A Distributed Proxy Server

CS8803: Graduate Introduction to Operating Systems

Omkar Bellare GTID#902835686

Vishnu Akhilesh Venkataraman GTID#902832656

Overview

The proxy server that we built checks if the file being requested is of jpg type and if it is, then it sends the complete url of the jpg file to the RPC server which downloads the file, compresses the image and sends back the compressed image to forward it back to the client. We tested our code on the www.ece.gatech.edu page and the screenshots clearly show the original page on the top and the page with the compressed images on the bottom.

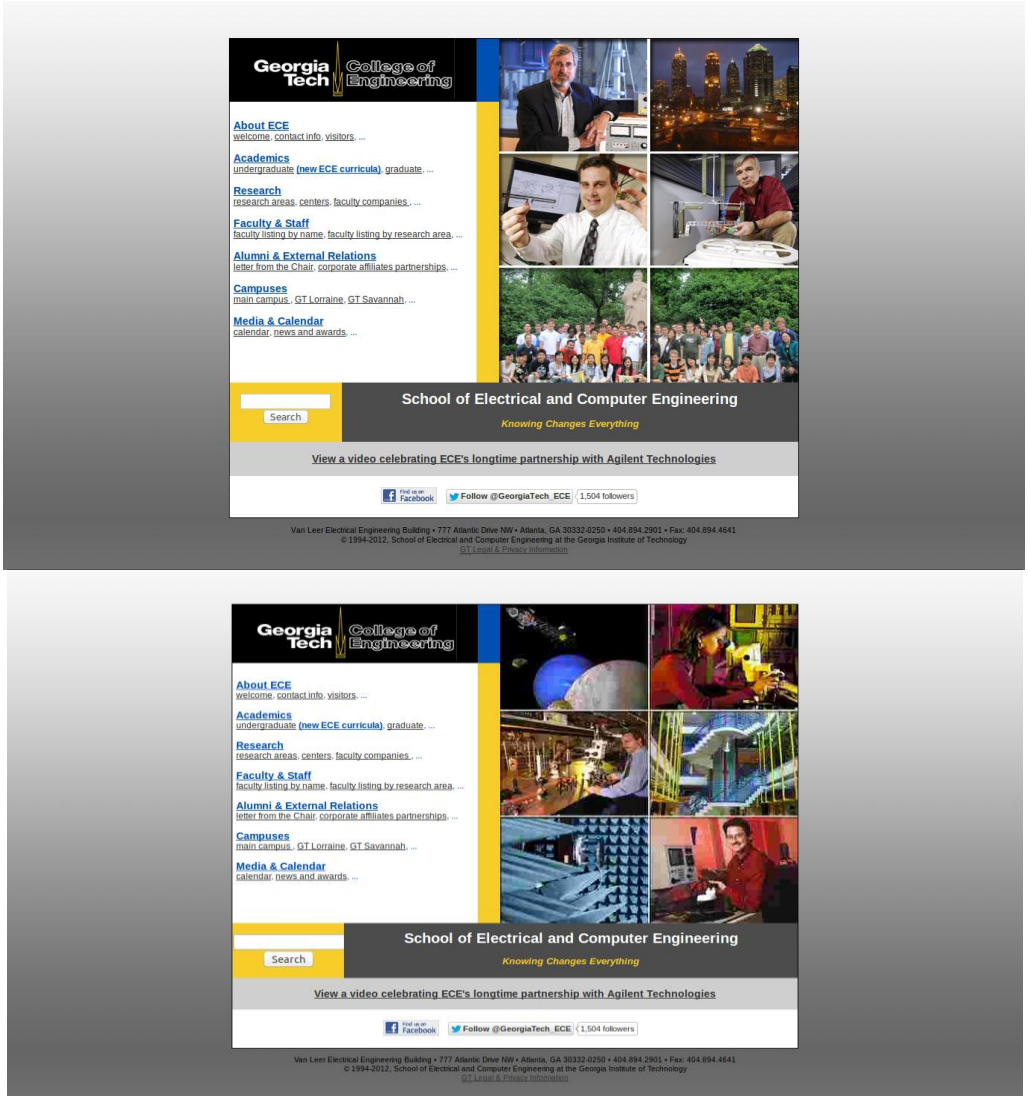

Fig 1: Top image is original page, bottom is page after compression

To test the memory usage of the proxy RPC servers, we queried for an image that was of size 1.7MB whose compressed image size was 68KB using 20 client threads each making 5 requests. Simultaneously we ran top to find the memory usage that our server was using and below are the screenshots for the experiment.

