#plt.style.use('seaborn-whitegrid')
for b in c["1"]["Instances"] :
      plt.plot(x, y[b], label = b)
```

```
plt.xlabel('Simulation number')
plt.xlim(0, 100)
plt.ylabel('Unsatisfied Demand')
plt.ylim(0, 0.5)
plt.title('Unsatisfied Demand of Instances')

plt.legend()

plt.figure(2)
for b in c["1"]["Instances"] :
      plt.plot(x, z[b], label = b)
plt.xlabel('Simulation number')
plt.xlim(0, 100)
plt.ylabel('Utilization')
plt.ylim(0, 1)
plt.title('Utilization of Instance')
plt.legend()

plt.figure(3)
plt.plot(x, z2, label = "Average Usage")
plt.xlabel('Simulation number')
plt.xlim(0, 100)
plt.ylabel('Utilization')
plt.ylim(0, 1)
plt.title('Utilization of All Instances')
plt.legend()
plt.show()
```

5.3.5 Static Scheduling File

```
def randomAssignment(conatiner, instances, avail):
      for contain in conatiner:
      count =0
      num =len(avail)
      if num != 1:
            num = random.randrange(1,len(avail))
      print(instances['Instances'])
      while True:
            if 'Instance' + str(num) in instances['Instances']:
                  difference = avail['Instance' + str(num)]
```

```
                  for n in instances['Instances']['Instance' +
str(num)]['deployedCon']:
                  difference -= instances['Instances']['Instance' +
str(num)]['deployedCon'][n]
                  if conatiner[contain] <= difference:
                  instances['Instances']['Instance' + str(num)]['AvailRes']
= avail['Instance' + str(num)]
                  instances['Instances']['Instance' +
str(num)]['deployedCon'][contain] = conatiner[contain]
                  break
            else:
                  if conatiner[contain] <= avail['Instance' + str(num)]:
                  instances['Instances']['Instance' + str(num)] = {}
                  instances['Instances']['Instance' +
str(num)]['deployedCon'] = {}
                  instances['Instances']['Instance' + str(num)]['AvailRes']
= avail['Instance' + str(num)]
                  instances['Instances']['Instance' +
str(num)]['deployedCon'][contain] = conatiner[contain]
                  break
            num = num + 1 % len(avail)
            count +=1
            if count >=len(avail):
                  print("could not assign resources")
                  sys.exit(1)
      return instances

def evenAssignment(conatiner, instances, avail):
      lst = [0] * len(avail)
      for contain in conatiner:
      num = lst.index(min(lst)) + 1
      print(instances['Instances'])
      while True:
            if 'Instance' + str(num) in instances['Instances']:
                  difference = avail['Instance' + str(num)]
                  for n in instances['Instances']['Instance' +
str(num)]['deployedCon']:
                  difference -= instances['Instances']['Instance' +
str(num)]['deployedCon'][n]
                  if conatiner[contain] <= difference:
                  instances['Instances']['Instance' + str(num)]['AvailRes']
= avail['Instance' + str(num)]
```

```
                  instances['Instances']['Instance' +
str(num)]['deployedCon'][contain] = conatiner[contain]
                  lst[num-1] += 1
                  break
            else:
                  if conatiner[contain] <= avail['Instance' + str(num)]:
                  instances['Instances']['Instance' + str(num)] = {}
                  instances['Instances']['Instance' +
str(num)]['deployedCon'] = {}
                  instances['Instances']['Instance' + str(num)]['AvailRes']
= avail['Instance' + str(num)]
                  instances['Instances']['Instance' +
str(num)]['deployedCon'][contain] = conatiner[contain]
                  lst[num-1] += 1
                  break
            num = lst.index(min(lst), num) + 1
            if num >=len(avail):
                  print("could not assign resources")
                  sys.exit(1)
      return instances

if __name__ == "__main__":
      print("ok")
      print(sys.argv)

#arguement 1 arrangement file, arguemnt 2 supposed mean file, arguement 3
output file
if len(sys.argv) < 2:
      print("Not enough or too many arguements.")
      sys.exit(0)

stats = {"Instances" : {}}
print(stats)
contain = {}
avail = {}
print('Indicate how many Conatiners:')
x = input()
y = int(x)
for n in range(1, y+1):
      print('Assign Resources to container' + str(n))
      x = input()
      contain['C' + str(n)] = int(x)
```

```
print(contain)

print('Indicate how many Instances:')
x = input()
t = int(x)
for n in range(1, t+1):
      print('Assign Resources to Instance' + str(n))
      x = input()
      avail['Instance' + str(n)] = int(x)
print(stats)
print("Random?")
while 1:
      x = input()
      if x == "y" or x =="yes":
      stats = randomAssignment(contain, stats, avail)
      break
      elif x == "n" or x =="no":
      stats = evenAssignment(contain, stats, avail)
      break

print(stats)
with open("inputFiles/ContatinerDeployment/" + sys.argv[1], "wt") as out:
      json.dump(stats, out, indent = 4, sort_keys=True)
```

6. Data Analysis and Discussion

Output Generation

Our output takes the values located in our test data.json. In general, the json file will have the interval in which a period of data was taken. In addition, it will have the average resource amount and cost. The input data will have to look similar to this to generate data.

