Cloud Application HA using SDN to ensure QoS

Srinivasan Dwarakanathan
National ICT Australia (NICTA)
Sydney, Australia
srini.nathan@nicta.com.au

Len Bass, Liming Zhu
National ICT Australia (NICTA)
University of New South Wales
Sydney, Australia
{firstname.lastname}@nicta.com.au

Abstract—Users expect cloud applications to be highly available with minimum service disruption. Some of the cloud applications also have Quality of Service (QoS) requirements. The High Availability (HA) module needs to consider QoS requirements while placing or failing over the application, application’s components and its replicas. In this paper we propose a new QoS module in the SDN controller. This module creates QoS queues with certain minimum bandwidth on OpenFlow based switches on the route between components requiring certain minimum network bandwidth. Our approach is similar to ‘Aggregation of RSVP for IPv4 and IPv6 Reservations’ (RFC 3175). DiffServ based approaches reserve bandwidth for an entire class of flows and do not cater for requirements from individual applications. Our approach reserves bandwidth on a per-flow basis while overcoming the well-known scalability problem of IntServ. Also, the QoS queues are dynamically configured based on the current requirements of the applications. The QoS module is also responsible for ensuring QoS when topology and route changes. The HA module will interact with SDN controller (which includes our QoS module) to ensure QoS requirements of cloud applications. We also tried an intermediate approach where the QoS flows are directed to a DiffServ queue with a fixed rate (mentioned in RFC 3175) and the reservation of the queue is reconfigured only when the total reservation for the QoS flows exceed the initial reservation. We demonstrate the feasibility and performance aspect of our approach from experiments using Mininet.

Keywords—High Availability; SDN; QoS; OpenFlow

I. INTRODUCTION

There are different types of cloud applications. Web applications, multimedia and soft real-time applications, enterprise applications, mobile cloud applications, and map-reduce based applications are some of the classes of applications on cloud. Users expect these cloud applications to be highly available with minimum service disruption. Most of the current cloud systems use HA solutions to detect host, VM failure [1] and application failure [2]. Some of the cloud applications like multimedia and soft real-time applications have QoS (network bandwidth) requirements in addition to HA requirements. The HA solutions discussed above do not consider QoS requirements of the applications. They may just check that the host or VM where the applications are placed can support required network bandwidth. Some of the applications need assurance of certain network bandwidth between the application’s components and between the application and replica placed on different hosts. Some applications also need assured bandwidth between them and the cloud gateway connecting the users. This assurance is need to provide the expected QoS. Authors in [3] prove that optimized placement of replicas for soft real-time application can result in better responsiveness of the application. Application response time is also a Service Level Agreement (SLA) for some cloud applications. As per [3], placement of application replicas and components is critical for providing required QoS. Hence, HA module need to place application’s replica and components considering the QoS requirements and to meet the SLA guaranteed by the cloud provider. In [21], authors mention about the importance of meeting QoS targets to meet the SLA agreed with the users. They highlight the importance of provisioning VMs considering the QoS and performance aspects. Hence, it is imperative for the HA module to consider QoS requests while placing applications, replicas and application components. The applications’ QoS requirements need to be ensured even when there are changes in the network like link down, and switch failures. In this paper, we have tried to provide a framework for meeting the QoS requirements for particular type of applications while providing HA guarantees.

