Chapter 5

Conclusion

Autoscaling Web services on cloud computing environments such as Amazon EC2 has a technical appeal as well as an economic appeal. We try to address the issue of allocating unnecessary safety-net capacity and also make the system increase or decrease its cumulative computing power depending on the actual load, instead of having a static configuration which has been allocated based on the anticipated peak load. Such a traditional setup on cloud computing environments is a wastage of resources, and we can provide a better solution to the peak load problem using some of the inherent features of the cloud. The available computing power should vary proportionate to the actual load experienced. This can be achieved by starting up new EC2 instances when the load increases & terminating running EC2 instances when the load decreases.

Multicast-based membership discovery does not work in such an infrastructure. Traditional multicast based member discovery mechanisms have to be replaced with a Well-Known Address (WKA) based scheme on cloud computing infrastructures such as EC2 since multicast has been blocked on such networks and since there is no guarantee that all members will belong to the same multicast domain. This is because network and IP addresses are randomly assigned from an available set of addresses. This thesis introduced a WKA based membership discovery protocol & showed how the protocol can robustly handle different failure scenarios.

This thesis also introduced the concept of membership-aware dynamic load balancing, where the load balancer can itself made part of the group it is distributing the load across, so that membership changes in these application nodes can be detected by the load balancer. Traditionally, the list of nodes across which the load is distributed had to be provided to the load balancer. Such a load balancer is known as a static load balancer. Hence, new members could not join the application group at runtime. However, such a setup will not work economically on EC2 since the only way of achieving static load balancing is to reserve a set of elastic IP addresses, assign them to the application instances & and then provide those IP addresses to the load balancer. The maximum number of application members will also be bound by the number of elastic IP addresses reserved, if such an approach is followed. This defeats the idea of autoscaling since new members cannot join the cluster & IP addresses of members cannot change to anything other than those assigned from the set of elastic

IP addresses. Our approach enables membership discovery by the load balancer, hence making it a membership-aware dynamic load balancer.

The fault resilience features provided by Amazon EC2 such as elastic IP addresses & availability zones, can be utilized so that the system is truly fault tolerant, irrespective of failure causes, even in some catastrophic situations such as an entire availability zone failing. Even failures of well-known members who have elastic IP addresses assigned to them can be recovered from by dynamically reassigning the elastic IP address. The members can be distributed across availability zones so that complete failure of a region will not result in failure of the system.

We also introduced the concept of an autohealing cluster is a mechanism where the system continues to maintain its minimum configuration, even when some nodes fail. A task which periodically executes on a node checks the current configuration of the cluster against a specified minimum requirement, and ensures that this minumum requirement is always satisfied.

Amazon recently introduced its own autoscaling load balancer implementation called Elastic Load Balancer (ELB) [25]. Some of the concepts between the Amazon implementation & this implementation are quite similar. The Amazon autoscaling takes into consideration additional parameters such as the CPU usage when deciding to scale-up or down. The main advantage in adopting this approach is that it is not bound to the Amazon APIs or implementation, hence we can easily adopt this system to run on any other cloud as well. Different autoscaling algorithms can also be plugged-in this implementation. The main disadvantage in this approach as opposed to using the Amazon load balancer is that, in this implementation, additional redundant load balancer instances need to be run, whereas in Amazon's approach, the Amazon infrastructure itself takes care of this.

Even though this thesis concentrates on autoscaling Web services, the major concepts introduced here such as Well-known address based membership discovery in environments which do not allow IP multicasting, Membership-aware dynamic load balancing & Autohealing cluster can be easily adopted for other areas.

We have also shown how the system's response time drastically increases with load in the non-autoscaling scenario, and how this can be translated to constant response time with autoscaling enabled. This also means that the overall system throughput increases with load in the autoscaled scenario, whereas it is constant in the non-autoscaled case.

5.1 Future Work

As future work, we could implement the same autoscaling setup with different group communication frameworks & carry out a performance comparison study of these different systems in an autoscaling environment. We could also provide tooling that can be used for managing an autoscaling Web services application server. Such tooling will need to handle aspects such as atomically changing the policies of running services and atomically deploying & undeploying services.

Another enhancement to this implementation will be to develop an Amazon EC2 aware load balance algorithm which will be able to take into consideration the actual load on each EC2 instance when deciding on load balancing across the members. Parameters such as the current processor usage & memory usage can be taken into account. This approach will be much more effective than simply taking into consideration the number of messages in flight.