```
{
"Interval" : "100",
"Containers" :
        {
                "Container1" :
                        { "Usages" :
                                [
                                        {"PeriodStartTime" : "0", "ResourceDemand" : "92"},
                                        {"PeriodStartTime" : "5", "ResourceDemand" : "82"},
                                        {"PeriodStartTime": "10", "ResourceDemand" : "92"},
                                        {"PeriodStartTime": "95", "ResourceDemand" : "92"}
                                ]
                        },
                "Container2" :
                        { "Usages" :
                                [
                                        {"PeriodStartTime" : "0", "ResourceDemand" : "95"},
                                        {"PeriodStartTime" : "5", "ResourceDemand" : "82"},
                                        {"PeriodStartTime": "10", "ResourceDemand" : "82"},
                                        {"PeriodStartTime": "95", "ResourceDemand" : "25"}
                                ]
                        },
                "Container3" :
                        { "Usages" :
                                [
                                        {"PeriodStartTime" : "0", "ResourceDemand" : "95"},
                                        {"PeriodStartTime" : "5", "ResourceDemand" : "82"},
                                        {"PeriodStartTime": "10", "ResourceDemand" : "82"},
                                        {"PeriodStartTime": "95", "ResourceDemand" : "123"}
                                ]
                        }
        },
"Instances" :
        {
                "Instance1" :
                {
                        "AvaR" : "100",
                        "Cost" : "100"
                },
                "Instance2" :
                {
                        "AvaR" : "200",
                        "Cost" : "200"
                },
                "Instance3" :
                {
                        "AvaR" : "300",
                        "Cost" : "300"
                }
        }
}
```

Figure 1: Our first set of test data, test data 1, following the format of our input data

Afterwards, using the input data, our program will generate 2 files. One to calculate the average amount of resources needed for a specific container and the other file which lists the container assignments.

