Learnings from Deploying Network QoS Alignment to Application Priorities for Storage Services

Matthew Buckley and Parsa Pazhooheshy, *Google and University of Toronto;* Z. Morley Mao, Nandita Dukkipati, Hamid Hajabdolali Bazzaz, Priyaranjan Jha, Yingjie Bi, and Steve Middlekauff, *Google;* Yashar Ganjali, *University of Toronto*

https://www.usenix.org/conference/nsdi25/presentation/buckley

This paper is included in the Proceedings of the 22nd USENIX Symposium on Networked Systems Design and Implementation.

April 28–30, 2025 • Philadelphia, PA, USA

978-1-939133-46-5

Open access to the Proceedings of the 22nd USENIX Symposium on Networked Systems Design and Implementation is sponsored by

جامعة الملك عبدالله للعلوم والتقنية
King Abdullah University of Science and Technology

Operational Systems Track: Learnings from Deploying Network QoS Alignment to Application Priorities for Storage Services

Matthew Buckley[†‡], *Parsa Pazhooheshy*[†‡], *Z. Morley Mao*[†], *Nandita Dukkipati*[†], *Hamid Hajabdolali Bazzaz*[†], *Priyaranjan Jha*[†], *Yingjie Bi*[†], *Steve Middlekauff*[†], *Yashar Ganjali*[‡]

[†]***Google LLC.***, [‡]***University of Toronto***[*]

Abstract

To ensure that application network traffic is prioritized correctly within data center networks, it is critical to align the configuration of network QoS in packets to the intended priority of the application. These QoS configurations, typically encoded in the DSCP bits in the IP header, are interpreted by network switches and routers to determine the resources such as buffer space and scheduling priorities, for network traffic. Conceptually, it appears fairly straightforward to map the application priorities within data center networks to network QoS configurations, as long as the mapping is well defined. In this work, we describe our experience of aligning network QoS settings for intra-cluster storage traffic to application priorities on a per-RPC basis for a large data center network, with well-defined static mappings from priorities to QoS traffic classes. We describe some unexpected insights learned from the deployment experiences, e.g., downgrading traffic to use a lower QoS does not always imply worse network latency due to over-used QoS bands in the network. We also share some challenges encountered along the way to reach the goal of a fleet-wide deployment, including the concerns of potential performance regressions due to QoS downgrades. These lessons provide guidance on the use of a QoS-based scheduling strategy to meet service guarantees and can be deployed to networks of any scale.

1 Introduction

Performance of network services heavily depend on the delay along the network path, as well as other factors such as server load [1–4]. Network delays can be much higher under congestion when the load imposed by arriving traffic exceeds network capacity [5–9]. This affects both wide-area network (WAN) links and paths within data centers, as traffic traversing both types of networks can experience queuing delays from congestion. While various host-based and network-based techniques such as admission control [10–14] can help reduce the traffic demands to minimize congestion, transient congestion can still occur due to bursty traffic patterns [3, 15–17]. This causes switches along the network path to drop packets when faced with full buffers.

The latency of higher-level application entities, such as Remote Procedure Calls and Remote Memory Accesses, plays a crucial role in the applications' performance. Different applications exhibit varying sensitivities to these delays; some, particularly those that prioritize tail latency, cannot tolerate any packet loss, while others focus on average latency or overall throughput. To manage network performance, switches utilize Quality of Service (QoS) bits in packet headers to deliver tailored service levels and latency expectations based on application demands. In general traffic is distributed across various QoS levels based on the specific requirements of workloads, allowing QoS queues to establish priorities according to their latency expectations. When congestion occurs, this prioritization enables the dropping of packets for lower-priority traffic, thus minimizing the impact on more latency-sensitive workloads. Thus, designing the right priority enforcement within data-center networks requires marking traffic according to the intended priority at the application layer.

In [14], Aequitas was introduced as Google's solution for aligning application priorities with network QoS. It was argued that this enforcement was most effective at the Remote Procedure Call (RPC) level and the work primarily focused on the design aspects with some production examples from early rollout. In this work, we expand on ideas introduced there and focus on our deployment experience from implementing the primary phase of Aequitas in Google's storage systems. We show production examples that justify the originally proposed design and offer new insights regarding the complex relationship between QoS queues and network latency.

In summary, our work focuses on these practical aspects of deploying Aequitas in Google networks for the storage services, which all fall under Phase 1 in [14]: (1) defining the static mapping from application priority to network QoS, (2) enforcing these QoS settings on traffic at the RPC level based on the priorities, and (3) supporting any exceptions to bypass this scheme due to special requirements.

[*]Work completed while the students were at Google

Thus, we clarify the goals of deploying Aequitas for storage to prevent priority inversion, which occurs when traffic of higher priority receives less network resources compared to traffic of lower priority. These are a refined set of goals as compared to [14] based on our deployment experience.

- *RPC level enforcement*: The granularity of RPC level enforcement of QoS configuration is necessary so that it is sufficiently fine-grained to enable different priorities of RPCs within a task or job for a given application.
- *Distinction between network regions with different bandwidth management policies*: The enforcement mechanism should distinguish network boundaries when adjusting network QoS. This is needed as traffic provisioning and engineering may vary significantly across different regions. For instance, bandwidth is often more constrained for WAN links—where users are typically charged for bandwidth—compared to intra-fabric bandwidth within the data center, which is generally abundant for applications, except in rare tail cases.
- *Efficient and practical deployability*: We require a method to classify RPCs according to application-level priority and then assign QoS to this traffic based on a pre-defined mapping. This must be performed without incurring excessive overhead or high resource usage.
- *Positive overall performance incentive for applications*: The new enforced QoS mapping needs to bring an overall positive performance for applications (in terms of latency). Thus, it is important that the highest priority traffic experiences minimal regression (if any) in performance after the QoS alignment.

The journey to deploying Aequitas as the default QoS authority across Google storage systems has been a long one, spanning multiple years, and there is still a lot of work to be done. We have gained valuable insights regarding the complicated interaction between network QoS and delay as well as the diverse needs of users and how that influences their view of the network. Even with a partial rollout, Aequitas is currently affecting millions of RPCs per second. Aequitas is a mature technology that has been integrated into RPCs to and from LL storage for an extended period. It has also been actively utilized for various client-facing services. Our findings can serve as a guide for other network operators aiming to differentiate traffic through QoS. The main takeaways are the following:

- **Priority is the correct network scheduling unit:** We show that prioritizing RPCs based on priority, not size, leads to significant latency reductions for critical traffic with minimal impact on lower-priority classes. This highlights that priority is the more effective unit for network scheduling.
- **Lower QoS does not imply higher latency:** Contrary to popular belief, we demonstrate that users can experience decreased latency when moving to a lower weight QoS queue if that queue is sufficiently underutilized relative to its weight (§2.3 and §4.2).
- **All can win:** We show that Aequitas is not a zero-sum game and we observe examples across production clusters where traffic at all priorities see decreased latency (§4.3). We show that these learning are consistent and hold true across different layers of the storage stack.
- **Benefits arise even under incremental deployment:** We provide production examples detailing how even under a gradual rollout process users can experience performance improvements (§5). This shows that network operators can still realize benefits from QoS assignment schemes when an all-or-nothing approach is infeasible.

The paper is organized as follows. In §2, we describe background for this project and the motivation for Aequitas, followed by a discussion of operational challenges in §3.2. Key learnings from our deployment experience are explained in §4. The detailed progress in QoS alignment is covered in §5, followed by a discussion of results and future work in §6. Related work is covered in §7, before we conclude in §8.

2 Background and Motivation

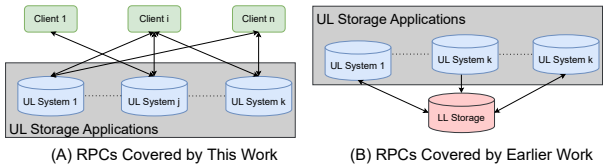
We give an overview of the Aequitas system first introduced in [14], discuss the relevant details of Google storage systems, and highlight the insights that motivated Aequitas' design.

2.1 Aequitas Overview

Aequitas operates at the RPC layer. An RPC is a programmatic request for information in which a client instructs a server to execute a procedure and receives a response. Application-level operations can be composed of hundreds of RPCs so that application performance is dependent on RPC performance. Currently, in Google production datacenters, more than 95% of application traffic is generated by RPCs [14]. Aequitas aims to inform the network of RPC level objectives and increase the number of RPCs that meet their service-level objectives (SLOs). In general, RPCs from micro-services often have SLOs that vary widely based on application needs. Here, we focus on Google storage as $\sim 75\%$ of all fleetwide non-ML related RPCs are to or from these systems [14].

Aequitas uses QoS to differentiate traffic and ensure that the most critical RPCs are given the needed network resources. The feasibility of this design was shown in [14] with results from both simulation and production rollouts. A theoretical framework was also given, based on network calculus [1], detailing the complicated relationship between QoS weights and worst-case network delay.

Figure 1 shows the high-level architecture of Google storage systems. We have n internal clients which connect to any number of k Upper Level (UL) databases (Bigtable [18], Spanner [19], etc.). These UL systems connect to Colossus [20], a



(A) RPCs Covered by This Work (B) RPCs Covered by Earlier Work

Figure 1: Storage RPCs Covered by Aequitas

common Lower Level (LL) system responsible for physically storing data. Thus, the UL systems provide different abstraction layers atop LL storage and clients can use any number of these abstractions depending on application needs.

The figure also highlights the differences between our previous results and the current work. In [14], the deployment and analysis focused on RPCs between UL systems and LL storage (Figure 1(B)). We have since expanded this to also handle traffic between clients and the various UL systems (Figure 1(A)) while also finalizing the deployments started in the previous work.

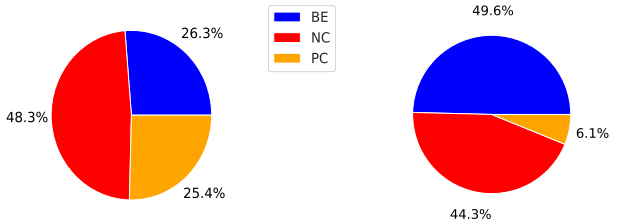
2.2 The Case for Aequitas

In [14], the notion of race-to-the-top was introduced for the first time. That is, in times of network overloads, some applications experience increased delay, causing them to miss their SLOs. In response, the corresponding developers request (and are often granted) higher QoS under the belief that a higher QoS will decrease delay and the probability of the corresponding RPCs missing their SLOs. However, these decisions are often independent of the overall QoS mix in the datacenter. In the extreme all users move their traffic to the highest QoS bands and there is no differentiation among traffic. For a given QoS band, QoS_h, there is a point at which it is no longer beneficial to upgrade traffic to QoS_h.

