Fast and Accurate Load Balancing for Geo-Distributed Storage Systems

Kirill L. Bogdanov
KTH Royal Institute of Technology
kirillb@kth.se

Waleed Reda
Université catholique de Louvain
KTH Royal Institute of Technology
wfhsr@kth.se

Gerald Q. Maguire Jr.
KTH Royal Institute of Technology
maguire@kth.se

Dejan Kostić
KTH Royal Institute of Technology
dmk@kth.se

Marco Canini
KAUST
marco@kaust.edu.sa

ABSTRACT

The increasing density of globally distributed datacenters reduces the network latency between neighboring datacenters and allows replicated services deployed across neighboring locations to share workload when necessary, without violating strict Service Level Objectives (SLOs).

We present Kurma, a practical implementation of a fast and accurate load balancer for geo-distributed storage systems. At run-time, Kurma integrates network latency and service time distributions to accurately estimate the rate of SLO violations for requests redirected across geo-distributed datacenters. Using these estimates, Kurma solves a decentralized rate-based performance model enabling fast load balancing (in the order of seconds) while taming global SLO violations. We integrate Kurma with Cassandra, a popular storage system. Using real-world traces along with a geo-distributed deployment across Amazon EC2, we demonstrate Kurma's ability to effectively share load among datacenters while reducing SLO violations by up to a factor of 3 in high load settings or reducing the cost of running the service by up to 17%.

CCS CONCEPTS

• **Networks** *Network performance analysis*; • **Computer systems organization** **Cloud computing**; • **Information systems** *Distributed storage*; **Key-value stores**;

KEYWORDS

Distributed Systems, Wide Area Networks, Cloud Computing, Service Level Objectives, Server Load Balancing

ACM Reference Format:
Kirill L. Bogdanov, Waleed Reda, Gerald Q. Maguire Jr., Dejan Kostić, and Marco Canini. 2018. Fast and Accurate Load Balancing for Geo-Distributed Storage Systems. In *Proceedings of SoCC '18: ACM Symposium on Cloud Computing , Carlsbad, CA, USA, October 11–13, 2018 (SoCC '18),* 15 pages.
https://doi.org/10.1145/3267809.3267820

SoCC '18, October 11–13, 2018, Carlsbad, CA, USA
2018. ACM ISBN 978-1-4503-6011-1/18/10…$15.00
https://doi.org/10.1145/3267809.3267820

1 INTRODUCTION

Modern interactive Web services require both predictable and low response times [21, 43]. These requirements are often specified in terms of Service Level Objectives (SLOs) and expressed as a maximum bound on a target percentile (e.g., 95th) of the response time. Failure to meet SLOs results in penalties, lost revenue for service providers, or both [59, 87].

Meeting strict SLOs is a challenging task [34], because Web services demonstrate temporal and spatial variability in load [23, 40]. Moreover, the workload can change due to sudden spikes in content popularity [49, 93] caused by major events [48] or failures. Datacenter-level load balancers [9, 36, 39, 75] are restricted by the capacity of the cluster in which they run and cannot meet service time guarantees when the load exceeds this capacity.

To satisfy increasing demands, cloud providers are continuously expanding the number of datacenters and their geographic coverage [16, 90]. This has led to an increased geographic density of datacenters. Service providers exploit increased datacenter density to deploy Web services closer to users, which reduces median and tail response times, and to replicate data, which increases service reliability and ensures survivability even during a complete datacenter failure [17, 97, 98].

We leverage increased data center density to realize Kurma, a fast and accurate geo-distributed load balancer. By accurately estimating the rate of SLO violations, Kurma can reduce SLO violations by redirecting requests across the Wide Area Network (WAN) as shown in Fig. 1.[1] Whereas, by operating at the granularity of seconds, Kurma can work in tandem with modern elastic controllers, thereby reducing over-provisioning and SLO violations incurred during provisioning delays.

[1] Fig. 1 shows selected results from Fig. 8a, see §7.1 for details.

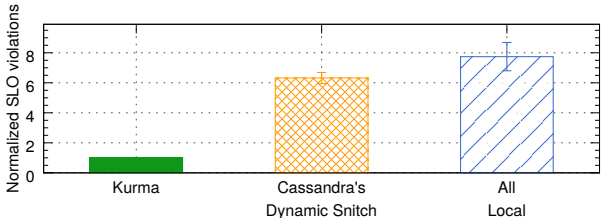


Figure 1: Due to its fast and accurate load balancing, Kurma achieves significant reduction in SLO violations relative to the strictly local request serving strategy and Cassandra's load balancer operating across the WAN. Plot normalized to Kurma.

1.1 Challenges in Load Balancing

To overcome capacity limitations, service providers automatically scale local resources within individual datacenters by utilizing techniques such as automatic resource allocation and speed scaling [41, 57, 58, 69, 86]. Unfortunately, these approaches have fundamental limitations as illustrated in the load curves shown in Fig. 2.[2] First, automatic resource scaling requires time to (i) detect the need to scale up, (ii) acquire and start new service instances, and (iii) warm up (integrate) new instances into a working cluster (all combined into "Provisioning delay" shown in the figure). For example, Amazon EC2 recommends scaling at a frequency of 1 minute to quickly adapt to load changes [8]. However, the average VM startup time on EC2 is around 2 minutes [22]. Moreover, it can take over 2 minutes for a Cassandra instance to start operating at full capacity [69] (excluding the time necessary for data replication). As a result, the provisioning time may be much longer in practice than shown in the figure.

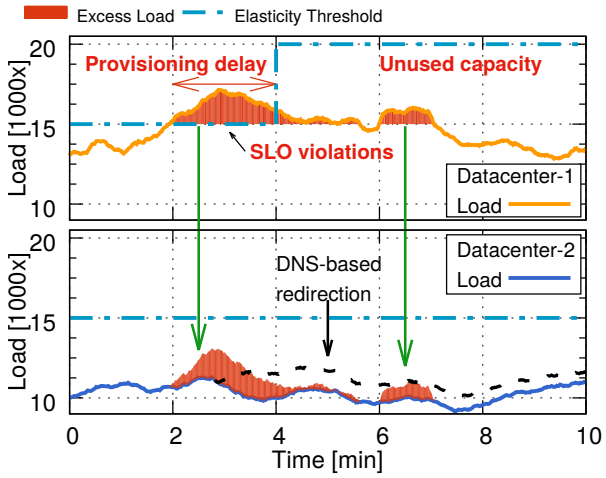


Figure 2: Challenges of elastic scaling and geo-distributed load balancing (red fill under the curve for Datacenter-1 represents load that would lead to SLO violations), and Kurma's approach (green arrows).

To avoid SLO violations *during* the provisioning period, techniques are needed to forecast upcoming workloads sufficiently far into the future to account for provisioning delays. However, this forecasting is known to be difficult, given the unpredictable nature of flash crowds and failures [41, 69]. This results in wasteful over-provisioning [14, 54]. While over-provisioning can reduce SLO violations, the challenge is to provide the best quality of service to customers while minimizing the cost of operating the service. Kurma, addresses this challenge by providing *fast and accurate* geo-distributed load balancing.

