Preventing Network Bottlenecks: Accelerating Datacenter Services with Hotspot-Aware Placement for Compute and Storage

Hamid Hajabdolali Bazzaz[1], Yingjie Bi[1], Weiwu Pang[1], Minlan Yu[2], Ramesh Govindan[3], Neal Cardwell[1], Nandita Dukkipati[1], Meng-Jung Tsai[1], Chris DeForeest[1], Yuxue Jin[1], Charlie Carver[4], Jan Kopański[1], Liqun Cheng[1], Amin Vahdat[1]

[1]*Google*, [2]*Harvard University*, [3]*University of Southern California*, [4]*Columbia University*

Abstract

Datacenter network hotspots, defined as links with persistently high utilization, can lead to performance bottlenecks. In this work, we study hotspots in Google's datacenter networks. We find that these hotspots occur most frequently at ToR switches and can persist for hours. They are caused mainly by *bandwidth demand-supply imbalance*, largely due to high demand from network-intensive services, or demand exceeding available bandwidth when compute/storage upgrades outpace ToR bandwidth upgrades. Compounding this issue is *bandwidth-independent task/data placement* by datacenter compute and storage schedulers. We quantify the performance impact of hotspots, and find that they can degrade the end-to-end latency of some distributed applications by over 2× relative to low utilization levels. Finally, we describe simple improvements we deployed. In our cluster scheduler, adding hotspot-aware task placement reduced the number of hot ToRs by 90%; in our distributed file system, adding hotspot-aware data placement reduced p95 network latency by more than 50%. While congestion control, load balancing, and traffic engineering can efficiently utilize paths for a fixed placement, we find hotspot-aware placement – placing tasks and data under ToRs with higher available bandwidth – is crucial for achieving consistently good performance.

1 Introduction

Distributed cloud applications and services rely on high-bandwidth networks to provide high-throughput, low-latency data transfers. Ideally, the datacenter network provides applications with the illusion of an *unconstrained network* — one in which applications receive high throughput independent of the placement of their components across the network. Techniques such as topology design [41] and/or topology and traffic engineering [32] can support this ideal.

In practice, however, the datacenter network is not unconstrained at all places and all times. In our study of production networks (§2), there exist *network hotspots* — links with persistently high utilization — both within the datacenter interconnect and at top-of-the-rack (ToR) switches (§3). At Google, most hotspots occur at ToR switches and can persist for tens of minutes to hours. ToRs are split between compute hosts managed by Borg [45] and storage hosts. A dominant fraction of traffic in our clusters[1] can be attributed to our distributed storage stack. This includes the Colossus distributed file system [17], the *RamStore*, an in-memory store similar to RamCloud [29], *QuerySys*, an internal query processing system similar to [5, 27, 36, 44], and Bigtable [18]. Colossus places chunks on storage racks, and *QuerySys* worker tasks (§4.2) access these over the network.

Datacenter network planning balances ToR bandwidth supply with anticipated compute and storage traffic demand. Due to the high cost of non-blocking bandwidth provisioning, statistical multiplexing is employed, resulting in an oversubscription ratio – the ratio of ToR downlink to uplink bandwidth. This ratio, set at deployment, typically remains fixed throughout a ToR's multi-year lifespan, based on initial workload forecasts. Consequently, hotspots emerge from two primary, interconnected causes (§3). The first is rack *bandwidth demand-supply imbalance*: an imbalance between bandwidth demand from compute or storage provisioned on a rack, and the provisioned ToR bandwidth supply. This imbalance may occur under specific extreme traffic patterns, such as large incasts, due to the inherent oversubscription ratio. Alternatively, it can arise unexpectedly from deviations in either demand or supply compared to initial provisioning forecasts. Demand-side changes might stem over time from inorganic increased bandwidth demands of network-intensive workloads or other workload efficiency improvements. Supply-side changes could be triggered by incremental upgrades that increase compute or storage capacity without corresponding ToR uplink upgrades, or by planned/unplanned ToR uplink outages that reduce available bandwidth temporarily. The second reason is *bandwidth-independent task/data placement* by datacenter compute or storage schedulers assuming an unconstrained network. Datacenter scheduling is primarily focused on host resources, like compute and storage, that can be evaluated independently for each scheduling decision (because the network was designed to typically not be a bottleneck). Scheduling for network load is a much harder problem, since a hotspot is a property of a set of hosts and not attributable to any individual host. Further, characterizing the bursty communication demands of a distributed job is

[1]In this paper, our scope is traditional compute and storage, excluding ML clusters.

challenging relative to the per-task CPU and memory requirements. Hence, for simplicity, our initial job schedulers did not consider network requirements in scheduling decisions. This could in turn lead to situations where the tasks or storage chunks on a rack had sufficient traffic demand to exceed the provisioned ToR bandwidth supply.

Highly utilized ToRs can degrade the performance of our distributed storage systems (§3.4). For instance, Colossus hard disk drive (HDD) write latencies are inflated by 2× relative to an unloaded ToR even at 40% ToR utilization. Higher levels of the storage stack are also sensitive to ToR hotspots; for example, query latencies in *QuerySys* are inflated by 1.5× at utilizations ranging from 75–95%. It is surprising that these systems are so sensitive to ToR utilization. They access high-latency storage devices like HDDs and can incur significant compute latency when, for example, aggregating query results. Clearly, the increased network latency due to high ToR utilization starts to dominate these other latency components at some utilization level.

Application impact from ToR hotspots cannot be avoided with traditional network performance techniques like congestion control and traffic engineering, since these solutions cannot provide more ToR bandwidth. Hotspots *can*, in theory, be addressed by maintaining ToR resource-balance at every instant but is hard and expensive to do in practice (§3.3). It is much easier to replace a few failed disks on a rack with new higher-capacity disks (resulting in an imbalance) than it is to upgrade ToR capacity to restore balance. Instead, we have to change the input traffic matrix by placing work in a hotspot-aware manner.

In this paper, we modified compute and storage schedulers (Borg and Colossus) to account for resource-imbalance and ToR utilization when making task and storage chunk placements. This is challenging because the scheduler and file system are complex software systems that simultaneously balance a range of competing objectives: ensuring performance for applications with diverse workloads, high utilization, load balance, failure tolerance, and so on.

Despite this, for two reasons, we were able to introduce relatively simple modifications to the scheduler and the file system to mitigate the latency impact of high ToR utilization. First, end-to-end application-perceived query performance is only sensitive to ToR utilization above 75% or so. Second, across our fleet, the *average* ToR utilization is low, so many ToRs are well below 75% utilization. Our *hotspot-aware* task placement (§5) on Borg proactively places tasks on low utilization ToRs, and reactively migrates them away from hotspot ToRs. Similarly, our *hotspot-aware* chunk placement biases storage chunk placement away from those imbalanced racks with less capacity.

These mechanisms reduced the number of hotspot ToRs by 90% in our network without regressing scheduler and file system objectives. Furthermore, they significantly reduce storage access and query latency. We reduced Colossus p95 network latency by 50–80% and p95 total latency by 30–60%. Similarly, *QuerySys* shuffle flush and materialize benchmarks achieve latency reductions of 13% and 9%, respectively.

This paper makes three key contributions:

1. We demonstrate that ToR hotspots remain an unsolved but critical problem, originating from rack bandwidth demand-supply imbalance in dynamic network environments with evolving workloads, incrementally increasing compute/storage resources, and ToR uplink bandwidth prone to planned/unplanned outages. The ToR ↔ aggregation block cross-section is the most common location for hotspots inside our datacenters, and these hotspots can persist for extended durations, ranging from minutes to days. (§3).
2. We show that these hotspots can significantly impact application performance, even for operations dependent on disk or compute latency. Specifically, ToR hotspots can double the latency of both simple storage read/write operations and complex application operations (§4).
3. We find that simple augmentations to existing compute and storage schedulers to better balance network load in a best-effort manner significantly reduce ToR hotspots and their impact on application latency without the added complexity of treating the network as a fully managed resource (§5).

2 Background

2.1 Datacenter Network

Clos topology. Google's datacenter network is based on a hierarchical design (Figure 1) that uses Clos topologies to provide high-bandwidth interconnectivity between servers [41]. The lowest level of the network consists of servers mounted on racks. A top-of-rack (ToR) switch connects to all the servers within the rack. Each ToR switch connects to an aggregation switch. A group of interconnected aggregation switches forms an aggregation block. In turn, aggregation blocks connect to the datacenter network interconnect (DCNI). The DCNI provides links that directly connect different aggregation blocks or indirectly connect them through spine blocks [32, 41]. The DCNI connects to a fabric border router (FBR) that provides connectivity to other datacenters or WAN.

Rack types. To simplify procurement, deployment, and subsequent lifecycle management, Google's datacenters contain different types of racks. A *Disk* rack contains only storage appliances, servers dedicated to I/O using hard disk drives (HDDs) or solid-state drives (SSDs). Optimized for high-capacity storage, they store and serve data for Google's entire suite of user-facing services, as well as analytics, backup, and archival applications. A *Compute* rack contains compute resources, such as servers, with minimal local storage. Finally, a *Mixed* rack contains both compute and storage resources.

2.2 The Storage Stack

Several storage systems provide Google's applications with a range of interfaces to storage hardware. These systems

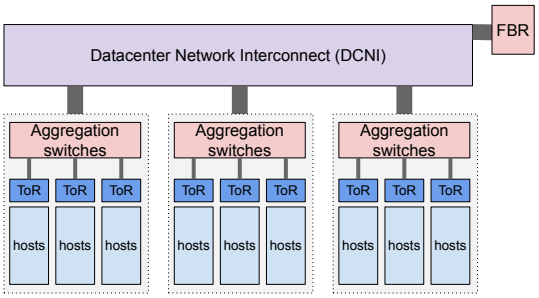


Figure 1: The typical topology of a datacenter network deployed at Google.