Over the last several years there has been an explosion of works that aim to integrate application knowledge into network scheduling decisions [21–26]. However these solutions treat all traffic belonging to an application or a coflow as identical from the network's perspective. Aequitas takes the view that there exist different classes of traffic even within one application. In our storage systems we observe that RPCs can be broadly categorized as Performance-Critical (PC), Non-Critical (NC), or Best-Effort (BE). PC RPCs have the most strict SLOs in term of latency and are generally interactive user-facing traffic. NC traffic often constitutes bulk storage operations[1], while BE RPCs are generally the background analytic traffic in our storage systems.

Figure 2 shows the priority distribution for request (response) traffic to (from) Colossus and demonstrates that there is a large amount of traffic for each of the three priorities. Moreover, the UL storage systems are composed of a multitude of jobs from clients around the globe and we see that even within an individual job multiple priorities can be used.

[1] these are large throughput-intensive procedures destined to storage



(a) Request (b) Response

Figure 2: Priority Distribution for Colossus Traffic

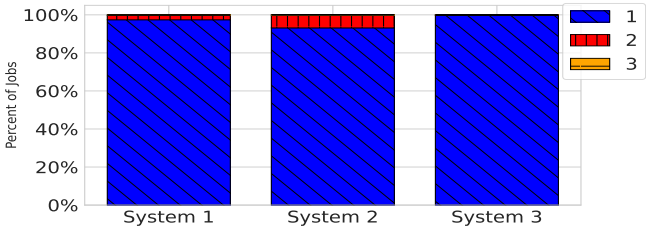


Figure 3: Percentage of Jobs Using 1, 2, or 3 Different Priorities Across Three UL Systems

Figure 3 shows the percentage of jobs that use one, two, or three priorities among BE, NC, and PC, across three different UL storage systems. We see that most jobs stick to only one priority. However, there are a small proportion of jobs that use two or more priorities and these jobs tend to be much larger, accounting for the majority of traffic. This can be seen in Figure 4 which shows the percent of request traffic in each of the three UL storage systems for jobs that use one, two, or three priorities among BE, NC, and PC. The story for response traffic from the UL systems to clients is similar.

These measurements demonstrate that simply setting a network priority at the application or job level fails to account for the different categories of traffic within a job. Instead, the evidence suggests that RPCs are the right unit for QoS enforcement as proposed in the initial Aequitas design [14]. More specifically, Aequitas takes the approach of broadly classifying RPCs as either BE, NC, or PC. It then maps these priorities to different QoS classes, implemented with Weighted Fair Queuing (WFQ)[2]. That is, BE, NC, and PC are mapped 1:1 to QoS classes QoS_l, QoS_m, and QoS_h where QoS_i corresponds to a queue with scheduling weight w_i and $w_l < w_m < w_h$. Thus the network is informed of the relative importance of different classes of RPCs, without the need to differentiate at the application level.

2.3 Determining the Correct QoS Mix

As discussed, clients are usually hesitant to change QoS, especially to one with worse latency guarantees. However, as

[2] In WFQ, switches are set up with various queues where queue i has weight w_i. Then the traffic in queue i receives w_i fraction of a link's capacity in the long run.

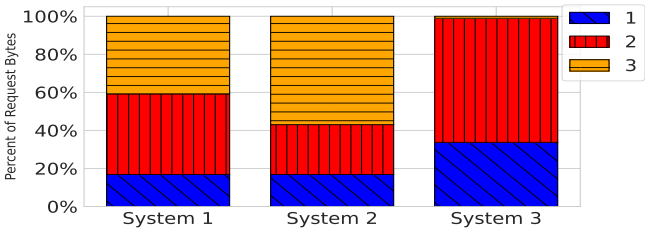


Figure 4: Percent of Request Traffic for Jobs Using 1, 2, or 3 Different Priorities Across Three UL Systems

evidenced by the race-to-the-top phenomenon, clients' QoS selection rules are usually arbitrary. So we aim to answer the following question: *what are the conditions under which a user should prefer one QoS class over another?*

Consider a WFQ with k QoS classes corresponding to k queues with weights $w_0, w_1, ..., w_{k-1}$. We let λ_i be the total load on QoS_i and let β_i^c be the load belonging to client c on QoS_i. Pick two of these QoS classes, QoS_0 and QoS_1 with $w_0 > w_1$. We can show that under a mild set of assumptions, client c can achieve a better service rate by shifting a small amount ε of traffic from QoS_0 to QoS_1 if

$$w_1\left[\frac{\lambda_1-\beta_1^c}{\lambda_1(\lambda_1+\varepsilon)}\right] > w_0\left[\frac{\lambda_0-\beta_0^c}{\lambda_0(\lambda_0+\varepsilon)}\right] \quad (1)$$

This improved service rate occurs despite the fact that a small amount of traffic is moved to a lower weight QoS band and stems from the fact that QoS_0 is overloaded relative to QoS_1. More details on this can be found in Appendix A.

This serves to illustrate that a QoS class with a higher scheduling weight does not always imply better service. The work in [14] showed the conditions for priority inversion (when a higher weight QoS class has worse performance as compared to a lower weight QoS class). But that analysis focused on all traffic sharing the same queue. Here we analyzed the conditions under which a particular user receives better service on one QoS class compared to another. We see that there are scenarios in which it is beneficial for a user to transfer a portion of their traffic from a QoS class with a higher weight to one with a lower weight. Thus, there are situations in which priority inversion exists. Indeed, in §4 we see scenarios in Google production datacenters in which users benefit from a lower weight QoS class.

As the storage systems represent a significant portion of Google traffic, we desire that the mix of (QoS_l, QoS_m, QoS_h) within the storage systems is close to an ideal QoS mix fleetwide. The Google-wide mix should not have any priority inversions, and so for a particular application a we say that a has an unhealthy mix for (QoS_i, QoS_j) with $w_i > w_j$ if β_i^a:β_j^a is greater than w_i:w_j. Note, if a is the only job in the cluster then this is equivalent to the notion of priority inversion we outlined above; a has overloaded QoS_i relative to its weight as compared to QoS_j so that we would expect moving a sufficiently small volume of a's traffic from QoS_i to QoS_j would yield a better service rate. Thus, this is a restriction of priority inversion to a specific application and serves to identify applications or users that are most responsible for an overall unhealthy QoS mix. Appendix B examines instances of unhealthy QoS in Google storage systems.

In the presence of priority inversion, performance on each of the QoS classes becomes unpredictable, making it difficult to decide to which QoS queue traffic should be assigned. RPCs in the wrong queue can miss their SLOs when the latency experienced is higher than the queue weights would suggest. Aequitas' goal is thus to manage the assignment of traffic to the different QoS queues in order to avoid priority inversion and thus provide more predictable performance for each queue. This in turn ensures that SLOs for the given queues can be set accordingly so that traffic priorities can be assigned appropriately, leading to fewer SLO violations.

3 Deployment: Strategy and Challenges

In this section we discuss our strategy to deploy Aequitas and challenges faced along the way.

3.1 Deployment Strategy

At the surface, it appears that the work required to extend Aequitas is fairly straightforward and is simply a matter of enforcing QoS mappings based on application priority. However, even the initial step of defining these mappings is non-trivial as RPCs are scattered across multiple storage systems, each with their own notion of priority. This disparate collection of priorities must each be mapped to the same finite number of QoS queues, while accounting for the different SLAs across these systems. Moreover, in contrast to our previous results, we are now handling RPCs that either directly originate from or are destined to end users. These clients have a diverse set of demands on the network and many directly measure RPC performance. Thus, a large number of users have configured their own QoS and do not necessarily want to conform to the QoS assignment dictated by the mappings, making it difficult to coordinate large-scale changes.

On a per RPC basis, Aequitas leverages information about the source and destination IP addresses as well as the application priority and requested QoS to make its decisions. However, which packets of an RPC are affected by Aequitas' selection depends on the destination. The LL storage is often obscured from users and as a result the Aequitas team elected for a server-side implementation for QoS enforcement on the traffic between UL and LL systems. Under this approach, the first few packets of an RPC are sent using the original, unaltered QoS. Upon arriving at an LL server, Aequitas chooses a QoS for the RPC based on priority and all subsequent packets sent on this connection, in either direction, use the updated QoS. Such an approach allows for quicker adoption as all

users interacting with the same server experience the change.

In contrast, each of the UL systems is visible to clients and clients monitor RPC-level metrics for the storage systems with which they interact. As such, a server-side implementation in these UL systems would obscure information from users and would not account for their varied SLOs. For example, in the case of Spanner, client-side implementations allow Spanner clients to see the QoS changes of their RPCs before they are sent over the network. In contrast, under a server-side implementation, the client's chosen QoS is reflected in the Spanner client library but the actual QoS chosen by Aequitas at the server can be different, causing a mismatch. This obscures information from users, making it unclear why this mismatch occurred. The Aequitas team therefore decided on client side implementations to expose these changes to users and to more easily reason about how the changes affect different clients. In this setting, Aequitas intercepts RPCs before they enter the network and chooses the correct channel based on the decided QoS. Note that clients do not directly interact with Aequitas. Each storage system has its own API for setting priority and Aequitas is integrated into these APIs to seamlessly pick QoS based on the pre-defined mappings. Appendix C discusses the impact these decisions had on deployment.

The Aequitas implementation is currently restricted to intra-cluster traffic. Aequitas over the WAN is a much more challenging issue due to the longer network latencies and the interaction with BwE [27]. While the proportion of intra-cluster RPCs varies across time and clusters, it accounts for a significant portion of all traffic. Customers typically try to keep traffic within the same cluster due to the inherently lower latency. Aequitas uses data in the backend UL systems to map source and destination IP addresses to clusters and determine which traffic is eligible for QoS enforcement.

3.2 Operational Challenges

Downgrading QoS: By far, the most challenging aspect of Aequitas was convincing individual clients and the teams managing the various storage systems to assign a lower QoS than originally requested for some RPCs. Aequitas distributes RPCs across QoS_l, QoS_m, and QoS_h by RPC priority.

Figure 5 shows the distribution of requested QoS within priority for all request traffic associated with Spanner. There is a high degree of misalignment between the scheme proposed by Aequitas and the current state of the cluster. In particular, there is heavy usage of QoS_h for NC RPCs and more than half of the BE RPCs use QoS_m. There is very little usage of QoS_l acoss all priorities. Imposing Aequitas across the cluster leads to both QoS downgrades (ie. BE RPCs on QoS_h downgraded to QoS_l) and QoS upgrades (ie. PC RPCs on QoS_m upgraded to QoS_h). We see similar misalignment across clusters and various storage systems, for both request and response traffic.