Coarse-grained load balancing via Domain Name System (DNS) servers operates at the level of individual clients and provides only limited control [26, 30]. Moreover, it does not take the actual load of the target server into account [25, 80], and, due to caching, it cannot respond quickly to changes in workloads [24, 74].

Fine-grained load balancing of requests among geo-distributed datacenters presents a number of difficult challenges. To operate effectively, a geo-distributed load balancer needs to answer the following difficult questions in a timely manner: From the point of view of each datacenter, can requests be redirected such that responses will return within the SLO bound? How many requests should be redirected without overloading remote datacenters? What rates of SLO violations are to be expected?

Existing work on geo-distributed load balancing does not fully address these challenges, as it either targets the average response time [12, 45, 80] (but cannot guarantee SLO enforcement), uses a modeling approach to estimate server performance [12, 52, 61, 101] (which may not accurately capture complex system dynamics), or overlooks variability of WAN latency [45, 52, 60].

Solving these challenges requires us to look beyond end-to-end response time percentiles among datacenters; we must dissect how these percentiles change as load is balanced among datacenters and as WAN conditions change. Moreover, our design must be able to quickly react to global changes while avoiding oscillations, herd behaviors, and stale data decision.

1.2 Kurma Research Contributions

We present Kurma, a *fast and accurate* geo-distributed load balancer for backend storage systems of Web services. To the best of our knowledge, Kurma is the first system that accounts for the actual service time and inter-datacenter WAN latency distributions to accurately estimate the rate of SLO violations when redirecting requests among datacenters. Kurma's primary objectives are to: (i) globally minimize or bound SLO violations under a dynamic, global workload or (ii) reduce the cost of running a service.

Contribution (1): Taming SLO violations. Kurma decouples request completion time into (i) service time, (ii) base network propagation delay, and (iii) residual WAN latency caused by network congestion.[3] At run-time, Kurma tracks changes in each component independently and thus, accurately estimates the rate of SLO violations *among* geo-distributed datacenters from the rate of incoming requests. Kurma *tames* the rate of SLO violations (local or global) while load balancing requests across a geo-distributed storage system. This allows Kurma to satisfy SLO objectives while redirecting as few requests as possible, effectively minimizing inter-datacenter traffic and associated costs.

Contribution (2): Fast adaptability. Each datacenter periodically (at the granularity of a few seconds) solves a decentralized rate-based performance model to compute the rates at which datacenters should redirect requests among each other. This allows Kurma to take advantage of short-term decorrelated changes (spikes) in load across datacenters by redirecting requests towards neighboring datacenters that currently have spare capacity (e.g., redirection can take place between datacenters in neighboring regions, with different time zones or cultural patterns).

[2] The load curves shown in Fig. 2 are an illustrative example of two experimental traces. Actual elasticity thresholds can differ based on hardware. The rate of DNS redirection is based on an estimate of client's session departure rate discussed in §7.1.2.

[3] We define residual network latency as a one-sided distribution obtained by subtracting base propagation delay from packet delays.

Kurma achieves cost savings (i) by avoiding unnecessary scaling out (e.g., as a result of intermittent spikes in load) when the load can be shared among neighboring datacenters *without* violating SLO targets (which relies on Kurma's ability to accurately estimate the rate of SLO violations, see §7.1.3), and (ii) by reducing global over-provisioning by allowing spare capacity to be shared among neighboring datacenters. Kurma can be used stand-alone or in combination with existing cloud elasticity techniques [27, 46, 69]. In the latter case, Kurma's redirection can buy time for the associated elasticity techniques to scale up *without* incurring excessive SLO violations during the provisioning delay (see Fig. 2).

Contribution (3): Practical evaluation in a real system. We implement Kurma in the Datastax CQL driver of the Cassandra database. Using real-world traces, we evaluate Kurma across datacenters of Amazon EC2 and simulations. Kurma reduces SLO violations over existing techniques by up to a factor of 3 and reduces the operational costs by up to 17%.

2 RELATED WORK

A large body of work has been conducted in the area of load balancing for geo-distributed clusters [12, 35, 45, 52, 60, 80, 96, 98]. Content Delivery Networks (CDNs) [35] rely on request redirection, which is done via DNS. Donar [96] builds a general-purpose service selection mechanism that can also be used for this purpose. In our evaluation (§7), we highlight Kurma's fast adaptation by comparing it with the DNS- and Donar-like approaches. Relative to the modeling approaches such as WARD [79, 80] and the work by Kanizo et al. [52], Kurma operates at a granularity of seconds and rapidly adapts to workload changes. Moreover, Kurma adapts to both the variability in network latency and uneven load distribution among datacenters. Ardagna et al. [12] integrate geo-distributed load balancing with elastic scaling; however, unlike Kurma, they can only provide SLO bounds in terms of average response time.

Cardellini et al. [25] let an overloaded Web server initiate request redirection to other servers based on a threshold metric, such as percentile of the end-to-end response time. Dealer [45] computes a Weighted Moving Average (WMA) of service time and network latencies among service components of a geo-distributed service. In contrast to both approaches, Kurma decomposes network latency into base propagation delay and residual latency. While Wendell et al. [96] mention the possibility of incorporating network latency variance, Kurma achieves this in practice.

Dynamic data replication can be used as a form of load balancing [4, 13, 85, 97]. Shankaranarayanan et al. [85] minimize response time percentiles for geo-distributed datastores by solving a data placement model. In contrast, Kurma considers service time delays that could be affected by changes in replication policies and reacts much faster to median WAN latency changes (seconds vs. hours). Spanstore [97] replicates data by adhering to a target SLO percentile; however, it does not take service time into consideration and cannot estimate how the rate of SLO violations will change with a change in load. Tuba [13] and Volley [4] perform storage system reconfigurations periodically in the order of hours, whereas Kurma works at the level of seconds and can adapt much faster to changes in load.

Cloud elasticity. Numerous reactive [8, 51, 57, 72, 73] and proactive [27, 69, 86, 99, 100] elastic scaling techniques aim to maintain applications' SLOs under dynamic workloads by sizing the number of nodes that handle requests. However, third-party cloud providers (such as EC2) do not provide access to the hypervisor, thus certain techniques are inapplicable [41, 69, 86, 103].

The common challenge of these techniques is to accurately forecast workloads sufficiently far into the future to spawn additional VMs and quickly warm up the application. This is typically compensated for by some form of over-provisioning [38, 86], which is wasteful. In contrast, Kurma aggregates the spare capacity of a few neighboring datacenters that are accessible within the SLO bound, thus reducing global over-provisioning. Moreover, by rapidly adjusting to changes in load and redirecting requests, Kurma provides time for the elasticity techniques to scale up.

3 KURMA DESIGN

Reference system. Kurma targets a multi-tier service architecture, which is common for modern Internet-scale Web services. The target service is assumed to be deployed across a set of geo-distributed datacenters interconnected by a WAN. Clients access the service at one of the datacenters based on traditional DNS-based load balancing. Once clients' requests arrive at application servers (load balanced through frontend servers), these servers in turn generate tens to thousands of individual requests for the backend servers (e.g., a distributed database such as Cassandra). Meeting a strict SLO for the overall client requests' completion times depends on consistently delivering low-latency responses from the service's backend, despite multiple sources of performance variability [34, 89].

