OpenCloud Distributed IAAS

Saptarsi Saha, *M.E in Computer Science & Information Systems* and Ayushi Sharma, *M.E in Computer Science & Information Systems*

***Abstract*—The next time you step away from your desk for a quick latte at your local coffee bar, your computer can get to work. It's possible when you volunteer your PC or laptop's unused time to OpenCloud. Grid computing joins together thousands of individual computers, establishing a large system with massive computational power equal to a supercomputer. This supercomputer can then be provided to solve world's most challenging problems i.e Weather Forecast, Genome Computations for Cancer Research, Drug Development Providing education etc. The use-case scenarios are immense for this type of product. It will ideally be a Cloud by the people, for the people, of the people with distributed governance providing infrastructure as a service to people when the need arises. The model we have discussed in this paper will also help solve the problem of minuscule number of edge locations around the world. Every city/town/locality can have a 'OpenCloud' where the unused computing resources can be shared equally amongst the people in need providing ubiquitous access of computing to the people.**

***Index Terms*—cloud, computing, edge computing, kubernetes, docker, distributed computing, IAAS**

I. Introduction

What is a Cloud? It's a way of thinking which defines a set of computing nodes publicly accessible to everyone so that the people in need of it can access and use it. It abstracts away all the infrastructure, networking components to provide a clean user-interface to users so that they can just focus on the work in hand. The cloud is a great distributor of resources too, it works on its own infrastructure farm providing ubiquitous access to its resources to anyone around the world. Infrastructure as a Service (IAAS) is probably the most important model which is in use in cloud computing. It defines the customers requirements and provide them with the exact amount of resources they want. It is flexible too with the provision of scaling up and down when the need arises.

So why OpenCloud and what is it exactly? OpenCloud as defined in this paper is a way of thinking which takes the cloud computing a step further. It defines a set of two people, Providers & Consumers. Providers are the people with great equipment and hardware lying around in their home and consumers are people who want to access the unused computing power that the providers want to provide. An important note in this part would be that Providers don't use their computers 100% all the time. The idle-time and the under-utilized resources of these computers can be a great way to provide access to resources to other people who need it without hampering the work done by the Provider's computers.[1]

What exactly is an Edge Computing and how is it relevant in this scenario? Edge is defined as the place where end-devices access the rest of the network. The edge used to be a place where devices used to connect to download and upload software and data to the cloud. With the explosion of IOT these models have shortcomings. The IOT devices collect so much more data that it requires more connections to the data-centers and the cloud. The nature of the works performed by this IOT devices are also creating a problem since they need access to real-time monitoring and the speed of the connection between these devices and the cloud has to be very fast for efficient real-time decision-making. AWS, one of the top providers of the Cloud-Computing environment only has 5 edge locations in India (Bangalore, Chennai, Delhi, Hyderabad, and Mumbai). This creates a problem

Saptarsi Saha, Ayushi Sharma are with Computer Science & Information System, Birla Institute of Technology and Sciences, Hyderabad, Telengana, India, e-mail: h20201030109@hyderabad.bits-pilani.ac.in, e-mail: h20201030157@hyderabad.bits-pilani.ac.in

for computing nodes where the latency has to be minimized. OpenCloud to the rescue : In our model we tie-up with the local ISP's to make an efficient Cloud Network in our own town/locality/city. This removes the unnecessary latency issues, and we are left with a publicly accessible cloud where the providers provide ubiquitous access of their unused resources which the consumers(IOT devices) can work on and the latency can be reduced by a large factor since both the Provider and Consumers are located in the same locality only.

II. Model Definition

The design of the model will be more clear by going through the components of the figure 1.

