

**TITLE: Exploring the possibilities of Load Balancing in
distributed systems**

by

NAME: Eshan Das

Atharva Dhande

Shoumyadeep Dhani

REGISTER NUMBER: 19BCE1575

19BCE1571

19BCE1543

A project report submitted to

Dr. Renuka Devi S.

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

in partial fulfilment of the requirements for the course of

CSE2005 – OPERATING SYSTEMS

in

B. Tech. COMPUTER SCIENCE AND ENGINEERING



VIT[®]
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

Vandalur – Kelambakkam Road

Chennai – 600127

JUNE 2021

Abstract:

This project describes the necessary, newly developed, principal concepts for several load balancing techniques in a distributed computing environment. Our project also includes various types of load balancing strategies, their merits, demerits and comparison depending on certain parameters. Distributed computing is a promising technology that involves coordinate and involvement of resources to carry out multifarious computational problems. One of the major issues in distributed systems is the design of an efficient dynamic load balancing algorithm that improves the overall performance of the distributed systems. Scheduling and resource management plays a decisive role in achieving high utilization of resources in grid computing environments. Load balancing is the process of distributing the load among various nodes of a distributed system to improve both job response time and resource utilization while also avoiding a situation where some of the nodes are heavily loaded while other nodes are idle or lightly loaded.

Introduction about the project:

Distributed systems offer the potential for sharing and aggregation of different resources such as computers, storage systems and other specialized devices. These resources are distributed and possibly owned by different agents or organization. The users of a distributed system have different goals, objectives and strategies and their behavior is difficult to characterize. In such systems the management of resources and applications is a very complex task. A distributed system can be viewed as a collection of computing and communication resources shared by active users. When the demand for computing power increases, the load balancing problem becomes important. The purpose of load balancing is to improve the performance of a distributed system through an appropriate distribution of the application load. Load balancing involves assigning tasks to each processor and minimizing the execution time of the program. Load balancing has several advantages:

- Load balancing improves the performance of each node and hence the overall system performance.
- Load balancing reduces the job idle time
- Small jobs do not suffer from long starvation

- Maximum utilization of resources
- Response time becomes shorter
- Higher throughput
- Higher reliability
- Low cost but high gain
- Extensibility and incremental growth

Module Description:

❑ Analysis of static load balancing techniques

Static Load Balancing is a type of Load Balancing which is often referred to as the mapping problem, the task is to map a static process graph onto a fixed hardware topology in order to minimize dilation and process load differences.

In static algorithm the processes are assigned to the processors at the compile time according to the performance of the node. Once the process are assigned, no change or reassignment is possible at the run time. Number of jobs in each node is fixed and these algorithms do not collect any information about the nodes. The assignment of jobs is done to the processing nodes on the basis of the factors such as incoming time, extent of resource needed, mean execution time and inter-process communications. Hence it is also called probabilistic algorithm.

❑ Analysis of various dynamic load balancing techniques in terms of performance

It is desirable in a distributed system to have the system load balanced evenly among the nodes so that mean job response time is minimized. In dynamic load balancing algorithm assignment of jobs is done at the runtime. Here jobs are reassigned at the runtime depending upon the situation that is the load will be transferred from heavily loaded nodes to the lightly loaded nodes. In this case communication overheads occur and become more when number of processors increase. No decision is taken until the process gets execution. This strategy collects information about the system state and about the job information.

❑ Drawing a comparison between the static and dynamic load balancing algorithms

In this module we will be showing comparison between the static and dynamic load balancing algorithms in the form of resources utilized, response time and thorough analysis of the slope of the resultant graphs.

Hardware / software used:

We are using Ubuntu platform as the software for our project and Debian for analysing various algorithms on three servers.

Ubuntu is a complete Linux operating system, freely available with both community and professional support. The Ubuntu community is built on the ideas enshrined in the Ubuntu Manifesto: that software should be available free of charge, that software tools should be usable by people in their local language and despite any disabilities, and that people should have the freedom to customize and alter their software in whatever way they see fit.

- Ubuntu will always be free of charge, and there is no extra fee for the “enterprise edition”, we make our very best work available to everyone on the same Free terms.
- Ubuntu includes the very best in translations and accessibility infrastructure that the Free Software community has to offer, to make Ubuntu usable by as many people as possible.
- Ubuntu is shipped in stable and regular release cycles; a new release will be shipped every six months. Every two even years an Ubuntu long term support (LTS) release will become available, that is supported for 5 years. The Ubuntu releases in between (known as development or non-LTS releases) are supported for 9 month each.
- Ubuntu is entirely committed to the principles of open source software development; which encourages people to use open source software, improve it and pass it on.