Overview. Kurma tames SLO violations at the service's backend by realizing an efficient geo-distributed load balancer that accurately estimates the rates of SLO violations for requests that are served locally and those that are redirected across the WAN. Fig. 3 presents an overview of our approach. An instance of Kurma runs at each of the service's datacenters. Each Kurma instance periodically performs the following tasks: (i) it monitors the load (specifically the rate of requests to this backend, read/write ratio, and request sizes), measures WAN latency to remote datacenters, and monitors SLO violations for requests served locally and remotely; (ii) exchanges the measured load, WAN latency, and SLO violations with other Kurma instances; and (iii) computes *inter-datacenter* request redistribution rates and enforces these rates at the application servers. The problem of intra-datacenter load balancing is well understood [36, 39, 71, 75]. Hence, Kurma does not address intra-datacenter load balancing, but rather relies on existing load balancing within the datacenter. We further assume that by redirecting requests, Kurma does not cause network congestion.

Kurma solves an optimization problem to determine the request redistribution rates (i.e., how to load balance requests among datacenters) to minimize *or* bound the global number of SLO violations at a target level (e.g., 5%). In particular, each Kurma instance computes the redistribution rates based on three inputs: (1) current loads, (2) distribution of WAN latencies, and (3) a family of SLO curves, one per pair of datacenters. Loads and WAN latencies were gathered in task (ii). For each pair of source application tier and a destination storage tier, an SLO curve describes the relationship

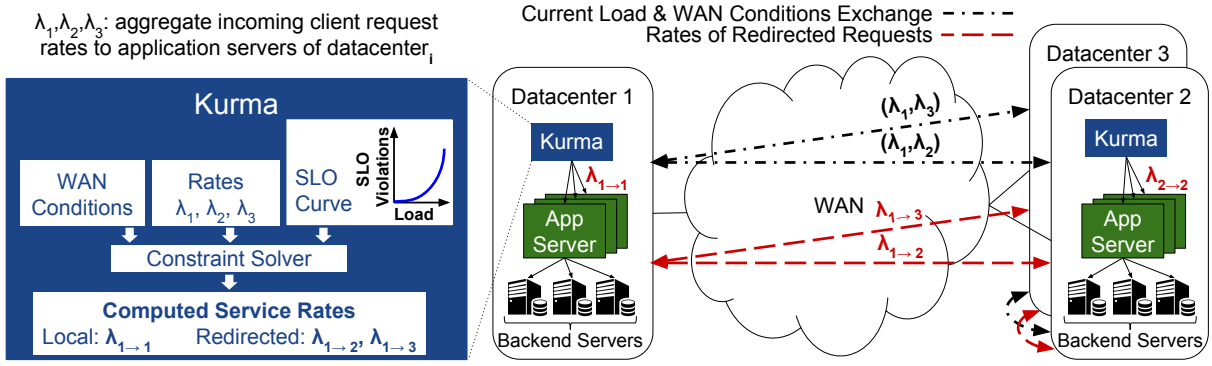


Figure 3: Kurma overview.

between the offered load and the expected fraction of requests that would violate the SLO. An SLO curve is parametrized based on the current load (i.e., request sizes, arrival rate, and read/write ratio), datacenter capacity (i.e., the number of backend servers currently running), and the WAN latency distribution from the sender to the datacenter. An initial set of SLO curves is obtained via offline backend profiling (see §4) or can be estimated at run-time using queue modeling techniques. SLO curves are adjusted at run-time according to measured inter- and intra-datacenter network latencies (base propagation and residual latency).

If interference is detected, Kurma performs SLO curve substitution by selecting the best fitting curve from a family of previously obtained SLO curves based on the closest match between the expected and actual rates of SLO violations. When solving the optimization problem at run-time, each Kurma instance deterministically chooses an appropriate SLO curve based on current conditions. The process of selecting an appropriate SLO curve is quick (see §7.1.4).

Application servers in a datacenter enforce the request redistribution rates computed by that datacenter's Kurma instance. Because Kurma only computes aggregate rates, these need to be enforced by the application servers in a distributed manner. This problem is well suited to distributed rate-limiting techniques [9, 78, 88]. These techniques have been applied within datacenter environments where servers can communicate frequently and with very low latencies. We assume that a similar approach can be employed in our design, but for clarity present our solution in terms of aggregate rates. Furthermore, using aggregate rates is appealing as it makes the approach scale better.

4 LOAD VERSUS SLO VIOLATIONS: LOCAL AND REMOTE

Fig. 4 shows the relationship between a system's throughput and SLO violations in the case of a Cassandra cluster. In this experiment, we profiled a five-server cluster deployed at Amazon EC2 in Frankfurt. The SLO target was set to obtain a 95^{th} percentile latency of 30 ms. We gradually increased the offered load until we hit the cluster's saturation point at around 55k req/s (shown by the black arrow in the bottom plot). Beyond this point, the arrival rate exceeds the service rate, and the servers' queues start to grow (unbounded).

Fig. 4 also shows that this cluster of five servers can sustain at most 43k req/s before 5% of the responses to requests exceed the SLO. Thus, a load of 43k req/s defines the cluster's saturation point for the 95^{th} percentile (shown by the blue arrow in the top graph). In the presence of an elastic controller, this level of load would trigger the addition of a VM. Similar load and resource pressure models (e.g., CPU and RAM utilization) are fairly accurate and are described in works on elastic scaling [38, 57, 69, 73]. However, applying them directly to geo-distributed load balancing was not done previously; primarily due to the difficulty in accurately estimating SLO violations in remote datacenters given dynamically changing WAN conditions.

The *three-way relationship* between the load, WAN latency distribution, and rate of SLO violations has important implications when attempting to load balance across a geo-distributed system. Consider a scenario where a remote datacenter located in Ireland attempts to redirect its requests to a neighboring datacenter in

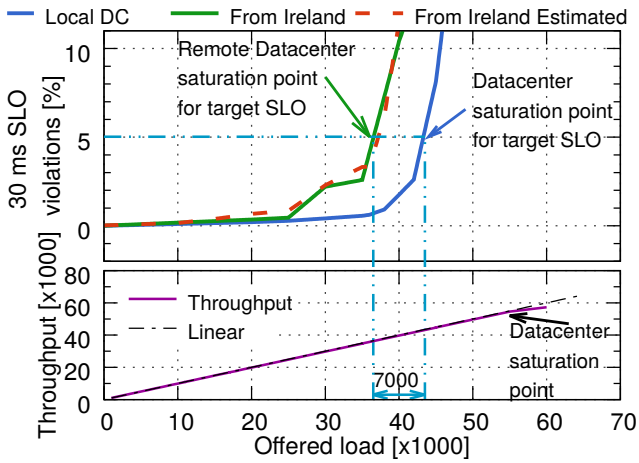


Figure 4: Relationship between throughput and the rate of SLO violations for a five-server Cassandra cluster running on Amazon EC2 on `r4.large` instances. The workload was generated using an open loop workload generator with a Poisson request interarrival distribution.