As discussed in §2.2, the notion of race-to-the-top suggests that users are often supportive of moving traffic to a higher

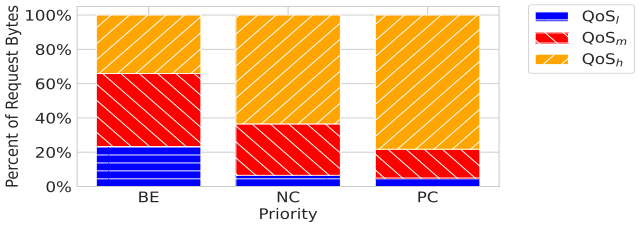


Figure 5: Intra-Cluster QoS Breakdown Within Priority (by Request Bytes)

QoS. This is due to a commonly held belief that higher QoS implies better network performance. This implies that the converse is also a widespread belief: lower QoS leads to worse network performance. §2.3 argued that this is not always the case and traffic can benefit from lower QoS when that QoS class is underloaded relative to its weight. Yet, there can also be cases where users have carefully tailored QoS in order to meet their SLOs and a change in QoS can be the difference between acceptable performance and SLO violations.

As such, enabling Aequitas fleet-wide within a short time could cause multiple production issues. Instead, the Aequitas team used a gradual approach; in some cases enabling the change for individual clients and monitoring performance before moving onto cluster level or multi-cluster rollouts. While this required considerable effort from the team and greatly extended the timeline for widespread adoption, it also allowed for detailed studies of the effects of Aequitas to understand the impact on individual users as well as entire clusters.

Avoiding other Performance Bottlenecks: While Aequitas aims to avoid priority inversion and help more applications meet their SLOs, we do not want to introduce bottlenecks in other areas. A reasonable measure to assess the impact of Aequitas is how it affects RPC network latency (RNL), which is the time that the RPC spends in the network. However, not all developers have intimate knowledge of how the network functions and thus often do not know how to correctly configure network QoS, one of the main motivations for Aequitas.

Developers put most of their attention on end-to-end RPC performance metrics like total RPC latency, memory usage and other general performance indicators. While Aequitas can improve overall network performance through better QoS configuration, for an individual RPC the choice of QoS must be made with minimal overhead to avoid adding unnecessary latency at other points throughout the life of an RPC.

In our evaluations we consider both RNL and overall RPC latency. Throughout the rollout process we also tracked memory usage, error rates, cache misses, and many other metrics, observing no noticeable changes attributed to Aequitas.

Inferring Effects under Incremental Changes: As detailed earlier, due to both Google's scale and availability concerns, we cannot simply roll out Aequitas for all users simultaneously. This challenge is well-known in practice and several methods for running experiments on production servers

while ensuring system availability have been proposed previously [28]. This is especially challenging for client side implementations and often led to several user-level pilot rollouts before moving to cluster level rollouts.

As discussed in [14] and further argued in §2.3, for a particular client the performance of their traffic on one QoS band compared to another depends not only on the QoS distribution of their traffic but also the QoS distribution of all traffic traversing the same switches. This is in addition to all the usual network factors that impact performance: overall load, burstiness of traffic, presence of congestion, etc. This implies that enabling Aequitas for a few clients or a few servers at a time does not paint an accurate picture of its effects. To fully understand the impact of Aequitas requires measurements when no traffic uses Aequitas versus measurements after the policy has been enabled for all RPCs.

However, due to the incremental rollout strategy, the traffic distributions can change significantly between a point in time before Aequitas is enabled and when Aequitas has fully rolled out. This is especially true in the case of client side changes; clients are often synced to different versions of the code. Thus, even in cases where we can enable Aequitas for all clients simultaneously there is often a significant time gap between when the first client picks up the change and when the change is adopted by the final user. Indeed, we saw production examples in which the distribution of a clients' RPCs across priorities greatly changed during the rollout. In the extreme, we observed scenarios where a user's volume of PC traffic grew by 5x during rollout. We thus adopted two major strategies as part of the deployment process.

The first was to target clusters in which a small number of users (ideally one) account for the majority of the cluster's traffic. In this way, once these small number of users have adopted Aequitas we can obtain accurate measures of overall cluster performance. Moreover, it normally requires significantly less time for a small group of users to adopt the change as compared to all clients using the cluster, allowing for a more accurate before and after comparison; in the case of a single dominant user we can measure performance just before the user adopts the change versus performance immediately after adoption. Note that there are in fact clusters at Google that exhibit this behaviour. We have users that account for more than 50% of request and response traffic within a cluster. As a result, applying Aequitas to such a client provides a good indication of overall cluster performance.

Second we implemented a feature that allows us to enable Aequitas for a fraction of traffic, either at the cluster level or at the granularity of individual clients. For example, enabling at 50% for one client means that 50% of the client's RPCs will be randomly chosen to be included in Aequitas whereas the remaining RPCs are unaffected. This is especially useful in cases not covered by the first strategy, where cluster traffic is composed of many small clients, with no client dominating. This random sampling adequately controls for other confounding factors and allows us to tease out Aequitas' effects by comparing traffic sampled versus the RPCs not sampled[3]. This approach is also more appealing to individual customers and owners of the storage systems as it offers a more conservative alternative to the all-or-nothing approach where clients or clusters are either in Aequitas or not.

4 Key Learnings from Production Examples

We next present case studies that highlight key insights we have gained from enabling Aequitas across a diverse user base. As alluded to earlier, the actual code changes for Aequitas are only a small part of the overall challenge. The biggest obstacle is integrating the changes for various clusters that span the globe. In §4.1 we discuss the methodology used to perform analysis and infer the affects of Aequitas. §4.2 and §4.3 provide examples demonstrating our main takeaways.

4.1 Analysis Methodology

Aequitas aims to maximize the volume of PC RPCs that meet their network SLOs. This means lowering the RNL for PC RPCs, but without introducing intolerable latency to NC and BE RPCs. We also want to ensure that Aequitas does not lead to performance regressions in other areas, such as increasing overall latency at the expense of decreased RNL latency, increased memory usage, or general connection errors.

Throughout this section we present experiments that compare traffic that obeys the Aequitas policy (aligned) against traffic that does not (misaligned). For each of these experiments, 50% of traffic was randomly sampled for Aequitas. While cluster utilization and latency numbers can fluctuate during the experiment, we chose highly loaded clusters for these studies and users reported no adverse effects in terms of increased packet loss nor unacceptable memory usage. In fact, in some cases where Aequitas reduced latency we also observed decreased packet loss rates and higher throughput. During Aequitas rollouts, users have a general idea that the change is coming but they are not aware of the exact time of change as it depends on internal deployment cycles. While not a true A/B experiment, we made efforts to simulate it as closely as possible with methods like random sampling to control for other confounding factors. A true A/B experiment is not feasible in our production environments.

Monarch [29] allows clients to monitor the performance of their jobs. It exposes dashboards showing job performance across several key metrics, aggregated over all the RPCs of the job. As detailed in [30], it is not possible to collect a vast array of metrics for every RPC due to Google's scale. Thus, these dashboards allow users to assess overall performance

[3] While there are other sampling techniques to more directly control for confounding factors, we find that either we do not have a priori information to perform such sampling (in the case of RPC size) or that it would add unnecessary overhead (in the case of priority based sampling). However, we see that due to the scale of the Aequitas experiments, random sampling often does a good job of controlling other confounding factors.

but do not contain all the metrics necessary to assess the impact of Aequitas. Crucially, Monarch does not contain the application-specific priorities as these differ by application and collecting these would limit Monarch's scalability.

Instead, Dapper [31] is our primary analysis tool. Dapper is Google's distributed tracing infrastructure that exposes intuitive APIs for developers to add custom annotations to their traces. Fathom [30] metrics have been integrated into Dapper, providing fine-grained decomposition of latency numbers attributed to the various systems traversed by the RPC. We also export custom annotations from the Aequitas logic indicating the priority of the RPC, the requested QoS, and the QoS chosen by Aequitas. This allows us to examine RNL latency, and its contributing factors, on a per priority basis.

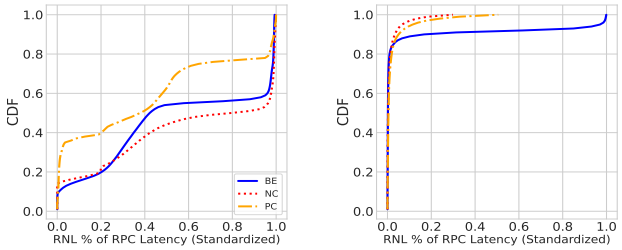
In order to operate Dapper at Google's scale, the RPCs are necessarily sampled. Sampling rates can differ depending on characteristics such as job size, RPC importance, and frequency of errors (with higher error rates necessitating higher sampling to debug performance issues). To account for this, the Aequitas analysis restricts to uniformly sampled Dapper traces and weights each trace by the inverse of its sampling rate. For example, consider three traces from Dapper, one of which represents an RPC of size 1MB sampled at rate 1, while the other two traces represent RPCs having size 2MB and are sampled at rate $\frac{1}{2}$. Then this represents a total of 5 RPCs ($\frac{1}{1} + \frac{1}{1/2} + \frac{1}{1/2}$) having an average size of 1.8MB ($1 * 1$MB $+2 * 2$MB $+2 * 2$MB = 9MB / 5 RPCs).

To assess the performance of Aequitas we are not only interested in average performance but also behaviour at the tail. There is a concern that due to sampling, some behaviour at the extreme is not accounted for. Thus we supplement our Dapper analysis by comparing the results to the high level metrics exposed in Monarch dashboards. We do this for clients with the most traffic impacted by Aequitas and continue to monitor performance after Aequitas adoption. We observe that the findings from the two tools are consistent.

4.2 Key Finding 1: Priority, not Size, is the Correct Network Scheduling Unit

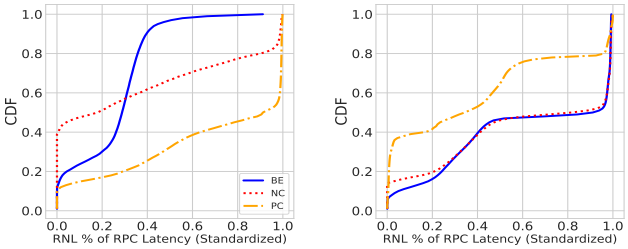
As the name suggests, PC RPCs are the most critical, and thus we want these RPCs to spend as little time as possible in the network. Instead these RPCs should spend most of their time in computation. In contrast, BE is treated as a scavenger class for RPCs, which can thus tolerate slightly increased time in the network in favor of prioritizing NC and PC RPCs [32]. We thus examine RNL as a percent of total RPC latency and would expect that this quantity is lower for PC RPCs as compared to BE or NC RPCs. We deem this quantity as RNL proportion. RNL is also the portion of total latency that the network has control over, as discussed in [14].