To actually run the simulation, the terminal requires 3 arguments. The 2 arguments are for the deployment of containers and the mean value of the containers. The last argument is the output of simulation. This output will be located in the output file.

Finally to generate a graph of a given output, the graph.py needs one argument which is the output file from the simulation. From there, it will generate three graphs, one indicating the unused resources, one indicatciate the utilization of each instance and the final one indicating the utilization of all resources.

Output analysis

For both our randomly distributed and evenly distributed container we assigned values that were slightly higher than the mean resource value of the containers. We assigned static values for the containers because simulates what a person would do when assigning resources to containers. The simulation also excluded instances that had no containers assigned to it, meaning that an instance is not generated if there are no containers assigned to it.

For our evenly distributed container, it had the lowest utilization out of the bunch because it used a lot of instances. However, it also had an average unavailable resource percentage as well. This is because it was unlikely for the container to run out of resources if there is a bunch of buffer space in between the containers and the actual amount.

For our randomly distributed container, the results are actually closer to an evenly distributed scheduling algorithm than the custom scheduling algorithm. The random assignment also had a decent unavailable resource percentage. The algorithm just assigns containers without worrying about the overall size of the container. So out of our tests, the random assignment scheduling algorithm managed to schedule in a way such that the resource percentage was decent.

Our custom scheduling algorithm had the highest utilization rate out of all the other scheduling algorithms. In addition, it also had the lowest unavailable resource percentage as

well. The scheduling algorithm optimally assigned containers to instances such that containers had enough resources to run their assigned to them application. The graphs shown below demonstrate the analysis.

Compare Output Against Hypothesis

As shown by our output, we originally thought that the normal utilization would be about 80 to 90 percent but it turned out to be lower. The utilization of our scheduling algorithm was higher than the other scheduling algorithm. This shows that we improved the resource efficiency of containers. Plus, we managed to create an algorithm that adaptively determines the scheduling process. The output the algorithm generates fulfils the requirement of being adaptive.