Sample coding:

Static Algorithms

Round Robin

```
n=int(input("Enter total number of requests: "))
position=0
count=1
while(n!=0):
    if position>2:
        position=0
    else:
        position+=1
        print("Client Request "+str(count)+": served by server "+str(position))
        n-=1
        count+=1
```

Weighted Round Robin

```
print("Enter the weight of the 3 servers with spaces: ")
l=list(map(int,input().split()))
t=int(input("Enter the total number of requests: "))
position=0
count=0
while t!=1:
    if position>2:
        position=0
    else:
        position+=1
        for i in range(l[position-1]):
            count+=1
            print("Client Request "+str(count)+": served by server "+str(position))
            if t!=1:
                t-=1
            else:
                break
```

Random

```
import random
n=int(input("Enter the number of requests: "))
l=[]
l1=[]
```

```

l2=[]
l3=[]
l4=[]
for i in range(1,n+1):
    l.append(i)
for i in range(1,n+1):
    l4.append(i)
while len(l)!=0:
    a=random.randint(1,n)
    if a in l:
        if len(l1)<=len(l2) and len(l1)<=len(l3):
            l1.append(a)
        elif len(l2)<=len(l3) and len(l2)<len(l1):
            l2.append(a)
        else:
            l3.append(a)
    l.remove(a)
l1.sort()
l2.sort()
l3.sort()
for i in l4:
    if i in l1:
        print("Client request "+str(i)+" served by server 1")
    elif i in l2:
        print("Client request "+str(i)+" served by server 2")
    else:
        print("Client request "+str(i)+" served by server 3")

```

Weighted Random

```

import random
n=int(input("Enter the number of requests: "))
print("Enter the weights of the 3 servers with space")
k=list(map(int,input().split()))
l=[]
l1=[]
l2=[]
l3=[]
l4=[]
key1=k[0]
key2=k[1]
key3=k[2]

```

```

sum1=sum(k)
for i in range(1,n+1):
    l.append(i)
for i in range(1,n+1):
    l4.append(i)
while len(l)!=0:
    a=random.randint(1,n)
    if sum1==0:
        key1=k[0]
        key2=k[1]
        key3=k[2]
        sum1=sum(k)
    if a in l:
        if key1!=0:
            l1.append(a)
            key1-=1
            sum1-=1
        elif key1==0 and key2!=0:
            l2.append(a)
            key2-=1
            sum1-=1
        elif key1==0 and key2==0:
            l3.append(a)
            key3-=1
            sum1-=1
        l.remove(a)
l1.sort()
l2.sort()
l3.sort()
for i in l4:
    if i in l1:
        print("Client request "+str(i)+" served by server 1")
    if i in l2:
        print("Client request "+str(i)+" served by server 2")
    if i in l3:
        print("Client request "+str(i)+" served by server 3")

```

Dynamic Algorithms

Weighted Response Time

```

import time
n=int(input("Enter total number of servers: "))
count=4
l=[]
start = time.time()
print("Client Request 1 served by server 1")
time.sleep(1)
end = time.time()
l.append(end-start)
start = time.time()
print("Client Request 2 served by server 2")
time.sleep(1)
end = time.time()
l.append(end-start)
start = time.time()
print("Client Request 3 served by server 3")
time.sleep(1)
end = time.time()
l.append(end-start)
n-=3
position=0
count=4
k=[]
k.append(1)
k.append(int(l[1]/l[0]))
k.append(int(l[2]/l[0]))
while n!=1:
    if position>2:
        position=0
    else:
        position+=1
        for i in range(k[position-1]):
            count+=1
            print("Client Request "+str(count)+": served by server "+str(position))
            if n!=1:
                n-=1
            else:
                break

```

Least Connections

```

n=int(input("Enter total number of client requests: "))

```



```

m=int(input("Enter total number of clients: "))
count=1
ac1=0
ac2=0
ac3=0
dc1=0
dc2=0
dc3=0
k=0
while n!=0:
    if ac1<=ac2 and ac1<=ac3:
        print("Client request "+str(count)+" served by server 1")
        ac1+=1
        n-=1
    elif ac2<=ac3 and ac2<ac1:
        print("Client request "+str(count)+" served by server 2")
        ac2+=1
        n-=1
    else:
        print("Client request "+str(count)+" served by server 3")
        ac3+=1
        n-=1
    k+=1
    count+=1
    if k==m:
        dc1=int(input("Enter number of connections disconnected from server 1: "))
        dc2=int(input("Enter number of connections disconnected from server 2: "))
        dc3=int(input("Enter number of connections disconnected from server 3: "))
        ac1-=dc1
        ac2-=dc2
        ac3-=dc3
        k=0