In Figure 6, we examine RNL proportion across priorities and RPC sizes. This snapshot was taken from Spanner when



(a) RPC size < 1KB (b) RPC size $\geq$ 2MB

Figure 6: RNL for different RPC sizes under High Misalignment



(a) RPCs on QoS$_l$ (b) RPCs on QoS$_h$

Figure 7: RNL for small RPCs ([0, 1KB)) across different QoS under High Misalignment

there was still a high degree of misalignment[4], allowing us to tease out the effects of QoS. These latency numbers are taken across clusters with differing load characteristics: from underutilized to clusters at or above capacity. Figure 6a shows results for small RPCs ([0, 1KB)) while Figure 6b shows the same plot but for RPCs that are at least 2MB. In both plots we normalize by the maximum RNL proportion so that both figures range from 0.0 to 1.0 with at least one value achieving 1.0. In the case of large RPCs, both NC and PC traffic have a much shorter tail compared to BE, as expected. Moreover, for small RPCs, RNL accounts for a smaller portion of total latency for PC traffic as compared to BE and NC. However, as there is still a high degree of misalignment in this snapshot, we observe that for small RPCs median RNL proportion is higher for NC as compared to BE.

To further understand the effect of QoS on RNL, Figure 7 looks at RPC performance across priorities for different QoS for small RPCs. These numbers were also taken across all clusters showing varying network load profiles. Figure 7a depicts RPCs that use QoS$_l$ whereas Figure 7b shows RPCs using QoS$_h$. In both figures the values are normalized by the maximum value of RNL proportion across all small RPCs, regardless of QoS, to facilitate a comparison across figures. We see a substantial reduction in RNL proportion for PC

[4] As this is production traffic, some traffic is naturally already aligned and so we cannot compare to a perfect control with 100% misalignment

	p50		p95		p99	
Priority	**Mean**	**Max**	**Mean**	**Max**	**Mean**	**Max**
BE	0.02	0.04	0.14	0.17	1.02	1.55
NC	0.02	0.04	0.17	0.56	0.42	1.75
PC	0.17	-0.73	-1.49	-34.01	-3.89	-263.95

Table 1: Aligned RNL Improvement over Misaligned by Priority (Negative Values Indicate Improvement)

	p50		p95		p99	
Priority	**Mean**	**Max**	**Mean**	**Max**	**Mean**	**Max**
BE	0.00	0.00	0.00	-0.02	0.00	-1.82
NC	0.00	0.00	0.00	0.05	-0.04	-3.65
PC	-3.06	-3.50	-81.57	-2419.22	-134.45	-2419.51
All	0.00	0.00	-0.11	-0.17	-0.66	-5.66

Table 2: Aligned RPC Latency Improvement over Misaligned by Priority (Negative Values Indicate Improvement)

RPCs for traffic using QoS_h as compared to QoS_l. Moreover, we see high median and tail numbers for BE and NC RPCs even when they use QoS_h, with slightly better performance for NC RPCs. RNL represents a larger proportion of total latency for NC RPCs on QoS_l as compared to BE traffic. This suggests that better prioritization in the network, in the form of higher-weight QoS, has a more significant impact on PC RPCs as compared to lower priority RPCs and BE traffic is the least sensitive to network prioritization.

Note that Figure 6 shows that both small and large RPCs are distributed across all three priorities, suggesting that prioritizing based on RPC size is too coarse grained and fails to capture the needs of RPCs operating on different priorities. This is consistent with the findings in [14] and combined with the above analysis demonstrates that Aequitas can correctly translate priorities to the network through its QoS mapping.

To further tease out the effects of Aequitas, we look at a particular client (denoted c) in a large, highly loaded cluster. c accounts for over 50% of all traffic in the cluster and was one of the initial pilot customers of Aequitas for Spanner. We look at a snapshot when Aequitas was rolled out at 50% for c. In the absence of Aequitas, all of c's traffic uses QoS_h with a small amount of traffic on QoS_m. Thus Aequitas upgrades a small fraction of c's PC traffic on QoS_m and either downgrades or leaves unchanged all of c's BE and NC traffic.

Table 1 shows the improvements (or regressions) in RNL for traffic aligned under Aequitas versus misaligned traffic for c. A CDF of RPC performance is computed every hour for a 4-day window and then we compute the mean and maximum values seen over the analysis period for the p50, p95, and p99 for each priority. These calculations are done for both aligned and misaligned traffic and then the differences are normalized by the standard deviation of p99 of misaligned traffic across all 3 priorities[5]. For example, if $s^c_{RNL,m}$ is the standard deviation of p99 RNL of misaligned traffic, then the mean p99 of -3.89 for PC traffic means that the 99^{th} percentile latency averaged over all the 1-hour windows for aligned traffic is lower than misaligned traffic by $3.89 \cdot s^c_{RNL,m}$. We see that while the alignment increases RNL latency for BE and NC traffic, this is a small fraction of the reduction in latency experienced by PC RPCs. The reduction in maximum p99 latency for PC traffic is over 150x (263.95 / 1.75) the increase in maximum p99 latency for NC RPCs.

Table 2 shows the same statistics for total RPC latency. Here we normalize by the standard deviation of p99 RPC latency for misaligned traffic, denoted $s^c_{RPC,m}$. We observe practically no change in the total latency numbers between aligned and misaligned traffic at p50 and p95 for BE and NC RPCs. Somewhat surprisingly however, the maximum p99 values see a reduction for both BE and NC traffic. While it is difficult to definitively determine the causes of these reductions, analysis suggests that a small fraction of BE and NC RPCs experience longer queuing delays in the QoS_h queues due to the high volume of traffic assigned to this queue (in the absence of Aequitas, more than 99% of client c's traffic uses this QoS class) and Aequitas is able to better isolate traffic in the different queues, causing less contention overall.

The table also highlights significant reductions in PC RPC latency numbers at all the measured percentiles. The standard deviation of p99 of RPC latency is an order of magnitude larger than the standard deviation of p99 RNL, so these reductions in total RPC latency are significantly larger than the corresponding reductions in RNL. This suggests that for many misaligned PC RPCs, network is the bottleneck, as is consistent with the results in Figure 10 of [32]. By assigning this traffic to the highest QoS we can effectively alleviate the bottleneck without introducing major latency degradation for traffic using the other priorities. The last row of Table 2 shows the same statistics, but aggregated across all 3 priorities. Here we see far less extreme values than those seen for PC RPCs as around 90% of client c's traffic uses BE or NC. However, we still observe a large reduction in maximum p99 RPC latency for aligned traffic compared to misaligned.

These results demonstrate that PC RPCs are far more sensitive to network latency as compared to BE and NC traffic. Thus, by assigning PC traffic to the highest QoS class, Aequitas can significantly reduce both RNL and overall RPC latency for this traffic with comparatively small increases for BE and/or NC RPCs. Moreover, other clients exhibited similar performance and we continued to observe behaviour consistent with these findings over longer time horizons and as the rollout reached 100% for various users across diverse clusters with different load characteristics.

4.3 Key Findings 2&3: Lower QoS does not imply Higher Latency, All can win

As discussed in §2.3, for QoS_i and QoS_j with $w_i > w_j$ it is possible for a client to experience lower latency on QoS_j, despite the lower weight, if QoS_i is significantly overloaded

[5] This is done to keep Google latency numbers confidential and to facilitate comparison across different clients and types of latency.

	QoS_m:QoS_l	QoS_h:QoS_l	QoS_h:QoS_m
Unhealthy Cutoffs	4:1	8:1	2:1
Mix in Cluster	10.69:1	0.52:1	5.54:1

Table 3: Unhealthy QoS mix thresholds and actual QoS mix in one cluster prior to Aequitas rollout

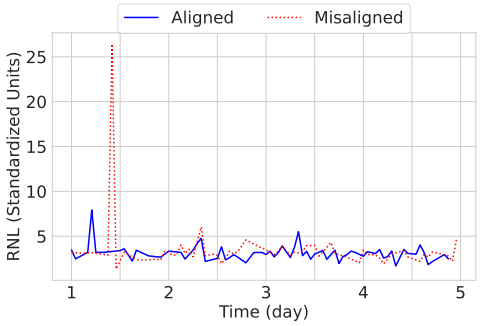


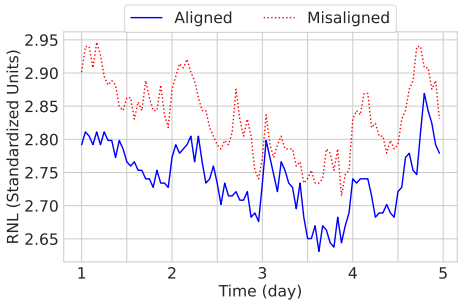
Figure 8: Aligned vs Misaligned p99 RNL for BE traffic of client $\bar{c}$

relative to QoS_j. Here we show a production example of this phenomenon. We piloted changes with a large Spanner client (denoted $\bar{c}$) that had previously exclusively used QoS_m across all three priorities. When approached for the pilot study, client $\bar{c}$ was very willing to participate and acknowledged that the current QoS settings had been decided through trial-and-error without any definitive guiding principle. The Aequitas team started a pilot in one cluster before expanding fleetwide for this user. This user was chosen as they have a high volume of traffic across several clusters that routinely experience high utilization. We study the result of a 50% rollout in one such cluster to highlight the behavior of interest.

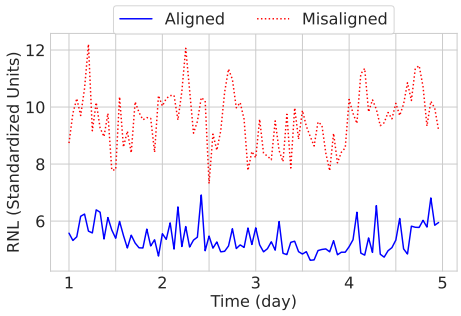
The ratio of weights for QoS_h:QoS_m:QoS_l is 8:4:1. Using the terminology of §2.3 the cluster has an unhealthy QoS mix between QoS_l and QoS_m if the volume of traffic on QoS_m is more than 4 times the amount of traffic on QoS_l. The first row of Table 3 shows the unhealthy QoS cutoffs for the 3 pairs of QoS bands. Ratios higher than these cutoffs correspond to an unhealthy mix. The second row shows the actual mix within the cluster, prior to enabling Aequitas at 50% for $\bar{c}$. QoS_m is severely overloaded relative to QoS_l. Aequitas downgrades BE traffic to QoS_l, so using the analysis of §2.3 we would expect this downgraded traffic to experience reduced latency compared to its misaligned counterpart.

Indeed, Figure 8 shows the p99 RNL for aligned and misaligned BE RPCs. For both types of BE traffic we compute a CDF every hour for a four day period and then normalize by $s^{\bar{c}}_{RNL,m}$, the p99 RNL of all misaligned traffic for client $\bar{c}$.