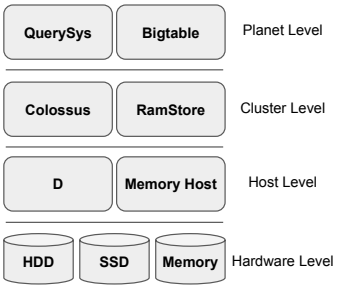


Figure 2: The storage stack at Google.

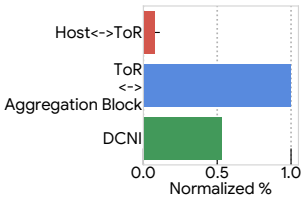


Figure 3: The percentage of hot links at different cross-sections of the network, normalized by the percentage at the ToR ↔ aggregation block layer.

collectively constitute the *storage stack* (Figure 2).

At the lowest level, *D* provides access to chunks of data residing on a single server, but it does not have an API to access user-level files or provide reliability guarantees.

Colossus [17], a distributed file system, is a successor to, and shares some architectural heritage with, the Google File System [16]. It breaks files up into encoded units called *stripes*. Each stripe has a particular encoding, perhaps replicated or erasure-coded, and is comprised of *chunks*, which it stores on *D*. Colossus transparently manages chunk replication and/or encoding, scalably updates filesystem metadata, and ensures adherence to consistency semantics during concurrent writes and appends. Clients access Colossus through a client library which obtains metadata from Colossus and directly accesses chunks, or parts of chunks, from *D* servers.

RamStore, a distributed in-memory file system (similar to [29]) built upon *memory hosts*, is primarily used to stage data during a *shuffle* [10]. Database grouping and join operations frequently use this form of collective communication, and an in-memory file system enables fast shuffles.

At the top of the stack, *QuerySys*, an internal query processing system similar to [5, 27, 36, 44], provides a query interface to distributed storage. *QuerySys* stores persistent data in Colossus, and uses *RamStore* to stage intermediate data during query execution. Multiple such systems with varying APIs and features exist, including Bigtable [18], another database service built on the same infrastructure.

3 Hotspot Characterization

In this section, we characterize the prevalence and duration of network hotspots in Google's network, discuss their causes, and present examples of their impact. We begin by defining hotspots and describing our methodology.

3.1 Methodology and Metrics

To characterize hotspots, we use telemetry data from switches, which provides the average utilization of each network link over 5-minute intervals. A link is a *hotspot* in an interval if its utilization during that interval is higher than 75%. Since a hotspot might occur on either of the two directions of a given link, we treat each direction separately in our characterization.

We choose a 75% threshold for two reasons. First, we [32, 41, 50] and others [14, 39, 47] maintain a 25% capacity headroom to minimize application impact during maintenance, upgrade, and failure mitigation. For this reason, we use a 75% threshold to determine a hotspot. Second, applications we study in this paper exhibit significant latency increases when network utilization exceeds this threshold (§4.1).

3.2 Prevalence

Figure 3 shows the percentage of hot links (i.e., the *hotspot rate*) across three types of network links in different cross-sections: host ↔ ToR, ToR ↔ aggregation block, and links within the DCNI, aggregated from the link utilization data over a month across the entire fleet.

ToR uplinks have the highest hotspot rate. As we have observed that ToR ↔ aggregation block links are ∼10x more likely to be hot than host ↔ ToR links and ∼2x more than DCNI links, ToR ↔ aggregation block hotspots have the highest traffic impact and are the main focus of this paper. Moreover, ToR hotspots are evenly divided across upstream and downstream directions (results omitted for brevity), so solutions cannot focus on a specific traffic direction. In what follows, the term hotspot always refers to a ToR uplink (i.e., ToR ↔ aggregation block) hotspot in either direction unless otherwise stated.

ToR hotspots span a large range of durations. Figure 4 shows the distribution of the duration for all ToR hotspots in a month. Here, the *duration* of a hotspot is defined as the total length of successive 5-minute time intervals during which the ToR uplink is consistently hot. About a third of the hotspots last less than 1 hour, while nearly 40% of them last 1–12 hours. Other timescales show non-trivial persistence as well; for example, about 13% of hotspots last between 1 day and 2 weeks. This wide range of persistence suggests that a variety of approaches might be necessary to address hotspots: reactive job migration at smaller time scales (minutes), proactive placement of data and network-intensive long-running jobs at medium timescales (hours), and capacity planning and re-provisioning at longer timescales (days or longer).

Storage traffic accounts for most ToR hotspots. Figure 5 shows the distribution of hotspots across different types of racks in the same month. *Mixed* racks, which contain both storage and compute (§2), account for nearly 50% of hotspots,

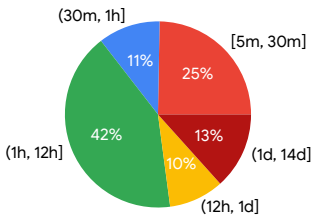


Figure 4: The distribution of the duration of hotspots.

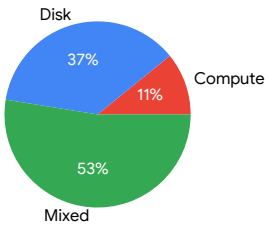


Figure 5: The distribution of hotspots by rack type.

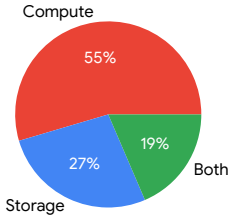


Figure 6: The attribution of hotspots to different workloads on *Mixed* racks.

and *Disk* racks for nearly 40%. Thus, between them, racks with storage devices account for about 90% of hotspots.

To understand which type contributes more to hotspots on *Mixed* racks, we first calculate the share of the capacity provisioned for storage and compute for a given rack based on the configured CPU and disks on the rack. Then, we attribute a hotspot on a *Mixed* rack to storage (compute) if the average throughput of storage (compute) traffic divided by the corresponding provisioned capacity is larger than 75%. From Figure 6, both compute and storage jobs contribute significantly to hotspots on *Mixed* racks; nearly 45% of hotspot traffic on a *Mixed* rack can be attributed to storage.

Taken together with Figure 5, this data suggests that storage traffic contributes significantly to hotspots. Prior work [49] also observes that storage read/write traffic constitutes 75% of the application traffic in Google's datacenters.

3.3 What Causes ToR Hotspots

ToR hotspots occur in our network because of a combination of two factors: *bandwidth demand-supply imbalance* on racks, and *bandwidth-independent task/data placement* by cluster schedulers and distributed storage systems.

Rack bandwidth demand-supply imbalance. Datacenter network planning involves *balancing* ToR bandwidth supply with anticipated traffic demand. For compute racks, bandwidth provisioning targets a specific bandwidth per normalized compute unit. For storage racks, link capacity is provisioned based on HDD capacity on the rack.

Over time, as datacenter ecosystems evolve, demand and supply can diverge from targets, resulting in *bandwidth demand-supply imbalance*. On a compute rack, new network-intensive applications with unprecedented network demands, efficiency improvements in existing applications, or server hardware refreshes can increase bandwidth demand beyond design targets and result in *compute bandwidth demand-supply imbalance*. On a storage rack, increases in installed storage capacity, or incremental replacement of existing or failed disks with faster ones, can increase storage supply without increasing bandwidth supply proportionally, likewise increasing bandwidth demand beyond design targets and resulting in *storage bandwidth demand-supply imbalance*.

These imbalances can persist for days, weeks, or months until network operators restore balance. We argue that these imbalances are inherent to cost-effective datacenter management. It is easier to upgrade or replace a few servers or disks on a rack, but harder to upgrade ToR capacity, since the latter requires taking all ToR-hosted compute and storage offline, and may require aggregation block upgrades as well.

Bandwidth-independent task and data placement. By themselves, rack resource imbalances do not necessarily cause ToR hotspots. ToR hotspots arise, additionally, because our cluster scheduler and our distributed file system make *bandwidth-independent task/data placement* decisions. This was a deliberate choice. At the scale of our datacenters, designing bandwidth-aware cluster scheduling and file systems while also satisfying performance, reliability, and utilization objectives was (and is) considered extremely hard. We have instead attempted to provide the illusion of an unconstrained network to these systems (§1).

As a result of bandwidth-independent task/data placement, these systems can create ToR hotspots. Consider our cluster scheduler, Borg, which schedules, on compute racks, worker tasks belonging to storage systems like *QuerySys*, *RamStore* and Bigtable. Making bandwidth-independent task placement decisions, Borg can schedule a large number of these network-intensive workers on a single rack, resulting in significant traffic demand and triggering a hotspot. Figure 7 shows the distribution of bandwidth per compute unit across all the jobs on our fleet. The distribution is wide, with some network-intensive jobs using more than 10× the median value. Figure 8 plots ToR utilization as a function of the ratio of the number of all network-intensive tasks, including those from *QuerySys*, *RamStore* and Bigtable, per unit network bandwidth for each rack. There is a clear correlation between utilization and the number of these tasks per bandwidth unit.

Bandwidth-independent data placement in a distributed file system can place a large number of storage chunks on racks. When applications access these chunks concurrently, bandwidth demand can approach or exceed supply, and ToR hotspots can arise. Figure 9 quantifies the storage bandwidth demand-supply imbalance in our network; the normalized amount of link capacity divided by HDD capacity for *Disk* racks across our fleet exhibits a more than 2× difference between the smallest and largest values.

3.4 Hotspot Incidents

Network hotspots can cause significant production incidents, including SLO (service-level-objective) violations, user expe-

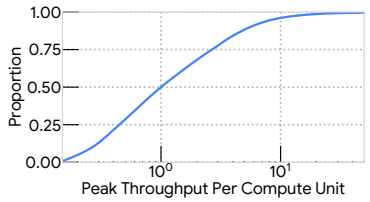


Figure 7: The distribution of outbound peak network throughput per compute unit for jobs across the fleet with non-trivial network usage, normalized by the median value.

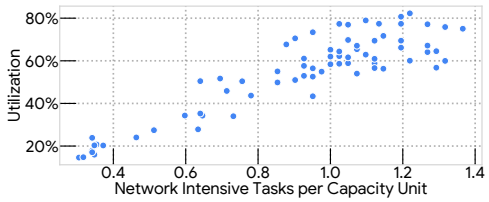


Figure 8: The correlation between ToR utilization (maximum of outbound and inbound directions utilization) and the number of network-intensive tasks on a *Compute* rack per unit of ToR capacity. The x-axis is normalized by the median value. ToRs with a higher number of network-intensive tasks run hotter.