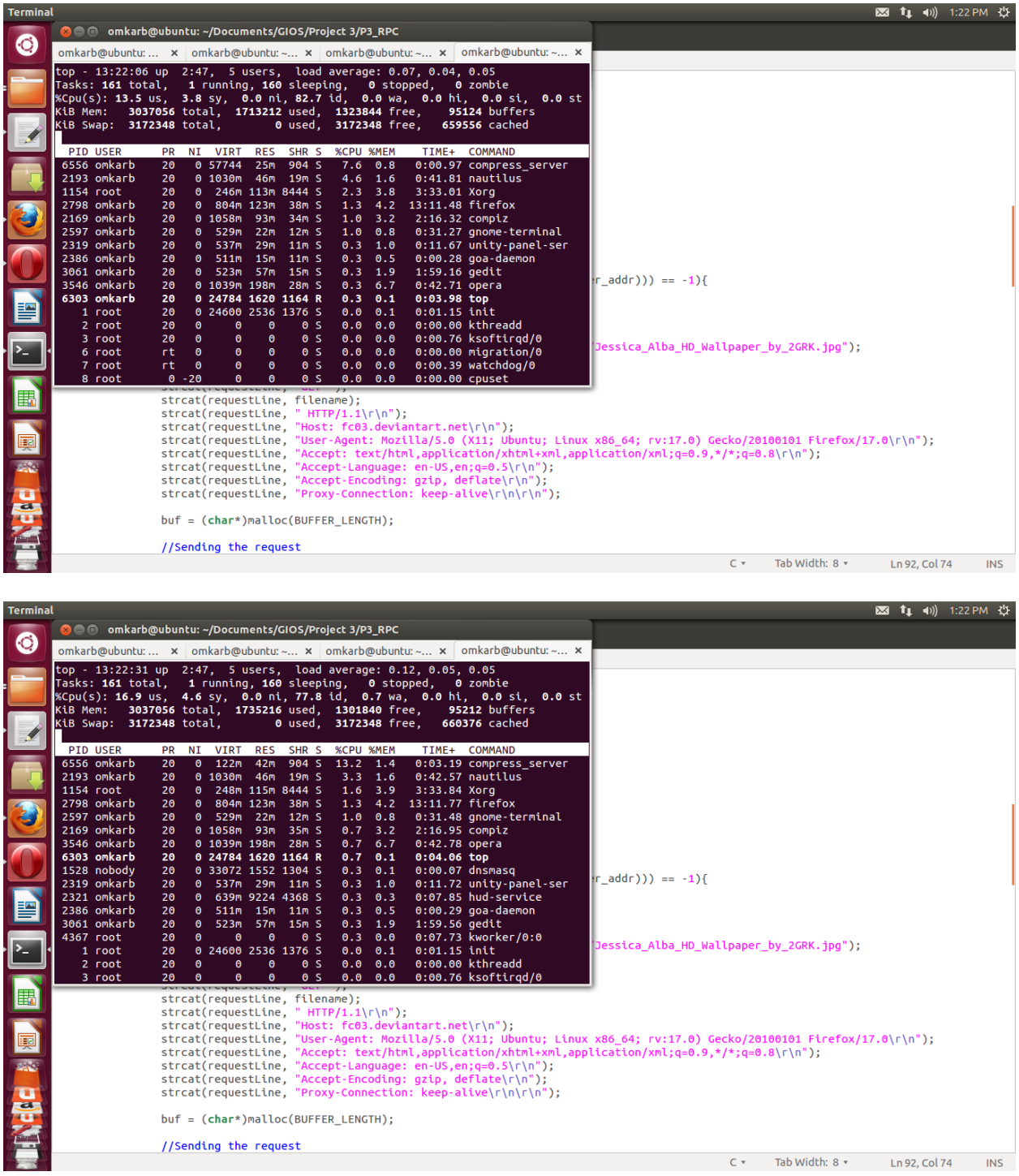

Fig 2: Top image was taken before bottom image (observe compress_server)

From the above screenshots, we clearly observe that the compress_server is using a large amount of memory which it does not free as time progresses. The memory will be freed automatically but only after some time. The previous experiment took the memory readings when the files were still being requested for. In the next experiment we tried requesting for just one large file, keep the compress_server reading and check for memory usage after probably 4 seconds after the file was sent to the client. This time we observed that the memory goes down soon after the processing is done, meaning Sun RPC probably has a memory cleanup thread which runs when the server is not doing anything else. Screenshots are again shown below:

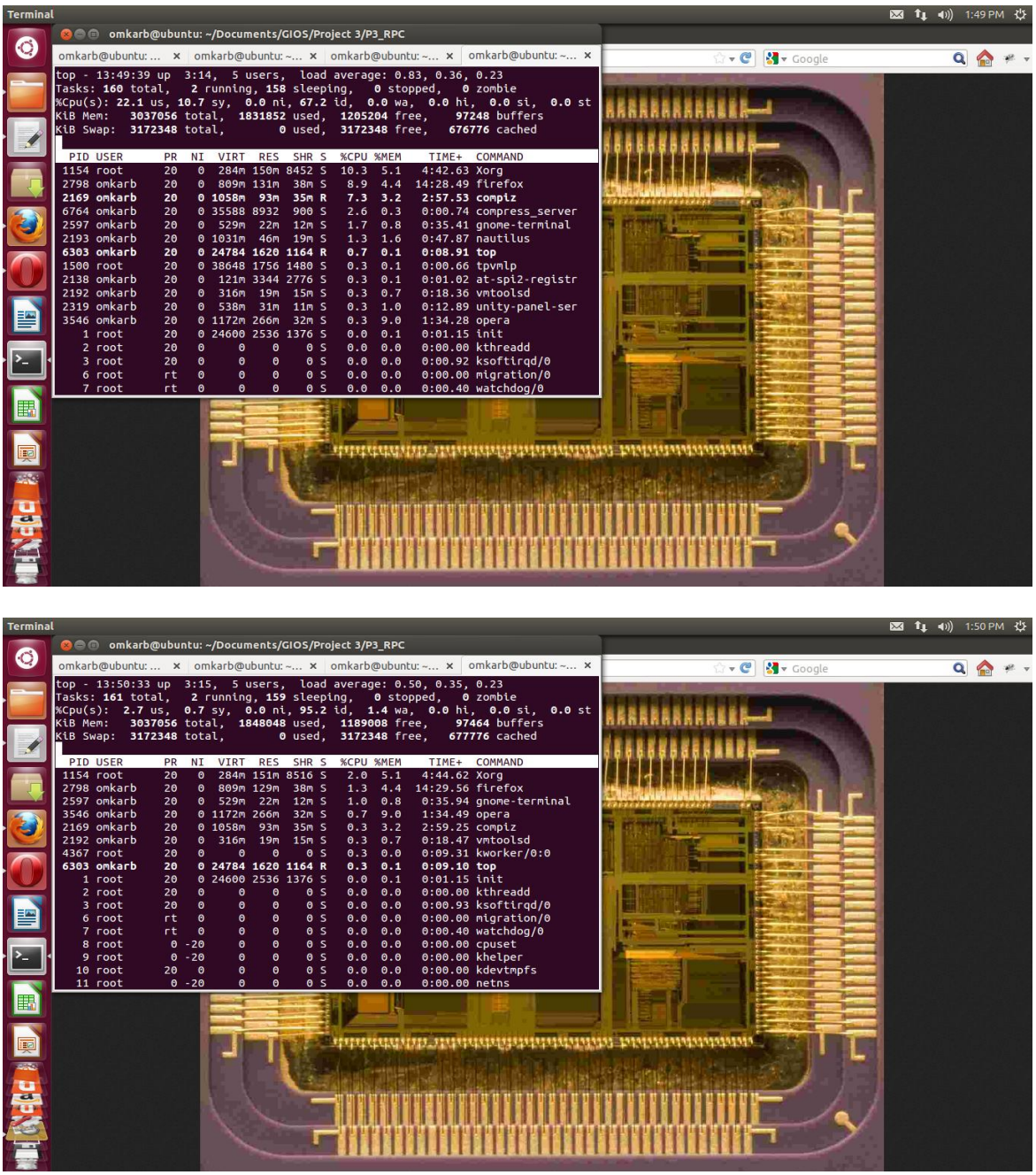

Fig 3: Top image shown while RPC server was working on compression, bottom shows 4 seconds later

Experiment 2: Testing the impact of varying the image file size

Setup: 20 Client Threads with each making 5 requests.

