UNIVERSITY OF TROMSØ

INF-3203: A Load Balancing Simulation

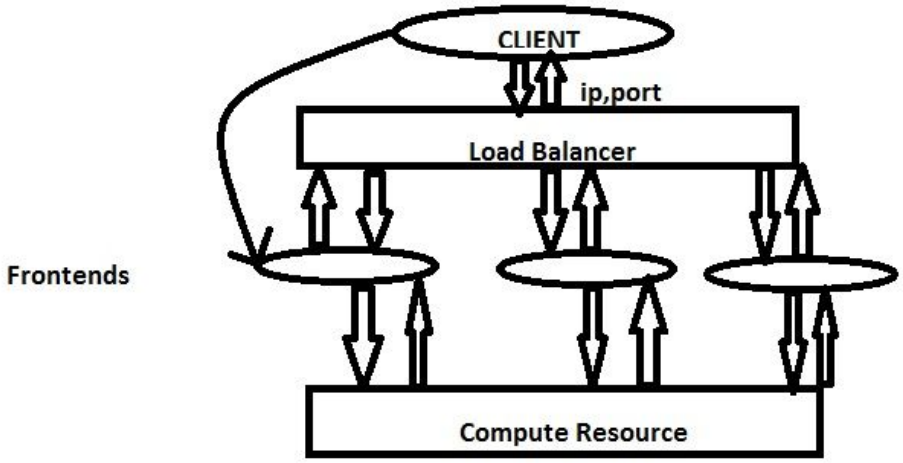
Nishant Verma

1 Introduction

In this assignment we are asked to design and implement a simple internet service.We will explore a simple load balancer design known as feedback mechanism .We will also explore the idea of managing timeouts in such system.

2 Design

The design is already given in the assignment requirements.The chosen load balancer design results in client making request to Frontend.For our system to be asynchronous ,the communication between client and frontend should be asynchronous (given the communication



between frontend and compute resource is synchronous). This will require our client to act as a server .This means client will be tightly coupled to frontend .Assuming the client can be anything (to keep our client decoupled from the overall system),I chose against the asynchronous design .We will see the impact of this design in discussion/conclusion.

2.1 Load Balancer Design

Load balancer respond back with an ip and port of one of the frontends to the client .It uses a feedback mechanism to decide the frontend which will be responsible for handling the request .It checks with the frontend for current load .Frontend responds back with 500 response code for more than 7 request .In this case load balancer respond back with ip of the next frontend from the list of frontends (also in case if the frontend did not respond).Client then request to frontend for compute resource.It can be noted that ,Idea is similar to DNS lookup.

2.2 FrontEnd Design

Front end has two different server running on different ports. The first server listens for feedback requests.It reponds with 200 response code(it is available) if the number of requests is less than equal to seven, 500 otherwise. The second server processes request from clients by requesting the resource from compute resource. This request can be time intensive, so we have added a timeout adjustment feature .It starts with a very low timeout and increases by 100ms for every timeout and decreases by 10ms for every successful response.

3 Evaluation

For a single node, we did not see any timeout at all with multithreaded compute resource.However an expected decrease in performance can be noted as we increase the number of nodes.The program scales well enough to handle 21 nodes but we start to see frequent timeouts and delayed frontend response. The program does not scale well in case of single threaded compute resource and we start seeing lots of timeouts even with 5 nodes. This can be explained due to synchronous design of the system.

4 Discussion (Lessons Learned)

As previously suggested ,an asynchronous design will improve the overall performance of the system .But this comes with a higher cost where our client is tightly coupled with frontend.To handle this problem ,we have to propose a better load balancer design .The proposed load balancer will provide a pure abstraction of the underlying implementation so that the client can

request computation as long as it knows ip and port number of the load balancer. Our load balancer and frontend can communicate asynchronously. Each request can be assigned a unique request id and load balancer will close the connection after sending the request. Front End will respond back with the compute resource and load balancer can identify this request with the request id and Henceforth notifying the client with the compute resource. We should be able to see a considerable performance gain as we are no longer waiting for responses anymore. We can suggest further improvements in design of the system .The system design is highly multithreaded .We have not considered concurrency issues that it faces .However the request counter is changed in a thread safe environment using locking.We can also scale the system for more number of nodes by increasing number of frontends .I was able to see better response to client when the number of frontends was increased.Initially a bug in the system caused only one frontend to respond. However the increase in performance be limited as computing resource is the main performance bottleneck. A further improvement in the system can be making fault tolerant .This will require a little tweak in design though .If one of the front end goes down,requests can be forwarded to the available frontends.But this will mean ,we are able to handle only 14 requests at a time (the tweak in design).

Being a noob in both python and bash, it was really great learning experience for me which I enjoyed a lot.I also put a question on stackoverflow after being stuck for hours on a simple bash if condition (link in references) .Learning to redirect outputs of program to file or screen so that I can debug my program was crucial when the display wall was not up .I learned a bit of pygame as well like everyone else did in the class.

4 Conclusion

We have presented a simple design and implementation of a simple internet system .The system is not able to scale to a single threaded compute resource and the probable reason is asynchronous design of the system.Asynchronous design was the result of choosing generally a good load balancer design which made overall design of the system suffer from scalability issues.

References

http://stackoverflow.com/questions/36940851/syntax-error-near-unexpected-token-fi