Centralize the Aequitas Admission Control Mechanism

Divesh Chowdary, Wentai Xie, Yuzhou Zhong
GitHub repository: Github link

Abstract:

Admission control mechanisms play a crucial role in preventing network congestion and ensuring adherence to service-level objectives (SLOs) for remote procedure calls (RPCs) in modern data centers. This work proposes a centralized admission control approach that leverages a global view of network conditions to make more informed decisions compared to distributed schemes like Aequitas. By moving the admission decision logic to a centralized server, our design tries to simplify the management, monitoring, and deployment of new features across the fleet.
In this Project, We explore two main approaches for the centralized controller: 1) a token bucket filter that dynamically adjusts token rates based on SLO violations, and 2) an averaging technique that separates the network and processing components of RPC latency. Extensive simulations evaluate our methods against the baseline Aequitas system in terms of SLO compliance, fairness, and convergence time under various traffic patterns.

1. Motivation:

Ensuring that Remote Procedure Calls (RPCs) meet their Service Level Objectives (SLOs) is crucial for maintaining the performance and reliability of applications in modern data centers. A centralized admission control approach can play a vital role in achieving this goal by leveraging a global view of network conditions and RPC latency metrics across the entire fleet.
By consolidating the admission decision logic into a central server, centralized control simplifies the management, monitoring, and consistent deployment of new features or policies across all nodes. This centralized approach enables more informed admission decisions by considering the overall network state and resource availability, potentially leading to better SLO compliance, improved fairness in resource allocation, and faster convergence times.
Furthermore, a centralized controller can account for the complexities of disaggregated microservices and varying RPC workloads, employing novel algorithms and techniques to effectively translate the global view into accurate admission decisions. This holistic approach to admission control has the potential to provide a coordinated solution for managing RPC traffic and ensuring SLO compliance in large-scale distributed systems.
The benefits of improved performance, fairness, and simplified operations motivate the investigation of centralized admission control approaches. By addressing the challenges of scalability, fault tolerance, and communication overhead, this research aims to develop a centralized admission control system that can effectively manage RPC traffic and ensure SLO compliance in modern data center environments.

2. Introduction:

Modern datacenter applications are composed of numerous microservices that interact with each other and remote disaggregated storage through Remote Procedure Calls (RPCs). To satisfy the diverse business requirements of these applications, the RPCs generated by the microservices often have varying Service Level Objectives (SLOs) that are critical to meet. Aequitas, a distributed sender-driven admission control scheme designed by Google, leverages commodity Weighted-Fair

Queuing (WFQ) to guarantee RPC-level SLOs. In the presence of network overloads, Aequitas enforces cluster-wide RPC latency SLOs by limiting the amount of traffic admitted into any given Quality of Service (QoS) class and downgrading the rest.

While Aequitas operates in a distributed manner, our work explores the potential benefits of a centralized admission control approach. By moving the admission decision logic to a centralized server, we aim to simplify the management, monitoring, and deployment of new features across the fleet. Moreover, a centralized controller can leverage a global view of network conditions and RPC latency metrics to make more informed admission decisions compared to the localized view of individual nodes in a distributed scheme.

In this paper, we propose and evaluate two main approaches for the centralized admission control server. The first approach employs a token bucket filter that dynamically adjusts token rates based on observed SLO violations. The second approach separates the network and processing components of RPC latency through averaging techniques, enabling more precise admission control decisions. Extensive simulations are conducted to evaluate our methods against the baseline Aequitas system under various traffic patterns, focusing on key metrics such as SLO compliance, fairness, and convergence time.

3. Design overview and details

We have developed a centralized protocol based on the Aequitas admission control algorithm. The overall design is shown below.

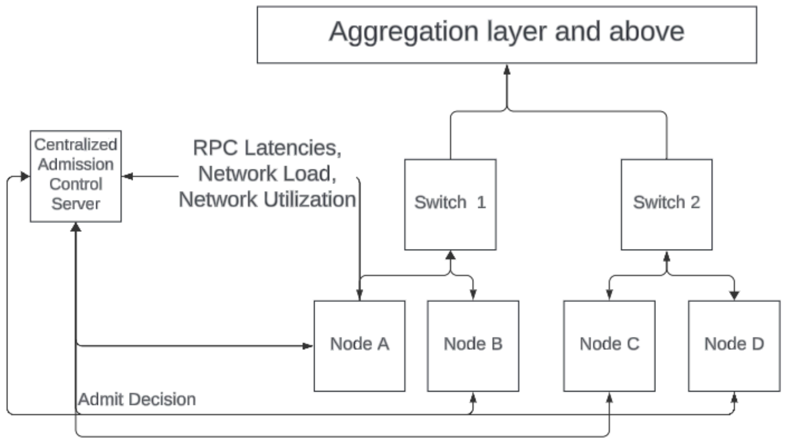
Aggregation layer and above
Centralized Admission Control Server
RPC Latencies, Network Load, Network Utilization
Switch 1
Switch 2
Node A
Node B
Node C
Node D
Admit Decision