```

Weighted least connections

```

n=int(input("Enter total number of client requests: "))
m=int(input("Enter total number of clients: "))
print("Enter the weight of the servers with space")
weight=list(map(int,input().split()))
count=1
ac1=0
ac2=0

```

```

ac3=0
dc1=0
dc2=0
dc3=0
k=0
while n!=0:
    if ac1<=ac2 and ac1<=ac3:
        for i in range(weight[0]):
            print("Client request "+str(count)+" served by server 1")
            ac1+=1
            count+=1
            if count>n:
                n=0
                break
    elif ac2<=ac3 and ac2<ac1:
        for i in range(weight[1]):
            print("Client request "+str(count)+" served by server 2")
            ac2+=1
            count+=1
            if count>n:
                n=0
                break
    else:
        for i in range(weight[2]):
            print("Client request "+str(count)+" served by server 3")
            ac3+=1
            count+=1
            if count>n:
                n=0
                break
    k+=1
    if k==m:
        dc1=int(input("Enter number of connections disconnected from server 1: "))
        dc2=int(input("Enter number of connections disconnected from server 2: "))
        dc3=int(input("Enter number of connections disconnected from server 3: "))
        ac1-=dc1
        ac2-=dc2
        ac3-=dc3
        k=0

```

Nearest Neighbour

```

n=int(input("Enter total number of requests: "))
print("Enter initial load of the three servers with space")
l=list(map(int,input().split()))
count=1
key1=1
key2=0
key3=0
while n!=0:
    if l[0]<=l[1] and key1==1:
        print("Client request "+str(count)+" served by server 1")
        n-=1
        count+=1
        key1=1
        key2=0
        key3=0
        l[0]+=1
        continue
    elif l[0]>l[1] and key1==1:
        print("Client request "+str(count)+" served by server 2")
        n-=1
        count+=1
        key1=0
        key2=1
        key3=0
        l[1]+=1
        continue
    if l[1]<=l[0] and l[1]<=l[2] and key2==1:
        print("Client request "+str(count)+" served by server 2")
        n-=1
        count+=1
        key1=0
        key2=1
        key3=0
        l[1]+=1
        continue
    elif l[1]>l[2] and l[2]<l[0] and key2==1:
        print("Client request "+str(count)+" served by server 3")
        n-=1
        count+=1
        key1=0
        key2=0

```

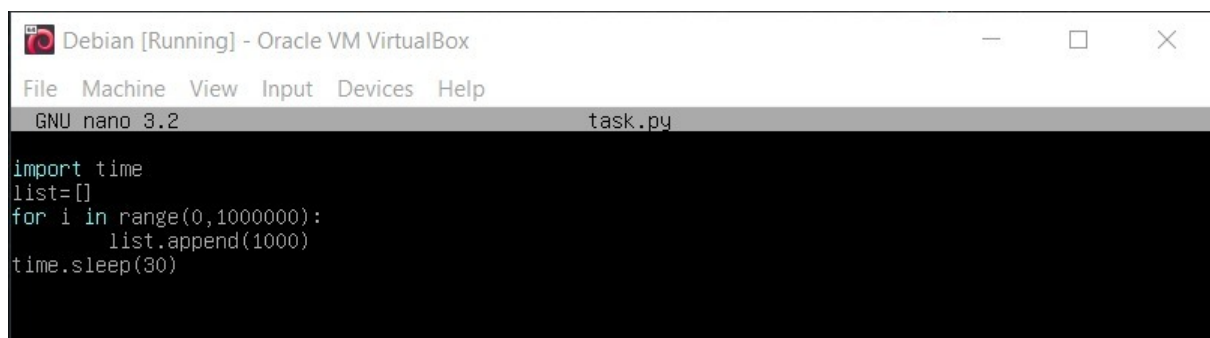
```
key3=1
l[2]+=1
continue
elif l[1]>l[0] and l[0]<l[2] and key2==1:
    print("Client request "+str(count)+" served by server 1")
    n-=1
    count+=1
    key1=1
    key2=0
    key3=0
    l[0]+=1
    continue
elif l[2]>l[1] and l[2]>l[0] and key2==1:
    print("Client request "+str(count)+" served by server 1")
    n-=1
    count+=1
    key1=1
    key2=0
    key3=0
    l[0]+=1
    continue
elif l[1]>l[0] and l[1]>l[2] and key2==1:
    print("Client request "+str(count)+" served by server 1")
    n-=1
    count+=1
    key1=1
    key2=0
    key3=0
    l[0]+=1
    continue
if l[2]<=l[1] and key3==1:
    print("Client request "+str(count)+" served by server 3")
    n-=1
    count+=1
    key1=0
    key2=0
    key3=1
    l[2]+=1
    continue
elif l[2]>l[1] and key3==1:
    print("Client request "+str(count)+" served by server 2")
```

```

n-=1
count+=1
key1=0
key2=1
key3=0
l[1]+=1
continue