Frankfurt. The WAN RTT between these datacenters is 22 ms (at the time of this measurement); therefore, keeping SLO violations under 5% when doing request redirection is viable only when Frankfurt can serve 95% of requests in under 8 ms. This requires that the utilization at the Frankfurt datacenter be below 78%. This relationship is captured by the green line in Fig. 4, which shows SLO violations observed by the Ireland datacenter for requests redirected to Frankfurt. The green line shows that the SLO violations for redirected requests increases faster than for requests that are served locally. Consequently, the load at the remote datacenter (assuming the same hardware configuration) should not exceed 36k req/s in contrast to 43k req/s when requests are served locally. In other words, this implies that the farther away the remote datacenter is, the less loaded it should be in order to serve remote requests within their SLO target. Naturally, this creates a trade-off between the *effective* distance between neighboring datacenters and the load (on the receiver's side) that a datacenter would experience while serving redirected requests.

To navigate these trade-offs, Kurma constructs a set of SLO curves that is local to each datacenters (discussed in §4.1), then for each pair of source and destination datacenters, Kurma combines the destination datacenter's SLO curves with the WAN latency between the two datacenters, in order to estimating the expected rate of SLO violations for requests redirected from the source to destination datacenters (discussed in §4.2).

4.1 Constructing Local SLO Curves

To construct SLO curves, we profile a warmed-up backend cluster of a fixed size within a single datacenter under gradually increasing loads and variable read/write ratios. For each profiled configuration we (i) measure the percentile that corresponds to the SLO target latency (e.g., 30 ms at 95^{th} percentile)[4] *and* (ii) preserve the measured service time distribution for use at run-time in combination with WAN latency distribution in order to accurately estimate the rate of SLO violations for requests sent from remote datacenters (see §4.2). Collecting the entire service time distribution is crucial, as pressure models (obtained either offline or online) commonly used in elastic controllers (i.e., load vs rate of SLO violations) *cannot* accurately be combined with a joint distribution of network delays to estimate the rate of SLO violations in remote datacenters.

The sample loads are spaced exponentially, but with more sample points closer to the datacenter's saturation point; thus, giving Kurma greater accuracy around the inflection point of the SLO curve. Each profiled configuration produces a single point in a multi-dimensional space and represents the expected rate of SLO violations for that configuration. At run-time, based on the current workload for each datacenter, Kurma selects an individual SLO curve that is a three dimensional surface that maps a workload mix (reads and writes) to the expected rate of SLO violations (blue line in Fig. 4 shows the curve for reads only). Kurma uses bilinear interpolation to estimate the rate of SLO violations for read/write ratios that were not explicitly profiled.

[4]Some services might distinguish between read and write operations by having different SLO targets for each (i.e., due to distinctly different service times), and this can be accounted for when constructing the SLO curve.

Our current prototype relies on offline profiling to establish the initial relationship between the load and the rate of SLO violations; in §7.1.1 we also show how this relationship can be estimated using queue modeling formulae [44, 52, 80, 101]. Furthermore, assuming linear or near-linear scaling of modern storage systems [29], the SLO curve of a datacenter can be derived from a joint distribution of service times of individual servers in the cluster. Kurma can utilize SLO curves constructed using any of the above techniques.

4.2 Including WAN Latency Distribution

To reason about the SLO violations at remote datacenters, we incorporate a WAN latency distribution into the SLO curves at run-time. One way to achieve this would be to repeat the offline profiling while generating the workload from remote datacenters, thus measuring end-to-end response time distributions of redirected requests that incorporate both current WAN latency and service time. However, this approach does not scale well (as it is quadratic in the number of datacenters). Furthermore, WAN conditions often change [28, 47], which would require *re-profiling* the system on a regular basis.

To address this issue, we view the total request completion time as two components: service time within a datacenter and WAN latency between datacenters. We perform service time profiling only once (as described above), then at run-time, we reuse these service time distributions and combine them with WAN latency to obtain an accurate SLO curve for each pair of datacenters.

To accurately and timely incorporate WAN latency into an SLO curve, we account for both the *base* propagation delay (which depends primarily on physical distance and can change when routing changes) as well as the residual latency (which is the result of queuing and congestion and depends on the level of network utilization).

Routing changes can appear as distinct shifts in network latency [28, 76] that can cause temporal skew in the measured end-to-end delay distribution. Methods that are oblivious to these shifts (e.g., variants of Exponentially Weighted Moving Average (EWMA)) will experience delays in adaptation to changes in latency distribution caused by such routing changes. In contrast, by measuring these quantities separately, Kurma rapidly reacts to detected routing changes and selects the pre-computed SLO curve that matches the current base propagation latency. When combined with the (locally-profiled) service time distribution, this yields an accurate remote response time distribution and thus can be used to estimate the rate of SLO violations for redirected requests (for more details see [19]).

Fig. 5 shows the estimated remote service time between the sender and receiver located in the Frankfurt and Ireland datacenters, respectively. We use Monte Carlo sampling to jointly sample from the distribution of residual latency (blue line) and service time (orange line) distributions. The joint distribution is then combined with the base propagation delay (vertical dash-dotted line)[5] to estimate the remote service time distribution (green line). We empirically validated this curve by comparing it with the distribution of service times measured from the remote datacenter. We find that

[5]Note, the base propagation latency is not constant throughout the measurement interval and shows the last known value.

the curves are well aligned, suggesting this estimation technique has *good accuracy*.

Moreover, measuring network latency as a single metric (WMA [45] or a specific percentile [85, 97]) is insufficient as percentiles of two distributions are not additive, thus such a metric cannot be combined with the service time distribution to estimate percentile of the joint distribution.

In contrast, the black dashed curve in Fig. 5 shows the remote service time obtained by jointly sampling from the raw latency distribution and service time distribution *without* decoupling the WAN latency into base propagation latency and residual latency. The difference is substantial even for a well provisioned network with a relatively small range of propagation delays (between 22 and 27 ms) and would result in under- or over-estimating the rate of SLO violations.

The process of incorporating two WAN components with a set of service time distributions to estimate SLO curves is not computationally intensive and can be completed within milliseconds. Network congestion is typically infrequent across WAN links [28], thus, the re-computation does not need to occur often. Furthermore, SLO curves can be precomputed for the expected range of base propagation delays (e.g., with a step of 1 ms) allowing for instantaneous run-time selection of the appropriate curve under routing changes; this can happen at the frequency of model recomputation.

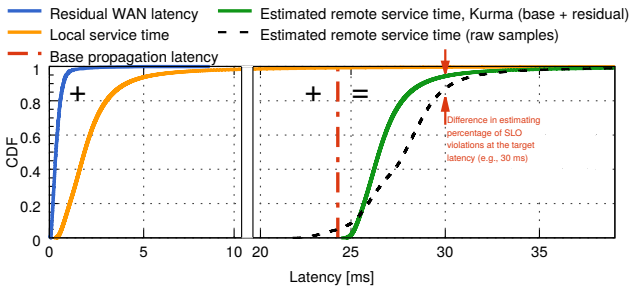


Figure 5: Remote service time estimation.

5 COMPUTING REDISTRIBUTION RATES

We now introduce the load redistribution model used in Kurma. Table 1 summarizes the primary notations used in this paper. Throughout, we use $i, j \in N$ to denote datacenters in the set of datacenters N. The model's outputs are the rates λ_{ij} at which application servers at i redistribute requests to backend servers at j.