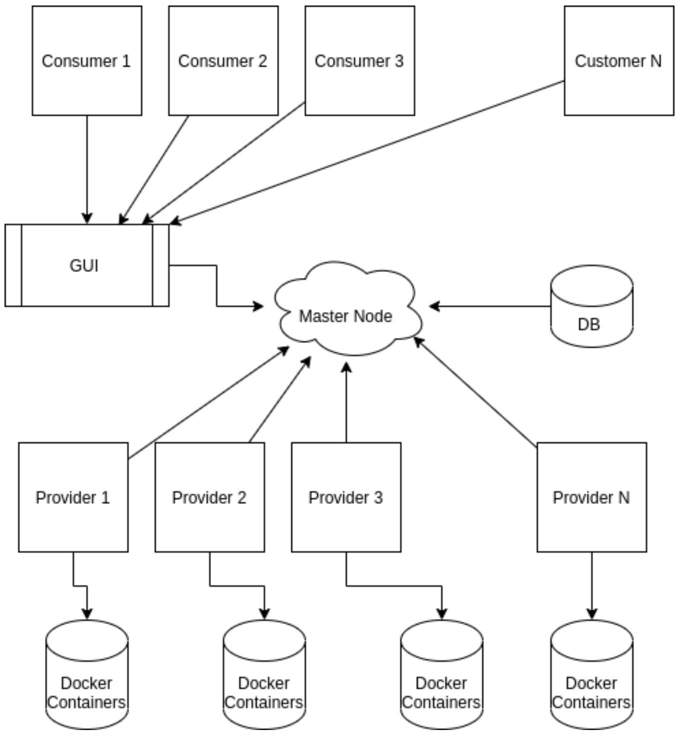

Fig. 1. OpenCloud Distributed Model

Providers: The systems that will be providing their under-utilized resources and the idle time to the Consumers without hampering their work.

Consumers: The systems in need of the resources for research/educational/industrial purposes.

MasterNode: Every locality/town/city should have one and ideally a backup should be there. The master node will be critical for taking decisions of mapping every consumer to every provider. Moreover, in case of node failure in Provider-side the master node should be efficient enough to divert the traffic to another available Provider Node. MasterNode is essentially the load-balancer of the service.

DB: This is the database which will be used by the MasterNodeBackup-MasterNode to decide which Consumer Node will be mapped to which Provider Node and vice-versa.

GUI: This will be a dashboard like web application for the Consumers where they can simply provision resources according to their needs from the list of available resources made available by the Providers. This provides the Consumers with an abstract view of the whole system and the instant they provision a resource an API call will be generated and a docker instance (Any Linux flavor) will be spun up in Provider Side. The Consumer would be able to SSH into the Provider Docker Instance and use the machine according to their needs. On termination another API call will be generated and the Docker Instance inside the Provider nodes will be killed gracefully.

Docker Containers: These will be instances running inside the Provider Host OS either (Windows/Linux) which provide consumers with the necessary requirements for its use-case scenario. These will also be restricted[2] in such a way that it doesn't cause lag issues for the Provider side. Providers with powerful machines can spin up to N instances of docker containers keeping in mind the required resources needed for their own use. These docker containers need to be accessible via the MasterNode so that Consumers can use them.

III. Local Edge-Cloud Deployment Model

Amazon Web Services (AWS)[4] one of the top providers for Cloud Computing services only has 5 edge locations in India (Bangalore, Chennai, Delhi, Hyderabad, and Mumbai). In this model, Figure 2, we describe an ideal Edge Networking Model which can be setup in a locality with the help of ISP's since they control the public IP addresses of the network. In Figure 2, we talk about an ideal scenario where the cloud is behind a NAT and the whole network has been subnetted by the ISP. Here the Providers and Consumers are in the same network only and the Consumer Nodes(IOT devices) can easily use the excessive resources provided by the Provider Nodes. This will improve the overall latency of the network, and we don't have to query the data centers located so far away thus decreasing the round trip time by a massive amount. Real-time decisions can be taken by the IOT nodes and massive processing can be done in the docker containers deployed in the provider nodes. The only caveat to this model would be fault-tolerance. In case of node failure in the provider side the ISP's (the MasterNodes now) should be fast in diverting traffic to another working provider node with minimum downtime.

The MasterNode should be configured to deal with these issues by sending a heartbeat message to every working Provider nodes in a fixed interval of time. In case of any failure it should spin up a new docker container and perform the same tasks which were being performed in the faulty node. DockerSwarm[5] and Kubernetes[6] might help us in managing the cluster of these resources. Security also needs to be setup so that the Provider OS doesn't get corrupted in case of any issues. Any issues should be easily resolvable by killing the docker instance and removing the required images.

This is just an example scenario for a locality/town. If this model can be deployed latency issues can be resolved easily since the nodes are configured very close to each other. We don't have to depend on the cloud provider's Edge Locations and Availability Regions which are located pretty far from the inherent location and are very few in number. Every town, Every city, Every locality will have its own cloud for the benefit of its people and everybody would have uniform access to the resources overall. There will be a one-to-one mapping of the resources from the Consumer to the Provider side and resources would never be wasted, not even a single idle-time. PC's are more suited for this purpose instead of laptops since laptops are not built for providing such long use-case scenarios and may thermal throttle. But with the right restrictions in the docker containers[2] the Provider User won't even notice the background task working in the background and providing resources to the Consumers.

