OPERATING SYSTEMS

F2 Slot - PROJECT COMPONENT

Improving Performance of a Web Server using Caching and Load Balancing Techniques

by Abhinav. S
17BCE0554

Submitted to -
Saleem Durai .M.A
Associate Professor
SCOPE

Abstract

Introduction:

Hosting a website requires consideration of several factors such as including dynamic addresses, bandwidth constraints, electricity costs and often requires comparing between various options available.

With a web server, developers can easily test their code without paying for a hosting service. Often, when a website is too big for a commercial web host or to be able to quickly make changes on your website, running a web server on one’s own PC is a feasible idea. Setting up a web server on a local computer(local host) is a practically possible solution, often overlooked.

Several options have come up to host a web server such as using an old laptop, android device or a raspberry pi. However to efficiently manage incoming network traffic across a group of back-end web servers(also called server pool/ server farm), load balancing is essential.

This project is about efficient ways to optimising the web server for static applications using caching and load balancing.

Web server requirements:

System specs:

1)At least 512MB of RAM
2)At least 1Ghz of processing power
3)At least 1GB of Hard disk space

4)Ethernet Card for Network Connection

Software specs:

For Windows based server: 1)Apache 2.4
2)MYSQL
3)PHP 5.x

(or)
For Linux based server: 1)NGINX

Hosting locally requires generally following steps:
1)An appropriate web server computer is chosen
2)A good web server program. Apache is used here.
3)Configure MySQL
4)Create a basic main page to test the server, make that file the web server's document path
5)Test the site

However, to have a load balancing web server, we can simply install NGINX in an ubuntu operating system, instead of the first three steps.

Then we create a NGINX configuration file, in which we can use default setting available to define number of worker processes, maximum number of simultaneous connections for each worker process and so on.

NGINX is a powerful web server and uses a non-threaded, event-driven architecture that enables it to outperform Apache if configured correctly. It can also do other important things, such as load balancing, HTTP caching, or be used as a reverse proxy.

As a software-based load balancer, NGINX Plus is much less expensive than hardware-based solutions with similar capabilities. The comprehensive load balancing capabilities in NGINX Plus enable you to build a highly optimized application delivery network.

When you insert NGINX Plus as a load balancer in front of your application and web server farms, it increases your website's efficiency, performance, and reliability.

Disadvantages of hosting locally:

1)Slow connections will be experiences compared to professional hosts. ISP upload speed is much slower than download speed, so serving content to website visitors will be slow too.

2)To deal with an ever-changing (dynamic) IP address. Though there are DNS configuration tools to help with this somewhat, this can potentially cause problems at any time

3)Costs a lot of electricity

4)One is responsible for one's own hardware and software maintenance and support

Ways to manage web traffic and tasks:

Adding a reverse proxy server to offload some of these tasks.

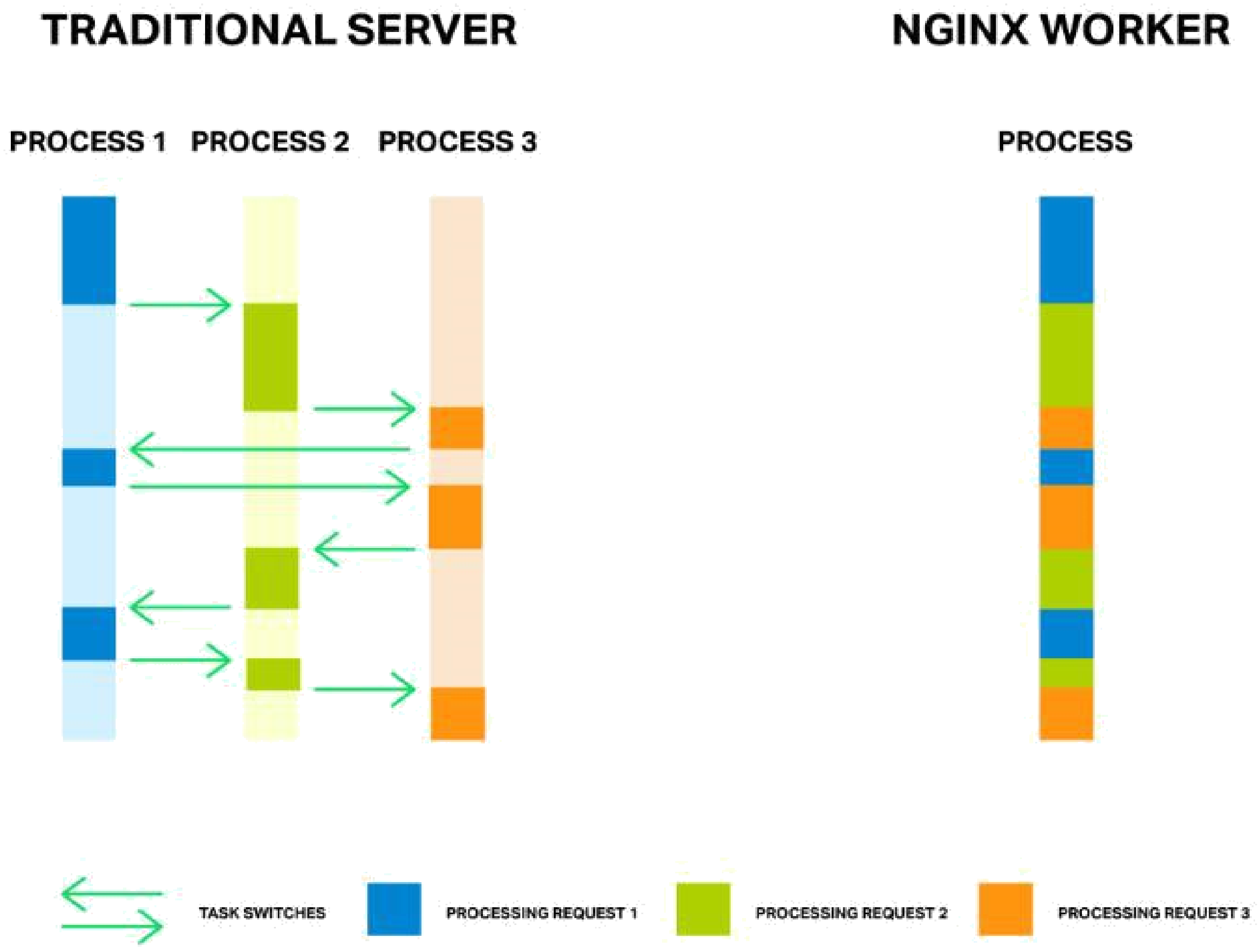
A reverse proxy server sits in front of the machine running the application and handles Internet traffic. Only the reverse proxy server is connected directly to the Internet; communication with the application servers is over a fast internal network.