We denote by D_i the input request rate at i. These requests are generated by application servers at i. Thus, the total demand at i is $\Lambda_i = D_i - \sum_j \lambda_{ij} + \sum_j \lambda_{ji}$.

Ω denotes a family of SLO curves. These SLO curves are obtained periodically as described in the previous section and are treated as an input from the viewpoint of model computation. In particular, Ω_{ij} is the SLO curve for the i, j pair of origin-destination datacenters, respectively. Each SLO curve is a function of request rate and ρ, the SLO violation threshold. For a datacenter i redirecting requests to datacenter j, $\lambda_{ij}\Omega_{ij}(\Lambda_j, \rho)$ gives the expected rate of SLO violations of requests redirected from i to j.

Table 1: Notations used in the model formulation.

N	Set of geo-distributed datacenters
λ_{ij}	Rate at which application servers in i redirect backend requests to j
D_i	Input rate of backend requests at datacenter i
Λ_i	Total demand of backend requests at datacenter i
ρ	SLO violation threshold (e.g., 5% for 95th percentile)
$\Omega_{ij}(\Lambda_j, \rho)$	SLO curve of requests redistributed from i to j; Ω_{ij} is a function of demand at j and ρ

Next, we introduce two optimization models: KurmaPerf, which aims to minimize global SLO violations and KurmaCost, which is designed to minimize cost while complying with SLO bounds.

5.1 KurmaPerf

The objective of KurmaPerf is to minimize global SLO violations across a geo-distributed service:

$$\begin{aligned} \min_{\lambda_{ij}} \quad & \sum_i \sum_j \lambda_{ij}\Omega_{ij}(\Lambda_j, \rho) \\ \text{subject to} \quad & \sum_i \lambda_{ij} = D_i, \quad \forall i \\ & \lambda_{ij} \geq 0, \quad \forall i, j \\ & \left(\sum_{j:j\neq i} \lambda_{ij}\right)\left(\sum_{j:j\neq i} \lambda_{ji}\right) = 0, \quad \forall i \end{aligned} \tag{1}$$

The first constraint establishes demand satisfaction. The second constraint requires non-negative rates. The last constraint means that a datacenter cannot concurrently redistribute requests to other datacenters while receiving requests from other datacenters. We added this last constraint after we experimentally verified that the kind of request redistribution it prevents: (i) is cost-inefficient, as it results on average in more redirects for little gain and (ii) greatly increases model computation time as the solution space is much larger.

5.2 KurmaCost

By minimizing global SLO violations, KurmaPerf improves overall application performance. However, it comes at the expense of redirecting more requests over WAN links than are strictly necessary to meet the SLO target. Therefore, we introduce KurmaCost, an alternative optimization model to satisfy SLO objectives while redirecting as few requests as possible, effectively minimizing inter-datacenter traffic:

$$\begin{aligned} \min_{\lambda_{ij}} \quad & \sum_i \sum_{j:j\neq i} \lambda_{ij} \\ \text{subject to} \quad & \sum_j \frac{\lambda_{ij}\Omega_{ij}(\Lambda_j, \rho)}{D_i} \leq \rho, \quad \forall i \\ & \text{and same constraints as in (1)} \end{aligned} \tag{2}$$

The additional constraint above imposes that the total SLO violations experienced by every datacenter must be below the SLO target.

In contrast to KurmaPerf's focus on global SLO violations, KurmaCost focuses on local SLO violations.[6] This difference is particularly important in relation to elasticity controllers. Elasticity controllers are typically deployed in a decentralized fashion and provision nodes based on performance indicators at each local datacenter (e.g., local SLO violations). By ensuring that each datacenter's SLO violations remain below a stated threshold, KurmaCost works in tandem with elastic controllers to avoid scaling out unnecessarily, which further reduces costs.

These optimization problems are non-convex (as can be observed by simply noting the non-convexity of the complementarity constraints in 1). Thus, it is challenging to solve them exactly. Our approach is to quantize SLO curves such that each λ_{ij} is a multiple of a minimum load balancing quantum (the default is 1% of the total capacity of a datacenter). This enables the solver to consider all possible solutions if necessary, which is not costly given the settings and the modest number of datacenters that we consider. By increasing the minimum load balancing quantum, we reduce the model's computation time at the expense of load balancing precision. Our technical report [20] provides sensitivity analyses for both the minimum load balancing quantum and the model re-computation interval.

6 IMPLEMENTATION

We implement the core logic of Kurma in the Datastax Java driver [33] — a library that provides an API for communicating with Cassandra. Even though our Kurma instances are logically-separated from the application tier, such a driver implementation consolidates both the Kurma instance and application-server logic at the same node. The driver establishes TCP connections with local and remote backend servers, allowing Kurma to have full control of where requests are redirected.

Kurma-to-Kurma communication. We distribute request rates, measured SLO violations, and WAN conditions among the Kurma instances — via a full-mesh broadcast that sends messages once per model recomputation interval. Global state dissemination and model computation are synchronized among all Kurma instances using NTP [66]. Fig. 6 shows the model execution and communication timeline. Triangular markers (m_k and m_{k+1}) indicate globally synchronized model recomputations at fixed intervals. The red circle indicates a globally synchronized state dissemination point, i.e., the moment when all nodes broadcast their state. The exact time of the broadcast is determined by the time necessary for all messages to reach all datacenters. Specifically, Δt is deterministically computed by all Kurma instances at run-time and equals the one-way WAN delay between the two farthest datacenters in the system and a fixed safety margin to compensate for NTP error and processing delays (set to 20 ms by default). This guarantees that all Kurma instances will receive identical up-to-date information just before the next scheduled model recomputation. This message exchange creates negligible overhead both in terms of network bandwidth and associated costs. Alternatively, gossiping protocols [37, 53] could be used to address potential communication scaling issues.

[6] By considering a weighted sum of SLO violations across all datacenters as a constraint, it is straightforward to extend KurmaCost to operate with a global SLO target.

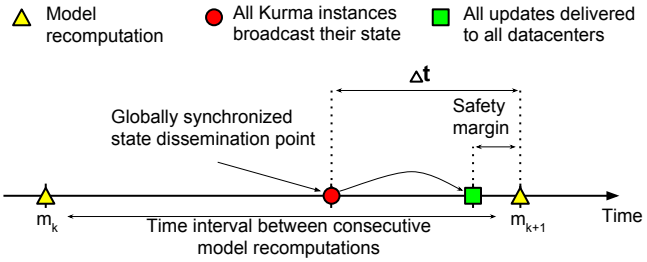


Figure 6: Kurma exchanges its states once per model recomputation interval (denoted m_k and m_{k+1}). The time when instances exchange their messages is determined at run-time such that all datacenters receive the latest update just before next model recomputation.

Solving the model. To implement the model, we use the MiniZing constraint modeling language [68]. We compile and solve the model using a Gecode constraint solver [84] at configurable intervals (default 2.5 s). However, other modeling languages and solvers can be used to solve Kurma's model. At run-time, Gecode is pinned to a single dedicated CPU core.