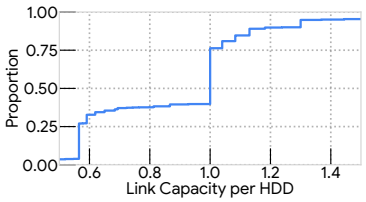


Figure 9: The distribution of the link capacity per HDD capacity for all *Disk* racks across the fleet, normalized by the median value.

rience degradation, and can require extensive troubleshooting effort. This section details several hotspot-related incidents.

Unintended capacity reduction causes ToR hotspots. Temporary hotspots can occur from reduced ToR uplink capacity during maintenance or failures. In a specific instance, a software rollout issue during network maintenance caused a 50% uplink capacity reduction on multiple ToRs. Unaware of this reduced capacity, Borg scheduled too many network-intensive jobs, including crucial batch jobs for Ads analysis, based only on compute resources. This demand-supply mismatch resulted in a 2–3x increase in task completion time.

Unintended network imbalances from scheduling constraints. While serving their intended purpose, Borg scheduling constraints can inadvertently create network imbalances. For instance, an I/O-intensive service implemented a custom constraint to avoid scheduling workers on machines with D servers, aiming to reduce network congestion. However, this constraint unintentionally excluded the majority of available machines, concentrating a large number of these same workers onto the remaining servers. This concentrated load resulted in persistent ToR hotspots and a 1.5x increase in read latency. This incident underscores the need for a more comprehensive approach to mitigating network hotspots.

Colossus degraded reads exacerbate ToR hotspots. During reads, a Colossus client initially attempts to read data directly from the disks storing the required chunk(s). If one or more of these are temporarily unavailable (due to overload on the disk, CPU, or network), and the file is erasure-encoded, the client performs a degraded read: retrieving the other chunks in the erasure-coded stripe and reconstructing the unavailable chunk. However, this mechanism is ineffective when the client resides on a ToR hotspot — it increases network demand on an already congested link. For example, in one incident, ToR congestion caused by incast traffic to Bigtable led to high Colossus read latencies. The resulting degraded reads significantly exacerbated ToR congestion.

Storage ToR hotspots cause high-level applications pain. There are many applications that are built on top of storage services. Therefore, hotspots impacting storage services ultimately translate into impact on users. For example, YouTube relies on storage services (e.g., Bigtable) to store, retrieve and display watch history. In one incident, a ToR hotspot led to increased packet losses and slowed reads from an SSD-backed Bigtable partition. As a result, Bigtable replication latency of user data doubled, resulting in stale or missing watch histories for some users, impacting user satisfaction.

4 Impact of Hotspots on Storage Systems

In this section, we present measurements of the impact of hotspots on the performance of our distributed storage systems. Because ToR hotspots occur predominantly on racks with storage elements, they impact the performance of access to distributed storage. In our infrastructure, applications access distributed storage via queries to a distributed database (*QuerySys*), or by performing reads/writes on a distributed file system (Colossus). In this section, we study how ToR utilization affects the latency of *QuerySys* query completion and Colossus file reads and writes.

4.1 Approach and Methodology

At a high level, to understand the impact of ToR utilization on storage access, we associate the latency of a *storage operation* (a file system read/write, or a database query) with the ToR utilization observed during the operation.

Latency. We use extensive logs to measure the latency of each storage operation, relying on production workloads for measurements of file system accesses and common benchmarks for database queries. These systems are instrumented to break down the time spent on the server versus on the network. The total operation time is measured from initiation to response, including network latency, which encompasses the time for underlying Remote Procedure Calls (RPCs) to traverse the network (including ToR switches).

Utilization. Each file system read or write, and each query, produces a *latency sample*. To understand how these latency measurements depend on ToR utilization, we associate each latency sample with a ToR utilization value. We do this as follows. A file system read/write, or a database query involves many ToR uplinks and downlinks, since these operations may involve multiple servers. We associate each latency sample with the maximum utilization of all ToR links involved in a storage operation, since that lies on the critical path of

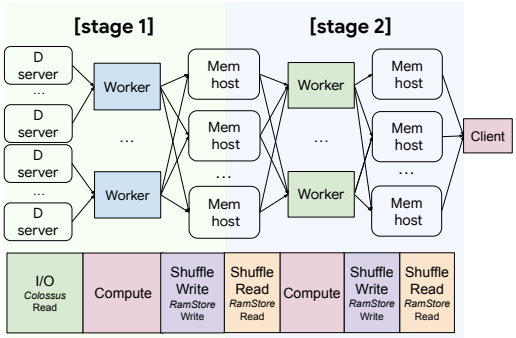


Figure 10: The data flow for an aggregation query in *QuerySys*.

an operation's completion. We obtain network utilization from a network telemetry system which reports the average throughput of each ToR during each 30 second window. For each latency sample, we use the average utilization during the time window containing the operation.

Metrics. To understand and quantify how storage access latency is affected by utilization, we divide utilization values into buckets in increments of 5%. For each bucket, we compute the *p95* latency of the samples within that bucket. We define the p95 *latency inflation* at a given utilization bucket as the ratio of the p95 latency in that bucket to the p95 latency of the lowest utilization bucket for which we have data. Latency inflation characterizes how storage access latency is affected by increasing ToR utilization.

We use a tail measure (p95) because tail performance is critical for many of our applications. We have also measured median latency inflation (defined similarly). This shows qualitatively similar results, so we omit this for brevity.

To understand how ToR utilization inflates storage access latency, we define *Load-tolerance*: the utilization at which p95 (or median) latency inflation is τ. In this paper, we consider two values of τ, 1.5 and 2. At $\tau = 2$, the storage access latency is double that at the lowest utilization which we consider unacceptable. For simplicity, we currently define unacceptable latency based on this fixed threshold. In future work (§6), we plan to define it based on application latency SLOs, which directly characterize the sensitivity of storage access to utilization.

4.2 Database Queries

A significant component of the offered load in our datacenters is traffic from queries of a distributed database, *QuerySys*. Query execution consists of a series of stages, and each stage runs multiple *workers* in parallel. Each worker (running on compute racks) processes a *partition* by reading data initially from a distributed file system, Colossus (or, in later stages, from an in-memory file system *RamStore*), performing some computation, and writing data to *RamStore*. Figure 10 depicts this for an *aggregation* query which produces statistics across one or more columns in a large relational table.

As discussed in §3, imbalanced infrastructure upgrades or network-intensive workloads can result in a *network/compute imbalance* that can increase ToR utilization and, in turn, affect query completion times. When infrastructure used for databases is upgraded such that compute on a rack increases (*e.g.,* because servers are upgraded) without a corresponding increase in ToR capacity, ToR utilization can increase. Database workload network-intensiveness can grow in several ways: *e.g.,* the rate of queries can increase, queries can become more complex, and queries can process larger tables; any of these factors can require more workers. More workers concurrently accessing distributed storage from a rack can increase ToR utilization.

In this section, we seek to quantify how the end-to-end query completion time, as well as the latency of each read or write operation, is affected by ToR utilization. When a query is issued, its completion time is affected by the concurrent workload (other queries and other application traffic) using the network. Some queries may traverse highly utilized ToRs. This gives us an opportunity to study how database queries are affected by ToR utilization.

Methodology. While most database queries execute in stages (*e.g.,* Figure 10), they differ widely in terms of the number of stages they use, the number of workers they employ, and the tables they access. We cannot thus assess ToR utilization impact on a single query, because each query may be affected to different extents at different utilization levels.

We use seven *benchmark* queries that our *QuerySys* team runs periodically to assess performance. These benchmarks include canonical queries (aggregations, materializations, etc.) that reflect real-world usage patterns, as well as industry standard benchmarks (*e.g.,* TPC-H and TPC-DS [46]), and thus capture the shape and semantics of most common queries against *QuerySys*. The breakdown of latency at each worker and each stage is logged for these benchmarks. This enables us to understand and quantify the extent to which network, compute, and storage affect end-to-end performance.

The rest of this section presents the impact of ToR utilization on these benchmarks.

Aggregation is the most common query in *QuerySys*: computing aggregates (*e.g.,* sum, count, average) from one or more columns in a large relational table. To scale to large tables, *QuerySys* employs multiple workers each of which processes distinct partitions of the table. This query executes in two stages (Figure 10). Both stages involve networked storage access and computation.

In this query, networked storage (Colossus) access account for approximately 30% of the query's total time. The time difference between Colossus reads and the query total time is primarily computation time, representing roughly 50% of the query's total time.

Figure 11 shows the p95 latency inflation and the varying *Load-tolerance* of these operations for $\tau = 1.5$: The *Load-tolerance* of the end-to-end query latency can be quite high (85%). For this query, the component which accounts for half the overall query latency, the computation time, is largely

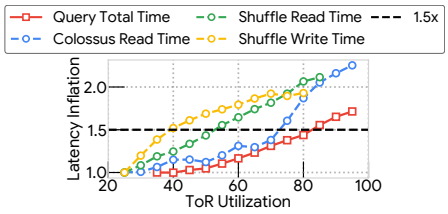
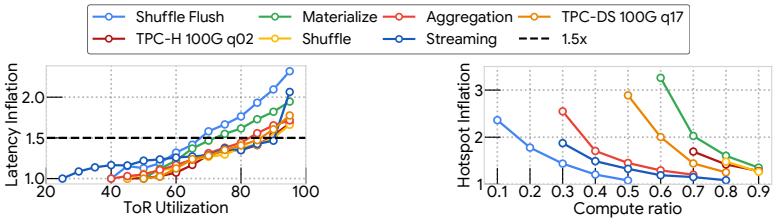


Figure 11: The query latency inflation vs. ToR utilization for the aggregation query.