The average p99 BE for aligned traffic is lower than the corresponding misaligned traffic by $0.35 \cdot s^{\bar{c}}_{RNL,m}$. Moreover, the maximum across all p99 values is $18.51 \cdot s^{\bar{c}}_{RNL,m}$ lower for BE aligned traffic as compared to BE misaligned traffic. The standard deviation of 99^{th} percentile for BE traffic is $2.15 \cdot s^{\bar{c}}_{RNL,m}$ lower for aligned RPCs. Thus, we see that despite



(a) p50



(b) p99

Figure 9: Aligned vs Misaligned RNL aggregated across all priorities for client $\bar{c}$ at the (a) 50^{th} and (b) 99^{th} percentiles

the fact that aligned traffic uses QoS_l while misaligned traffic uses QoS_m, aligned traffic experiences lower latency as QoS_l is severely underloaded relative to QoS_m. In addition, the p99 BE RNL for aligned traffic is more stable over time as compared to misaligned RPCs, providing more predictability for $\bar{c}$ in terms of worst case behaviour of BE traffic.

Figure 9 shows the p50 and p99 for aligned and misaligned traffic sampled from the same user aggregated across all priorities. Again, these numbers are computed hourly over a four day period and normalized by $s^{\bar{c}}_{RNL,m}$. We see clear differences in performance between aligned and misaligned latency with the aligned percentiles showing lower latency values, even at p50. While the gap between aligned and misaligned RNL is not large at p50 (aligned RNL is $0.09 \cdot s^{\bar{c}}_{RNL,m}$ lower on average), the aligned RNL is always lower than its misaligned counterpart, while for p99 we see a larger separation. On average, the aligned values are $4.19 \cdot s^{\bar{c}}_{RNL,m}$ lower than the misaligned values. The maximum p99 BE RNL for aligned traffic is $5.29 \cdot s^{\bar{c}}_{RNL,m}$ lower than the maximum BE RNL for misaligned RPCs. Moreover, QoS queues can sometimes become overloaded due to a high volume of traffic at that QoS, leading to latency spikes. We consistently observed that these spikes were more common with misaligned (non-Aequitas) traffic vs the Aequitas counterpart which shows smoother, more predictable latency, as evidenced by the figure.

Comparing Figures 8 and 9b, we see that on average the reduction in aligned RNL across all priorities is larger than the reduction for BE traffic. However, the reduction in maximum

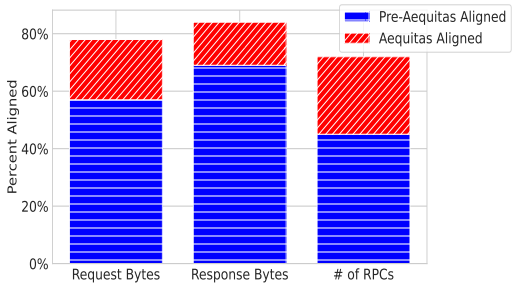


Figure 10: Aequitas Alignment Status of Spanner by Response bytes, Request bytes, and Number of RPCs

for BE traffic is larger as compared to the reduction across all priorities. This not only shows that we can decrease the latency of PC traffic through QoS upgrades but that Aequitas can also lead to improvements in worst-case behaviour for BE traffic by reassigning it to a lower, under-utilized QoS band. In the long run, this also benefits all traffic on the overloaded QoS_m band due to the reduced load at this QoS, lowering contention for RPCs in this queue. After the full rollout for client $\bar{c}$ we observe a 31.04% reduction in RNL for NC RPCs, despite the fact that these RPCs were already aligned before the Aequitas implementation. The results for BE and PC traffic after full rollout are consistent with those at 50% rollout.

This demonstrates that QoS alignment under Aequitas is not a pure zero-sum game and in fact all traffic under a particular user can benefit from better alignment. This is consistent with the theoretical justification given in Appendix A.

5 Fleetwide Performance

Previously, we provided examples of key findings from enabling Aequitas for a diverse set of clients. Here we discuss overall Aequitas performance across Google storage systems and argue that benefits of correct QoS alignment arise even under incremental deployment. Deployment results in [14] are related to traffic to/from the HDD LL storage systems. We have expanded Aequitas to also affect SSD LL storage traffic. The results for SSD LL storage traffic show that the improvements brought by correct QoS enforcement hold true across different layers of the storage stack. The detailed results are discuess in Appendix D.

As mentioned, Aequitas is implemented on the client-side for UL systems, making rollout much more gradual as compared to LL storage. Figure 10 shows the progress for a planet-scale UL system. The quantities for **Pre-Aequitas Aligned** represent the alignment status of the system in the absence of Aequitas, whereas the values for **Aequitas Aligned** represent the additional alignment gained through Aequitas deployment. While the marginal alignment through Aequitas can be small in some cases ($\sim$ 15% of Response bytes), without Aequitas the **Pre-Aequitas Aligned** RPC alignment is not enforced and it can change over time. As discussed in §2, we observe race-to-the-top QoS behavior with users selecting higher QoS for their jobs after SLO violations. Thus, we would expect to see the **Pre-Aequitas Aligned** numbers shrink over time without enforcement. With Aequitas, the alignment is the combination of **Pre-Aequitas Aligned** and **Aequitas Aligned** so that $\sim$ 72% of RPCs are aligned, representing $\sim$ 84% of all response bytes and $\sim$ 78% of all request bytes.

The Aequitas team conducted pilot studies for four of the five largest customers of Spanner and have made Aequitas the default in all clusters. However, some clients have developed a set of complex QoS assignment policies in collaboration with other internal teams and are not currently included in the Aequitas logic. We are currently tackling these exceptional cases, but even in the current state we have a large proportion of RPCs under Aequitas and clients can still see benefits. This shows the power of Aequitas even under partial rollouts.

The first pilot customer from Spanner was a large query service that operates across the fleet. This service can account for over 15% of all RPCs in the system and operates on PBs of data daily. Utilization across clusters varies with time. However, as this service is a crucial part of Google's infrastructure, it is exposed to varying load profiles across the fleet.

Figure 11 shows the priority distribution of the client's response traffic. The workloads are more response heavy (corresponding to reads) and the traffic is dominated by NC and PC RPCs with very little BE usage. The request traffic distribution is more PC-heavy due to the importance of reliable low-latency writes.

Figure 12 examines the QoS distribution of response traffic by priority for the service. The situation for request traffic is similar. The client almost exclusively uses QoS_m and QoS_h, even for its small number of BE RPCs. There is also a high degree of misalignment, especially in the case of BE and NC traffic. Figure 13 further shows that this misalignment varies across cells. There are many cells with a high degree of misalignment (up to 100%) regardless of whether we aggregate by request bytes, response bytes, or number of RPCs.

After Aequitas, almost no traffic is misaligned. The rollout process for this client was gradual, increasing the percentage of RPCs affected in several canary cells before expanding fleetwide. However, after complete rollout, the Aequitas team made a change to facilitate expanding the logic to other customers and inadvertently turned off Aequitas for this service. While this was unintentional, it allowed us to gather data just before enabling Aequitas again across all the clusters.

Figure 14a shows the p99 RNL hourly over a 4 day period for NC and PC RPCs prior to re-enabling Aequitas. Figure 14b shows the same data but for a 4 day period after the Aequitas rollout with the days ranging from 8 to 12 as this is a 4-day period exactly one week after the results in Figure 14a. Both plots are standardized by the value of PC latency. We omit BE numbers as there was almost no BE traffic in the pre-Aequitas period. By monitoring over a longer time horizon, we witnessed only a slight regression in BE RNL and there have been no episodes of intolerable latency in over 2

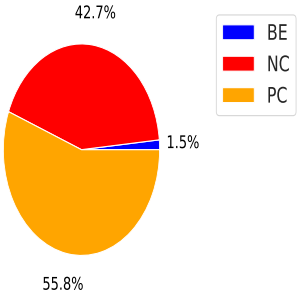


Figure 11: Priority Distribution of Query Service for Response Traffic

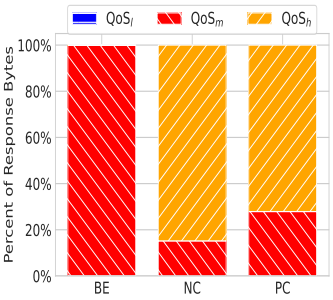


Figure 12: QoS Distribution of Response Traffic for Query Service

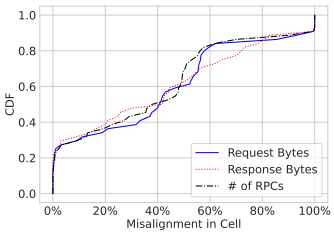
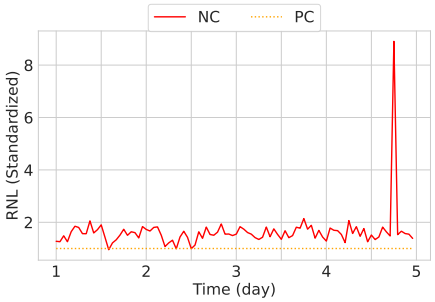
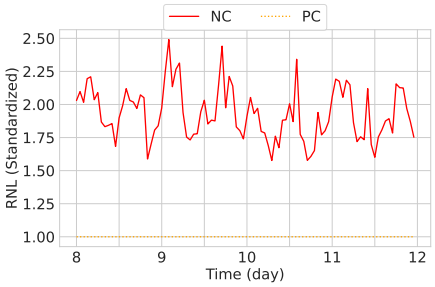


Figure 13: Query Service Misalignment Across Cells prior to Aequitas



(a) pre-Aequitas



(b) post-Aequitas

Figure 14: Latency by Priority (Standardized to PC Latency)

	Avg	Max	Std Dev
NC	16.24%	−68.91%	−68.62%
PC	−3.77%	−36.45%	−70.03%

Table 4: Percentage Change in p99 RNL Statistics for a Query Service Post-Aequitas (Negative Values Imply Improvement)

years since rollout. The small regression for BE is also much smaller than the improvement for PC.

Before Aequitas, the gap in latency between NC and PC RPCs was often small and at times NC p99 RNL was smaller than PC p99 RNL, indicating priority inversion. Post-rollout there is a clear separation in RNL between NC and PC RPCs with the p99 RNL for NC RPCs always more than 1.5x that of the p99 RNL for PC RPCs. We also see a point pre-Aequitas where NC RNL rose to more than 8x that of the PC RNL. Post-Aequitas, NC never reaches 2.5x higher. However, this only shows better separation in RNL between the 2 priorities.

Table 4 shows that the average, maximum, and standard deviation of PC p99 RNL is decreased post-rollout. While we witnessed a marginal increase in average NC p99 RNL, the maximum and standard deviation both decreased. Thus, Aequitas effectively differentiates between RPC priorities. It also leads to more stability overall with improvement in worst case p99 RNL for both PC and NC. The reductions in standard deviation are also expected as now each priority is mapped to a unique QoS, whereas in the absence of Aequitas the same priority can operate across different QoS.