Adding a reverse proxy server also adds flexibility to your web server setup. For instance, if a server of a given type is overloaded, another server of the same type can easily be added; if a server is down, it can easily be replaced.

Because of the flexibility it provides, a reverse proxy server is also a prerequisite for many other performance-boosting capabilities, such as:

•**Load balancing** – A load balancer runs on a reverse proxy server to share traffic evenly across a number of application servers. With a load balancer in place, you can add application servers without changing your application at all.
•**Caching static files**– Files that are requested directly, such as image files or code files, can be stored on the reverse proxy server and sent directly to the client, which serves assets more quickly and offloads the application server, allowing the application to run faster.

NGINX software is specifically designed for use as a reverse proxy server, with the additional capabilities described above. NGINX uses an event-driven processing approach which is more efficient than traditional servers. NGINX Plus adds more advanced reverse proxy features, such as application health checks, specialized request routing, advanced caching, and support.



Adding a load balancer is a relatively easy change which can create a dramatic improvement in the performance and security of your site. Instead of making a core web server bigger and more powerful, you use a load balancer to distribute traffic across a number of servers. Even if an application is poorly written, or has problems with scaling, a load balancer can improve the user experience without any other changes.

A load balancer is, first, a reverse proxy server– it receives Internet traffic and forwards requests to another server. The trick is that the load balancer supports two or more application servers, using a choice of algorithms to split requests between servers. The simplest load balancing approach is **round robin**, with each new request sent to the next server on the list.

Other methods include sending requests to the server with the fewest active connections. NGINX Plus has capabilities for continuing a given user session on the same server, which is called session persistence. Load balancers can lead to strong improvements in performance because they prevent one server from being overloaded while other servers wait for traffic.

Caching improves web application performance by delivering content to clients faster. Caching can involve several strategies: pre-processing content for fast delivery when needed, storing content on faster devices, storing content closer to the client, or a combination.
There are three main techniques for caching content generated by web applications:

•Moving content closer to users – Keeping a copy of content closer to the user reduces its transmission time.

•Moving content to faster machines – Content can be kept on a faster machine for faster retrieval.

•Moving content off of overused machines – Machines sometimes operate much slower than their benchmark performance on a particular task because they are busy with other tasks. Caching on a different machine improves performance for the cached resources and also for non-cached resources, because the host machine is less overloaded.

Load Balancing Algorithms:

Different load balancing algorithms provide different benefits; the choice of load balancing method depends on your needs:
•Round Robin – Requests are distributed across the group of servers sequentially.
•Least Connections – A new request is sent to the server with the fewest current connections to clients. The relative computing capacity of each server is factored into determining which one has the least connections.
•IP Hash – The IP address of the client is used to determine which server receives the request.
NGINX also facilitates the concept of ‘weights’, to give priority to a web server based on individual server capabilities.

Idea proposal:

A NGINX server with load balancing can dramatically improve load time of the web application and manage resources efficiently to fulfil server requests. Caching helps improve load time of static files when not accessed for the first time. It can satisfy server requests even when the server is down based on data saved in the cache from the first time the file was accessed.