Figure 1.The Proposed Centralized Control Design

Before initiating the RPC call, each host will first query the central admission control server to see if it needs to downgrade its RPC call. On receiving the last RPC packet, the host will send the latency information to the central server, which will then update the state to instruct the next RPC call.

The main challenge lies in how the central server updates its state and responds to queries from the hosts. To address this, we will propose two methodologies.

3.1 Method 1

This method consists of sampling the latency over a period of time and then making a decision based on the history of performance.

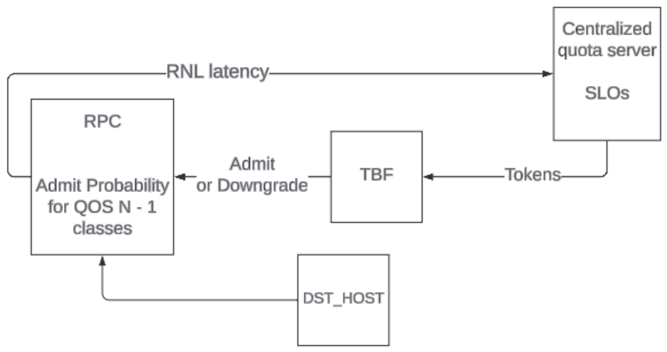

Figure 2: Diagram of the working process of method 1

For every sample period *t*, every destination host will be granted K (K is set as 5 in the experiments) chances to miss the SLO targets. For those end hosts that fail to miss the SLO target, the subsequent RPC calls will be downgraded until the next period *t*. When the host reports the latency to the server, the server will maintain the failure information and send back the decision on whether the RPC call should be downgraded.

3.2 Method 2

Method 2 tries to decompose the RPC latency and do an average on a global view. The original Aequitas paper mentioned the difference between RTT latency and RPC latency. To fully understand the logic behind it, we can imagine a case where an RPC call will need 3 packets to complete.

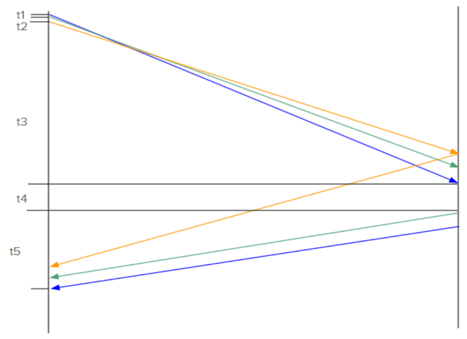

Figure 3. The Diagram of RPC Decomposition

The RPC latency equals:

$$RNL = t1 + t2 + t3 + t4 + t5$$

It is not difficult to find out t3 and t5 is the RTT latency, then can be written as: $t3 + t5 = RTT$. For t4, it is the remote NIC queueing delay and the processing delay. More in-cast traffic will result in a higher latency, so it can be written as $t4 \propto incast$. t2 is the delay caused by the local congestion window. A good TCP protocol will try to maintain the queueing size of the remote queue, so the local congestion window itself is impacted by the remote queue size, which is the in-cast number. We can then write it as: $t2 \propto incast$. Therefore, RNL can now be written as:

$$RNL = t1 + RTT + w * (t2 + t4)$$

this equation can be transformed into:

$$RNL = t1 + RTT + w * incast$$

t1 is impacted by the actual number of packets that need to be sent. If the number of packet that needs to be sent in one RPC is small, this number is trivial. If the number of packets is large, this is still very trivial compared to the congestion problem. As a result, t1 can be discarded. Then we will have the equation:

$$RNL = RTT + w * incast$$

RTT is assumed to be unique per (SRC, DST) pair, however, in-cast traffic is unique only to the remote server. So the algorithm will then average the per end-host unique delay while treating the RTT delay the same as the original Aequitas algorithm.

On every RPC that is finished, the sender host will send the RTT and RPC latency information to the central server. The server then subtracts RTT from the RPC to obtain the latency caused by the in-cast traffic. Then it will update the in-cast latency of that destination:

$$Incast_{new} = w * Incast_{old} + (1 - w) * (RNL - RTT)$$

Then the in-cast will be added back to the RTT provided by the sender to update the state using the original Aequitas algorithm.