Aequitas has been deployed to LL storage for more than 3 years whereas the query service has been using Aequitas for over 2.5 years. The results discussed have continued to be true over time, demonstrating the robustness of the scheme.

6 Discussion

Generalizability: As outlined in §2.2, Aequitas uses metadata from each storage system to classify RPCs as BE, NC, or PC. These RPC classes are then mapped 1:1 to the QoS classes QoS$_l$, QoS$_m$, and QoS$_h$, respectively. This is a flexible design in the sense that any available metadata can be used to classify RPCs into the three priority classes. In general, the Aequitas system accepts a set of RPC metadata features $x_0, x_1, ..., x_k$ from the application and uses these features to map the RPC to one of the priority classes.

This also gives a strategy to extend Aequitas to other Google systems; we simply classify traffic from these systems into a finite number of priorities that adhere to an ordering. These priorities are then mapped to QoS queues such that the distribution among the queues maintains a healthy QoS mix as described in §2.3. Similarly, Aequitas can be extended to networks outside of Google. In addition to the above, networks would need the ability to enforce differentiated service levels in the form of switch scheduling support.

Extending Aequitas to the WAN: The current implementation is restricted to RPCs having a source and destination in the same cluster. A promising future direction is to extend the implementation to the WAN but there are several features of WAN traffic that make this implementation more challenging. First, latencies inside a cluster are typically only a small

fraction of total latency. In contrast, network latency between clusters can be orders of magnitude larger, thus dominating the breakdown of total latency. Thus, changing QoS of such an RPC can have a profound affect on its latency. As such, changes that assign traffic to a lower QoS as compared to what they are currently using must be carefully managed.

Second, BwE [27] is responsible for bandwidth enforcement over the WAN. It allocates bandwidth to applications based on priority. From a network user's perspective this is seen as quota for the amount of traffic at a given priority level. In times of congestion, when usage exceeds capacity, traffic may need to be downgraded to a lower priority. Currently, this downgrading does not account for application-level considerations. So an Aequitas-for-WAN extension would have to work with BwE to ensure that the right QoS is set across various applications. Yet in doing so, Aequitas must be careful not to upgrade RPCs in such a way that the total usage at a given priority exceeds its assigned capacity, leading to BwE throttlings. This is an avenue for future work.

7 Related Work

Congestion Control: Typical congestion control algorithms, such as the Additive-Increase Multiplicative-Decrease (AIMD) approaches [33, 34] focus on trying to find a close-to-ideal tradeoff betwen high utilization and low delay. But these schemes wait for loss to occur and then adjust sending rates, adding unnecessary delays as they reach the send threshold. More recent schemes try to avoid loss altogether by inferring congestion from signals in the network: such as setting an indicator when buffers exceed a threshold [5] or using some combination of delay, estimated send time, and bottleneck link capacities to infer when applications are sending at too high of a rate [35–37]. Yet these schemes are reactive in nature, signalling back to clients to throttle their rates as queues start to build up. Recent efforts have accelerated this signalling by utilizing innovations in switch design [38], but still there is a delay between when congestion signals are sent and when they are reacted to. Aequitas takes a more proactive approach by encoding some application knowledge via network QoS. It can be run in conjunction with these schemes, allowing service owners to tune QoS weights to ensure critical traffic meets their SLOs in times of congestion.

Application-Network Integration: Coflow [21–23] was an early application-network integration idea that proposed informing the network of application semantics. However, this abstraction was tailored to the applications of the time and it cannot encode the end-to-end semantics of some of the more modern applications. Other solutions target an application-network API, providing a way for applications to inform the network of changes in sending rates or their SLOs [39–42].

These schemes target application integration across a general set of applications, but there has also been considerable work on optimizations for specific workloads [24–26, 43]. Many of these either require input from users in the form of workload patterns or use some form of offline profiling before sending flows and thus are better suited to long flows. Aequitas aligns to the priorities used by a small number of storage systems but does not require any input from the various services that use these systems[6]. Moreover, there are no additional profiling phases or other delays introduced by Aequitas, making it suitable for flows of all sizes.

Scheduling and Prioritization: There are many works that incorporate scheduling or prioritization schemes, both in the network domain and the wider systems community. SRPT is a scheduling algorithm that schedules shortest jobs first. It has been proven to minimize mean response times [44–46] and often performs very well in practice [6, 7, 47, 48]. Yet, it is often difficult to know flow or RPC sizes a priori. In response, a collection of SMART schemes were developed that approximate SRPT scheduling [49, 50] and others have aimed to separate large and small flows [51–55]. Even some of these techniques assume job/flow sizes are known in advance, or use techniques to estimate them. However, flow size alone is often not a sufficient predictor of a flow's network requirements [14]. Instead, Aequitas uses only the RPC priority to set QoS, without any knowledge of flow size.

8 Conclusion

In this work we detailed our experience of deploying Aequitas in Google's planet scale production network. This implementation of Aequitas is aimed at storage and enforces static mappings from RPC priorities to QoS classes for these systems. We present production examples showcasing that prioritizing through QoS assignment is not a zero-sum game and the ecosystem as a whole can benefit from the right policy. These findings are backed by theoretical analysis. While Aequitas requires no input from users, deploying it at the scale of our systems provides many challenges. We outline these concerns and our use of an incremental deployment approach. While this greatly extended the rollout timeline, the scheme was enabled in a safe manner that gave users confidence in its benefits. We hope that our experience can serve as a framework for QoS-based deployments in other networks, especially when an all-or-nothing approach is not possible.

Acknowledgements

The authors would like to thank Hassan Wassel, David Bacon, David Culler and the anonymous NSDI reviewers for their invaluable feedback and our shepherd Ankit Bhardwaj for guiding us through the revision process.

[6] While rolling out Aequitas was a long process at Google, this was because we wanted to be transparent to users and ensure that Aequitas performed as intended. But for an individual application, a user can simply opt-in and does not need to provide any information to the Aequitas system

References

[1] Rene L Cruz. A calculus for network delay. i. network elements in isolation. *IEEE Transactions on information theory*, 37(1):114–131, 1991.

[2] Rene L Cruz. Service burstiness and dynamic burstiness measures: A framework. *Journal of High Speed Networks*, 1(2):105–127, 1992.

[3] Theophilus Benson, Aditya Akella, and David A Maltz. Network traffic characteristics of data centers in the wild. In *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*, pages 267–280, 2010.

[4] David Zats, Tathagata Das, Prashanth Mohan, Dhruba Borthakur, and Randy Katz. Detail: Reducing the flow completion time tail in datacenter networks. In *Proceedings of the ACM SIGCOMM 2012 conference on Applications, technologies, architectures, and protocols for computer communication*, pages 139–150, 2012.

[5] Mohammad Alizadeh, Albert Greenberg, David A Maltz, Jitendra Padhye, Parveen Patel, Balaji Prabhakar, Sudipta Sengupta, and Murari Sridharan. Data center tcp (dctcp). In *Proceedings of the ACM SIGCOMM 2010 Conference*, pages 63–74, 2010.

[6] Mohammad Alizadeh, Shuang Yang, Milad Sharif, Sachin Katti, Nick McKeown, Balaji Prabhakar, and Scott Shenker. pfabric: Minimal near-optimal datacenter transport. *ACM SIGCOMM Computer Communication Review*, 43(4):435–446, 2013.

[7] Chi-Yao Hong, Matthew Caesar, and P Brighten Godfrey. Finishing flows quickly with preemptive scheduling. *ACM SIGCOMM Computer Communication Review*, 42(4):127–138, 2012.

[8] Mohammad Alizadeh, Tom Edsall, Sarang Dharmapurikar, Ramanan Vaidyanathan, Kevin Chu, Andy Fingerhut, Vinh The Lam, Francis Matus, Rong Pan, Navindra Yadav, et al. Conga: Distributed congestion-aware load balancing for datacenters. In *Proceedings of the 2014 ACM conference on SIGCOMM*, pages 503–514, 2014.

[9] Hong Zhang, Junxue Zhang, Wei Bai, Kai Chen, and Mosharaf Chowdhury. Resilient datacenter load balancing in the wild. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, pages 253–266, 2017.

[10] Sally Floyd and Van Jacobson. Random early detection gateways for congestion avoidance. *IEEE/ACM Transactions on networking*, 1(4):397–413, 1993.

[11] Rong Pan, Balaji Prabhakar, and Konstantinos Psounis. Choke-a stateless active queue management scheme for approximating fair bandwidth allocation. In *Proceedings IEEE INFOCOM 2000. Conference on Computer Communications. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies (Cat. No. 00CH37064)*, volume 2, pages 942–951. IEEE, 2000.

[12] Rong Pan, Lee Breslau, Balaji Prabhakar, and Scott Shenker. Approximate fairness through differential dropping. *ACM SIGCOMM Computer Communication Review*, 33(2):23–39, 2003.

[13] David Zats, Anand Padmanabha Iyer, Ganesh Ananthanarayanan, Rachit Agarwal, Randy Katz, Ion Stoica, and Amin Vahdat. Fastlane: making short flows shorter with agile drop notification. In *Proceedings of the Sixth ACM Symposium on Cloud Computing*, pages 84–96, 2015.

[14] Yiwen Zhang, Gautam Kumar, Nandita Dukkipati, Xian Wu, Priyaranjan Jha, Mosharaf Chowdhury, and Amin Vahdat. Aequitas: Admission control for performance-critical rpcs in datacenters. In *Proceedings of the ACM SIGCOMM 2022 Conference*, pages 1–18, 2022.

[15] Theophilus Benson, Ashok Anand, Aditya Akella, and Ming Zhang. Understanding data center traffic characteristics. *ACM SIGCOMM Computer Communication Review*, 40(1):92–99, 2010.

[16] Srikanth Kandula, Sudipta Sengupta, Albert Greenberg, Parveen Patel, and Ronnie Chaiken. The nature of data center traffic: measurements & analysis. In *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement*, pages 202–208, 2009.

[17] Soudeh Ghorbani, Zibin Yang, P Brighten Godfrey, Yashar Ganjali, and Amin Firoozshahian. Drill: Micro load balancing for low-latency data center networks. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, pages 225–238, 2017.

[18] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C Hsieh, Deborah A Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E Gruber. Bigtable: A distributed storage system for structured data. *ACM Transactions on Computer Systems (TOCS)*, 26(2):1–26, 2008.

[19] James C Corbett, Jeffrey Dean, Michael Epstein, Andrew Fikes, Christopher Frost, Jeffrey John Furman, Sanjay Ghemawat, Andrey Gubarev, Christopher Heiser, Peter Hochschild, et al. Spanner: Google's globally distributed database. *ACM Transactions on Computer Systems (TOCS)*, 31(3):1–22, 2013.

[20] Daniel Peng and Frank Dabek. Large-scale incremental processing using distributed transactions and notifications. In *9th USENIX Symposium on Operating Systems Design and Implementation (OSDI 10)*, 2010.