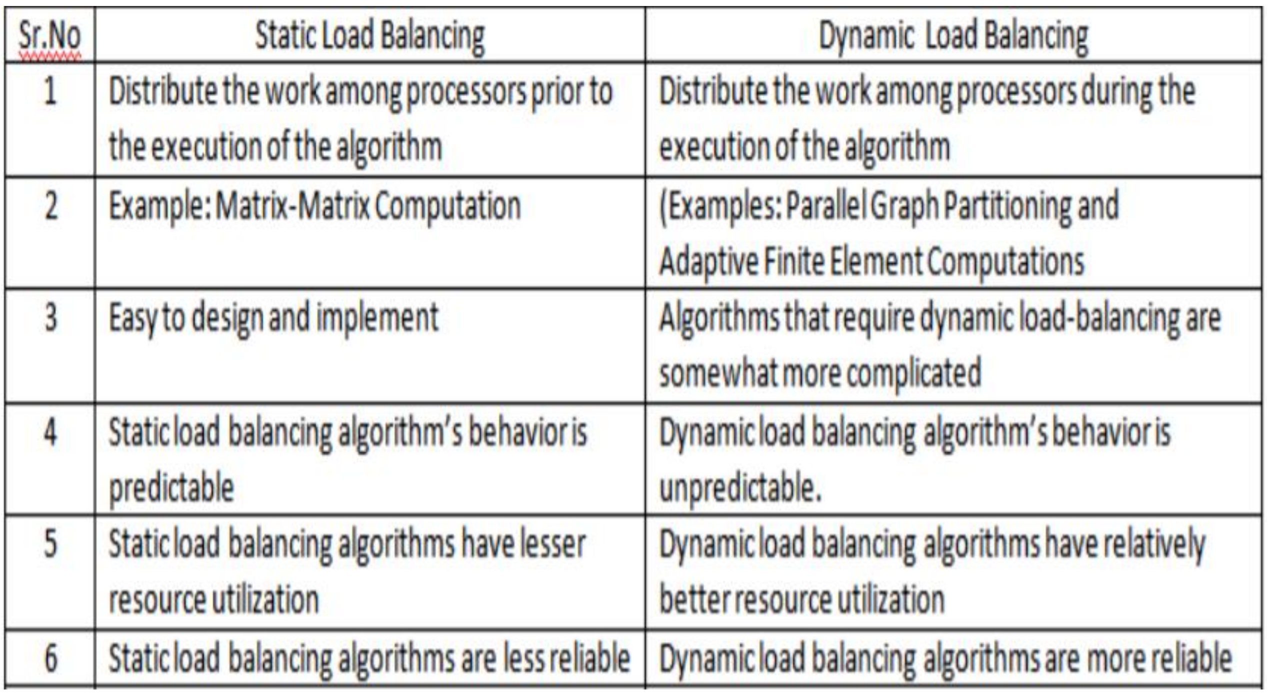
The content cache sits between a client and the 'origin server' and saves all the content it sees. If client requests content which is available in the content cache, it directly accesses data from the cache rather than send the request to the origin server.

There could potentially be several caches such as the client's browser cache, intermediary caches, content delivery network and load balancer's; between the web browser and application server.

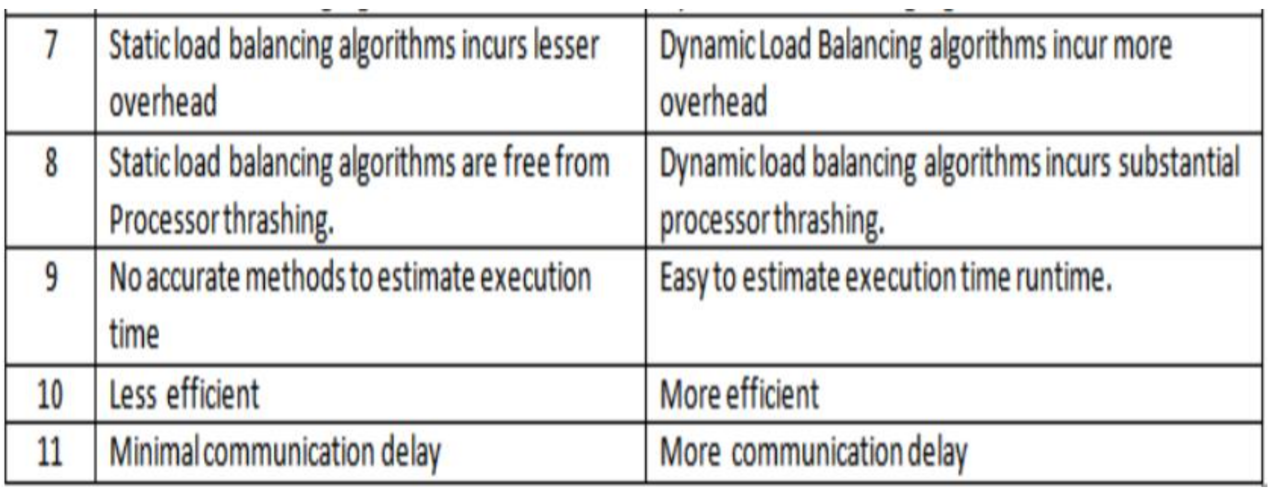
Only two directives are required to set up basic caching: proxy_cache_path and proxy_cache. The proxy_cache_path sets the path and configuration of the cache and proxy_cache directive activates it.
A powerful feature of NGINX content caching is that NGINX can be configured to deliver stale content from its cache when it can't get fresh content from the origin servers. This can happen if all the origin servers for a cached resource are down or temporarily busy. Rather than relay the error to the client, NGINX delivers the stale version of the file from its cache. This provides an extra level of fault tolerance for the servers that NGINX is proxying, and ensures uptime in the case of server failures or traffic spikes. To enable this functionality, include the proxy_cache_use_stale directive. It helps identify an error timeout and delivers stale version of cache.

Load balancing algorithms can be classified into two types, static and dynamic, and the appropriate algorithm of either type can be chosen based on requirements.

A comparison between static and dynamic load balancing



Sr.No	Static Load Balancing	Dynamic Load Balancing
1	Distribute the work among processors prior to the execution of the algorithm	Distribute the work among processors during the execution of the algorithm
2	Example: Matrix-Matrix Computation	(Examples: Parallel Graph Partitioning and Adaptive Finite Element Computations
3	Easy to design and implement	Algorithms that require dynamic load-balancing are somewhat more complicated
4	Static load balancing algorithm's behavior is predictable	Dynamic load balancing algorithm's behavior is unpredictable.
5	Static load balancing algorithms have lesser resource utilization	Dynamic load balancing algorithms have relatively better resource utilization
6	Static load balancing algorithms are less reliable	Dynamic load balancing algorithms are more reliable



7	Static load balancing algorithms incurs lesser overhead	Dynamic Load Balancing algorithms incur more overhead
8	Static load balancing algorithms are free from Processor thrashing.	Dynamic load balancing algorithms incurs substantial processor thrashing.
9	No accurate methods to estimate execution time	Easy to estimate execution time runtime.
10	Less efficient	More efficient
11	Minimal communication delay	More communication delay

Weighted least connection is one of the most efficient static algorithms since it takes into account two major factors- info about server capabilities and connection status. It has a dynamic approach towards managing server traffic. So, this algorithm has been chosen for utilising along server capabilities. Load balancing along with caching can help access files quickly even when a server is down or taking time to load by simply accessing it from the browser cache and content cache. This makes it more efficient than a simple weighted least connection algorithm.