Discussion

Effect on Utilization of Resources

For our simulation of our first set of inputs, our program optimally allocated all the containers' resources into a single instance to maximize efficiency and save allocation costs. This can be seen by figure 1. We can clearly see that we have a near 100% utilization rate, optimizing how our resources are being allocated to the instances.

This was a dramatic improvement compared to the outputs of the other distributed methods. For instance, the evenly distributed container had utilization records shown in figure 2. Not only did it employ all three instances, only one of the instances was fully utilized, creating an abundant amount of excess resources that results in wasted cost for any users of the container.

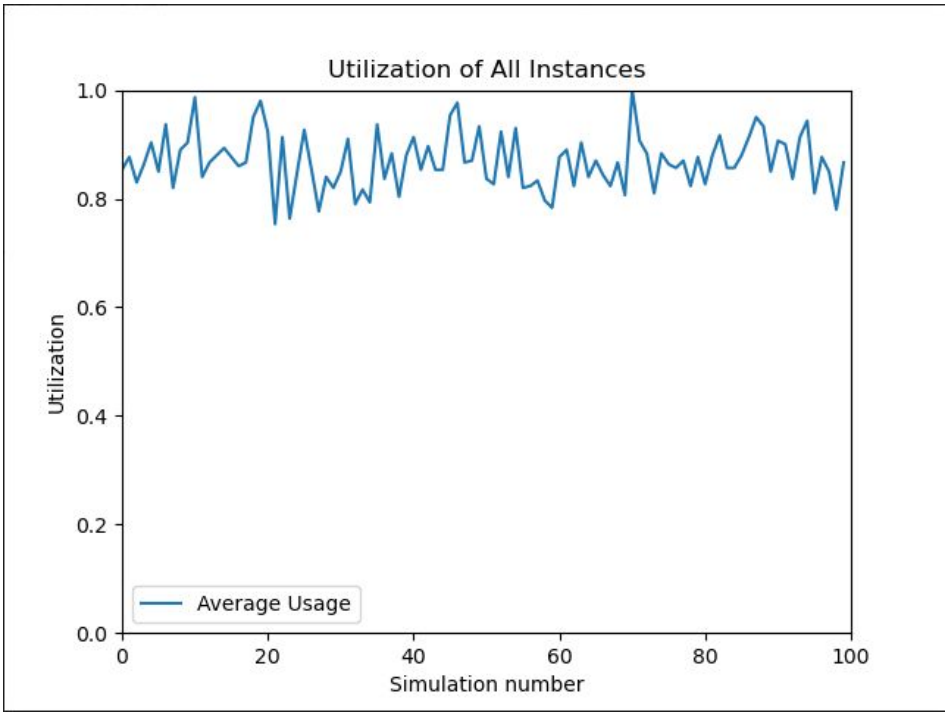

Figure 2: Utilization of all instances for test data 1. Only instance 3 was used, resulting in the high average usage.

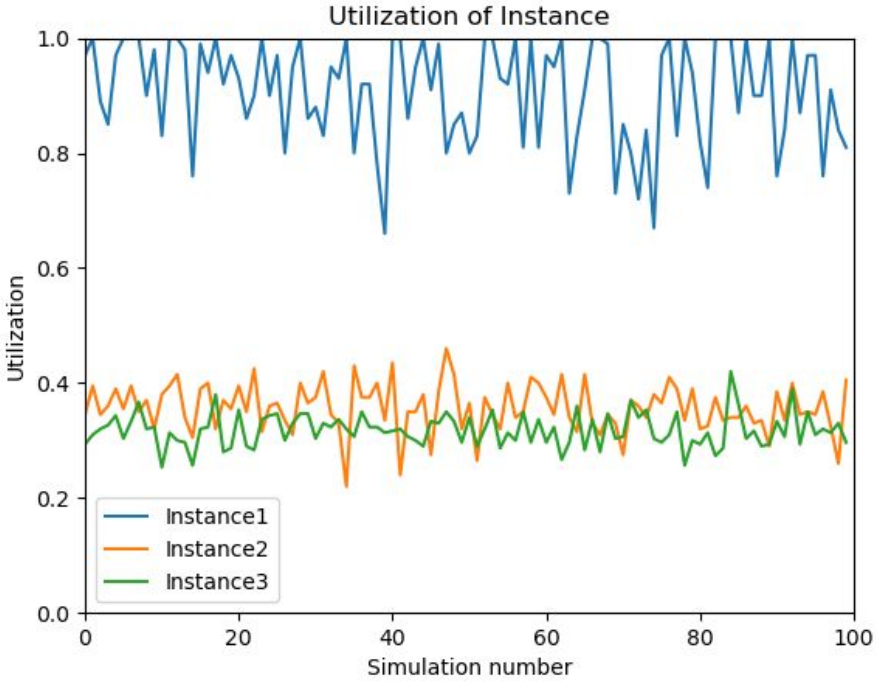

Figure 3: Utilization of Instances using Evenly Distributed Containers on test data 1

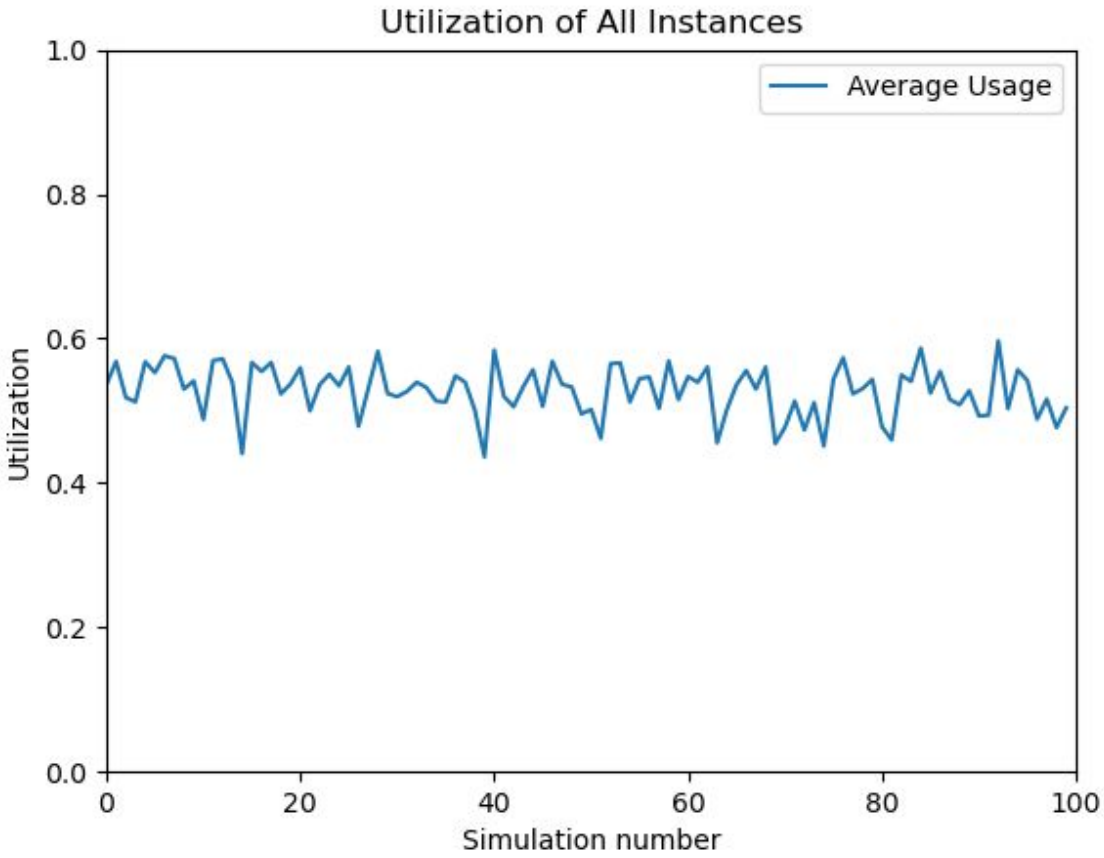

Figure 4: The average utilization of all instances using the evenly distributed containers. The result is an abundant amount of excess resources being unused.

On examining the randomly distributed container allocation, a surprisingly high utilization rate is achieved, as shown by figure 4. However, it still inefficiently accesses two instances, whereas our algorithm clearly shows that the resources can be confined to the use of one instance.

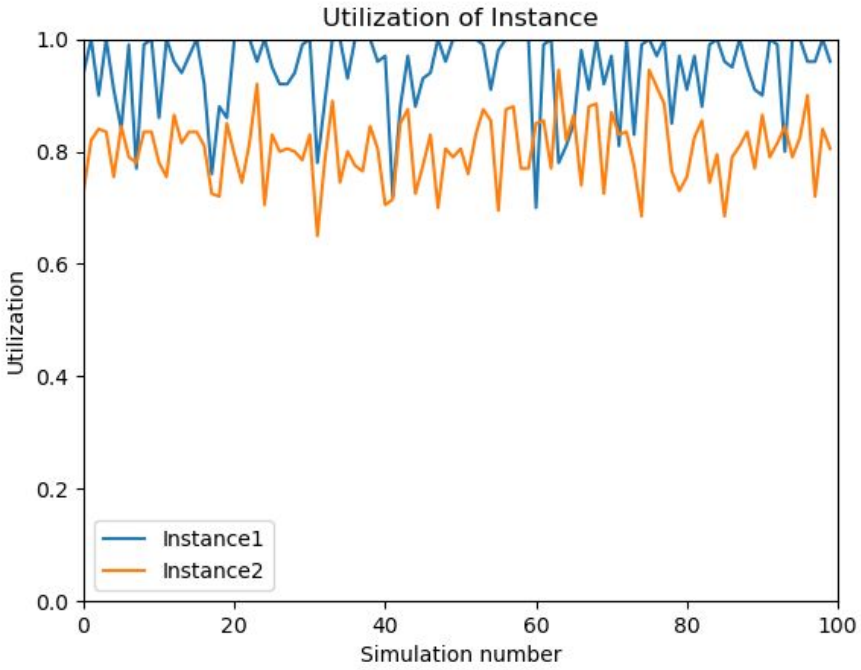

Figure 5: Utilization of Instances with randomly distributed containers on test data 1

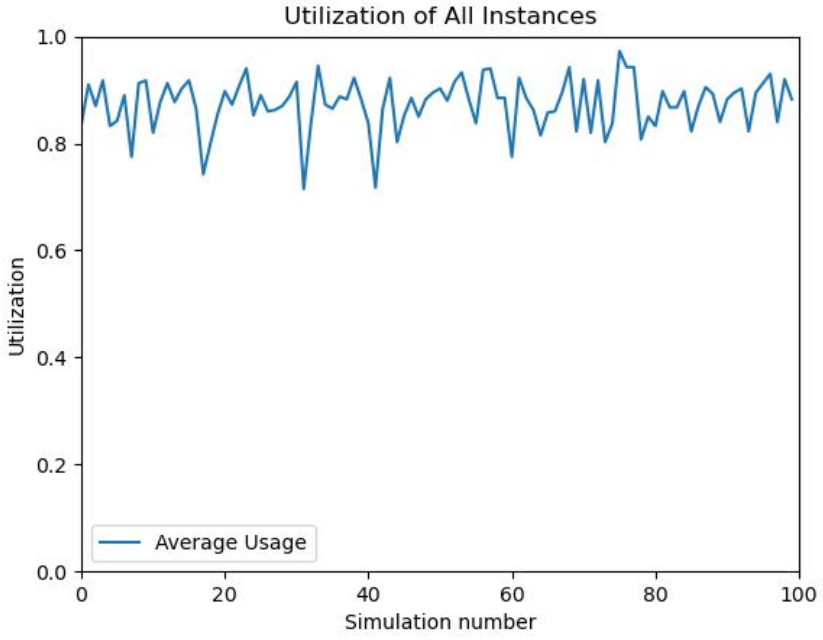

Figure 6: The average utilization of all containers using the randomly distributed method. It achieves decently high utilization rates, yet still remains lower than our linear programming model.

Furthermore, to test that our program would work with any set of test data, rather than a single example, we generated a second set of input values, shown in figure 7. Our results continued to demonstrate that our linear programming model provided a higher utilization rate than the other distribution methods, as shown by the figures below.