4. Evaluation:

Extensive evaluations have been conducted to assess how the centralized method compares to the original distributed Aequitas. These evaluations include testbed experiments in various scenarios anticipated to be practical in production environments. The focus is on three aspects: firstly, examining whether the centralized method remains SLO-compliant by controlling the RNL at the 99.9th percentile. Secondly, evaluating whether the centralized method can ensure fairness by allowing a fair amount of traffic when multiple nodes are issuing traffic simultaneously, regardless of the input traffic mix and their incentives. Lastly, assessing the performance of our centralized method in terms of saving network resources by achieving faster convergence. Based on the three main focuses above, three metrics have been selected for our evaluations: performance, fairness, and convergence. Specifically, performance should be defined as the 99.9th percentile RNL. Fairness should be defined as the probability of high-priority aggregation channels among any pair of nodes corresponding to their issued traffic to the QoS class. Convergence time should be defined as the duration until the network reaches a stable state.

Since our work is largely based on the original distributed Aequitas algorithm, and the Aequitas paper also used all the evaluation metrics above, it provides us with a solid and fair baseline for comparison.

3.1 Evaluation methods:

The performance and convergence time are both evaluated under one experiment setup. This is reasonable because SLO-compliance performance is the most important metric in the original Aequitas paper. The main contribution of Aequitas is reducing the RNL, thereby finding a reasonable SLO. Hence, the experiment setup used to evaluate SLO-compliance should be the setup most similar to that used in real data center environments, which is the one evaluating SLO-compliance.

In this evaluation, a 33-node topology and an all-to-all traffic pattern are used. This means there are 33 nodes, with each node sending traffic to all other nodes. All other parameters remain the same as the setup used in the Aequitas experiments, such as using a 32KB packet size, a 1.4 burst load ratio, etc.

For the fairness evaluation, the main differences lie in the topology and traffic pattern setup. Since the experiment is designed to evaluate whether traffic is shared fairly between nodes, it must be compared and measured in a detailed manner. Therefore, a 3-node topology and an "incast" traffic pattern are used. The "incast" pattern involves only one node receiving traffic from all other nodes. In this particular case, there are 2 nodes sending traffic to one receiving node. Other parameters remain the same as those used in the Aequitas experiments, which are also the same as the setup used to evaluate SLO-compliance performance.

3.2 Evaluation results

3.2.1 Evaluation results of Method One

Generally, the first method performs well in convergence time, but has an impact on the other metrics. The experiment results are shown below:

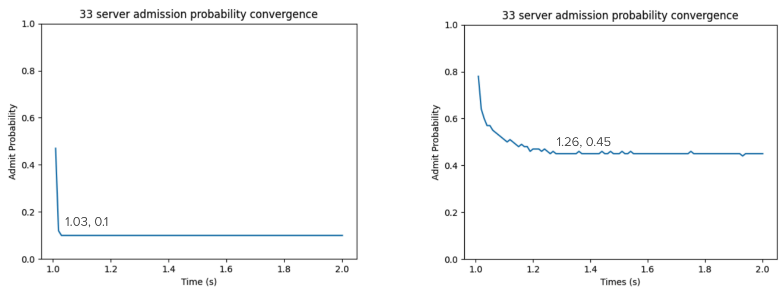

Figure 4: Overall Admission Probability changes using method 1 (left) and baseline (right)

This is the experiment results obtained from setup 1. From the graphs above, it could be clearly seen that our centralized method reaches a stable situation in 1.03 seconds, while the original Aequitas algorithm achieves its stable situation in 1.26 seconds. This brings an improvement of 18.2%, while having a low sum of admission probability.

As for the evaluation of fairness, it is tested using the second setup described above. There are two aspects related to the fairness experiments: throughput and admission probabilities. The traffic shared between two nodes could only be considered as fair while both its throughput and admission probabilities are conforming with expectations. The results are shown below:

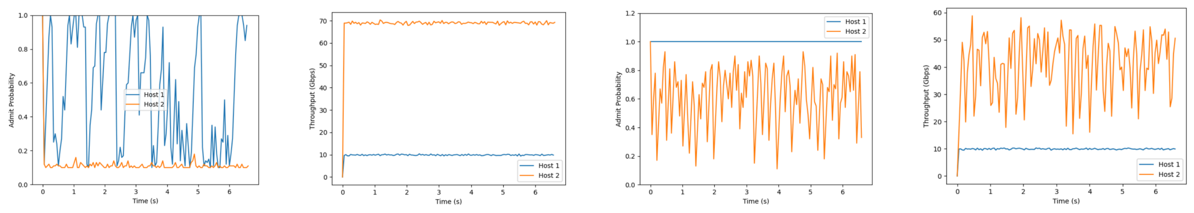

Figure 5: Admit probability and throughput of the two RPC channels from results of method 1 (left pair) and baseline (right pair).