Health checks are important when managing traffic. When a particular server is down, the load balancer should manage traffic such that requests are managed seamlessly.

The directories used for this purpose are max_fails and fail_timeout which are used to mark number of failures allowed and time allowed until the server is considered to be down.

Gzip compression is an option available with the NGINX server which, with the help of compression, will be sending smaller objects to clients, thereby the pages will be faster to load.

Unfortunately, compressing every object can take significant CPU resources from your server hardware. That, in turn, will cause a slowdown in NGINX, thus rendering the configuration null. So it is important to compress optimally.

Instead of compressing every object, configure NGINX to only compress large files and avoid the temptation to compress smaller files (such as images, executables, etc.).

- gzip on; - enables gzip compression
- gzip_vary on: - tells proxies to cache both gzipped and regular versions of a resource
- gzip_min_length 1024; - informs NGINX to not compress anything smaller than the defined size
- gzip_proxied - compress data even for clients that are connecting via proxies (here we're enabling compression if: a response header includes the "expired", "no-cache", "no-store", "private", and "Authorization" parameters)
- gzip_types - enables the types of files that can be compressed

Future scope:

An improved WLC algorithm that maintains the load among various servers by preventing requests being transmitted to the new real server can be implemented. When web requests are continuously assigned to only a new real server more than the maximum continuous allocation number(C), the proposed algorithm excludes the new real server from activated real server scheduling list and deactivates it. . Finally after C-1 allocation round times, the new real server is activated and included into the server scheduling list. The proposed algorithm balances the load among real servers by avoiding overloads of the new real server. This algorithm allows us to efficiently utilise a server added during a process such that the new server doesn't get overloaded. Implementing this algorithm will assure maximum results.

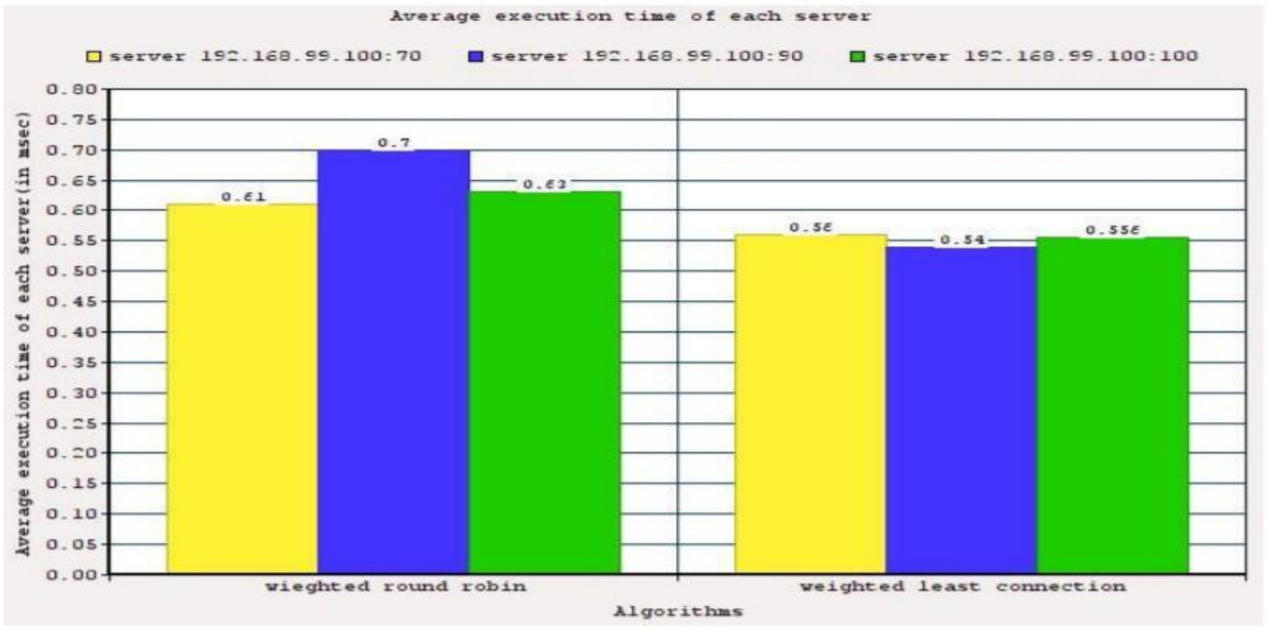


Fig. Average execution time of each server(in msec)