```
{
"Interval" : "100",
"Containers" :
        {
                "Container1" :
                        { "Usages" :
                                [
                                        {"PeriodStartTime" : "0", "ResourceDemand" : "83"},
                                        {"PeriodStartTime" : "5", "ResourceDemand" : "82"},
                                        {"PeriodStartTime": "10", "ResourceDemand" : "62"},
                                        {"PeriodStartTime": "95", "ResourceDemand" : "52"}
                                ]
                        },
                "Container2" :
                        { "Usages" :
                                [
                                        {"PeriodStartTime" : "0", "ResourceDemand" : "75"},
                                        {"PeriodStartTime" : "5", "ResourceDemand" : "42"},
                                        {"PeriodStartTime": "10", "ResourceDemand" : "62"},
                                        {"PeriodStartTime": "95", "ResourceDemand" : "35"}
                                ]
                        },
                "Container3" :
                        { "Usages" :
                                [
                                        {"PeriodStartTime" : "0", "ResourceDemand" : "58"},
                                        {"PeriodStartTime" : "5", "ResourceDemand" : "66"},
                                        {"PeriodStartTime": "10", "ResourceDemand" : "48"},
                                        {"PeriodStartTime": "95", "ResourceDemand" : "99"}
                                ]
                        }
        },
"Instances" :
        {
                "Instance1" :
                {
                        "AvaR" : "100",
                        "Cost" : "100"
                },
                "Instance2" :
                {
                        "AvaR" : "150",
                        "Cost" : "150"
                },
                "Instance3" :
                {
                        "AvaR" : "200",
                        "Cost" : "200"
                }
        }
}
```


Figure 7: Our input data for test data 2

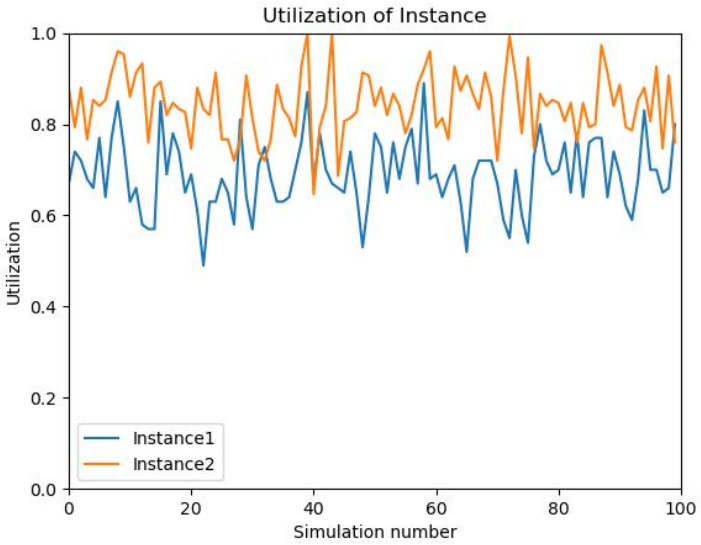