(a) The query latency inflation vs. ToR utilization. **(b)** The CPU ratio of a query vs. the latency inflation at utilization 75%.

Figure 12: The query latency inflation of different *QuerySys* benchmarks.

independent of network hotspots, resulting in a higher *Load-tolerance*. The other components of the storage stack have a lower and widely varying *Load-tolerance*: shuffle writes at 40%, shuffle reads at 52%, and Colossus reads at 72%. Shuffle read/write have lower *Load-tolerance* because they don't involve compute and storage operation on the *RamStore* hosts. Colossus read is slightly higher because it involves storage but minimal compute.

Figure 12a shows p95 latency inflation and Table 1 shows *Load-tolerance* of the other queries in the benchmark. **Shuffle flush** is most sensitive to ToR utilization, primarily because it triggers an intensive data-flush from *RamStore* memory hosts to Colossus when the host runs out of memory to complete the write. This operation increases the share of network transfer time in the total query time, resulting in the lowest 1.5× *Load-tolerance* among all benchmarks (70%). **Materialize** writes results through Colossus; its *Load-tolerance* of 75% is similar to that of shuffle flush and Colossus reads during aggregation. The remaining benchmark queries are less network-intensive, so have a *Load-tolerance* of 90–95%.

The impact of computation. Queries access storage, but also process the retrieved data (*e.g.,* aggregate the data). This processing time affects the tolerance of a query to hotspots.

To illustrate this, we defined *Hotspot-inflation*: the latency inflation at the hotspot threshold (75% utilization, §3). We explored *Hotspot-inflation* for different queries as a function of *query compute ratio* — the ratio of the sum of the processing time for the 10 slowest workers[2] within each stage by the sum of their worker time (aggregated across all stages).

Figure 12b depicts the relationship between these two quantities. To understand this figure, it helps to understand that different samples from a single benchmark query may exhibit different compute ratios (because storage operations and networking take different amounts of time). The figure groups samples by compute ratio, and plots the *Hotspot-inflation* for each group. Thus, for example, samples of the shuffle flush benchmark which have a compute ratio of 0.1 have a *Hotspot-inflation* of about 2.25. The figure demonstrates a clear negative dependence between compute ratio and *Hotspot-inflation*. This explains why higher compute ratios result in higher *Load-tolerance*s. In other words, the more compute-bound a query is, the less sensitive it is to network hotspots.

[2]Currently, our telemetry system tracks processing times only for the 10 slowest workers.

	Load-tolerance (1.5×)	*Load-tolerance* (2×)	*Hotspot-inflation* (75%)
Shuffle Flush	70	90	1.67
Materialize	75	–	1.55
Aggregation	85	–	1.38
TPC-DS 100G q17	90	–	1.35
TPC-H 100G q02	90	–	1.34
Shuffle	95	–	1.29
Streaming	95	95	1.37

Table 1: *Load-tolerance* and *Hotspot-inflation* of different *QuerySys* benchmarks.

Different queries have different *Hotspot-inflation* at the same compute ratio because they may, for example, use different storage operations. For example, when the CPU ratio is 0.8, *materialize* prober has more *Hotspot-inflation* than *streaming* prober because the former incurs Colossus writes while the latter incurs Colossus reads. For Colossus, writes are more sensitive to ToR hotspots than reads (§4.3).

Summary. In Table 1, we compare the *Load-tolerance* and *Hotspot-inflation* across queries. Most queries also have high *Load-tolerance*; many did not reach 2× inflation. For these, their high *Load-tolerance* comes from the significant time spent on compute — ranging from 10% to 90% (Figure 12b).

This discussion implies that for some *QuerySys* benchmarks substantial tail latency reductions can be had by reducing the occurrence of ToR hotspots. §5 discusses hotspot-aware placement mechanisms we use to achieve that.

4.3 File Access

In a distributed file system like Colossus, files are split into chunks, with each chunk replicated across multiple servers. Consider a rack with servers that host chunks. If these servers are upgraded to add more storage I/O capacity (e.g.,new storage hardware that increases the storage read/write rate), but the ToR's capacity is not correspondingly upgraded, a network/storage imbalance can result. This imbalance can result in a ToR hotspot, which can, in turn, affect the latency of reads and writes to the file system. In this section, we (a) study how Colossus read/write latency is impacted by ToR hotspots and by ToR utilization, and (b) quantify the prevalence of network/storage imbalance across our fleet.

Methodology. We collect the total latency of datacenter HDD chunk read and write requests on racks with HDD stor-

age (§2.1) with chunk sizes between 100KB and 1MB from the Dapper tracing system [40] from production workloads over one day.

HDD reads. Figure 13a shows the p95 latency *inflation* (ratio against latency at lowest utilization, §4.1) for network and total latency. Both network and total latency increases with ToR utilization. At the 75% hotspot threshold, network latency is inflated by 4×, but Colossus HDD reads have a *Hotspot-inflation* (the inflation at the hotspot-threshold of 75%, §4.1) of only 1.5×. This is because only the network portion of the Colossus read time is inflated. In all but the last utilization bucket, network accounts for less than 20% of the total HDD read latency; in the last utilization bucket, network doesn't contribute more than 40%. *Load-tolerance* (§4.1) is another way to depict the impact of ToR utilization on storage access latency. For Colossus reads, the 2× *Load-tolerance* is 95%; this means that read latency is less than 2× for all ToR utilizations lower than 95%.

HDD writes. Colossus HDD writes show a qualitatively different behavior than HDD reads because Colossus employs write-back caching on *D* servers (§2.2): writes are cached in battery protected server-side memory and later flushed to persistent storage. Thus Colossus HDD write latency is dominated by network latency, and hence is impacted significantly by increases in network latency as a result of high ToR utilization. Network contributes to 30–40% for low utilization buckets and more than 50% for higher utilization buckets (>75%). By contrast, though Colossus caches reads, read cache hit rates are low, because our workloads mostly access large files sequentially, resulting in little read locality. Hence similar behavior is not observed on reads.

Writes are very sensitive to ToR utilization. As Figure 13b shows, their 2× *Load-tolerance* is a mere 50%, so, even in a lightly loaded network, write latency can be twice the latency when compared to an unloaded network. Their *Hotspot-inflation* is around 4; writes to a rack with a ToR hotspot can be 4 times slower than writes to a rack with an unloaded ToR.

While HDD write operations are impacted more than reads due to the amount of storage access, the network portion of the latency is similarly affected in both operations. The network latency components include round-trip delay (RTT), serialization delays, and congestion control shaping traffic to fill the ToR link capacity. Measurements of RTT and packet losses (not shown here) remain consistently low across utilizations as expected from our deployments of congestion control and load balancing techniques [22, 25]. The shape of the network latency curve reflects the workload's inherent burstiness relative to the available capacity.

The impact of imbalanced infrastructure. Having characterized the impact of network utilization on reads and writes, we now show how network/storage imbalance impacts file reads and writes in Figure 14. To derive this figure, we classified each rack in a datacenter into one of three categories based on its level of resource balance: *High-Uplink* if its installed ToR uplink capacity exceeds provisioning guideline relative to installed storage, *Medium-Uplink* if the capacity matches the guideline, and *Low-Uplink* if the capacity is lower than the guideline. Figure 14 shows the fraction of storage reads and writes to each type of rack across different ToR utilizations, across all of our datacenters. The figure dramatically illustrates the impact of network/storage imbalance. Storage requests that experience low ToR utilization access *High-Uplink* ToRs (in green). Conversely, storage requests that experience extremely high ToR utilization almost exclusively access *Low-Uplink* ToRs (in red).

5 Hotspot-Aware Placement

As discussed in §3, ToR hotspots occur as a result of an interplay between network supply and demand for compute and storage resources. As such they cannot be resolved by *in-network* mechanisms *e.g.,* in switches or host networking stacks. Therefore, we integrate *hotspot awareness* into our cluster scheduler and distributed file system. We've specifically developed two hotspot-aware placement strategies:

1. *ToR-utilization aware task placement and migration* (UTP) takes ToR utilization into account when placing tasks (such as query workers §4) on servers, or migrating tasks away from them, thereby improving the balance between network demand-supply for compute resources.
2. *ToR-capacity aware chunk placement* (CCP) takes the imbalance between ToR capacity and rack storage capacity into account when making storage chunk placement decisions. Thereby improving the balance between network demand-supply for storage resources.

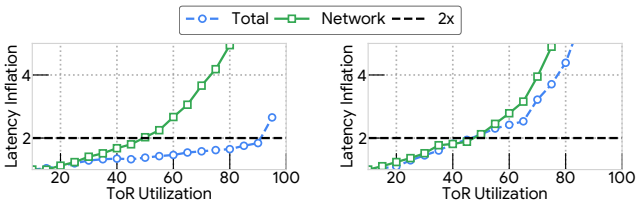
As we describe below, the design of these hotspot-aware placement strategies is informed by the measurements presented in §3 and §4. Moreover, neither of these placement decisions is trivial, since each needs to balance complex, competing objectives. At the same time, avoiding hotspots is important, since doing so can potentially improve network utilization and applications' perceived latency.

5.1 Task Placement and Migration

We modified the Borg cluster scheduler to bias task placement away from, and reactively migrate tasks from, ToR hotspots. Before describing this technique, abbreviated UTP, we briefly describe Borg; see [9, 45] for details.

Background. Borg selects the server on which to place each task (e.g., worker) in a job. It utilizes historical measurements to estimate task resource requirements, and uses this to select suitable servers. When an estimate is inaccurate or workload changes, Borg can reactively migrate affected tasks to improve job performance, but task migration must be careful to avoid impacting availability SLOs offered to service owners.

In practice, a cluster scheduler must (a) scale to clusters with a large number of servers and (b) satisfy many other, often competing, objectives. It must: balance load between servers, prevent over-subscription to resources such as CPU



(a) HDD read. **(b)** HDD write.

Figure 13: The total latency and the network latency inflation (*i.e.,* p95 latency values normalized against the latency at the lowest utilization).