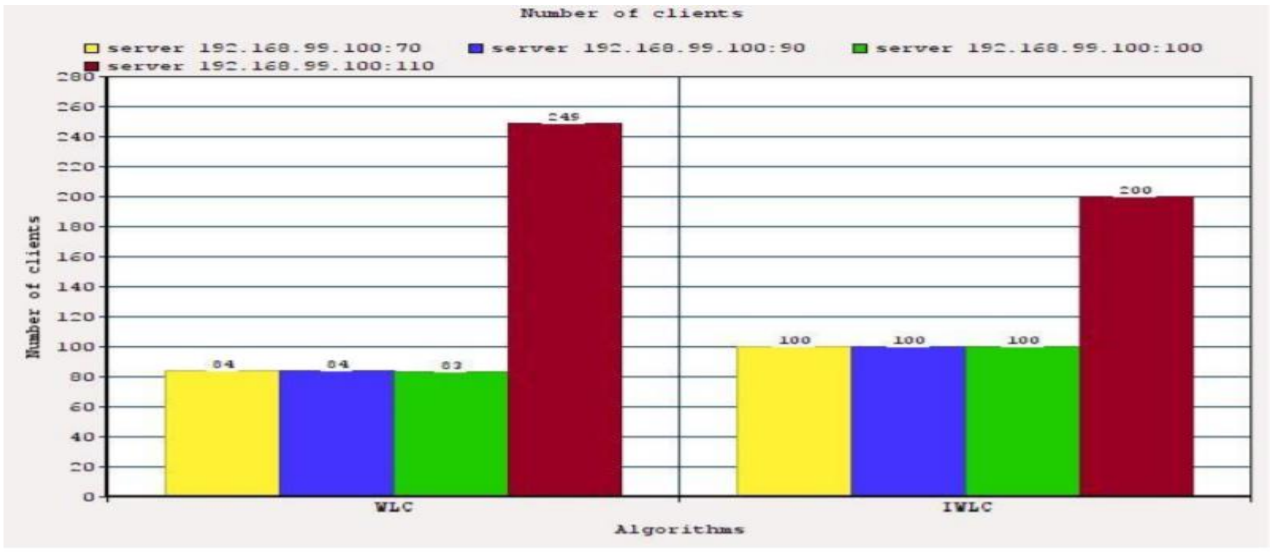
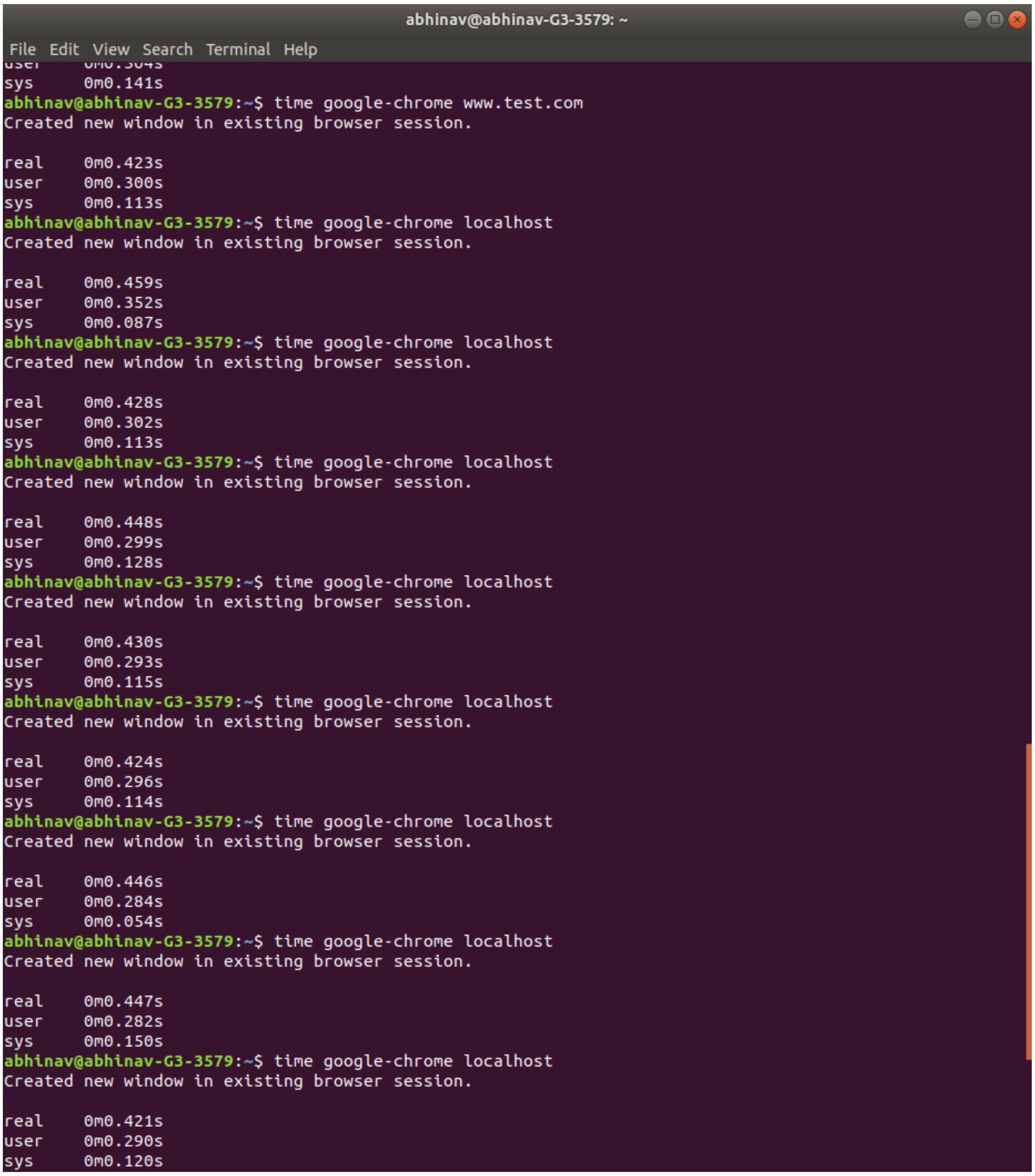