Figure 8: The utilization of containers using the custom scheduling method for test data 2.

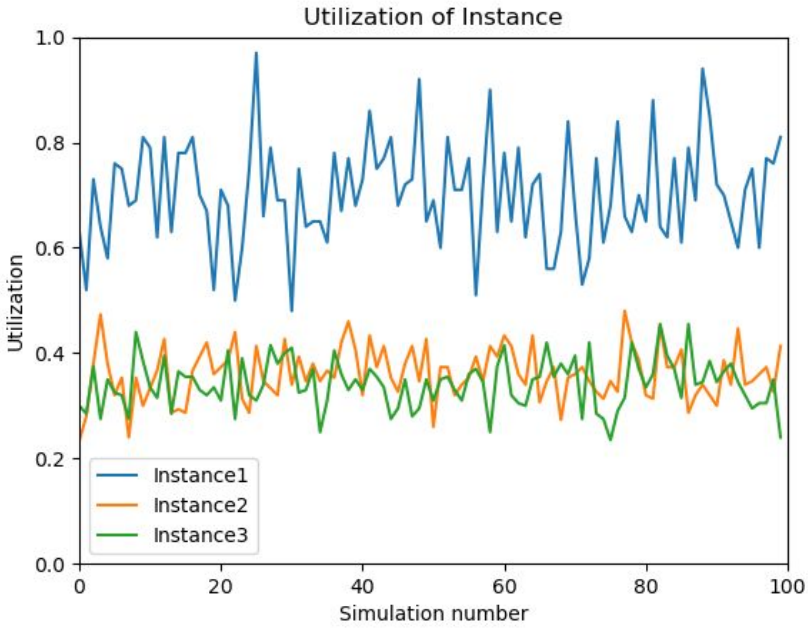

Figure 9: The utilization of containers using the even distribution method for test data 2.

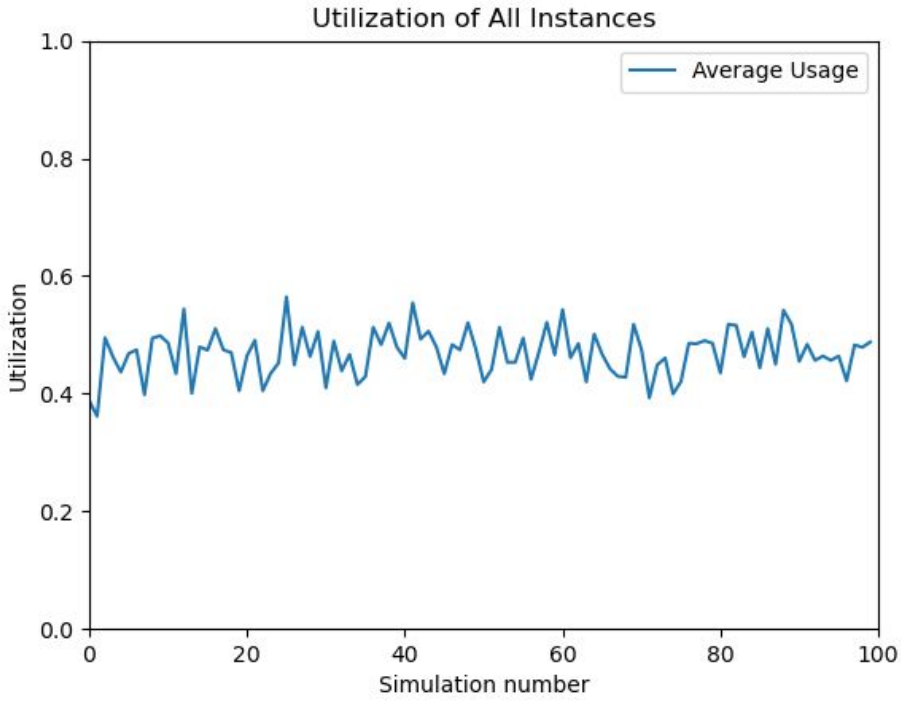

Figure 10: The utilization of all containers using the even distribution method for test data 2.

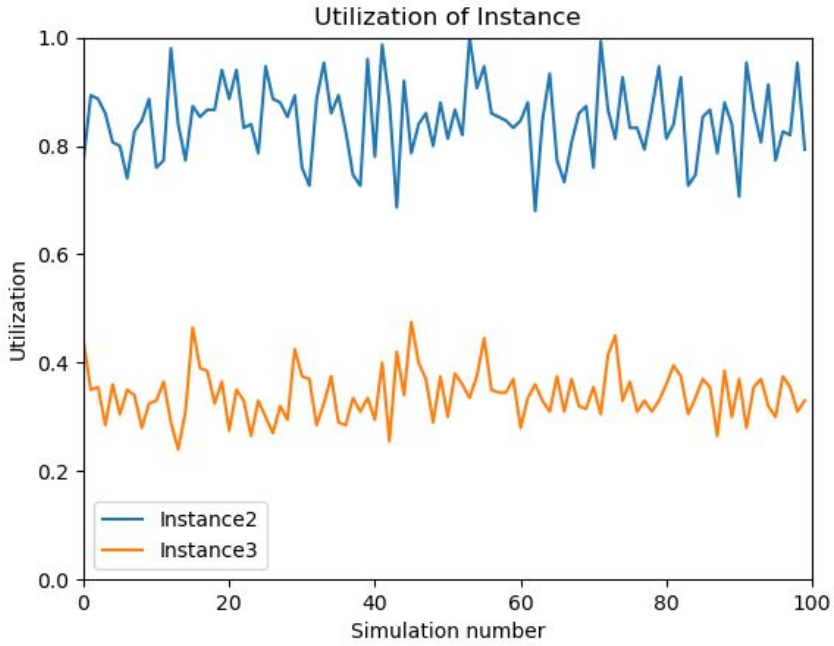

Figure 11: The utilization of containers using the random distribution method for test data 2.

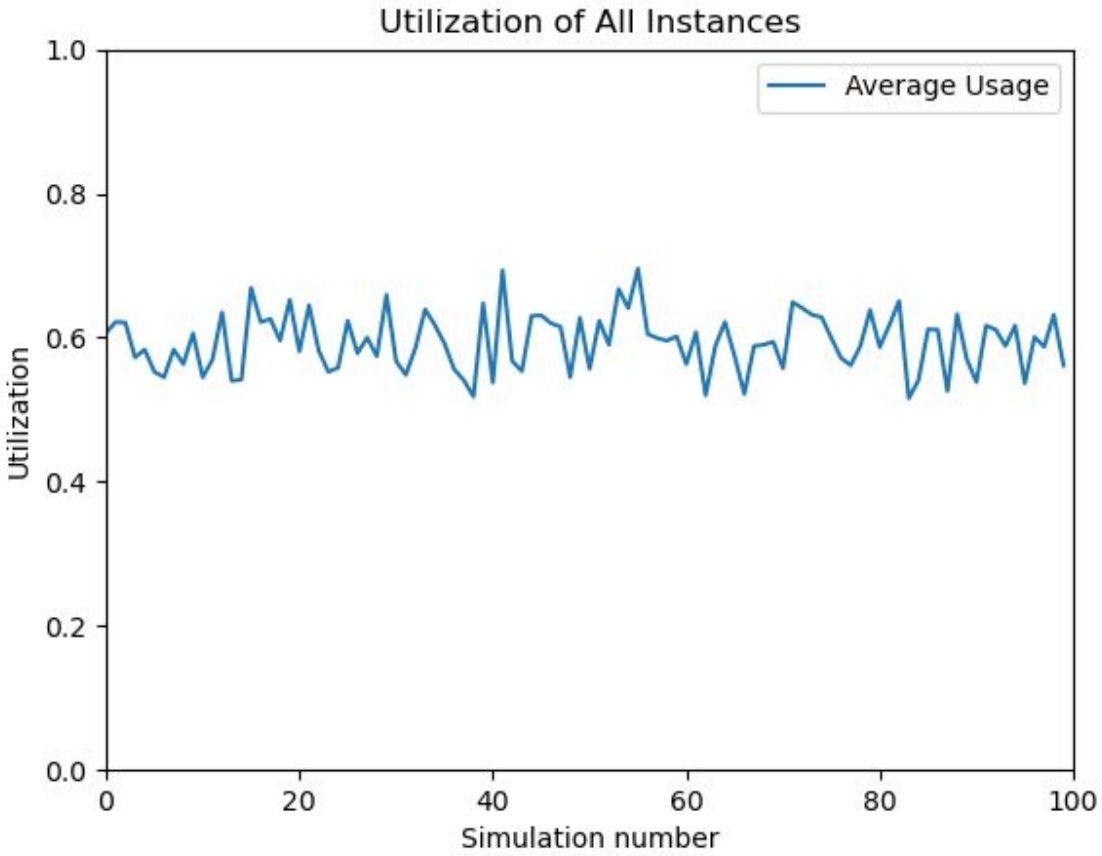

Figure 12: The utilization of all containers using the random distribution method for test data 2.

Effect on Underallocation of Resources

Similarly, we tested to ensure that our algorithm would not underallocate resources, which happens if the containers were allocated to an instance which did not have enough space to compensate for the new resources. In test data 1, we achieved near perfect results, with only a small bump in the data, where a small amount of unsatisfied demand was found, as seen in figure 13.

However, the evenly distributed and randomly distributed methods were considerably less successful in this aspect. Both methods demonstrated high amounts of underallocation of resources consistently throughout the simulations. These results can be seen in figures 14 and 15.

On our second set of test data, all three methods achieved similar results to their performance on test data 1. On our linear programming model, there was yet another bump found in the second instance. This could be caused by an outlier of resource allocation found in the test data. For instance, one of the container's resource demands spiked dramatically, which

was an outlier to the overall mean of the resource demands. This could have caused the simulation to portray the small bump in the data.

Incidentally, the evenly distributed scheduling method achieved perfect results in test 2. This may be attributed to us properly allocating enough resources as the evenly distributed amount, which resulted in a lower utilization rate instead. It is still clear that our linear programming model achieved the best results in terms of ensuring all the satisfied demands are met, while maximizing the utilization rate.