Software Defined Networking (SDN) [4] decouples control plane from data plane. The networking devices can be programmed by SDN controller using OpenFlow [5] protocol. OpenFlow provides QoS support through queuing mechanism. Queue can be attached to a port and minimum bandwidth rate can be configured for a queue. Flow entries matched to a particular queue are assured of the configured rate. HA module can use SDN to program switches to ensure QoS requirements of applications and their components using the OpenFlow queues. In this paper, we propose a new QoS module in the SDN controller. This QoS module exposes APIs for HA module to ensure that sufficient network bandwidth is available between components. There has been some work on using SDN for QoS support [6] [7]. These works mainly use Differentiated Services or DiffServ using Differentiated Services Code Point (DSCP) value in the Type of Service (ToS) field of IP header. DiffServ reserves resources for an entire class of traffic like voice, video streaming etc. Integrated Services or IntServ uses Resource Reservation Protocol (RSVP) to reserve resource for each flow requiring QoS on all the devices on the path.
IntServ suffers from scalability problem. Aggregation of RSVP for IPv4 and IPv6 Reservations [8] (RFC 3175) proposes a solution for internet based systems wherein aggregate reservation is used on intermediate switches to solve scalability problem. We have used a solution based on RFC 3175 to ensure bandwidth between application components and to address the scalability problem of IntServ. Our contributions are- 1) In our solution, we show that it is possible to provide QoS while providing HA guarantees for cloud applications. 2) We have considered the applications which have replicas and components that need to be placed on different VMs/host and which need network bandwidth guarantees between them to meet QoS requirements. We show through evaluation that for such applications, bandwidth can be ensured between the application components while providing HA guarantees. 3) In our evaluation, we show that when the number of applications requiring QoS guarantees increase dynamically, then the proposed method may provide better guarantees than DiffServ using DSCP [16]. The remainder of the paper discusses about related work, our framework, evaluation, and future work and conclusion.

II. RELATED WORK

Authors in [3] describe a HA solution for soft real-time applications. Their solution optimizes the placement of replicas to reduce the cost of communication between replicas. This improves performance and responsiveness of the application. This solution however does not guarantee the required bandwidth between the replicas which may be required for latency sensitive applications. OpenQos [9] dynamically computes QoS routes where the multimedia traffic can be routed without delay. It does the route computation based on the current packet delay and congestion information collected in the network. This solution is for multimedia traffic and does not address the problem of application replica/component placement and HA. In [6], authors propose a QoS solution for internet system. The SDN controller configures DiffServ Queues on OpenFlow routers at startup. It also adds flow entries to enqueue packets with ToS field in the appropriate queue. The edge routers identify traffic that need QoS and add appropriate DSCP value in the ToS field so that the routers configured by SDN controller enqueue the packet in QoS queues. This solution is not for cloud applications and it does not address the HA requirements and placement of replicas in cloud environment. In [21], authors propose to provision VMs based on performance analytics and QoS requirements. The technique used is based on the end user latency and performance statistics. It does not consider the current network load and the network device capacity while provisioning the VMs. In Pico Replication [22], authors propose a replication system for Middleboxes by exploiting their flow-centric structure. They use OpenFlow to achieve flow level replication. This is a specific HA solution for Middleboxes and does not consider meeting QoS requirements. DiffServ based QoS approach using Big Switch’s Floodlight [10] open-source SDN controller is presented in [7]. This solution is based on DiffServ approach and does not address individual flow’s/application’s requirements. Our solution caters to each flow/application’s requirements and at the same time also solves the scalability problem.

III. FRAMEWORK

The HA solution we have considered is similar to [2] where even the applications running inside the VM are monitored for failures. They are failover to other suitable VMs on VM or host failure and their state is restored after failover. The network bandwidth is reserved for these applications when the applications are created. QoS queues are created on the port connecting the VM interface to the bridge/vSwitch. This is done to reserve the required bandwidth for the application. Flow entries are added to direct the egress packets from the application to the corresponding QoS queue. The Transmission Control Protocol (TCP) ports, User Datagram Protocol (UDP) ports used by the application are used as the matching fields in the flow entries to identify the flows from the application to direct it to a particular QoS queue. So, at the source host each flow requiring bandwidth reservation gets a QoS queue with bandwidth reservation. In the initial implementation, the required network bandwidth is specified by the user. We have considered applications with multiple components and replicas that need assured bandwidth between the components to meet QoS requirements. We present our approach in the following paragraph.