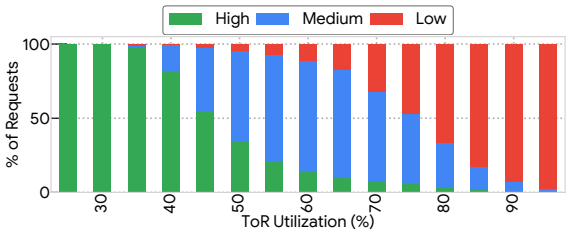


Figure 14: The distribution of chunk read and write request by uplink categories (*High-*, *Medium-* and *Low-Uplink*) for ToRs at a given level of utilization.

or memory, over-commit resources to take advantage of statistical multiplexing, spread tasks of a job across racks to minimize the likelihood of correlated failure, enable memory access locality, and pack jobs into as few servers as possible.

Borg addresses the first challenge by choosing a random sample of placement candidates from amongst servers in the cluster, rather than considering all servers. It addresses the second challenge by assigning a vector of scores to each server based on multiple objectives. Borg prioritizes server selection objectives, with lower-ranked objectives taking precedence. For instance, if load balancing is deemed more critical than bin packing, Borg may choose a server that optimizes load distribution even if it leads to suboptimal bin packing efficiency.

Requirements. Before we designed and implemented UTP, Borg made *bandwidth-independent task placement decisions* — its decisions did not take network utilization into account. Its primary task — to find a set of servers to satisfy application requirements and potentially competing infrastructure constraints — was complex enough that no attempt was made to add network as a resource to Borg earlier, since that would require imposing constraints on group of servers beneath each ToR rather than individual servers. Given that hotspots can degrade performance, and that Borg is a mature infrastructure, we imposed two requirements on UTP: (a) it should be designed as a minimal change to Borg (i.e., not requiring network as a full-fledged new resource type), and (b) it should not adversely impact any existing cluster objective.

Design. To make minimal changes to the scheduler, ideally we should minimally modify an existing objective so that it does not regress any other existing objective. This is hard to do for ToR utilization. Borg can easily determine if placing a task on a server would over-run the server's CPU resources; this depends only on the resources currently committed by the server to other tasks. It is harder to determine how placing a task on a server would impact the server's ToR utilization, since that depends on *other* servers on the rack (that may or may not be in the randomly chosen candidate set) and how their workload changes. Reasoning about this impact correctly would require Borg to consider a large number of servers, which can impact scaling.

Proactive UTP placement. Fortunately, UTP required a relatively simple change to Borg for the following reason: at any given instant, there are relatively few ToR hotspots in each cluster, since the average ToR utilization in our datacenters is low. Given this, UTP tweaks a low-priority objective in Borg, that of ensuring load-balance across servers: it prefers servers that would balance ToR-utilization across the cluster more evenly. Specifically, UTP achieves this by assigning a score to each candidate machine that reflects the ToR utilization for that machine should the task get placed on that machine. In computing this score, we take into account instantaneous ToR utilization as well as peak task demand.

This methodology works well even on a random sample of servers; given the low average ToR utilization, there are likely to be many servers in the sample with low ToR utilization, so in most of the cases Borg can satisfy other cluster-level objectives while finding a suitable candidate server which also improves better balancing of ToR utilization.

Reactive UTP migration. Even with ToR-utilization aware task placement (UTP), ToR hotspots can develop. This is because the placement decision is based on historical estimates, which can be incorrect as the usage pattern of a workload changes, or may not be available for new workloads. To address this, UTP migrates tasks away from ToR hotspots. Borg already supports the infrastructure to orchestrate this for other dimensions (e.g., cpu, memory bandwidth, etc) and we extended it toward network. To this end, UTP must make two decisions: (a) how to determine if a ToR is a hotspot, (b) which task(s) to migrate and when.

UTP employs a 75% utilization threshold to identify hotspots. This aligns with the load tolerance of common workloads, including those discussed in §4.2 and Colossus reads up to 1MB. It also adheres to the fleet-wide requirement of reserving 25% capacity for in-place upgrades and expansions. It then greedily finds the latency-tolerant tasks with the largest network bandwidth usage on a rack to migrate. This ensures the fewest possible migrations and avoids migrating latency-sensitive tasks when possible. Finally, UTP migrates tasks only if availability budgets would permit.

Results. Figure 15 shows the efficacy of UTP in addressing ToR hotspots after its fleetwide rollout. Specifically, it shows that the number of hot ToRs was reduced by 90% after deploying these features in the scheduler. The remaining hot

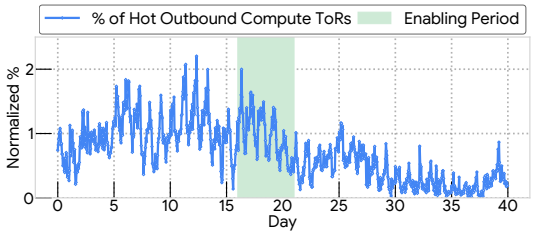


Figure 15: The fleet-wide percentage of compute ToRs hot in the outbound direction before and after deploying UTP, normalized by the average value before the deployment.

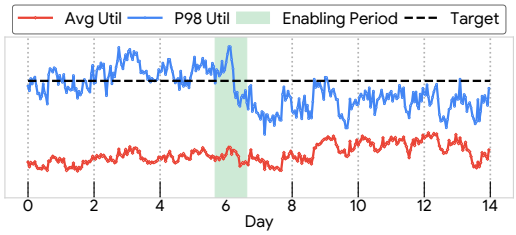


Figure 16: The change of the p98 outbound utilization of compute ToRs as well as the average cell level utilization, defined as the total egress throughput from compute ToRs divided by their total uplink capacity, after UTP was enabled in a pilot cluster.

ToRs are likely due to cases where higher-priority objectives in task placement would prefer a hot ToR rather than cold ToRs. As expected, given the low average ToR utilization in our datacenters, such cases are not common. Future work can consider ways to improve scoring and the relative order of objectives to further reduce the occurrence of hot ToRs.

Figure 16 shows a time series of the evolution of the p98 outbound utilization of *Compute* ToRs during the pilot of UTP in one of our clusters before the fleetwide rollout. It also depicts a timeseries of the average cell level utilization in the cluster (§5.2) during that same window. UTP reduces the tail of the ToR utilization to below the hotspot threshold (75%) after the deployment of UTP. This benefits comes despite a higher average utilization due to an unrelated workload change after UTP was enabled.

In theory, UTP could have been purely reactive: it could have omitted ToR-aware initial placement, and merely invoked task migration when a hotspot was detected. Figure 17 shows that a purely reactive design can increase task migrations by nearly 2×, relative to our design. Moreover, with a purely reactive approach, network-intensive jobs are 7× more likely to be scheduled on hot ToRs (Figure 18).

UTP enables significant reductions (up to 13%) in p95 latency across many of our *QuerySys* benchmarks (Figure 19). The shuffle flush and materialize benefit the most from UTP because they are the most sensitive to ToR utilization with the lowest *Load-tolerance* (Figure 12a). On the other hand, the improvement on the TPC-H benchmark is not significant, which is expected because the workload is not sensitive to ToR utilization.

As discussed in section §3.4, ToR hotspots were a recurring cause of service incidents. Enabling UTP resulted in a 70% reduction in the monthly occurrence of such incidents. The remaining incidents post-UTP were primarily attributed to either resolved bugs within UTP or occurred in specific clusters where UTP had not yet been deployed.

Metric	Change
Percentage of hot ToRs	−44.6%
p98th utilization	−18.5%
Average utilization	10.9%

Table 2: The change of the median value of the metrics reported in Figure 15 and Figure 16 in a 5-day window before and after UTP was enabled. We also used the Student's t-test on the metrics samples in the two periods to confirm that the change was statistically significant.

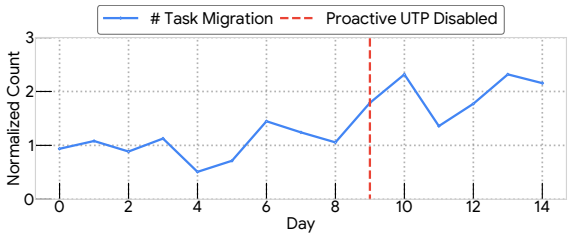


Figure 17: The number of task migration events in a day due to UTP before and after the disabling of the proactive UTP placement, normalized by the average value before.

Lastly, as discussed earlier, a key constraint that we needed to satisfy in the design of UTP was to not adversely impact any existing cluster objective. We didn't observe any regression in key Borg objective after the rollout of UTP. Corresponding graphs are omitted for brevity.

5.2 Storage Chunk Placement

ToR-capacity aware chunk placement (CCP) addresses imbalanced infrastructure upgrades by steering Colossus chunk placement away from racks whose storage capacity has grown out of balance with provisioned ToR bandwidth.

Background. Distributed file systems use heuristics to determine where to place new file chunks. These heuristics are designed to achieve several, often competing, objectives to determine the most appropriate storage servers at which to replicate these chunks. These objectives are similar to that of task placement: spreading for reliability, load balancing, access locality, and so on.

Design. As with UTP, CCP was designed to ensure minimal changes to the existing Colossus distributed file system. These changes were similar in principle to those for UTP, so we omit the details for brevity but describe the key idea underlying the approach.

In §4.3, we showed that storage requests to *High-Uplink* ToRs experience low ToR utilization. Conversely, storage requests that experience extremely high ToR utilization almost exclusively access *Low-Uplink* ToRs. Based on this observation, CCP prioritizes High-Uplink ToRs over Medium-Uplink and Low-Uplink ToRs for new chunk placement.

Results. We now discuss latency reduction resulting from

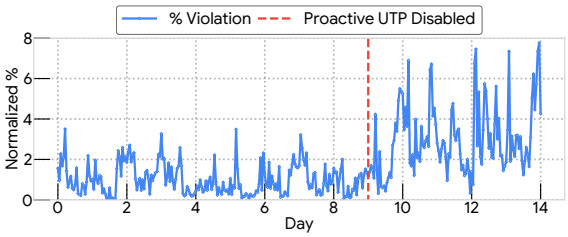


Figure 18: The percent of job scheduling events where a network-intensive job is scheduled on a hot ToR before and after the disabling of the proactive UTP placement, normalized by the average value before.