Currently, Kurma maintains the same ratio of reads and writes for redirected rates as in the source datacenter (i.e., λ_{ij} has the same read/write ratio as D_i).

In the normal operating mode, Kurma does not utilize a direct feedback loop for SLO violations, thus oscillations and herd behaviors are not possible despite the system's rapid reactions to changes in load. In its current implementation, SLO violation feedback is exploited only when VM interference is detected and run-time adjustments to the SLO curve are needed; however, this feedback operates at much slower pace (minutes vs. seconds) than model recomputation and does not cause oscillations.

WAN measurements. As noted before, we decouple residual network latency from the base propagation latency [18, 64]. Hence before each experiment, we conduct a short-term (5 minute) network measurement among all datacenters. First, in each datacenter we deploy a set of measurement probes (3 by default) that perform periodic TCP-level RTT measurements towards probes deployed in remote datacenters (default measurement interval is 200 ms). The obtained latency samples are post-processed by removing the base propagation latency from each latency sample, thus leaving only residual network congestion distribution (for more details see [19]).

Then, for each pair of source and destination datacenters, we pre-compute a family of SLO curves by combining each destination's datacenter service time distributions with the network congestion distributions between the datacenters (as described in §4). Furthermore, for each pair of datacenters, we expand the family of SLO curves by considering the previously observed range of base propagation network latencies with a step size of 1 ms.

At run-time, each instance of Kurma measures the base propagation WAN latency from itself to remote datacenters. For each destination, Kurma monitors the minimum response time over a one-second window. Then, we subtract the minimum service time that we obtained during offline system profiling. The resulting base propagation latency is then rounded off to the nearest ms and used as an index to select a specific SLO curve from the family of SLO

curves obtained in the previous step. These WAN measurements are then exchanged among the distributed Kurma instances.

Based on our measurements, the WAN between Amazon EC2's three neighboring datacenters (Frankfurt, Ireland, London) is well provisioned and congestion is rare; although, routing changes do occur regularly. Therefore, for the 30 minute evaluations we measured the residual latency distribution only once before each experiment. However, for long term production deployments it would be advisable to measure residual WAN latency distribution and recompute SLO curves at run-time.

7 EVALUATION

We evaluate Kurma and present experimental results comparing its performance with other geo-distributed load balancing techniques in real-world settings. We answer the following questions: (i) How effective is KurmaPerf at minimizing SLO violations (§7.1)? (ii) How accurately does KurmaCost adhere to a target SLO bound (§7.1.3)? (iii) How much cost savings can KurmaCost achieve (§7.2)?

Evaluation methodology. To evaluate Kurma, we deployed Cassandra clusters across three geo-distributed datacenters of Amazon's EC2 located in Frankfurt, London, and Ireland. Each datacenter hosted up to 5 `r4.large` on-demand instances comprising the actual cluster and one `c4.4xlarge` instance running the YCSB workload generator [31]. The replication factor was set to 3 (each key is replicated 3 times in each datacenter). In all our experiments we assume eventual consistency — which is commonly used in practice [50, 94]. We use consistency level ONE for both reads and writes. In line with the average value sizes found in production systems [15], we populate the database with 1 million keys that map to values of 150 bytes. The dataset was stored using Amazon's general purpose SSDs [6]. To minimize the impact of garbage collection on our measurements, we ran both Cassandra and YCSB instances on the Zing JVM [91]. For all evaluations the SLO target was set to 30 ms at the 95th percentile.

Workload traces. We evaluate Kurma using real-world traces with temporal variations in workload (obtained from [2]). These traces represent a Web-based workload and were recorded across multiple geo-distributed datacenters over a period of 88 days. The traces show the rate of object requests per datacenter at one second resolution. For each second of a trace we fit a Poisson distribution to allow us to estimate the inter-arrival request rates at sub-second resolution. Table 2 shows the mapping between the original traces to the datacenter where we have replayed them with the indicated shift in time to correlate these traces with the time zones used for our experiments.

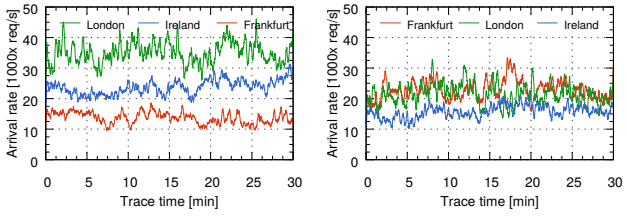
We modified YCSB to dispatch requests according to the timestamps recorded in a trace.[7] For each experiment we verify that the workload generator is able to keep up with the required sending rate, thus acting as an open loop workload generator. Key popularity was set according to a Zipf distribution (as in [31]).

For the evaluation on Amazon EC2, we selected two distinct intervals of 30 minutes (shown in Fig. 7). Both intervals were taken from a single day of the trace[8] and scaled by 450 to match the capacity of our hardware testbed while preserving workload variations. For brevity, we refer to them as Trace-1 and Trace-2, respectively. Trace-1 demonstrates a major load imbalance, where one of the datacenters is more loaded than the rest. This provides an opportunity for load redirection. In contrast, in Trace-2, spare capacity is very limited and constantly shifts among datacenters, making load balancing challenging. To operate effectively in this trace a geo-distributed load balancer needs to recognize and act upon load balancing opportunities.

[7] Source code is available at [82].

Table 2: Mapping between the datacenters where a trace was initially recorded to the datacenter where it was replayed (left). Observed range of WAN base propagation latencies between datacenters (right).

Source datacenter	Time shift (hrs)	Replayed in	Observed base propagation RTT (ms) London	Ireland	Frankfurt
Virginia	+0	London	0	9-10	11-14
Texas	+1	Ireland	9-10	0	22-27
California	+4	Frankfurt	11-14	22-27	0



(a) Trace-1: 147 M requests. 5-VMs per datacenter. **(b) Trace-2: 105 M requests. 3-VMs per datacenter.**

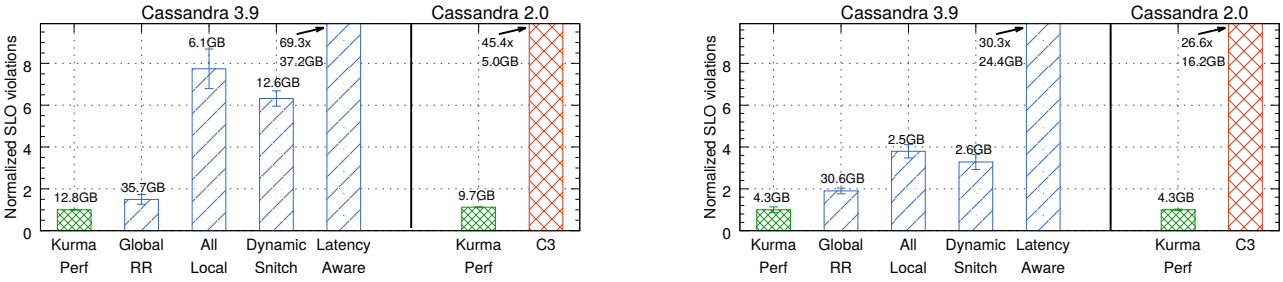
Figure 7: Two workload traces used in the evaluation.