```

SOCKET PROGRAMMING FOR THREE SERVERS AND IMPLEMENTATION TOOL



Debian [Running] - Oracle VM VirtualBox

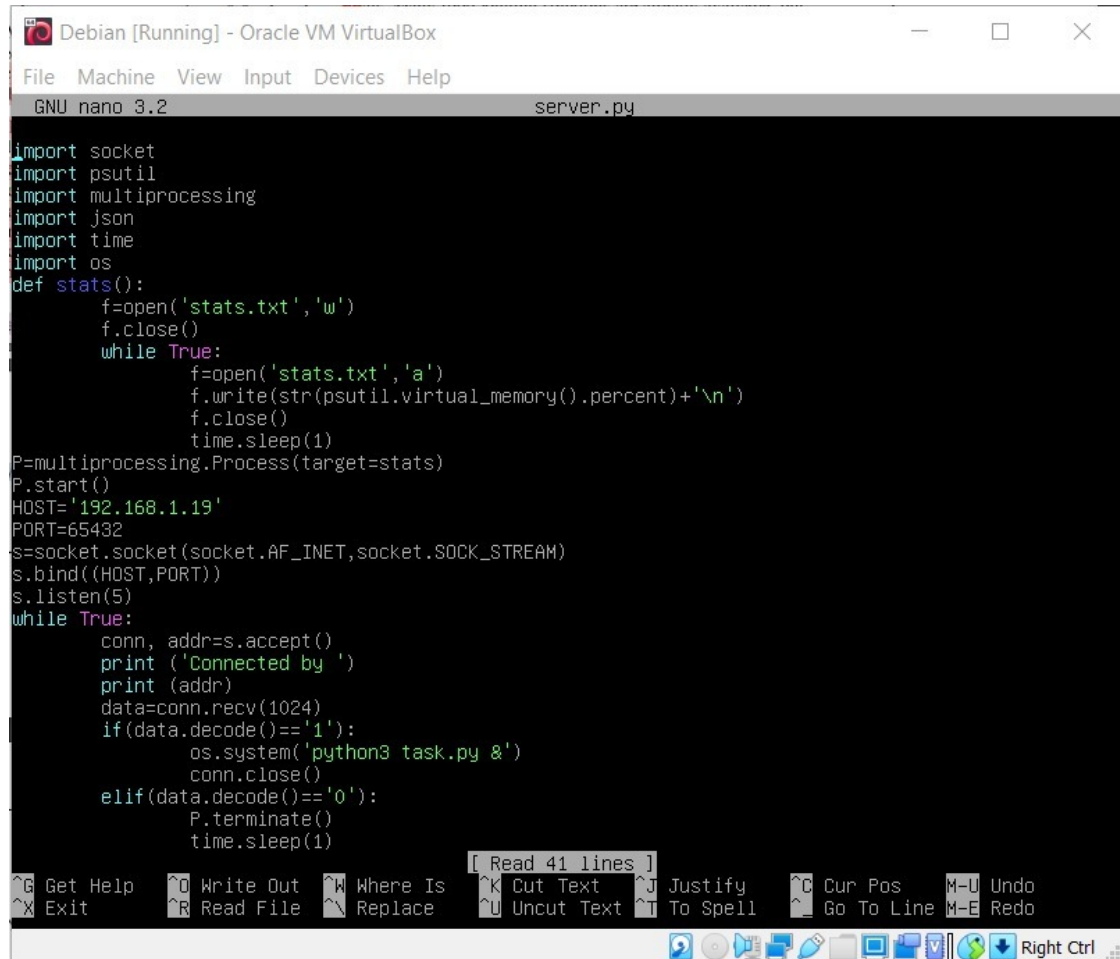
File Machine View Input Devices Help

GNU nano 3.2 task.py

```

import time
list=[]
for i in range(0,1000000):
    list.append(1000)
time.sleep(30)

```



Debian [Running] - Oracle VM VirtualBox

File Machine View Input Devices Help

GNU nano 3.2 server.py

```

import socket
import psutil
import multiprocessing
import json
import time
import os
def stats():
    f=open('stats.txt','w')
    f.close()
    while True:
        f=open('stats.txt','a')
        f.write(str(psutil.virtual_memory().percent)+'\n')
        f.close()
        time.sleep(1)
P=multiprocessing.Process(target=stats)
P.start()
HOST='192.168.1.19'
PORT=65432
s=socket.socket(socket.AF_INET,socket.SOCK_STREAM)
s.bind((HOST,PORT))
s.listen(5)
while True:
    conn, addr=s.accept()
    print ('Connected by ')
    print (addr)
    data=conn.recv(1024)
    if(data.decode()=='1'):
        os.system('python3 task.py &')
        conn.close()
    elif(data.decode()=='0'):
        P.terminate()
        time.sleep(1)