A. QoS queues on route between components

Fig. 1 shows how the bandwidth is reserved on the path between the application and its replica/components. There are two applications App1 and App2. App1 uses TCP port 2210 for communication with its replica and App2 uses TCP port 3310 for its communication with its component. App1 has bandwidth reservation of 100 Mbps and App2 has reservation of 50 Mbps. Let us assume that App1’s replica is first created before App2’s component. The HA module first checks whether the App1 replica can be placed at the particular target host and VM. It invokes the SDN controller’s QoS module’s API to check whether the network bandwidth can be reserved on the path. It then invokes the API to create queues and flow entries on the path. The SDN controller gets the route between App1 and the potential target for App1 replica. As per Fig. 1 this route has switches with ports port1, 2, 3, 4, 5, and 6. QoS queues with bandwidth 100 Mbps is reserved on the switches on this route. Flow entry to match the packet arriving with IP 10.0.0.1 and the TCP port 2210 is added with the action to enqueue the packet in the corresponding output port. When App2 component2 is launched, the HA module checks whether the route to the potential target can support the 50 Mbps reserved bandwidth. This route has switches with ports 1, 2, 3, 7. This means that the reservation at Q1 needs to be increased from 100 to 150 Mbps because both the input and output ports are same for both the routes used by App1 and App2. We used Open vSwitch commands [18] to configure and reconfigure queues. Q2 does not have to be changed
because the output ports are different. The QoS module increases the reservation at Q1 to 150 Mbps and adds flow entry to match packet with the particular source IP and TCP port of 3310 as shown in Fig. 1. Queue Q4 is created on port 7 with 50 Mbps reservation. So the idea is similar to RFC 3175 because on the switch at the source host, QoS queues are created for each flow and bandwidth is reserved. On the switches on the path, aggregated bandwidth is reserved on a single Queue. There is at most one queue for each output port. If the App2 or App2 Component2 is deleted, then the particular reservation is reclaimed from Q1 and Q4. The corresponding flow entries are also deleted. If the topology changes for App1’s route because of port 4 down, then the QoS module in the SDN controller which listens to topology changes will automatically reclaim the resources on the existing route and add the reservation for App1 on new route.

B. QoS queues on routes having tunnel

In virtualized environment tunnels are used to create logical networks. Fig. 2 shows the scenario where App1 is placed on a host. The host has a vSwitch/bridge which connects to VMs. There is also a tunnel bridge which connects to the outside world. The App1 replica is placed on the target host and it also has a tunnel bridge. The local IP at App1’s host is 192.168.0.0.1 and remote IP is 192.168.0.2, which is the local IP at App1 replica’s host. As mentioned in the previous section, when the App1 replica is placed, the QoS module creates queues with bandwidth reservation on the switches on the path and adds flow entries to direct the matching packets to the queues. During this step, it also checks whether the host has any tunnel port on its path to the destination. It identifies the tunnel ports and gets the remote and local IP to be used. At the tunnel port, it creates a QoS queue with aggregated bandwidth and adds flow entries. The flow entry has action to set the tunnel id to be used (based on the logical network used), action to set Type of Service (ToS) field and to enqueue the packet to the right QoS queue. In the subsequent switches, flow entries are added to match based on the local IP of the source tunnel bridge as source IP address and remote IP of the source tunnel bridge as destination IP address. The action is to enqueue the packet on the corresponding output QoS queue. The ToS field is just used to differentiate normal traffic from QoS traffic.

C. Periodic Check

The QoS module in SDN controller listens to topology changes and if the route has changed for a pair of components, then it reclaims the sub-reservation from the switches which are not relevant anymore. It adds the sub-reservation on new switches part of the new route. Sometimes it can happen that the reservations cannot be met on the new routes because of exceeding the capacity of the port. The HA module therefore periodically checks with the SDN controller to get the affected paths. If a particular application and replica pair is affected, it will move the replica to a new host where the bandwidth requirements on the path between the pair can be met.