[21] Mosharaf Chowdhury and Ion Stoica. Coflow: A networking abstraction for cluster applications. In *Proceedings of the 11th ACM Workshop on Hot Topics in Networks*, pages 31–36, 2012.

[22] Mosharaf Chowdhury, Yuan Zhong, and Ion Stoica. Efficient coflow scheduling with varys. In *Proceedings of the 2014 ACM conference on SIGCOMM*, pages 443–454, 2014.

[23] Mosharaf Chowdhury and Ion Stoica. Efficient coflow scheduling without prior knowledge. *ACM SIGCOMM Computer Communication Review*, 45(4):393–406, 2015.

[24] MR Siavash Katebzadeh, Paolo Costa, and Boris Grot. Saba: Rethinking datacenter network allocation from application's perspective. In *Proceedings of the Eighteenth European Conference on Computer Systems*, pages 623–638, 2023.

[25] Weiyang Wang, Moein Khazraee, Zhizhen Zhong, Manya Ghobadi, Zhihao Jia, Dheevatsa Mudigere, Ying Zhang, and Anthony Kewitsch. {TopoOpt}: Co-optimizing network topology and parallelization strategy for distributed training jobs. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, pages 739–767, 2023.

[26] Yihao Zhao, Yuanqiang Liu, Yanghua Peng, Yibo Zhu, Xuanzhe Liu, and Xin Jin. Multi-resource interleaving for deep learning training. In *Proceedings of the ACM SIGCOMM 2022 Conference*, pages 428–440, 2022.

[27] Alok Kumar, Sushant Jain, Uday Naik, Anand Raghuraman, Nikhil Kasinadhuni, Enrique Cauich Zermeno, C Stephen Gunn, Jing Ai, Björn Carlin, Mihai Amarandei-Stavila, et al. Bwe: Flexible, hierarchical bandwidth allocation for wan distributed computing. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, pages 1–14, 2015.

[28] Ali Basiri, Niosha Behnam, Ruud De Rooij, Lorin Hochstein, Luke Kosewski, Justin Reynolds, and Casey Rosenthal. Chaos engineering. *IEEE Software*, 33(3):35–41, 2016.

[29] Colin Adams, Luis Alonso, Benjamin Atkin, John Banning, Sumeer Bhola, Rick Buskens, Ming Chen, Xi Chen, Yoo Chung, Qin Jia, et al. Monarch: Google's planet-scale in-memory time series database. *Proceedings of the VLDB Endowment*, 13(12):3181–3194, 2020.

[30] Mubashir Adnan Qureshi, Junhua Yan, Yuchung Cheng, Soheil Hassas Yeganeh, Yousuk Seung, Neal Cardwell, Willem De Bruijn, Van Jacobson, Jasleen Kaur, David Wetherall, et al. Fathom: Understanding datacenter application network performance. In *Proceedings of the ACM SIGCOMM 2023 Conference*, pages 394–405, 2023.

[31] Benjamin H Sigelman, Luiz André Barroso, Mike Burrows, Pat Stephenson, Manoj Plakal, Donald Beaver, Saul Jaspan, and Chandan Shanbhag. Dapper, a large-scale distributed systems tracing infrastructure. 2010.

[32] Korakit Seemakhupt, Brent E Stephens, Samira Khan, Sihang Liu, Hassan Wassel, Soheil Hassas Yeganeh, Alex C Snoeren, Arvind Krishnamurthy, David E Culler, and Henry M Levy. A cloud-scale characterization of remote procedure calls. In *Proceedings of the 29th Symposium on Operating Systems Principles*, pages 498–514, 2023.

[33] Van Jacobson. Congestion avoidance and control. *ACM SIGCOMM computer communication review*, 18(4):314–329, 1988.

[34] Sangtae Ha, Injong Rhee, and Lisong Xu. Cubic: a new tcp-friendly high-speed tcp variant. *ACM SIGOPS operating systems review*, 42(5):64–74, 2008.

[35] Neal Cardwell, Yuchung Cheng, C Stephen Gunn, Soheil Hassas Yeganeh, and Van Jacobson. Bbr: Congestion-based congestion control. *Communications of the ACM*, 60(2):58–66, 2017.

[36] Gautam Kumar, Nandita Dukkipati, Keon Jang, Hassan MG Wassel, Xian Wu, Behnam Montazeri, Yaogong Wang, Kevin Springborn, Christopher Alfeld, Michael Ryan, et al. Swift: Delay is simple and effective for congestion control in the datacenter. In *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*, pages 514–528, 2020.

[37] Radhika Mittal, Vinh The Lam, Nandita Dukkipati, Emily Blem, Hassan Wassel, Monia Ghobadi, Amin Vahdat, Yaogong Wang, David Wetherall, and David Zats. Timely: Rtt-based congestion control for the datacenter. *ACM SIGCOMM Computer Communication Review*, 45(4):537–550, 2015.

[38] Serhat Arslan, Yuliang Li, Gautam Kumar, and Nandita Dukkipati. Bolt:{Sub-RTT} congestion control for {Ultra-Low} latency. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, pages 219–236, 2023.

[39] Amir Baniamerian, Ali Munir, Ashkan Sobhani, Mahmoud Mohamed Bahnasy, Ron Thomas, Si Yan, Xingjun Chu, and Yashar Ganjali. Nce: an ecn dual mechanism to mitigate micro-bursts. In *Proceedings of the ACM SIGCOMM Workshop on Network-Application Integration*, pages 7–12, 2022.

[40] Seyed Hossein Mortazavi, Hossein Shafieirad, Mahmoud Bahnasy, Ali Munir, Yuanhui Cheng, Anudeep Das, and Yashar Ganjali. Accord: application-driven networking in the datacenter. In *Proceedings of the 14th IEEE/ACM International Conference on Utility and Cloud Computing*, pages 1–10, 2021.

[41] Seyed Hossein Mortazavi, Ali Munir, Mahmoud Mohamed Bahnasy, Haiwei Dong, Shimiao Wang, and Yashar Ganjali. Earlybird: automating application signalling for network application integration in datacenters. In *Proceedings of the ACM SIGCOMM Workshop on Network-Application Integration*, pages 40–45, 2022.

[42] Ali Munir, Seyed Hossein Mortazavi, Mahmoud Mohamed Bahnasy, Amir Baniamerian, Shimiao Wang, Shichao Guan, and Yashar Ganjali. Smarttags: bridging applications and network for proactive performance management. In *Proceedings of the ACM SIGCOMM Workshop on Network-Application Integration*, pages 46–52, 2022.

[43] Timothy Zhu, Alexey Tumanov, Michael A Kozuch, Mor Harchol-Balter, and Gregory R Ganger. Prioritymeister: Tail latency qos for shared networked storage. In *Proceedings of the ACM Symposium on Cloud Computing*, pages 1–14, 2014.

[44] Linus Schrage. A proof of the optimality of the shortest remaining processing time discipline. *Operations Research*, 16(3):687–690, 1968.

[45] Donald R Smith. A new proof of the optimality of the shortest remaining processing time discipline. *Operations Research*, 26(1):197–199, 1978.

[46] Nikhil Bansal and Mor Harchol-Balter. Analysis of srpt scheduling: Investigating unfairness. In *Proceedings of the 2001 ACM SIGMETRICS International conference on Measurement and modeling of computer systems*, pages 279–290, 2001.

[47] Mor Harchol-Balter, Bianca Schroeder, Nikhil Bansal, and Mukesh Agrawal. Size-based scheduling to improve web performance. *ACM Transactions on Computer Systems (TOCS)*, 21(2):207–233, 2003.

[48] Isaac Grosof, Ziv Scully, and Mor Harchol-Balter. Srpt for multiserver systems. *ACM SIGMETRICS Performance Evaluation Review*, 46(2):9–11, 2019.

[49] Adam Wierman, Mor Harchol-Balter, and Takayuki Osogami. Nearly insensitive bounds on smart scheduling. *ACM SIGMETRICS Performance Evaluation Review*, 33(1):205–216, 2005.

[50] Chang-Woo Yang, Adam Wierman, Sanjay Shakkottai, and Mor Harchol-Balter. Tail asymptotics for policies favoring short jobs in a many-flows regime. In *Proceedings of the joint international conference on Measurement and modeling of computer systems*, pages 97–108, 2006.

[51] Mohammad Al-Fares, Sivasankar Radhakrishnan, Barath Raghavan, Nelson Huang, Amin Vahdat, et al. Hedera: dynamic flow scheduling for data center networks. In *Nsdi*, volume 10, pages 89–92. San Jose, USA, 2010.

[52] Wei Bai, Li Chen, Kai Chen, Dongsu Han, Chen Tian, and Hao Wang. {Information-Agnostic} flow scheduling for commodity data centers. In *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*, pages 455–468, 2015.

[53] Abdullah Bin Faisal, Hafiz Mohsin Bashir, Ihsan Ayyub Qazi, Zartash Uzmi, and Fahad R Dogar. Workload adaptive flow scheduling. In *Proceedings of the 14th International Conference on emerging Networking EXperiments and Technologies*, pages 241–253, 2018.

[54] Peter X Gao, Akshay Narayan, Gautam Kumar, Rachit Agarwal, Sylvia Ratnasamy, and Scott Shenker. phost: Distributed near-optimal datacenter transport over commodity network fabric. In *Proceedings of the 11th ACM Conference on Emerging Networking Experiments and Technologies*, pages 1–12, 2015.

[55] Behnam Montazeri, Yilong Li, Mohammad Alizadeh, and John Ousterhout. Homa: A receiver-driven low-latency transport protocol using network priorities. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, pages 221–235, 2018.

Appendices

A QoS Arbitrage

Here we aim to answer the question posed in §2.3: *what are the conditions under which a user should prefer one QoS class over another?* To answer this question we consider a Weighted Fair Queue (WFQ) with k QoS classes QoS_0, ..., QoS_{k-1} having weights w_0, ..., w_{k-1}, respectively. That is, the long run average service rate for QoS_i is $\frac{w_i}{w_0+\ldots+w_{k-1}}$. Note that a WFQ is work-conserving, thus we can assume the queues corresponding to QoS_0, ..., QoS_{k-1} are non-empty.

Without loss of generality, we will relabel the queues so that we consider the queues corresponding to QoS_0 and QoS_1 with $w_0 > w_1$ (note here we assume $>$ and not $\geq$ because if $w_0 = w_1$ then there is no differentiation between the QoS classes). So in this setting QoS_1 represents a lower priority than QoS_0. Now suppose that the current loads on QoS_0 and QoS_1 are λ_0 and λ_1, respectively. Race-to-the-top arises from the belief that for a particular client c, c's traffic on QoS_0 will experience lower delay as compared to c's traffic on QoS_1. This assumption is due to the fact that $w_0 > w_1$, however, we will show that this is not true in all cases, and depends on the load on both QoS classes.