Image Size (KB)	Number of bytes after compression (B)	Time (sec)	Throughput (Mb/s)	Requests per sec
10	360000	27.292	0.013191	3.664
20	921500	77.799	0.011844	1.285
64	1087400	116.667	0.009320	0.857
108	1020500	129.567	0.007876	0.772
151	786752	150.526	0.005226	0.664

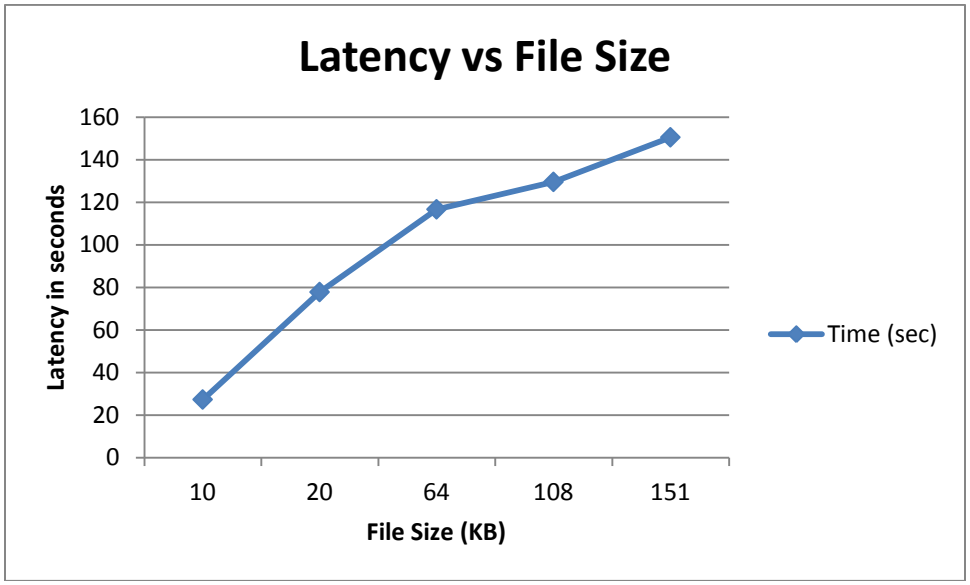

Fig 4: Latency vs File size

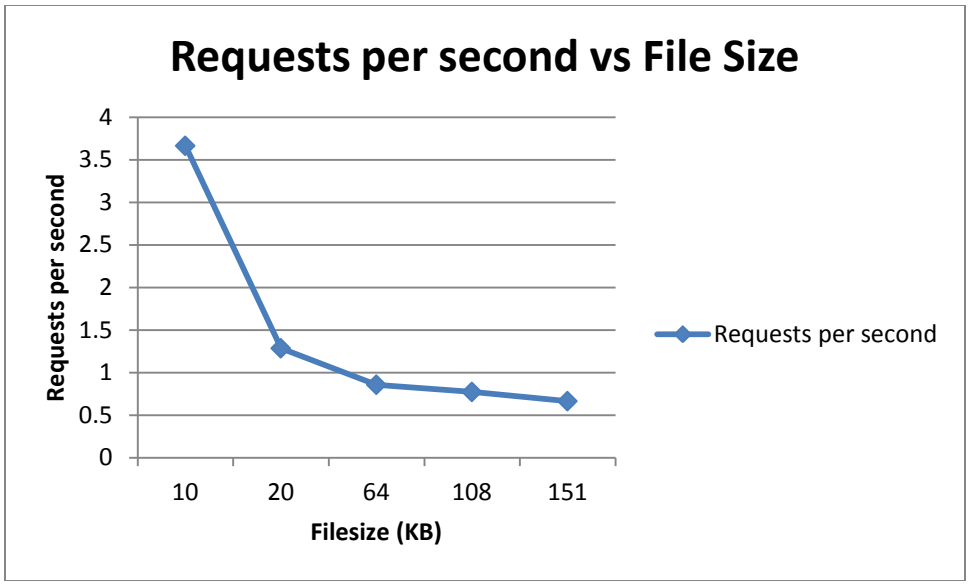

Fig 5: Requests per second vs Filesize

Conclusion

As we can see there is almost an exponential drop in the number of requests that can be handled by the RPC server per second. This is because the server is currently single threaded and processes requests sequentially. We can also see an almost linear trend in the throughput of the proxy. Also we think if we increase the number of RPC servers the time to process will be lower, and should ideally be half but will be a little more due to the RPC overhead. Thus the use of RPC for distributing the load on proxy shows that the bottleneck is not the network but the CPU.