Metric	Change
Number of task migration events	−41.0%
Percentage of scheduling violations	−74.1%

Table 3: The change of the median value of various metrics reported in Figure 17 and Figure 18 in a 5-day window with ToR utilization-aware placement enabled against in a 5-day window without the feature. We also used the Student's t-test on the metrics samples in the two periods to confirm that the change was statistically significant.

this approach, using measurements from a 15-day pilot in one of our clusters. Figure 20 shows the relative reduction in Colossus p95 total storage access and network latency in the pilot cluster. As CCP biases storage accesses towards high-capacity ToRs, p95 network latency reduces by 50–80% and p95 total latency reduces by 30–60%.

6 Future Directions

Automation techniques for hotspot mitigation. Hotspot mitigation in our storage systems required significant manual intervention, leveraging service owner knowledge of application metrics, dependencies, and request prioritization within the application. To scale mitigation to more applications, automation techniques must be developed. Future work can explore declarative methods for service owners to define application characteristics, dependencies, and constraints, enabling automated backend systems to identify and implement mitigation opportunities. Alternatively, more robust profiling techniques can extract network communication requirements from live deployments and map them to a range of system deployment conditions.

Service Level Objectives (SLOs). Our current placement strategy, which is hotspot-aware but best-effort, cannot guarantee performance under every network and workload condition. To ensure consistent performance, we must define SLOs and implement mechanisms to guarantee them. This necessitates substantial scheduler enhancements, moving beyond simple load spreading to include strict network constraint enforcement, thereby mitigating SLO risks.

Network fault tolerance for ML. The distinct characteristics of Machine Learning (ML) workloads, such as single points of failure and sensitivity to worker performance, necessitate novel network fault tolerance strategies. Placement and rescheduling techniques discussed in this paper can be

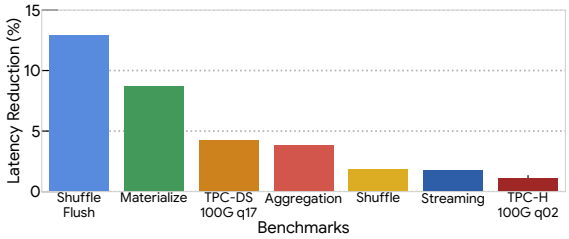


Figure 19: The relative improvement of the p95 query latency for each of the seven *QuerySys* benchmarks studied in Section 4.2 in a 5-day window before and after UTP was enabled in the pilot cluster.

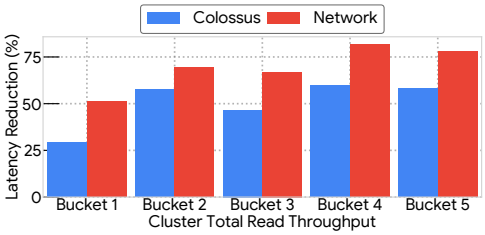


Figure 20: The reduction of the p95 Colossus and network latency of HDD reads in a pilot cluster with CCP enabled, compared with those at the time when the cluster is in the same bucket of total read throughput but with CCP disabled.

adapted for ML workloads to minimize the impact of faulty network components, such as NICs and ToRs. This is particularly crucial for jobs that rely on synchronous communication between numerous accelerators.

Network-awareness across layers. While we tackled persistent network hotspots with infrastructure-level network awareness, shorter-lived hotspots (seconds to few minutes) can make use of application-level load balancing. Application control further allows for customized hotspot handling based on request sensitivity and required quality of service. Future investigations can focus on determining optimal hotspot resolution timescales for various mitigation strategies (proactive/reactive) and layers (infrastructure/application), and developing coordinated decision-making frameworks.

Connecting placement and provisioning. Future work can explore the interplay between hotspot-aware placement and resource (re-)provisioning. While hotspot-aware placement can extend the lifespan of network infrastructure and allow for more conservative uplink:downlink oversubscription to deliver network efficiency, capacity planners need tools to determine when and where to upgrade network, storage, and compute resources to address capacity mismatches. A key area for future research is determining the optimal utilization target for provisioning and identifying utilization values to inform changes in job and data placement, enabling safely operating network at higher utilization while meeting application SLOs.

7 Related Work

Reducing network congestion. To reduce datacenter congestion, the literature explored various solutions, including congestion control [3, 4, 19, 25, 26, 28, 31, 51], network load balancing [1, 2, 8, 21, 22], and traffic engineering [7, 35]. These solutions mitigate packet drops and queuing delays by distributing traffic over time (congestion control) or across different paths (load balancing and traffic engineering), but they are bounded by bandwidth limits in ToR hotspots in which all the uplinks of a ToR are persistently hot. By contrast, hotspot-aware scheduling makes more bandwidth available, by modifying the traffic matrix.

Datacenter scheduling. Datacenter job scheduling can influence traffic patterns, but it traditionally prioritizes local resources like compute, memory and storage, assuming the network isn't a bottleneck. Various research has explored incorporating network attributes like latency, bandwidth, and reliability into storage systems to make them network-aware [6, 15, 42, 43]. Several storage systems attempt to mitigate network congestion. For instance, Sinbad [12] dynamically adjusts replica placement based on real-time link utilization to address imbalances. However, it is restricted to replicated file encodings and requires live network data, unlike our ToR-capacity aware chunk placement (CCP), which is encoding-agnostic and relies on simpler, static ToR uplink information. NetHint [11] proposes an interactive cloud tenant-provider mechanism to optimize application performance by identifying network hotspots. Yet, this approach necessitates application-level network awareness and is confined to single-tenant environments, contrasting with our infrastructure-level, multi-tenant hotspot management. Furthermore, systems like Diktyo [37] and Kubernetes [13] optimize service placement for latency by considering network bandwidth and topology, primarily focusing on microservice environments.

High-Performance Computing (HPC) and Machine Learning (ML) have long incorporated network awareness into job scheduling, as exemplified by CASSINI [34] and Corral [20]. This paper demonstrates that network awareness is equally critical for traditional compute and storage services, especially in mitigating application tail latency.

Network capacity planning. Recent research has explored leveraging application-specific information to enhance bandwidth allocation: Saba [23], for example, uses ahead-of-time profiling to distribute network bandwidth based on how sensitive applications are to changes in bandwidth. Jupiter's networks [41] employ both traffic engineering and topology engineering, which operate at different timescales. Traffic engineering swiftly adapts to changes in topology and traffic, whereas topology engineering is a slower, planned process for implementing new network structures. However, capacity planning alone cannot eliminate network hotspots due to the fundamental causes discussed earlier (§3.3). Our work focuses on analyzing how sensitive applications are to network hotspots, and how network-aware placement can reduce the occurrence and duration of these hotspots.

Characterizing application performance. Prior research has explored the impact of various factors on application performance. Ousterhout *et al.* [30] proposed a methodology for quantifying performance bottlenecks and used it to analyze the Spark framework's performance, concluding that network optimizations can only reduce job completion time by a median of at most 2%. Their conclusions are not in contradiction to this paper because (1) the network utilization in their datacenters is low and below 50%, so they do not observe the effect of hotspots, and (2) workloads in our datacenters are able to fully parallelize the execution, which magnifies the impact of the worst straggler. Seemakhupt *et al.* [38] analyzed latency bottlenecks in Remote Procedure Calls (RPCs), identifying network performance issues using passively-sampled RPCs [33] at Google. Zambre *et al.* [48] broke down the time involved in small message communication into CPU, I/O, and network usage. More recently, Kong *et al.* [24] modeled how NIC resource usage affects RDMA performance. Our work complements these studies by focusing on how ToR utilization influences performance, ranging from basic storage operations to complex application operations in *QuerySys*.

8 Conclusion

This paper demonstrates the need for hotspot-aware scheduling to mitigate the impact of network hotspots on application performance in datacenters. Traditional solutions like congestion control and traffic engineering prove insufficient; instead, directly managing workload placement is essential for reining in elevated tail latency. Our work reveals two surprising findings: first, even disk- or compute-bound application operations are vulnerable to elevated network utilization levels, with ToR hotspots capable of doubling their latency. Second, adding simple hotspot awareness to existing compute and storage schedulers dramatically reduces application tail latency and eliminates most ToR hotspots. This approach, while simpler than incorporating network as a fully schedulable resource, yields significant gains.

These results highlight the need for datacenter schedulers to comprehensively consider network demand and supply. Future research shall investigate holistic scheduling frameworks accounting for network bandwidth, topology, and compute and storage resources to pave the way for balanced datacenter environments with predictable application performance.

Acknowledgement

This work wouldn't have been possible without the contributions of many individuals and teams across Google, particularly within the Networking Infrastructure, Compute, and Storage organizations. We are especially grateful to to Arjun Singh, David Wetherall, Ian Whalley, Milo Martin, Thomas Wenisch, Larry Greenfield, Gaurav Dhiman, Flo-

rentina Popovici and the anonymous NSDI reviewers for their invaluable feedback and to our shepherd Ravi Soundararajan for guiding us through the revision process. This research was in part supported by NSF CNS NeTS 2211383.

References

[1] Mohammad Al-Fares, Sivasankar Radhakrishnan, Barath Raghavan, Nelson Huang, and Amin Vahdat. Hedera: Dynamic flow scheduling for data center networks. In *7th USENIX Symposium on Networked Systems Design and Implementation (NSDI 10)*, San Jose, CA, April 2010. USENIX Association.

[2] Mohammad Alizadeh, Tom Edsall, Sarang Dharmapurikar, Ramanan Vaidyanathan, Kevin Chu, Andy Fingerhut, Vinh The Lam, Francis Matus, Rong Pan, Navindra Yadav, and George Varghese. Conga: Distributed congestion-aware load balancing for datacenters. In *Proceedings of the 2014 ACM Conference on SIGCOMM*, SIGCOMM '14, page 503–514, New York, NY, USA, 2014. Association for Computing Machinery.