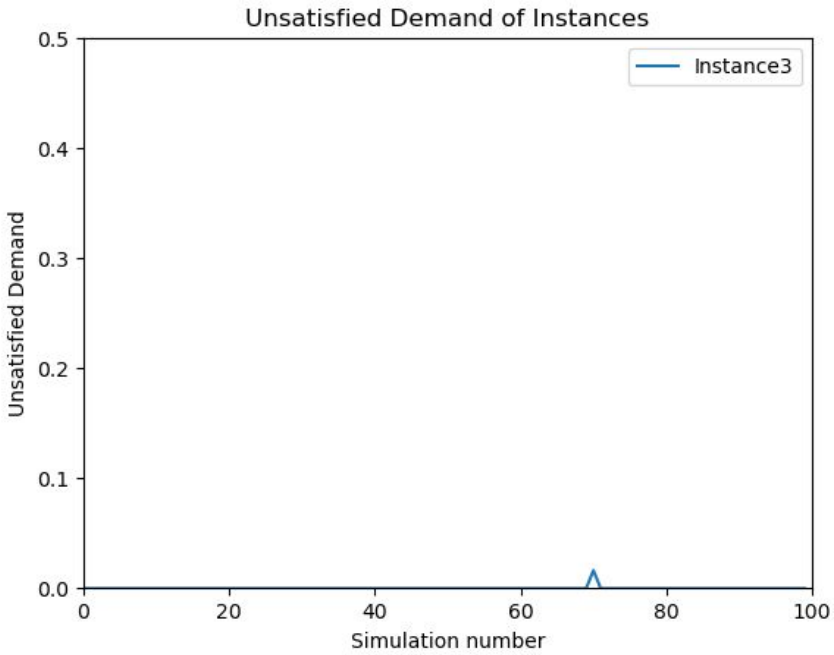

Figure 13: The unsatisfied demand of containers using the custom method for test data 1.

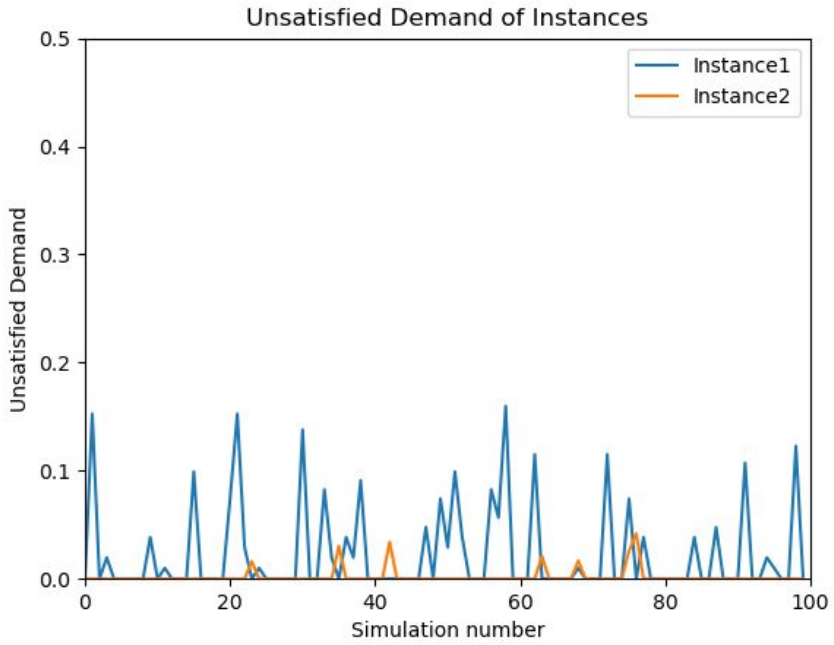

Figure 14: The unsatisfied demand of containers using the random method.

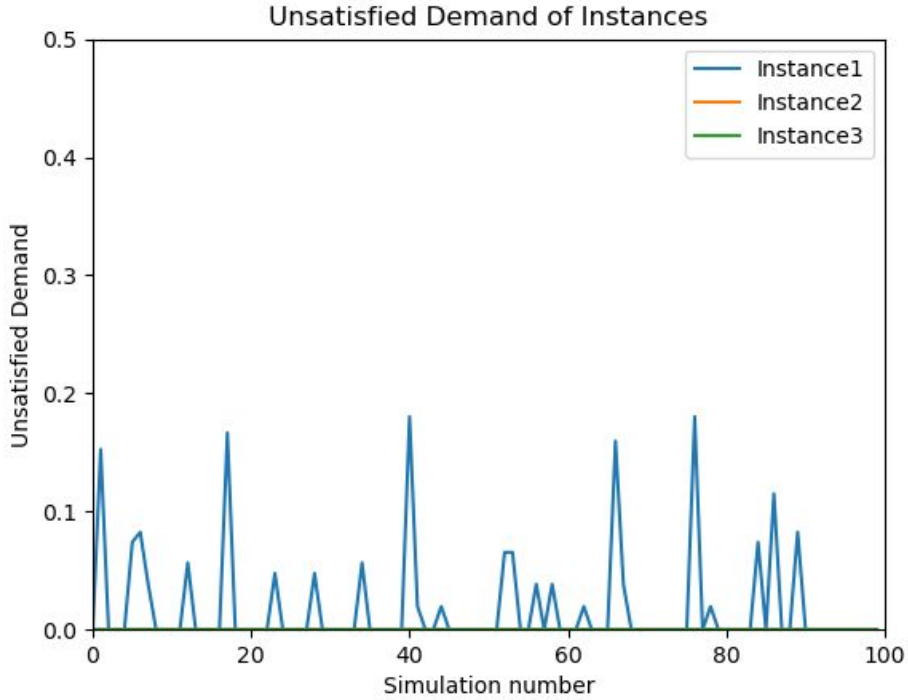

Figure 15: The unsatisfied demand of containers using the even distribution method.

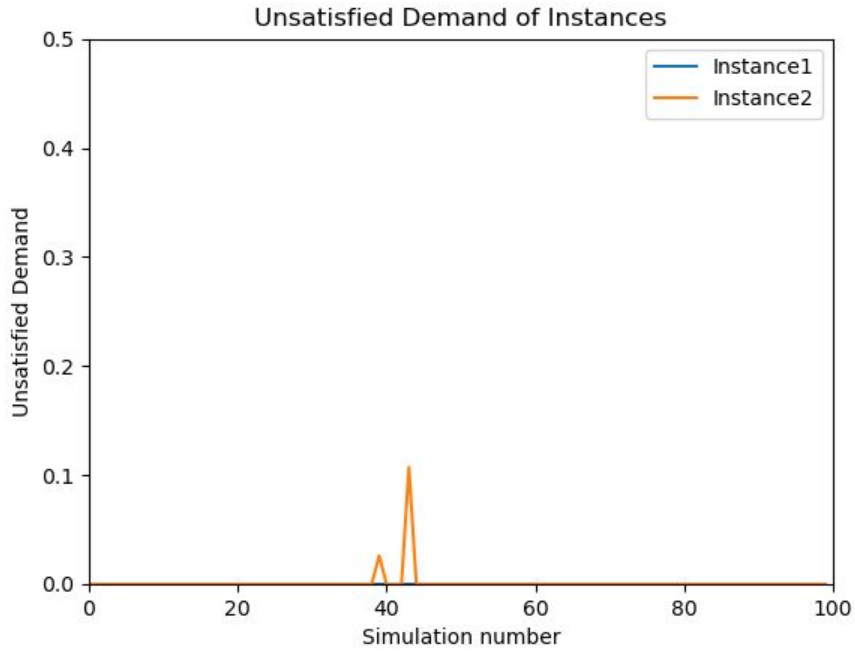

Figure 16: The unsatisfied demand of containers using the adaptive scheduling distribution method set 2.

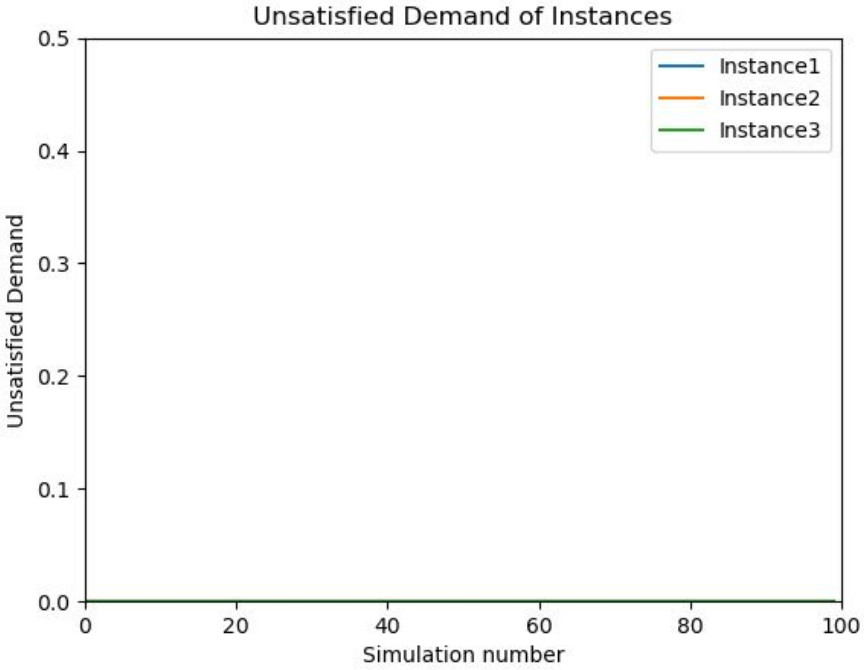

Figure 17: The unsatisfied demand of containers using the even scheduling distribution method set 2.

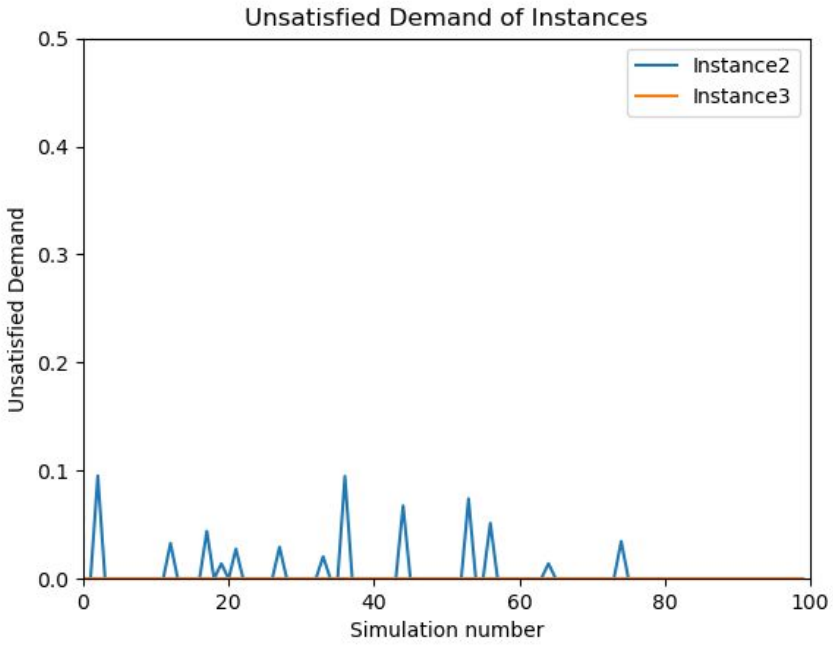

Figure 18: The unsatisfied demand of containers using the random scheduling distribution method set 2.

7. Conclusion and Recommendations

Summary

Our program had a higher utilization than the other scheduling algorithms. This means that our program was able to utilize our resources greatly. In addition, there were small times where we could not fulfill the request, but overall it was better than assigning the containers randomly or trying to distribute the containers evenly among the instances. So as a result, the scheduling program improved the utilization of the containers, while overall maintaining the integrity of the resource allocation.

Recommendation

Because of time issues, we had to limit our scope to not include collecting data from AWS. As a result, we tested only using theoretical values and simulations rather than actual values. So, it is likely recommended to use data to figure out how to schedule. Doing so will give a better review of the reliability of the scheduling algorithm.

In addition, our simulation did not contain a parameter indicating how much power was used in generating resources. Theoretically, our formula should give the optimal amount of instances required to schedule. So that means less instances and power should be used. We want to see if power is decreased as a result of using our scheduling algorithm.