If client c's traffic on QoS_0 experiences lower latency compared to c's traffic on QoS_1 then this is the desired outcome and we denote this by $\text{QoS}_0 \succ \text{QoS}_1$. If on the other hand, c's traffic experiences worse performance on QoS_0, we refer to this as priority inversion, which is denoted by $\text{QoS}_0 \prec \text{QoS}_1$. Let client c's load on QoS_0 and QoS_1 be β_0^c and β_1^c, respectively. Note that $\beta_0^c \leq \lambda_0$ and $\beta_1^c \leq \lambda_1$ as the client's load on a QoS class is lower than the total load on a QoS class.

Define an epoch to be a unit of time in which each QoS class is served in proportion to its weight. If we assume that a client's traffic is independent from the rest of the traffic in the queue[7] then we can reasonably assume that client c's traffic is evenly distributed in the queue[8]. That is, in a period in which QoS_i is served, we can expect $\frac{\beta_i^c}{\lambda_i}$ proportion of that traffic belongs to client c. Then of all the traffic served from QoS_0 and QoS_1 during the epoch the proportion that belongs to client c (regardless of how much traffic is in other QoS bands) is given by

$$w_0 \frac{\beta_0^c}{\lambda_0} + w_1 \frac{\beta_1^c}{\lambda_1} \tag{2}$$

Now suppose that we shift some small amount ε of client c's traffic from QoS_0 to QoS_1. Then the amount of traffic on QoS_0 and QoS_1 now becomes $\lambda_0 - \varepsilon$ and $\lambda_1 + \varepsilon$, respectively. Whereas, the amount of client c's traffic on these two queues is given by $\beta_0^c - \varepsilon$ and $\beta_1^c + \varepsilon$, respectively. Under this new distribution, equation (2) becomes

$$w_0 \frac{\beta_0^c - \varepsilon}{\lambda_0 - \varepsilon} + w_1 \frac{\beta_1^c + \varepsilon}{\lambda_1 + \varepsilon} \tag{3}$$

Note that client c's traffic distribution in the other $k-2$ queues did not change. So if the quantity in equation (3) is larger than the quantity in equation (2), then of all traffic served in the epoch, a larger proportion of this traffic belongs to client c after the redistribution, and thus client c receives better service. That is, more of client c's traffic is served in a given epoch if

$$\begin{aligned} & w_0 \frac{\beta_0^c - \varepsilon}{\lambda_0 - \varepsilon} + w_1 \frac{\beta_1^c + \varepsilon}{\lambda_1 + \varepsilon} > w_0 \frac{\beta_0^c}{\lambda_0} + w_1 \frac{\beta_1^c}{\lambda_1} \\ \implies & w_1 \left[\frac{\beta_1^c + \varepsilon}{\lambda_1 + \varepsilon} - \frac{\beta_1^c}{\lambda_1} \right] > w_0 \left[\frac{\beta_0^c}{\lambda_0} - \frac{\beta_0^c - \varepsilon}{\lambda_0 - \varepsilon} \right] \\ \implies & w_1 \left[\frac{\lambda_1 - \beta_1^c}{\lambda_1 (\lambda_1 + \varepsilon)} \right] > w_0 \left[\frac{\lambda_0 - \beta_0^c}{\lambda_0 (\lambda_0 + \varepsilon)} \right] \end{aligned}$$

Note that the last equation precisely matches (2.3) in §2.3. Moreover, $\lambda_0 - \beta_0$ is the total amount of traffic on QoS_0 across all customers excluding client c, and similarly, $\lambda_1 - \beta_1$ is the total amount of traffic on QoS_1 across all customers excluding client c. Also notice that there is an admissible region for equation (2.3): Take $w_0 = 0.6$, $\lambda_0 = 70$Gbps, $\beta_0 = 20$Gbps, $w_1 = 0.1$, $\lambda_1 = 5$Gbps, $\beta_1 = 0.1$Gbps, $\varepsilon = 1$Gbps. Then the left hand side of equation (2.3) simplifies to 0.0163 while the left hand side is 0.0060. Thus, even though $w_0 > w_1$, it is advantageous for the client to transfer a small amount of traffic from QoS_0 to QoS_1, due to the low load on QoS_1 relative to its weight.

B Unhealthy QoS

In this section, we examine the state of QoS in Google storage systems. We are specifically interested in clusters with an unhealthy QoS mix, as defined in §2.3. Aequitas is targeted towards the queues for QoS_l, QoS_m, and QoS_h so that is what we focus on here as well.

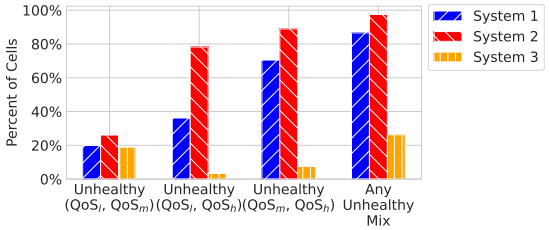
Figure 15 shows the proportion of clusters that have an unhealthy QoS mix across 3 UL storage systems, in the absence of Aequitas. We show the results for the three pairs of QoS classes[9] as well as a column that shows the percentage of cells that have an unhealthy mix between any of the three pairs. The figure contains results for both request and response traffic. For example, in the absence of Aequitas, System 2 request traffic has an unhealthy (QoS_h, QoS_m) mix in 89% of the clusters in which it is located, meaning that System 2's ratio of QoS_h to QoS_m request traffic in 89% of clusters is higher than $w_h : w_m$. Similarly, System 2 request traffic has an unhealthy mix for at least one pair of QoS classes in 97% of the clusters in which it is located.

As can be seen from the figure, a significant proportion of clusters have an unhealthy mix for at least one pair of QoS classes, especially for systems 1 and 2. An unhealthy mix between QoS_h and QoS_m is the biggest culprit, but there are instances of unhealthy QoS across all three pairs. This is in line with the notion of race-to-the-top discussed in [14], which suggests that users gravitate towards higher QoS over time.

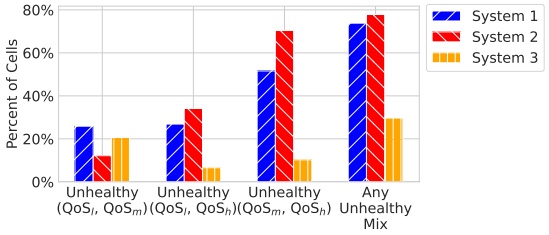
[7] While this assumption does not apply in every scenario, due to Google's scale we can reasonably assume one user's traffic is independent from other traffic in the datacenter

[8] Note that this does not assume that client c's traffic is necessarily smooth over time, it can occur in bursts. This just assumes a relatively small period of time in which client c's traffic is roughly evenly distributed among the rest of the traffic destined to that queue

[9] The QoS classes are naturally ordered by their weights; this also implies an ordering on the pairs



(a) Request Traffic



(b) Response Traffic

Figure 15: Percent of clusters with priority inversion across three UL systems based on (a) request traffic and (b) response traffic

C Client-side vs. Server-side Implementation

As detailed in section 2.2, Aequitas is implemented on the server-side for LL storage, whereas we used a client-side approach for UL storage implementations. The choice of where to implement depends on many factors that were discussed previously. Here we focus on how this affects the rollout process and analysis of the results.

A server-side implementation offers several advantages. First, it facilitates faster rollout and adoption as all traffic using a particular server is affected by Aequitas as soon as the implementation is deployed to the server. Thus the rollout is able to progress at the speed at which the servers are updated with the latest changes. This is generally much faster than client-side updates as the servers are controlled by a small number of teams that can coordinate simultaneous upgrades. Second, as seen in [14] and shown in section 2.3, the performance of traffic using a certain QoS class depends on both the weight of that QoS class and the load of all traffic passing through the same switch queues at that particular QoS. Server-side implementation implies that all traffic using the same server is affected simultaneously. Thus we see large-scale changes with each server updated and the behaviour of the system converges to the Aequitas QoS distribution quickly, making it possible to assess user-level and cluster-level performance changes more easily.

In contrast, for client-side implementations, the rollout process is much slower. It is far more difficult to coordinate simultaneous upgrades across users due to the various teams involved and the different jobs that these users are running. For example, some clients have long running jobs that take on the order of months to complete and thus will not pick up changes until these jobs finish. At the same time, there are other clients that can pick up changes almost immediately. This results in many incremental QoS changes over a long time period, making it difficult to see the impact at a datacenter level. Moreover, for clients who pick up the changes early, their QoS changes as a result of Aequitas can be a small portion of the overall QoS in the datacenter. As a result, it can be difficult to assess the long-term impact for an individual client as most of the datacenter remains unaffected by Aequitas and the client's performance is likely to change as the datacenter QoS distribution changes due to increased adoption of Aequitas. As discussed in §2.2 and seen in §4, this required extra considerations to facilitate a smooth rollout while carefully teasing out the effect of Aequitas.

D Aequitas for SSD LL Systems

In this section, we expand on the results for deploying Aequitas on LL systems previously presented in [14]. The production deployment results explained in [14] were presented for HDD storage traffic between LL and UL systems, wheres since then, we have expanded Aequitas to cover SSD traffic to/from LL systems as well. Here, we present the results related to SSD LL systems performance.

At a high level, $\sim$ 30% of all LL-bound SSD related RPCs were misaligned prior to Aequitas and this number is now $\sim$ 0%[10]. Prior to Aequitas deployment, 34%, 24% and 90% of PC, NC and BE.

The reduction in aggregate misalignment for RPCs, among clusters, varies. There are cluster with close to 100% misalignment, whereas the smallest percentage of misalignment in a single cell, observed during the sampling, is around 10%.

After expanding Aequitas to cover SSD LL system traffic, one of the users with most misaligned RPCs sees significant changes to their PC RPC latency. The average RPC latency is improved by 6.4% while the 99th percentile of RPC latency is improved by 11.2% for this user. Furthermore, we measured improvement of 2.6% and 5.4% for average and 99th percentile of NC RPC latency respectively. This is inline with our goal of having positive overall performance incentive for applications as the corresponding user experiences improvement in both of their latency sensitive RPC priorities.

The aggregated fleet-wide results, also show improvements after full deployment of Aequitas. The average PC RPC latency has reduced by 1.8% and the 99th percentile has reduced by 4.8%. These results, along results in [14] and results presented in rest of this paper for UL storage systems, show the robustness of our learning. That in fact, the correct QoS enforcement holds true consistently across different layers of storage systems stack.

[10]The number of misaligned LL-bound RPCs is $\sim$ 0% as Google has an intricate web of applications and a vast array of jobs running from users around the globe. As a result, while we have completed the Aequitas deployment for LL storage, there can be transient traffic that operates outside of the Aequitas framework