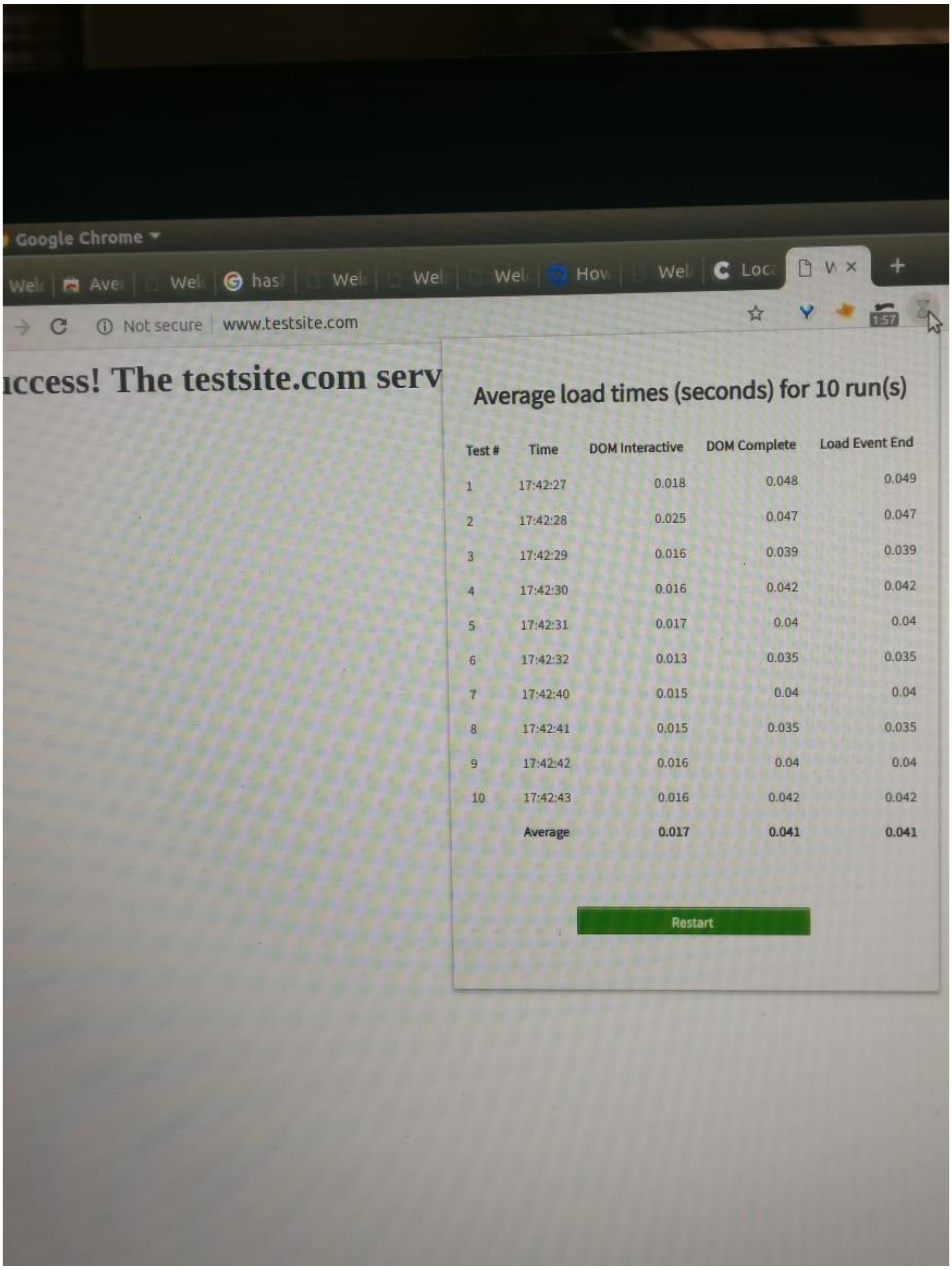