[3] Mohammad Alizadeh, Albert Greenberg, David A. Maltz, Jitendra Padhye, Parveen Patel, Balaji Prabhakar, Sudipta Sengupta, and Murari Sridharan. Data center TCP (DCTCP). In *SIGCOMM*, 2010.

[4] Mohammad Alizadeh, Shuang Yang, Milad Sharif, Sachin Katti, Nick McKeown, Balaji Prabhakar, and Scott Shenker. Pfabric: Minimal near-optimal datacenter transport. In *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, SIGCOMM '13, page 435–446, New York, NY, USA, 2013. Association for Computing Machinery.

[5] Amazon. Amazon redshift. https://aws.amazon.com/redshift/, 2024.

[6] Alexandros Batsakis, Randal Burns, Arkady Kanevsky, James Lentini, and Thomas Talpey. Ca-nfs: A congestion-aware network file system. *ACM Trans. Storage*, 5(4), dec 2009.

[7] Theophilus Benson, Ashok Anand, Aditya Akella, and Ming Zhang. Microte: Fine grained traffic engineering for data centers. In *Proceedings of the Seventh COnference on Emerging Networking EXperiments and Technologies*, CoNEXT '11, New York, NY, USA, 2011. Association for Computing Machinery.

[8] Olivier Bonaventure, Christoph Paasch, Gregory Detal, et al. Use cases and operational experience with multipath tcp. *RFC 8041*, 2017.

[9] Brendan Burns, Brian Grant, David Oppenheimer, Eric Brewer, and John Wilkes. Borg, omega, and kubernetes. *ACM Queue*, 14:70–93, 2016.

[10] Craig Chambers, Ashish Raniwala, Frances Perry, Stephen Adams, Robert Henry, Robert Bradshaw, and Nathan. Flumejava: Easy, efficient data-parallel pipelines. In *ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*, pages 363–375, 2 Penn Plaza, Suite 701 New York, NY 10121-0701, 2010.

[11] Jingrong Chen, Hong Zhang, Wei Zhang, Liang Luo, Jeffrey Chase, Ion Stoica, and Danyang Zhuo. NetHint: White-Box networking for Multi-Tenant data centers. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, pages 1327–1343, Renton, WA, April 2022. USENIX Association.

[12] Mosharaf Chowdhury, Srikanth Kandula, and Ion Stoica. Leveraging endpoint flexibility in data-intensive clusters. In *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, SIGCOMM '13, page 231–242, New York, NY, USA, 2013. Association for Computing Machinery.

[13] Kubernetes community. Network-aware scheduling. https://scheduler-plugins.sigs.k8s.io/docs/kep/260-network-aware-scheduling/readme/, 2023.

[14] Marek Denis, Yuanjun Yao, Ashley Hatch, Qin Zhang, Chiun Lin Lim, Shuqiang Zhang, Kyle Sugrue, Henry Kwok, Mikel Jimenez Fernandez, Petr Lapukhov, Sandeep Hebbani, Gaya Nagarajan, Omar Baldonado, Lixin Gao, and Ying Zhang. Ebb: Reliable and evolvable express backbone network in meta. In *Proceedings of the ACM SIGCOMM 2023 Conference*, ACM SIGCOMM '23, page 346–359, New York, NY, USA, 2023. Association for Computing Machinery.

[15] Amir Epstein, Elliot K Kolodner, and Dmitry Sotnikov. Network aware reliability analysis for distributed storage systems. In *2016 IEEE 35th Symposium on Reliable Distributed Systems (SRDS)*, pages 249–258. IEEE, 2016.

[16] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. The google file system. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles*, pages 20–43, Bolton Landing, NY, 2003.

[17] Google. A peek behind Colossus, Google's File System | Google Cloud Blog. https://cloud.google.com/blog/products/storage-data-transfer/a-peek-behind-colossus-googles-file-system, 2021.

[18] Google. Bigtable overview. https://cloud.google.com/bigtable/docs/overview, 2023.

[19] Mark Handley, Costin Raiciu, Alexandru Agache, Andrei Voinescu, Andrew W. Moore, Gianni Antichi, and Marcin Wójcik. Re-architecting datacenter networks and stacks for low latency and high performance. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '17, page 29–42, New York, NY, USA, 2017. Association for Computing Machinery.

[20] Virajith Jalaparti, Peter Bodik, Ishai Menache, Sriram Rao, Konstantin Makarychev, and Matthew Caesar. Network-aware scheduling for data-parallel jobs: Plan when you can. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, SIGCOMM '15, page 407–420, New York, NY, USA, 2015. Association for Computing Machinery.

[21] Abdul Kabbani, Balajee Vamanan, Jahangir Hasan, and Fabien Duchene. Flowbender: Flow-level adaptive routing for improved latency and throughput in datacenter networks. In *Proceedings of ACM CoNEXT*, pages 149–160, 2014.

[22] Abdul Kabbani, David J. Wetherall, Gautam Kumar, Junhua Yan, Kira Yin, Masoud Moshref, Mubashir Adnan Qureshi, Qiaobin Fu, Van Jacobson, and Yuchung Cheng. PLB: Congestion signals are simple and effective for network load balancing. In *Proceedings of the ACM SIGCOMM 2022 Conference*, SIGCOMM '22, New York, NY, USA, 2022. Association for Computing Machinery.

[23] M.R. 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*, EuroSys '23, page 623–638, New York, NY, USA, 2023. Association for Computing Machinery.

[24] Xinhao Kong, Jingrong Chen, Wei Bai, Yechen Xu, Mahmoud Elhaddad, Shachar Raindel, Jitendra Padhye, Alvin R. Lebeck, and Danyang Zhuo. Understanding RDMA microarchitecture resources for performance isolation. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, pages 31–48, Boston, MA, April 2023. USENIX Association.

[25] Gautam Kumar, Nandita Dukkipati, Keon Jang, Hassan Wassel, Xian Wu, Behnam Montazeri, Yaogong Wang, Kevin Springborn, Christopher Alfeld, Mike Ryan, David J. Wetherall, and Amin Vahdat. Swift: Delay is simple and effective for congestion control in the datacenter. In *SIGCOMM*, 2020.

[26] Yuliang Li, Rui Miao, Hongqiang Harry Liu, Yan Zhuang, Fei Feng, Lingbo Tang, Zheng Cao, Ming Zhang, Frank Kelly, Mohammad Alizadeh, and Minlan Yu. Hpcc: High precision congestion control. In *Proceedings of the ACM Special Interest Group on Data Communication*, SIGCOMM '19, page 44–58, New York, NY, USA, 2019. Association for Computing Machinery.

[27] Microsoft. Azure synapse analytics. https://azure.microsoft.com/en-us/products/synapse-analytics, 2024.

[28] 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*, SIGCOMM '18, page 221–235, New York, NY, USA, 2018. Association for Computing Machinery.

[29] John Ousterhout, Arjun Gopalan, Ashish Gupta, Ankita Kejriwal, Collin Lee, Behnam Montazeri, Diego Ongaro, Seo Jin Park, Henry Qin, Mendel Rosenblum, Stephen Rumble, Ryan Stutsman, and Stephen Yang. The ramcloud storage system. *ACM Trans. Comput. Syst.*, 33(3), aug 2015.

[30] Kay Ousterhout, Ryan Rasti, Sylvia Ratnasamy, Scott Shenker, and Byung-Gon Chun. Making sense of performance in data analytics frameworks. In *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*, pages 293–307, Oakland, CA, May 2015. USENIX Association.

[31] Jonathan Perry, Amy Ousterhout, Hari Balakrishnan, Devavrat Shah, and Hans Fugal. Fastpass: A centralized "zero-queue" datacenter network. *SIGCOMM Comput. Commun. Rev.*, 44(4):307–318, aug 2014.

[32] Leon Poutievski, Omid Mashayekhi, Joon Ong, Arjun Singh, Mukarram Tariq, Rui Wang, Jianan Zhang, Virginia Beauregard, Patrick Conner, Steve Gribble, Rishi Kapoor, Stephen Kratzer, Nanfang Li, Hong Liu, Karthik Nagaraj, Jason Ornstein, Samir Sawhney, Ryohei Urata, Lorenzo Vicisano, Kevin Yasumura, Shidong Zhang, Junlan Zhou, and Amin Vahdat. Jupiter evolving: Transforming Google's datacenter network via optical circuit switches and software-defined networking. In *Proceedings of the ACM SIGCOMM 2022 Conference*, SIGCOMM '22, page 66–85, New York, NY, USA, 2022. Association for Computing Machinery.

[33] Mubashir Adnan Qureshi, Junhua Yan, Yuchung Cheng, Soheil Hassas Yeganeh, Yousuk Seung, Neal Cardwell, Willem De Bruijn, Van Jacobson, Jasleen Kaur, David Wetherall, and Amin Vahdat. Fathom: Understanding datacenter application network performance. In *Proceedings of the ACM SIGCOMM 2023 Conference*,

ACM SIGCOMM '23, page 394–405, New York, NY, USA, 2023. Association for Computing Machinery.

[34] Sudarsanan Rajasekaran, Manya Ghobadi, and Aditya Akella. CASSINI: Network-Aware job scheduling in machine learning clusters. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*, pages 1403–1420, Santa Clara, CA, April 2024. USENIX Association.

[35] Arjun Roy, Hongyi Zeng, Jasmeet Bagga, George Porter, and Alex C. Snoeren. Inside the social network's (datacenter) network. *SIGCOMM Comput. Commun. Rev.*, 45(4):123–137, aug 2015.

[36] Bart Samwel, John Cieslewicz, Ben Handy, Jason Govig, Petros Venetis, Chanjun Yang, Keith Peters, Jeff Shute, Daniel Tenedorio, Himani Apte, Felix Weigel, David Wilhite, Jiacheng Yang, Jun Xu, Jiexing Li, Zhan Yuan, Craig Chasseur, Qiang Zeng, Ian Rae, Anurag Biyani, Andrew Harn, Yang Xia, Andrey Gubichev, Amr El-Helw, Orri Erling, Zhepeng Yan, Mohan Yang, Yiqun Wei, Thanh Do, Colin Zheng, Goetz Graefe, Somayeh Sardashti, Ahmed M. Aly, Divy Agrawal, Ashish Gupta, and Shiv Venkataraman. F1 query: Declarative querying at scale. *Proc. VLDB Endow.*, 11(12):1835–1848, aug 2018.