The other suggestion is to compare this algorithm to a dynamic scheduling algorithm. We were not able to come up with a dynamic scheduling algorithm for our project. However, if this algorithm is compared to a dynamic algorithm there is potential to see the advantages and disadvantages of using either adaptive scheduling or dynamic scheduling.

Lastly, in order to make our solution usable an online service can be developed to extract actual usage data from a cloud service. Then the service should run our scheduling algorithm and apply the results back to the cloud service.

8. Bibliography

Awada, Uchechukwu, and Adam Barker. "Improving Resource Efficiency of Container-Instance Clusters on Clouds." *2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, 2017, 929–34. https://doi.org/10.1109/ccgrid.2017.113.

Huang, Hang, Jia Rao, Song Wu, Hai Jin, Kun Suo, and Xiaofeng Wu. "Adaptive Resource Views for Containers." *Proceedings of the 28th International Symposium on High-Performance Parallel and Distributed Computing*, 2019, 243–54. https://doi.org/10.1145/3307681.3325403.

Li, Han, Limin Zhang, Wubin Li, and Jing Gao. "A Service Performance Based Dynamic Provisioning Approach in Containerized Cloud Environments." *Proceedings of the 2nd International Conference on Big Data Technologies - ICBDT2019*, 2019, 177–81. https://doi.org/10.1145/3358528.3358568.

Wang, Ming-Hwa. "Parallel Processing and Distributed Computing." http://www.cse.scu.edu/~mwang2/. Accessed 9 June 2020.

Zhiyong, Cai, and Xie Xiaolan. "An Improved Container Cloud Resource Scheduling Strategy." *Proceedings of the 2019 4th International Conference on Intelligent Information Processing*, 2019. https://doi.org/10.1145/3378065.3378138.

Zhong, Zhiheng, and Rajkumar Buyya. "A Cost-Efficient Container Orchestration Strategy in Kubernetes-Based Cloud Computing Infrastructures with Heterogeneous Resources." *ACM Transactions on Internet Technology* 20, no. 2 (2020): 1–24. https://doi.org/10.1145/3378447.

Zhou, Ruiting, Zongpeng Li, and Chuan Wu. "Scheduling Frameworks for Cloud Container Services." *IEEE/ACM Transactions on Networking* 26, no. 1 (2018): 436–50. https://doi.org/10.1109/tnet.2017.2781200.

9. Appendix

List of input file/output files

Input files

In P3 folder

testdata.json

In simulation folder

Schedule.json

Containers.json

Output files

SimOut.json