Fig. Number of client requests assigned

Results:


abhinav@abhinav-G3-3579: ~
File Edit View Search Terminal Help

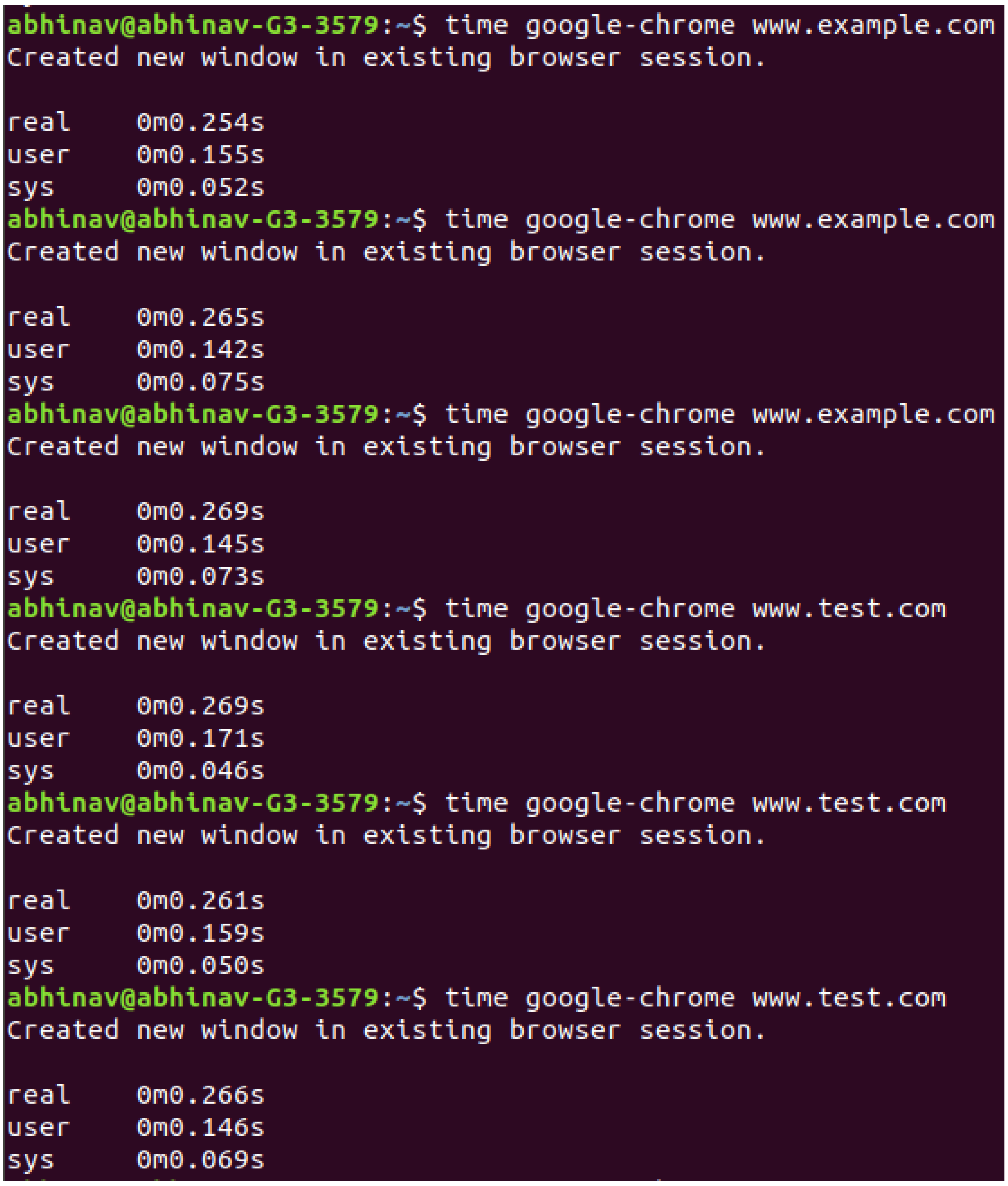
```
user    0m0.304s
sys     0m0.141s
abhinav@abhinav-G3-3579:~$ time google-chrome www.test.com
Created new window in existing browser session.

real    0m0.423s
user    0m0.300s
sys     0m0.113s
abhinav@abhinav-G3-3579:~$ time google-chrome localhost
Created new window in existing browser session.

real    0m0.459s
user    0m0.352s
sys     0m0.087s
abhinav@abhinav-G3-3579:~$ time google-chrome localhost
Created new window in existing browser session.

real    0m0.428s
user    0m0.302s
sys     0m0.113s
abhinav@abhinav-G3-3579:~$ time google-chrome localhost
Created new window in existing browser session.

real    0m0.448s
user    0m0.299s
sys     0m0.128s
abhinav@abhinav-G3-3579:~$ time google-chrome localhost
Created new window in existing browser session.

real    0m0.430s
user    0m0.293s
sys     0m0.115s
abhinav@abhinav-G3-3579:~$ time google-chrome localhost
Created new window in existing browser session.

real    0m0.424s
user    0m0.296s
sys     0m0.114s
abhinav@abhinav-G3-3579:~$ time google-chrome localhost
Created new window in existing browser session.

real    0m0.446s
user    0m0.284s
sys     0m0.054s
abhinav@abhinav-G3-3579:~$ time google-chrome localhost
Created new window in existing browser session.

real    0m0.447s
user    0m0.282s
sys     0m0.150s
abhinav@abhinav-G3-3579:~$ time google-chrome localhost
Created new window in existing browser session.

real    0m0.421s
user    0m0.290s
sys     0m0.120s
```