In this 3-node topology used in the experiment, host 1 is issuing 10% of its traffic, which is 10 Gbps with respect to the 100 Gbps link capacity, into high-priority class, which is less than its fair share. Therefore, the baseline shows that it has an admission probability around 1 and a throughput of 10Gbps. From the comparison of the two pairs of experiment results, it could be observed that the results from method 1 have also been affected in all aspects except from the throughput of host 1.

3.2.2 Evaluation results of Method Two:

For this method, it generally achieves the same performance as the baseline in all three metrics, but not performing better in any of them as expected. The experiment results are shown below:

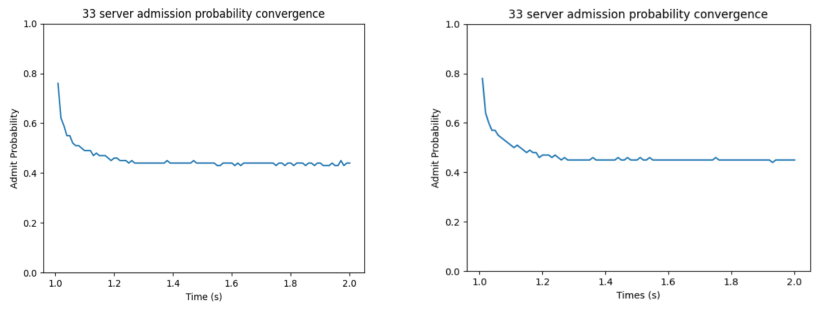

Figure 6: Overall Admission Probability changes using method 2 (left) and baseline (right)

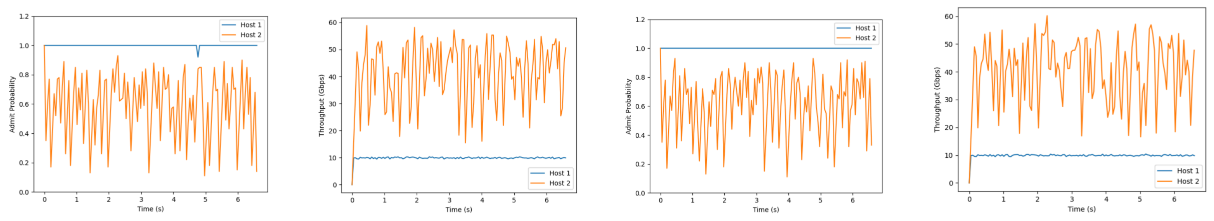

Figure 7: Admit probability and throughput of the two RPC channels from results of method 2 (left pair) and baseline (right pair).

3.2.3 Evaluation results of Method Two:

For the SLO-compliance metrics, the results from both methods are placed together below for formatting convenience:

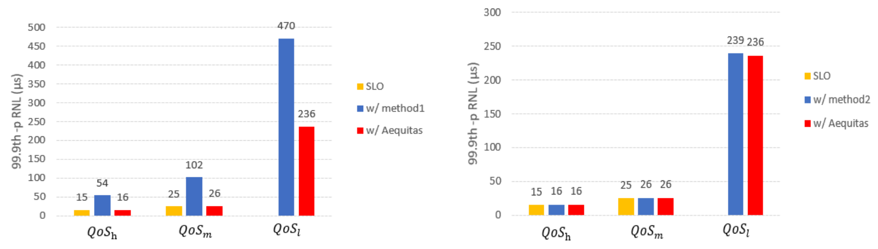

Figure 8: The SLO-compliance results from both methods

5. Conclusion:

In the project, we have developed two different centralized methods to augment the original distributed Aequitas algorithm, intending to achieve a better performance with the hypothesis of the advantages of a central server. In conclusion, the centralized algorithm has the potential to strive for a better convergence time and could achieve at least the same performance as the distributed server. For future work, one direction is: since it is difficult to find the most appropriate parameter value in method 1, and now it is too aggressively suppressing the admission of certain nodes, a machine learning algorithm could be used to figure out the right balance when there is a high burst load. Apart from this, testing the above algorithms in practical environments will be ideal to identify valuable insights into their real-time performance.