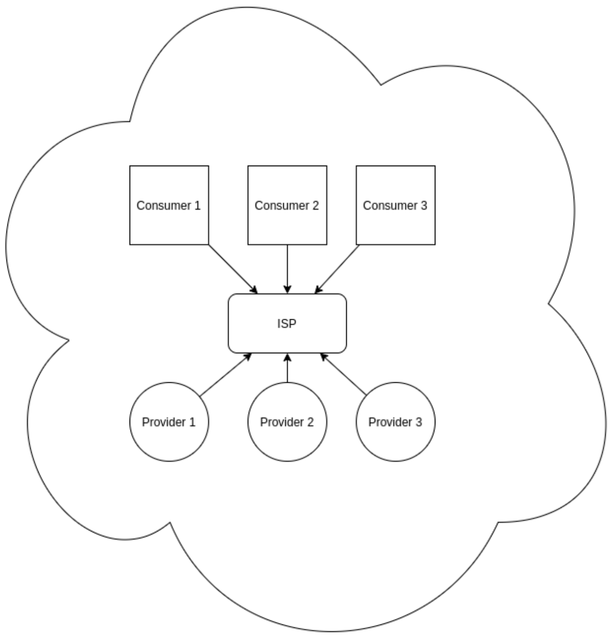

Fig. 2. Edge Networking Model

IV. Challenges

A. Security[3]

When a Provider is providing access to his/her own resources it should be ensured that the Consumer cannot attack the host system or access any restricted files.Docker containers are very similar to LXC containers, and they have similar security features. When you start a container with docker run, behind the scenes Docker creates a set of namespaces and control groups for the container. Namespaces provide the first and most straightforward form of isolation: processes running within a container cannot see, and even less affect, processes running in another container, or in the host system. Each container also gets its own network stack, meaning that a container doesn't get privileged access to the sockets or interfaces of another container. Of course, if the host system is set up accordingly, containers can interact with each other through their respective network interfaces — just like they can interact with external hosts. When you specify public ports for your containers or use *links* then IP traffic is allowed between containers. They can ping each other, send/receive UDP packets, and establish TCP connections, but that can be restricted if necessary. From a network architecture point of view, all containers on a given Docker host are sitting on bridge interfaces. This means that they are just like physical machines connected through a common Ethernet switch; no more, no less.

B. NAT Middleboxes

NAT Middleboxes provide a challenge to the implementation of the whole distributed system. Even if we expose a port for a docker container yet the Provider's host nodes are not configured with a public IP address. For an overall global scenario these will prove to be a challenge. In general the ISP's have a public IP address and the subnetted networks are provided in the locality with private IP's. How the Consumer will SSH[7] into the Provider docker container is a challenge in itself which will require some out of the box thinking.

C. Fault-Tolerance

Kubernetes and Docker-swarm should be in continuous monitoring of the resources and do efficient switching over off the traffic in case of any node failures. Moreover, the difficulties in implementation of the whole system should pose as a challenge for the developer as it would require a lot of help from the participating parties namely the Providers, Consumer & the ISP's(in case of edge-cloud model). The system needs to work in perfect unison to work perfectly.

D. Load Balancing

During runtime if the consumer requires more resources than those which is provisioned by the provider the system should be efficient enough to provision an extra docker container or migrate the existing machine to a bigger docker container with all the available resources which are pretty challenging at this time since docker doesn't provide any way to efficiently combine 2 containers. The system should move to creating EC2[8] instances in the AWS cloud then which should provide as a backup so that the system doesn't fail overall.

V. Conclusion

This model aims to improve upon the two most important aspects of cloud computing which we are trying to solve namely IAAS(Infrastructure as a Service) & Edge Computing. The challenges to be faced during the implementation will be immense but if successfully implemented this will provide the users with ubiquitous access to all the resources. IOT devices will be able to perform and take real-time critical mission decisions which will reduce the latency of the systems. It will not be necessary to send all the required data and processing to the datacenter and the computing nodes available nearby will also solve the total latency for the round-trip time. Our model is unique and novel that combines consumer with provider and solves the challenges of the modern society.

VI. References

[1]https://www.ibm.com/ibm/ideasfromibm/cy/en/howitworks/021307/index1.shtml

[2] https://docs.docker.com/config/containers/resource_constraints/

[3] https://docs.docker.com/engine/security/

[4] https://docs.aws.amazon.com/

[5] https://docs.docker.com/engine/swarm/

[6] https://kubernetes.io/docs/home/

[7] https://www.openssh.com/manual.html

[8] https://docs.aws.amazon.com/ec2/index.html