```

[Read 41 lines]

Get Help Write Out Where Is Cut Text Justify Cur Pos M-U Undo
Exit Read File Replace Uncut Text To Spell Go To Line M-E Redo

Right Ctrl

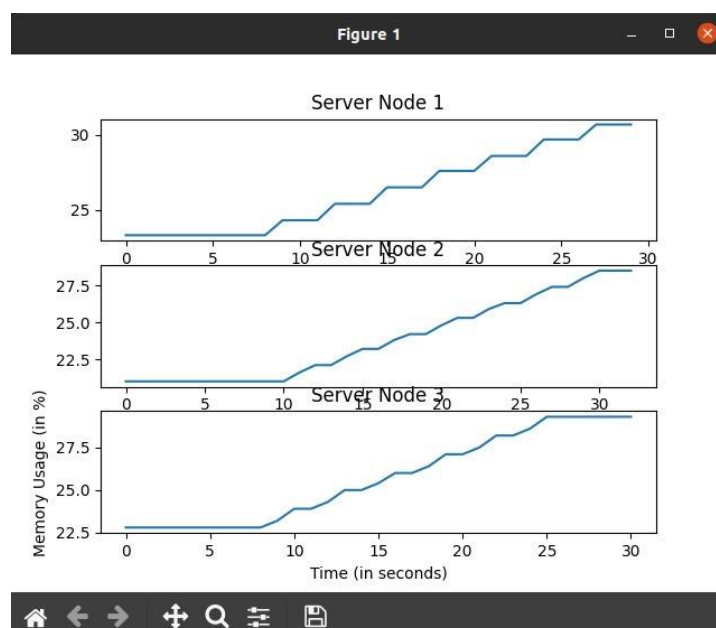
```
Debian [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
GNU nano 3.2 server.py

HOST='192.168.1.19'
PORT=65432
s=socket.socket(socket.AF_INET,socket.SOCK_STREAM)
s.bind((HOST,PORT))
s.listen(5)
while True:
    conn, addr=s.accept()
    print ('Connected by ')
    print (addr)
    data=conn.recv(1024)
    if(data.decode()=='1'):
        os.system('python3 task.py &')
        conn.close()
    elif(data.decode()=='0'):
        P.terminate()
        time.sleep(1)
        f=open('stats.txt','r')
        content=f.read().splitlines()
        data={'a':content}
        data=json.dumps(data)
        conn.sendall(data.encode())
        f.close()
        conn.close()
        break
s.close()
```

Screenshots and analysis:

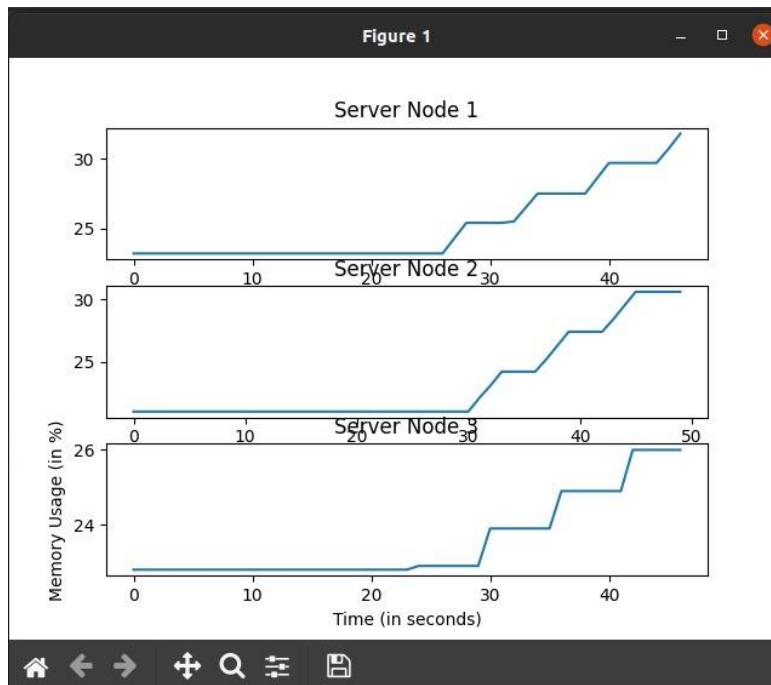
Static Algorithms

Round Robin



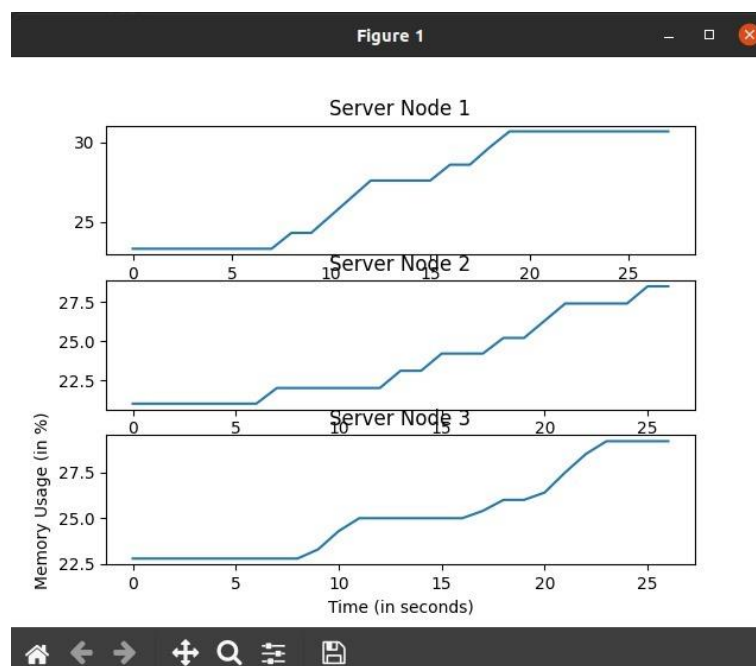
The graph above suggests that each server has different computing power but since same number of requests has been assigned hence equal iterations.

Weighted Round Robin



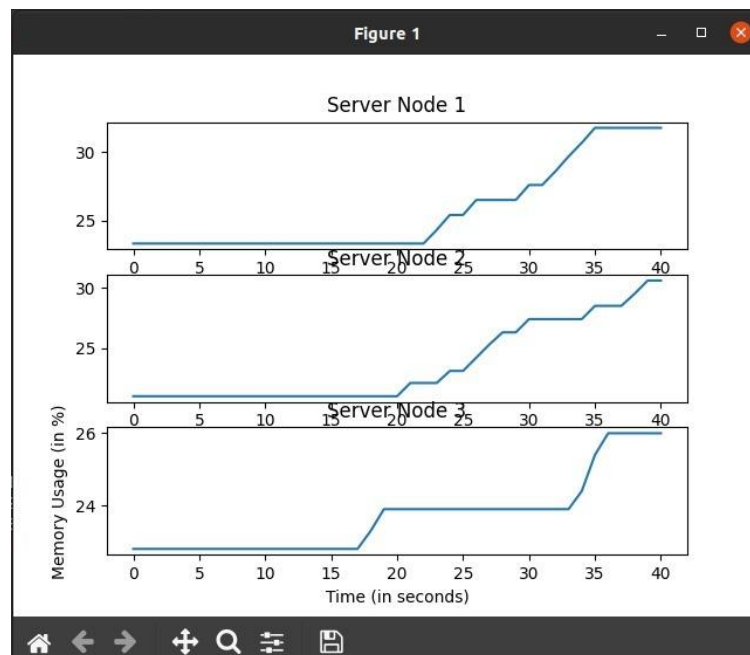
Here the iteration of each server depends on the weight assigned to each server.

Random



The requests are assigned randomly but equally hence graph may vary each time the algorithm is run.

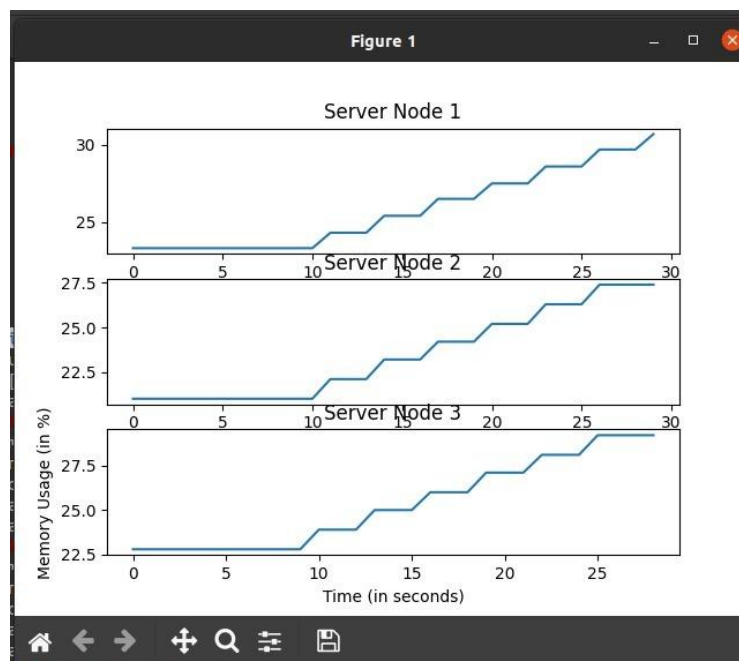
Weighted Random



The requests are assigned randomly, equally and according to the weight of the server.