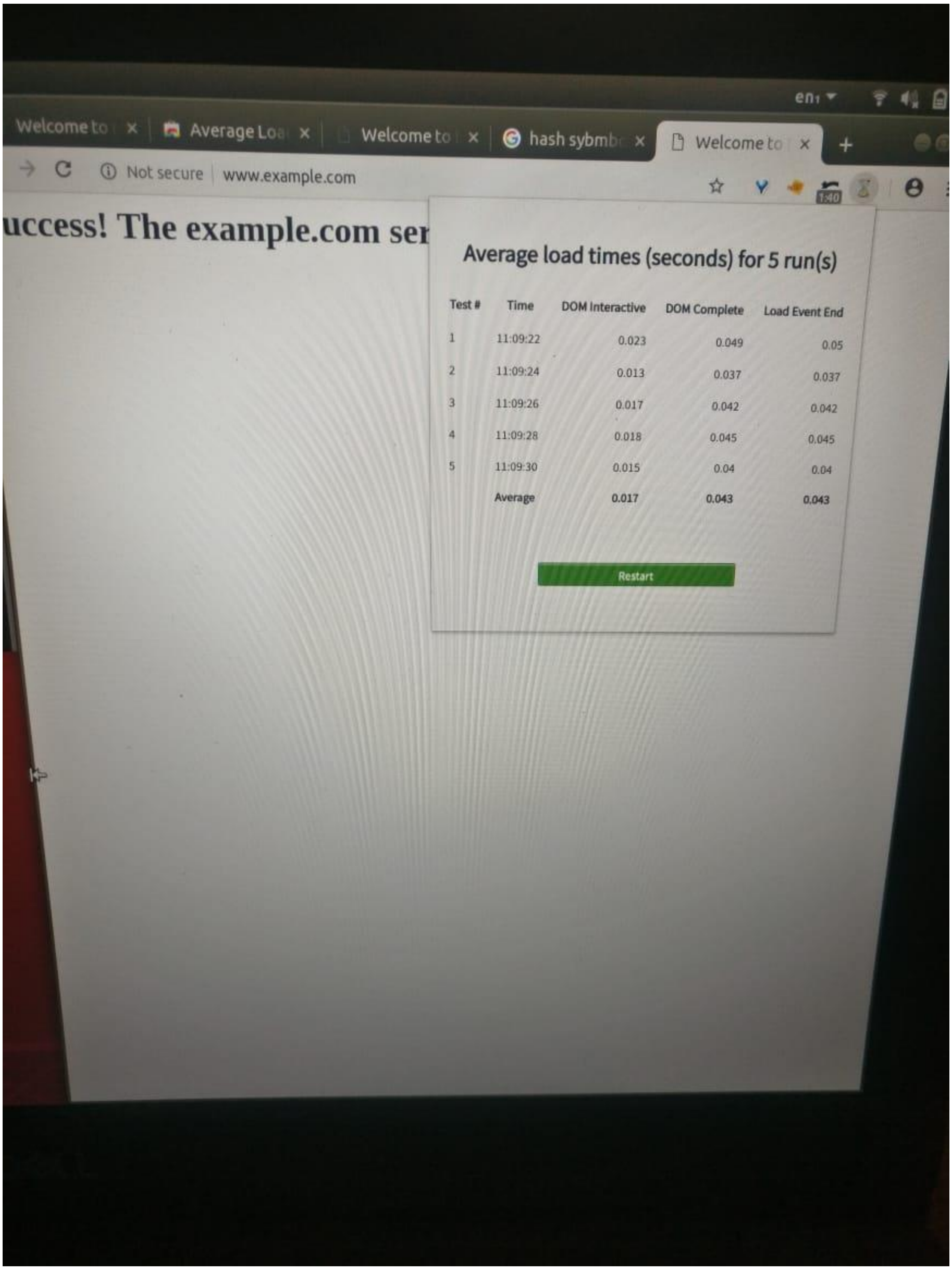
ıccess! The testsite.com serv

Average load times (seconds) for 10 run(s)

Test #	Time	DOM Interactive	DOM Complete	Load Event End
1	17:42:27	0.018	0.048	0.049
2	17:42:28	0.025	0.047	0.047
3	17:42:29	0.016	0.039	0.039
4	17:42:30	0.016	0.042	0.042
5	17:42:31	0.017	0.04	0.04
6	17:42:32	0.013	0.035	0.035
7	17:42:40	0.015	0.04	0.04
8	17:42:41	0.015	0.035	0.035
9	17:42:42	0.016	0.04	0.04
10	17:42:43	0.016	0.042	0.042
	Average	**0.017**	**0.041**	**0.041**



```
abhinav@abhinav-G3-3579:~$ time google-chrome www.example.com
Created new window in existing browser session.

real	0m0.254s
user	0m0.155s
sys	0m0.052s
abhinav@abhinav-G3-3579:~$ time google-chrome www.example.com
Created new window in existing browser session.

real	0m0.265s
user	0m0.142s
sys	0m0.075s
abhinav@abhinav-G3-3579:~$ time google-chrome www.example.com
Created new window in existing browser session.

real	0m0.269s
user	0m0.145s
sys	0m0.073s
abhinav@abhinav-G3-3579:~$ time google-chrome www.test.com
Created new window in existing browser session.

real	0m0.269s
user	0m0.171s
sys	0m0.046s
abhinav@abhinav-G3-3579:~$ time google-chrome www.test.com
Created new window in existing browser session.

real	0m0.261s
user	0m0.159s
sys	0m0.050s
abhinav@abhinav-G3-3579:~$ time google-chrome www.test.com
Created new window in existing browser session.

real	0m0.266s
user	0m0.146s
sys	0m0.069s
```



uccess! The example.com ser

Average load times (seconds) for 5 run(s)

Test #	Time	DOM Interactive	DOM Complete	Load Event End
1	11:09:22	0.023	0.049	0.05
2	11:09:24	0.013	0.037	0.037
3	11:09:26	0.017	0.042	0.042
4	11:09:28	0.018	0.045	0.045
5	11:09:30	0.015	0.04	0.04
	Average	**0.017**	**0.043**	**0.043**

References:

https://chubbable.com/how-to-setup-a-web-server-on-ubuntu-or-windows#web-server-setup-on-remote-or-local-server

https://medium.com/@jgefroh/a-guide-to-using-nginx-for-static-websites-d96a9d034940

https://www.nginx.com/blog/nginx-caching-guide/

https://www.digitalocean.com/community/tutorials/how-to-install-nginx-on-ubuntu-16-04

http://nginx.org/en/docs/http/load_balancing.html#nginx_load_balancing_with_least_connected

https://www.techrepublic.com/article/how-to-configure-gzip-compression-with-nginx/

https://chubbable.com/how-to-setup-a-web-server-on-ubuntu-or-windows#web-server-setup-on-remote-or-local-server

COMPARISON OF STATIC AND DYNAMIC LOAD BALANCING IN GRID COMPUTING Mr. Ramesh Prajapati1 , Mr. Dushyantsinh Rathod2 , Dr. Samrat Khanna3 1,2Dept. of Computer Science Engineering, Rai University, Center for Research & Development Saroda, Dholka, India 3Dept. of Information Technology, Istar, Sardar Patel centre for science & Tech. V.Vnagar, India

An Improved Weighted Least Connection Scheduling Algorithm for Load Balancing in Web Cluster Systems Gurasis Singh1, Kamalpreet Kaur2 1Assistant professor,Department of Computer Science, Guru Nanak Dev University College, Jalandhar ,India 2 Lecturer, Department of Computer Science and Engineering, Govt. Polytechnic College For Girls, Jalandhar, India