[37] José Santos, Chen Wang, Tim Wauters, and Filip De Turck. Diktyo: Network-aware scheduling in container-based clouds. *IEEE Transactions on Network and Service Management*, 20(4):4461–4477, 2023.

[38] 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*, SOSP '23, page 498–514, New York, NY, USA, 2023. Association for Computing Machinery.

[39] Harsha Sharma, Parth Thakkar, Sagar Bharadwaj, Ranjita Bhagwan, Venkata N. Padmanabhan, Yogesh Bansal, Vijay Kumar, and Kathleen Voelbel. Optimizing network provisioning through cooperation. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, pages 1179–1194, Renton, WA, April 2022. USENIX Association.

[40] 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. Technical report, Google, Inc., 2010.

[41] Arjun Singh, Joon Ong, Amit Agarwal, Glen Anderson, Ashby Armistead, Roy Bannon, Seb Boving, Gaurav Desai, Bob Felderman, Paulie Germano, Anand Kanagala, Jeff Provost, Jason Simmons, Eiichi Tanda, Jim Wanderer, Urs Hölzle, Stephen Stuart, and Amin Vahdat. Jupiter rising: A decade of clos topologies and centralized control in google's datacenter network. In *Sigcomm '15*, 2015.

[42] Márton Sipos, Josh Gahm, Narayan Venkat, and Dave Oran. Network-aware feasible repairs for erasure-coded storage. *IEEE/ACM Transactions on Networking*, 26(3):1404–1417, 2018.

[43] Moritz Steiner, Bob Gaglianello Gaglianello, Vijay Gurbani, Volker Hilt, W.D. Roome, Michael Scharf, and Thomas Voith. Network-aware service placement in a distributed cloud environment. In *Proceedings of the ACM SIGCOMM 2012 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, SIGCOMM '12, page 73–74, New York, NY, USA, 2012. Association for Computing Machinery.

[44] Tino Tereshko and Jordan Tigani. Bigquery under the hood: Google's serverless cloud data warehouse | google cloud blog, Jan 2016.

[45] Muhammad Tirmazi, Adam Barker, Nan Deng, Md E. Haque, Zhijing Gene Qin, Steven Hand, Mor Harchol-Balter, and John Wilkes. Borg: the next generation. In *EuroSys '20: Fifteenth EuroSys Conference 2020, Heraklion, Greece, April 27-30, 2020*, pages 30:1–30:14, 2020.

[46] TPC. Transaction Processing Performance Council (TPC). https://www.tpc.org, 2024.

[47] Yiting Xia, Ying Zhang, Zhizhen Zhong, Guanqing Yan, Chiun Lin Lim, Satyajeet Singh Ahuja, Soshant Bali, Alexander Nikolaidis, Kimia Ghobadi, and Manya Ghobadi. A social network under social distancing: Risk-Driven backbone management during COVID-19 and beyond. In *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*, pages 217–231. USENIX Association, April 2021.

[48] Rohit Zambre, Megan Grodowitz, Aparna Chandramowlishwaran, and Pavel Shamis. Breaking band: A breakdown of high-performance communication. In *Proceedings of the 48th International Conference on Parallel Processing*, pages 1–10, 2019.

[49] 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, SIGCOMM '22, page 1–18, New York, NY, USA, 2022. Association for Computing Machinery.

[50] Shizhen Zhao, Rui Wang, Junlan Zhou, Joon Ong, Jeffrey C. Mogul, and Amin Vahdat. Minimal rewiring: Efficient live expansion for clos data center networks. In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*, pages 221–234, Boston, MA, February 2019. USENIX Association.

[51] Yibo Zhu, Haggai Eran, Daniel Firestone, Chuanxiong Guo, Marina Lipshteyn, Yehonatan Liron, Jitendra Padhye, Shachar Raindel, Mohamad Haj Yahia, and Ming Zhang. Congestion control for large-scale rdma deployments. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, SIGCOMM '15, page 523–536, New York, NY, USA, 2015. Association for Computing Machinery.

Appendix

A *QuerySys* Dataflow

Aggregation queries involve two stages (Figure 10), communicating via a shuffle primitive provided by *RamStore*. In the first stage, workers read data from *D* servers using the Colossus read API, perform parallel computations, and write intermediate results to *RamStore* memory hosts. Second, workers read these results, perform further computations, and write final results to memory hosts.

This process involves many-to-many communication between workers, *D* servers, and memory hosts, making it difficult to correlate ToR utilization with operation latency (§4.2). To simplify this analysis, we focus on the maximum ToR utilization observed across all ToRs[3] involved in each worker's operation (Table 4). This approach allows us to capture the impact of the worst hotspot on the operation's performance.

B Impact of Chunk Sizes

Figure 21 shows how chunk sizes impact the *Load-tolerance*. We look at 3 different operation size buckets: *10KB – 100KB*, *100KB – 1MB*, and *1MB–10MB*, and measure the 2× *Load-tolerance* for HDD reads and writes. A larger *Op size* decreases *Load-tolerance*, *i.e.,* making it more sensitive to the utilization. Thus, for example, reads of small chunks of 10-100 KB can tolerate fully loaded ToRs with a latency inflation of less than 2. For these small chunks, read access time is dominated by disk read latency (disk seeking), and so should, in general, offer worse performance than larger reads. In contrast, large reads of a MB or more incur a latency inflation of 2 even with a 50% ToR utilization, because they are bottlenecked by the network. HDD write operations have a low *Load-tolerance* even for small chunks, since writes do not wait for disk accesses, and hence are dominated by network latency at any size.

[3]Except for Colossus read operations we only consider the ToR utilization of workers but not *D* servers due to telemetry limitations.

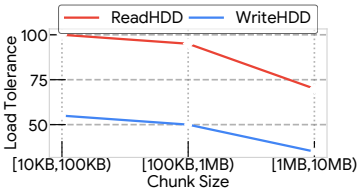


Figure 21: The impact of chunk sizes on the *Load-tolerance*.

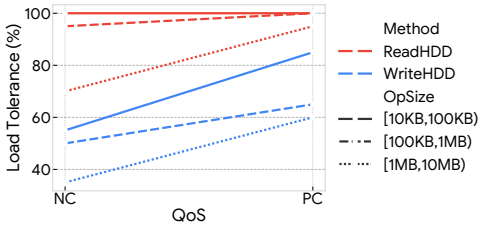
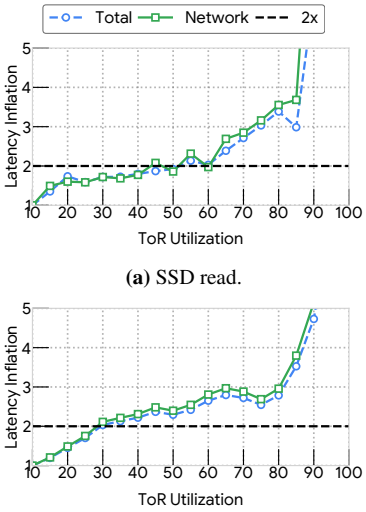


Figure 22: In all Op sizes, moving from NC QoS to PC QoS increases the *Load-tolerance*.



(a) SSD read.

(b) *RamStore* read.

Figure 23: The latency inflation of SSD and *RamStore* reads as a function of ToR utilization.

C Impact of QoS

Typical cluster applications in public clouds, including storage, classify their traffic into three priority classes: performance-critical (*PC*) traffic is associated with user-facing applications, non-critical (*NC*) traffic such as bulk storage operations generally cares about sustained rate, and best-effort (*BE*) traffic such as background analytics and backup traffic has the lowest priority. *PC*, *NC*, and *BE* traffic are mapped to high, medium, and low QoS classes respectively [49].

We measure network utilization every 30 seconds at each ToR for each quality of service (QoS) level. In our results reported below, the utilization always refers to the weighted-effective utilization of the given QoS class. The weighted-effective utilization is defined by the utilization of each

Operation	Description	Associated Network Utilization
Colossus Read	Read data from Colossus to worker	worker downlink
Shuffle Write	Write output from a worker to *RamStore*	**MAX**(worker uplink, *RamStore* memhost downlink)
Shuffle Read	Read the result generated by the previous stage from *RamStore* to worker	**MAX**(worker downlink, *RamStore* memhost uplink)

Table 4: The time of individual *QuerySys* worker's operation and the associated network utilization.

QoS and the weight ratio between QoSes. In practice, a flow of higher QoS can still be impacted by traffics of a lower QoS due to the weighted queues at the network devices. The weighted-effective utilization is capturing this subtlety with a weighted sum of all QoSes traffic occurs in the given ToR for each interval. Lower QoS contributions have smaller weight (< 1) and higher QoS traffic will have a larger weight (> 1).

Figure 22 shows that both HDD read and write has higher *Load-tolerance* when move from *Non-critical* (NC) to *Performance-critical* (PC) QoS. This is because traffic of the higher QoS observe less queuing time than that of the lower QoS. This less queuing delay will behave as a higher *Load-tolerance*.

D SSD and *RamStore*

SSD reads are much faster than HDD reads, and thus are more sensitive to network utilization. As shown in Figure 23a, SSD reads are more sensitive to high ToR utilization than HDDs. The *Hotspot-inflation* for SSD read is about $2.5\times$, and the $2\times$ *Load-tolerance* is about 55%. Thus, even when ToR utilization is not high enough to be classified as a hotspot, SSD read latency can be more than double the latency on an unloaded network.

RamStore reads have lower latency than Colossus since it stores the data in RAM. For this reason, *RamStore* is more sensitive to higher utilization and hotspots than any of the storage subsystems discussed so far. As shown in Figure 23b, *RamStore*'s $2\times$ *Load-tolerance* is 30%, and its *Hotspot-inflation* is 3.