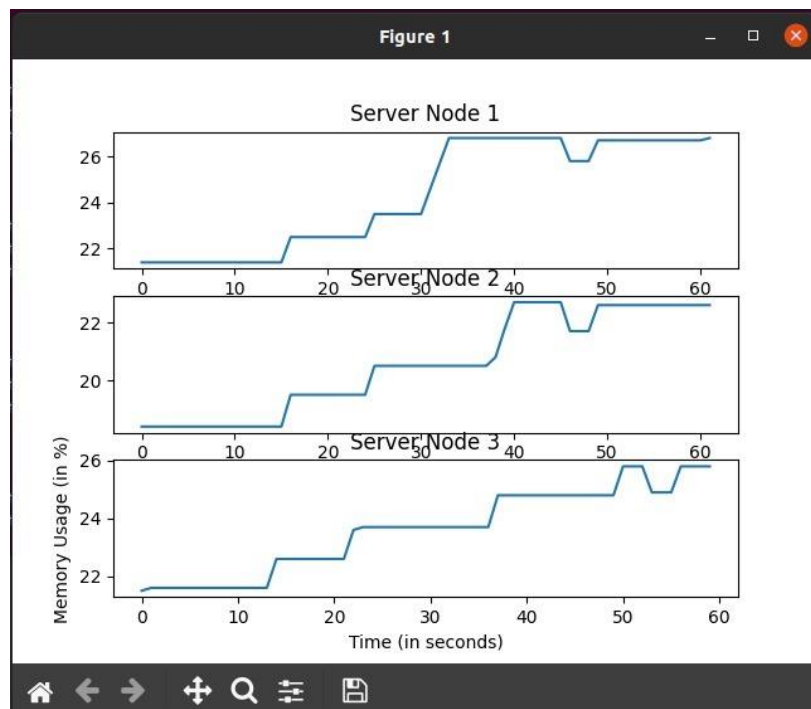
Dynamic Algorithms

Weighting Response Time



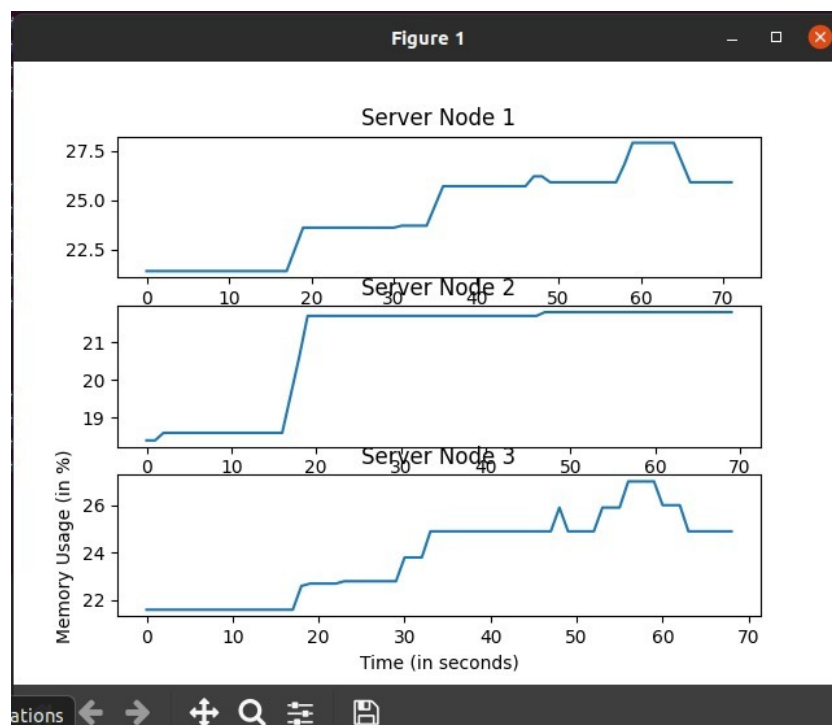
Here the requests are served by the servers which take the least time to execute.