7.1 How Effective is KurmaPerf at Minimizing SLO Violations?

We experimentally evaluated Kurma's ability to reduce SLO violations under dynamic workloads. Specifically, we evaluate the following alternative techniques: ***GlobalRR***: Classical Round Robin algorithm that uniformly balances requests among all backend servers of all datacenters. ***AllLocal***: All datacenters serve incoming requests locally without any redirection. ***LatencyAware***: Uses an EWMA of the response times to choose the best performing backend servers (as implemented in [33]). ***DynamicSnitch***: This is Cassandra's default strategy that performs dynamic replica selection [11]. ***C3***: A state of the art distributed load balancing technique [89].[9] ***MMc***: Kurma operates on SLO curves estimated using M/M/c modeling. We configure Kurma to track the average request arrival rate over a 5 s window, with its model re-computation interval set

[8] Day #49 [2] starting at 10:00 and 19:00 hours respectively based on Virginia's time zone.

[9] Because C3 has only been implemented in Cassandra version 2.0, we repeated Kurma's evaluation using two different versions of Cassandra (i.e., 2.0 and 3.9). Since Kurma was implemented in the CQL driver, it is backwards compatible with Cassandra 2.0.



(a) Trace-1, executed with 5 VMs per datacenter.

(b) Trace-2, executed with 3 VMs per datacenter.

Figure 8: Normalized SLO violations achieved on Amazon EC2 (reads only). Each bar represents an average value across five experiments. Kurma reduces the global SLO violations by about 3x when compared to schemes that do not blindly spraying requests across the WAN. The number above each bar is the total data transfer (in GB) between datacenters incurred by each technique over 30 minutes. Note, AllLocal technique also generates inter-datacenter WAN traffic, due to read repairs and gossiping among geo-distributed Cassandra nodes. The absolute average value of SLO violations achieved by Kurma in (a) and (b) are 1.1%/0.8% and 2.4%/1.1% (Cassandra 3.9 and 2.0 respectively).

to 2.5 s, with a load balancing resolution of 1%. Auto-scaling was turned off, thus the observed SLO violations is a direct indicator of each load balancing technique being able to actively redirect load while utilizing spare capacity among datacenters throughout the experiments.

7.1.1 Minimizing SLO Violations for Reads. Fig. 8 shows the SLO violations for the different techniques — normalized by Kurma's violations. Unsurprisingly, ***GlobalRR*** achieves the second best result after Kurma, as uniformly distributing requests among all datacenters ***GlobalRR*** avoids "hot spots" and because *all* datacenters were within the service's SLO bound (resulting in a relatively low rate of SLO violations). However, this is an ideal scenario for this technique as, in real settings, not all datacenters might be viable targets due to excessive WAN latency; hence, applying this technique could lead to unsatisfactory performance. Moreover, it consumes 2.9 and 7.1 times more bandwidth than Kurma in the two traces, respectively; hence, if deployed, it would incur a high cost.

LatencyAware maintains an EWMA of latencies to each node. It times out underperforming nodes with latencies higher than those of the fastest node by a pre-defined "exclusion threshold". However, in practice it is unclear how to set the timeout period and the exclusion threshold. Using the default values (2.0 and 10 s), this technique results in the second highest number of SLO violations, possibly because it enforces an aggressive exclusion algorithm that can result in herd behaviors [67, 83].

DynamicSnitch uses an exponentially decaying reservoir [32] to track median request completion time, but does not decouple network latency and service time components. The median latency of the requests sent remotely is much higher than for the local nodes; hence, dynamic snitch fails to exploit remote redirection opportunities and, as a result, heavily favors local reads.

Overall, ***C3*** provides poor performance for both traces. We argue that this is a direct consequence of the fact that C3's cubic function heavily penalizes nodes with larger queue sizes. Specifically, due to WAN delays, C3 greatly overestimates queue sizes at the remote servers, leading to suboptimal load balancing decisions. While this might work well within a single datacenter, we find that this scheme provides suboptimal partitioning of load on a geo-distributed scale.

Is profiling of a real system beneficial? To build SLO curves for an M/M/c model, we follow the steps outlined in §4; however, we estimate the percent of SLO violations under different levels of load by computing M/M/c sojourn time distribution (see page 46 of [3]). Based on our measurements we set $c = 10$ and $\mu = 5500k$. At an SLO target of 5%, M/M/c overestimates cluster capacity by 11000 req/s.

To evaluate the effects of using M/M/c we ran the same set of experiments as in Fig 8. Under both workloads (Trace-1 and Trace-2) Kurma performs identically to *AllLocal*. However, as the SLO curves estimated by M/M/c do **not** represent the actual system's behavior, Kurma does not redirect a sufficient number of requests.

While other queuing techniques could be used to produce a more accurate estimate of the response time, comparing these techniques is outside the scope of this work. Here, we merely demonstrate that Kurma can operate with different types of SLO curves as inputs. In summary, for our evaluations, we found that using real system profiling proved to be very beneficial because standard modeling approaches did not produce an accurate relationship between load and the rate of SLO violations.

7.1.2 Minimizing SLO Violations with a Mix of Reads and Writes. Next, we ran read/write experiments with Trace-2 and a 4% write ratio per datacenter. In Cassandra, writes are always propagated to all replicas that hold the given key, whereas the consistency setting merely implies the number of replicas required to confirm the write operation before a response can be sent back to the client. All datacenters propagate their fraction of writes to each other, causing each datacenter to experience a variable write ratio (8% to 16%) throughout the trace duration.

We introduce two additional baselines in which we used Kurma's SLO curves and model to decide on the actual load redistribution; however, we configured the system's responsiveness to match two prominent techniques: ***DNS*** and ***EWMA***. The ***DNS*** case is an approximation of DNS based load balancing that takes clients' session stickiness into account. We used the client's session departure rate from Fig. 5(b) in [60] to obtain an estimate of the rate

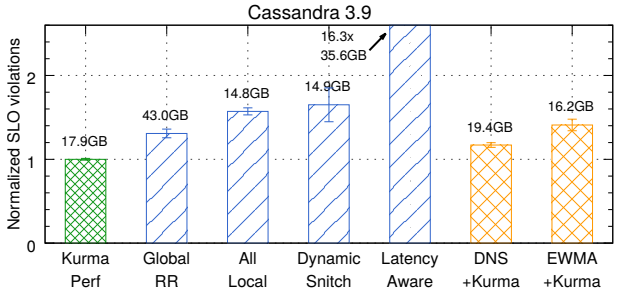


Figure 9: Normalized SLO violations achieved on Amazon EC2 (reads and writes on Trace-2). Each bar represents average value across five experiments. Absolute average value of SLO violations achieved by KurmaPerf is 5.07%

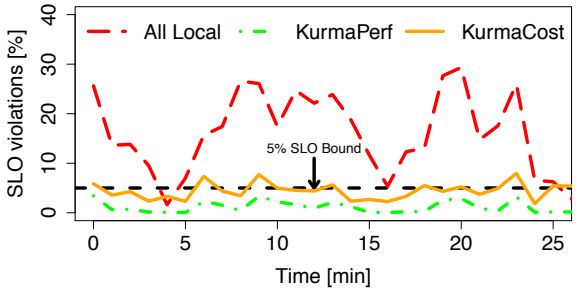


Figure 10: SLO violations in London with Trace-1, reads only. We show that, while KurmaPerf keeps the SLO violations well below 5%, KurmaCost still adheres to the SLO threshold while being more cost-effective.

limit at which load can be redirected among datacenters (we set this limit to 1.3%/s, i.e., 3.25% per 2.5 s of the model's re-computation interval). ***EWMA*** is a slow-paced, model-based load balancing approach tailored towards adapting to diurnal patterns (based on Donar's configurations [96]). Specifically, we tracked the rate of request arrival as an EWMA ($\alpha = 0.8$, interval 10 minutes) with model re-computation every 10 minutes.