IV. EVALUATION

OpenStack [11] Platform is used for the HA module of our framework. The HA module is built on top of OpenStack Nova [12]. Nova is the OpenStack Compute module which provisions and manages Virtual Machines. We extended it to
The Mininet network created was made to use our SDN controller. The SDN controller can be used to program the switches and routers created to support OpenFlow protocol. Mininet is a network emulator which can emulate hosts, switches, routers and links on a Linux host or VM. The software switches and routers created support OpenFlow protocol. The Mininet network created was made to use our SDN controller. The SDN controller can be used to program the emulated switches and routers without any modification in the code. We created a topology used in typical datacenters as mentioned in [15] to evaluate our framework. We tested the basic functionality of our framework. We also evaluated the scenario when the links go down to verify that our framework allocates bandwidth on new route.

During our experiments, we also tried an intermediate approach by modifying our QoS module. This approach is mentioned in RFC 3175 [8] and Aggregation RSVP flows are directed to a particular DSCP queue. According to this approach, the QoS flows are directed to a DiffServ queue with a fixed reservation and the reservation of the queue is reconfigured only when the total reservation for the QoS flows exceed the initial reservation. Per-flow reservations are still tracked and hence the bandwidth is ensured. In this approach, we created a specific DSCP queue with reservation on all output ports of switches at startup. We added flow entries on the switches to direct flows which had ToS field set with a particular DSCP code to the DSCP queue. When the application component or replica is to be placed, there is a check on the route between the application and the application’s component/replica to ensure that there is enough bandwidth available. In the second step, a check is performed, to ensure that DSCP queues on the output port of switches on the route have reservation to accommodate all the relevant QoS flows. If any of the DSCP queues on the route do not have sufficient bandwidth, its reservation is increased if the switch’s port/link can support that reservation. So, this method still works like aggregation of RSVP (per-flow QoS reservations are catered) but avoids reconfiguration of queue each time a new QoS flow is added. We ran experiments to compare the performance of the three approaches: “Aggregation of RSVP for IPv4 and IPv6", DiffServ method using DSCP, Aggregation of RSVP using DSCP queue.

We tested the bandwidth by running Iperf [17] on the hosts. Iperf can be used to measure the bandwidth available between two devices. So, to check whether the bandwidth reserved by QoS module is met, we used Iperf. The Iperf test is started between two hosts interconnected by switches. The Iperf client and server formed the application components. The SDN controller’s QoS module’s REST API was called to create the QoS queues with reservation and to add flow entries before Iperf was started. We assumed that each flow’s bandwidth requirement is 30 Mbps. With Aggregation of RSVP, as Iperf was added on new machines, the bandwidth reservation of 30 Mbps was added through the REST API. This resulted in getting the requested bandwidth. This was measured by the Iperf. In case of DiffServ using DSCP, there was a fixed reservation of 90 Mbps reserved at the start (assuming a maximum of 3 flows requiring QoS). As more and more Iperf client and servers were added, the bandwidth each Iperf got reduced because all the QoS flows had to share the reservation. For example, in the first method we added 3 Iperf server and client pairs which shared the same route. Each got the requested bandwidth of 30 Mbps for a total of 90 Mbps. Even when the fourth Iperf client-server pair was added, the QoS module could reserve the requested bandwidth on the route by increasing reservation on the path. In the DiffServ case, the DSCP queue of 90 Mbps was created at startup and this was shared with the 3 flows used

### Table 1: Comparison of QoS Methods Evaluated

<table>
<thead>
<tr>
<th>Type of QoS method</th>
<th>Number of Reconfigurations of queues</th>
<th>Bandwidth guarantees</th>
</tr>
</thead>
<tbody>
<tr>
<td>DiffServ</td>
<td>No reconfiguration</td>
<td>No, if the required bandwidth of QoS flows exceeds the DSCP queue bandwidth reservation.</td>
</tr>
<tr>
<td>Aggregation of RSVP</td>
<td>N flows + number of output ports on route. Also, on topology changes if it affects the particular output port.</td>
<td>100% guaranteed as long as there exists a route which can provide bandwidth.</td>
</tr>
<tr>
<td>Aggregation RSVP with DSCP</td>
<td>Only when the reservation exceeds initial reservation of DSCP queue. On topology change if required.</td>
<td>100% guaranteed as long as there exists a route which can provide bandwidth.</td>
</tr>
</tbody>
</table>