Least Connection



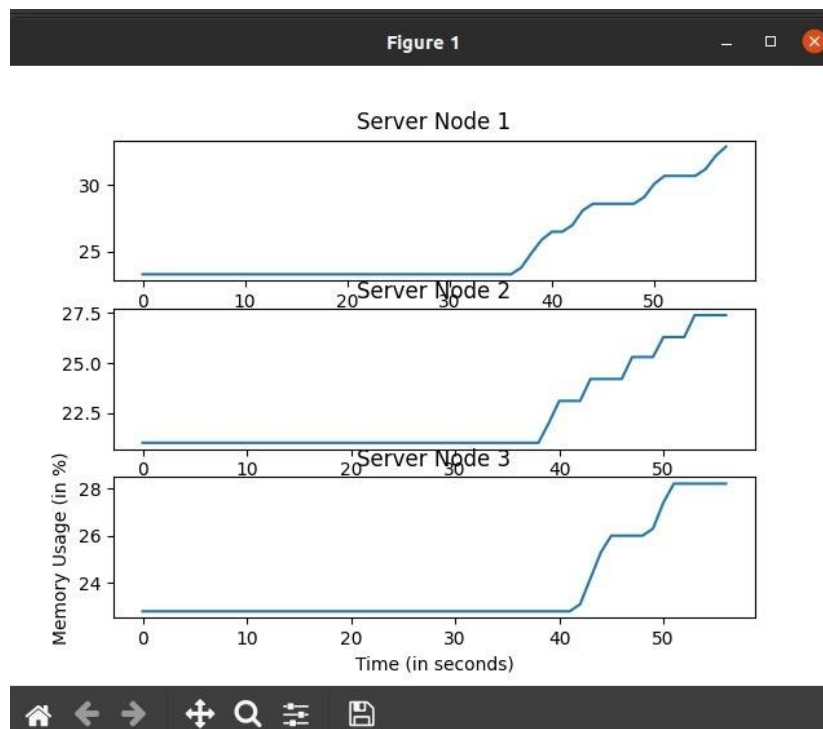
The slope of the graph where value rises is due to more number of disconnected connections while the decreasing slope means more active connections.

Weighted Least Connection



Similar to least connections with weight of server an added factor.

Nearest Neighbour



The graph with more iterations has most neighbour servers and vice versa.

Result and Conclusion:

S.No.	Factor	SLB Algorithm	DLB Algorithm
1	Nature	Work Load is assigned at compile time.	Work Load is assigned at run time.
2	Overhead Involved	Little overhead	Greater overhead
3	Resource Utilization	Lesser utilization	Greater Utilization
4	Predictability	Easy to predict	Difficult to predict
5	Adaptability	Less adaptive	More Adaptive
6	Reliability	Less	More
7	Response Time	Short	Longer
8	Stability	More	Less
9	Complexity	Less	More

After comparative study between both types of algorithms we come to the following conclusions:

1. Load balancing in distributed system is the most thrust area in research today as the demand of heterogeneous computing due to the wide use of internet.
2. More efficient load balancing algorithm more is the performance of the computing system.
3. There exists no absolutely perfect balancing algorithm but one can use depending on one's need.
4. Despite being more complex, dynamic load balancing algorithm is much better than static algorithm and hence is used in many areas in today's era.

References:

- [1] S. Begum and C. S. R. Prashanth, "Review of Load Balancing in Cloud Computing," *International Journal of Computer Science Issues*, vol. 10, no. 1, p. 343, 2013.
- [2] G. Xu, J. Pang, and X. Fu, "A load balancing model based on cloud partitioning for the public cloud," *Tsinghua Science and Technology*, vol. 18, no. 1, pp. 34–39, 2013.
- [3] Md. Firoj Ali and Rafiqul Zaman Khan. "The Study On Load Balancing Strategies In Distributed Computing System". *International SJournal of Computer Science & Engineering Survey (IJCSES)* Vol.3, No.2, April 2012
- [4] A. Khiyaita, M. Zbakh, and H. El Bakkali, "Load balancing cloud computing: state of art," in *Proceedings of the National Days of IEEE Network Security and Systems (JNS2 '12)*, pp. 106–109, Marrakech, Morocco, 2012.
- [5] P. Membrey, D. Hows, and E. Plugge, "Load Balancing in the Cloud," in *Practical Load Balancing*, pp. 211–224, Apress, 2012.
- [6] M. Randles, D. Lamb, and A. Taleb-Bendiab, "A comparative study into distributed load balancing algorithms for cloud computing," in *Proceedings of the 24th IEEE International Conference on Advanced Information Networking and Applications Workshops (WAINA '10)*, pp. 551–556, April 2010.