![Comparison of QoS queue reservation for the 3 methods](image)

**Figure 4.** Comparison of QoS queue reservation for the 3 methods.

Support creation of applications on VMs by adding new OpenStack APIs. The Nova Scheduler [12] is a component of Nova which selects the hosts to provision VMs. We extended it to select appropriate VM to place/failover the application considering CPU, memory, and network bandwidth requirements. While placing the replicas and components, even the network topology and the bandwidth available on the route between primary and replica are considered. We used Big Switch’s Floodlight SDN controller [16]. We added a new HaQos module to the Floodlight SDN controller to handle the QoS requirements of applications as mentioned in [7] but with the approach mentioned in the previous section. The interaction between HA module and SDN controller is presented in Fig. 3. The complete implementation can be found at GitHub repository ‘hadev’ [13].

Experiments were performed using Mininet [14]. Mininet is a network emulator which can emulate hosts, switches, routers and links on a Linux host or VM. The software switches and routers created support OpenFlow protocol. The Mininet network created was made to use our SDN controller. The SDN controller can be used to program the emulated switches and routers without any modification in the code. We created a topology used in typical datacenters.
by 3 Iperf server and client pairs. When the fourth Iperf server and client pair was added, the Iperf server and client pairs did not receive the required bandwidth of 30 Mbps because 90 Mbps was shared among 4 client server pairs. So, if the number of applications requiring QoS can increase dynamically and if there are sharp and unpredictable spikes then our method can provide better guarantees. This is the case usually in cloud environments. The third method ‘Aggregation of RSVP using DSCP queue’ works like the method proposed but avoids reconfiguration of queues each time a new QoS flow is added. TABLE 1 shows our findings through evaluation. Fig. 4 shows the queue reservation for the above example. Aggregation of RSVP dynamically reserves for each flow. Aggregation RSVP with DSCP increases reservation only when the current reservation is not sufficient.

More details on evaluation and our overall research goals and our future work can be found in our Technical Report [19]. Evaluating SDN controller performance is an ongoing work and currently we have not evaluated the overhead of using QoS module in SDN controller. We will evaluate the performance aspects of QoS module in our SDN controller similar to that mentioned in [20]. We will evaluate the response time for creating the QoS queues and adding flow entries in real cloud systems with thousands of applications. There has already been work on using Distributed SDN controllers [23] and multi-threaded controllers [20] to improve the performance of SDN controllers. We may in future try these approaches if our evaluation shows the need to improve performance of QoS module.

V. CONCLUSION AND FUTURE WORK

In this paper, we presented a way to provide QoS between an application and its replica or between application components while considering HA requirements. We used a method where the reservation at the source host is for each flow requiring QoS but on the switches on the path, aggregate reservation is used. The proposed solution reduces the number of queues on each output port to one and hence solves the scalability problem of RSVP. It also allows for fine grained resource reservation for each flow. Currently the required bandwidth between components and the total bandwidth required for an application is specified by the user while creating the application. One of the future works is to dynamically change the reserved bandwidth based on the network statistics collected for the communication between components. We want to evaluate our framework on the actual cloud system with hundreds of applications with QoS requirements. As discussed in previous section, we also want to evaluate the overhead of using QoS module in SDN. We also want to evaluate the overhead incurred by HA module for using SDN for placement and failover of applications. The proposed solution is just one way of ensuring QoS while providing HA guarantees and this solution may be suitable for latency sensitive applications and applications requiring bandwidth assurance between its components. Different applications have different HA and QoS requirements. Our overall goal is to come up with efficient way to achieve QoS for different types of applications based on their QoS requirements while providing HA guarantees.

REFERENCES