Fig. 9 shows the results. Using Kurma's model, ***DNS*** was able to outperform ***GlobalRR*** and ***AllLocal***, although due the stickiness of its clients, the technique generated more intra-datacenter traffic and showed lower SLO reduction when compared to KurmaPerf. ***EWMA*** was able to track at a coarse granularity the load trends at each datacenter; by redirecting a fraction of requests it showed improvements over no redirection at all. However, it was unable to take advantage of short-term variability in load, thus demonstrated much lower performance than Kurma.

While in this evaluation we considered only full replication ([56, 62]), Kurma can work with multiple keyspaces and dynamic replication policies (e.g., when a fraction of the most popular keys reside at the caching servers [94]). Kurma is inherently aware of keyspaces' replication policies, allowing it to direct requests to datacenters that can serve these requests while adhering to the redirection rates provided by the model.

7.1.3 Maintaining Target SLO. Fig. 10 shows a time series of the local SLO violations for London — our most loaded datacenter — averaged at one minute intervals. We can see that both KurmaPerf and KurmaCost significantly reduce SLO violations compared to AllLocal. This highlights the fact that KurmaCost maintains the SLO violations close to its configurable target of 5%, thus minimizing the number of redirections compared to KurmaPerf; hence it is more cost-efficient.

7.1.4 Adapting to Performance Variability. In this section we show how Kurma can adapt to detected cloud interference. We use KurmaPerf with the same setup as in §7.1, but use a synthetic workload with a constant arrival rate of 10k req/s at each datacenter.

Fig. 11 shows measurements of Kurma's SLO violations in Frankfurt and London averaged over 1 minute intervals. In the interval up to 2 minutes the average rate of SLO violations in both datacenters is low and within 0.1% of the expected value. At around 2 minutes, we introduce CPU intensive processes on 2 out of 5 VMs in Frankfurt. This causes SLO violations to rise above 2% and thus deviates markedly from the expected value. One minute later we emulate reception of an interference signal from a specialized tool (e.g., DeepDive [70]). At the next scheduled model recomputation interval, Kurma performs a search through the family of SLO curves to find a better match for the observed rate of SLO violations. The best fitting SLO curve is determined using least squares fitting by comparing the expected rate of SLO violations with the observed rate over a set of recent measurements exchanged via the global state dissemination. Natural workload variability allows Kurma to obtain multiple sample points on the curve at run-time. The process is fast and deterministic, thus all instances of Kurma find identical matches and start using the appropriate SLO curve in the next round of model recomputations.

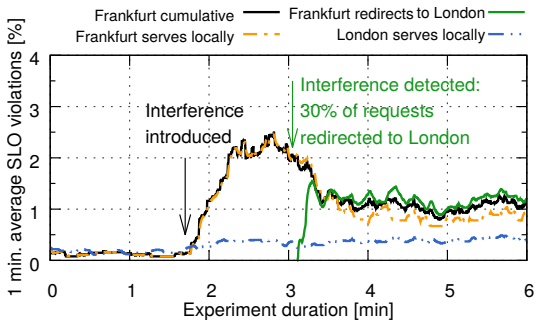


Figure 11: Kurma adapts to detected performance interference by selecting appropriate SLO curves and adjusting its request redirection rates.

The total rate of SLO violations for Frankfurt is a sum of SLO violations for requests that are being served locally (orange dashed line) and requests that are being redirected to London (green solid line). Note, due to WAN latency between two datacenters, requests redirected from Frankfurt to London have a higher rate of SLO violations than requests that originated and are served in London approximately 1.2% and 0.5% respectively at the 4 minutes mark.

Kurma was able to adapt to (externally) detected interference and adjusted its selection of SLO curve for Frankfurt, consequently it was able to greatly reduce the rate of SLO violations in Frankfurt while marginally increasing the rate of SLO violations in London.

Alternatively, to account for performance variability associated with multitenancy in clouds, Kurma could use multiple approaches: rely on performance isolation [9], rate control [102], smart resource controller [55, 63, 72], deploying on dedicated VM instances [5], or dynamically rebuild SLO curves through VM isolation and online re-profiling [46, 69]. We leave to future work incrementally adapting and rebuilding SLO curves.

7.1.5 How Well Can Kurma Scale? Kurma computes its model sufficiently fast for today's scale of several to ten datacenters per provider; solving the model for 5 datacenters using a load balancing quantum of 1% (default) requires a median computation time under 10 s, while solving for 8 datacenters with quantum of 8% takes 1 s without inflating SLO violations. Full details are available in [20]. Deployments with higher densities of datacenters will likely still have a limited number of data centers that theoretically allow for the SLO to be met, thus we claim that Kurma will have no difficulty addressing future needs even in its current form.

7.2 How well can Kurma reduce cost?

Kurma can reduce the cost of running a service by avoiding excessive global over-provisioning. Specifically, it attempts to redirect load away from a datacenter *before* it becomes overloaded and would require scaling out. In this section, we leverage simulations to evaluate potential cost savings achievable using KurmaCost and KurmaPerf models. We utilize our previous testbed settings with three datacenters and use static inter-datacenter WAN latencies, i.e., without routing changes and network congestion. We selected continuous 30-days of workload traces.[10]

We assume the presence of a threshold based elastic controller in each datacenter (e.g., EC2 Auto Scaling[7]). When the incoming load in a given datacenter exceeds a threshold that matches 5% SLO violations, the controller adds an additional VM. The actual threshold values were obtained during our offline profiling (see §4). We configured the controller to operate at the granularity of one minute (as suggested by Amazon EC2 [8]). Thus, for every minute of the trace, we estimate the expected rate of SLO violations, pass this information to the elastic controller that subsequently makes a scaling decision on a per datacenter level.

The operating cost, for each evaluated technique, was computed as a sum of the costs of VM provisioning (1 VM costing $0.133/hour) and inter-datacenter WAN traffic (costing $0.01/GB).[11] The total cost of WAN traffic was computed as a product of the total number of redirected requests and the average request/response size measured experimentally (375 bytes in our setup).

Evaluated techniques. As an upper bound for operating costs we used the *AllLocal* strategy where each datacenter has to serve all incoming requests locally without redirects. Cost savings were calculated relative to this upper bound. For the lower bound we compute the VM provisioning in the *AllShared* setting where all load can be shared amongst datacenters without any penalty for WAN latency or costs for redirected traffic. While this is not achievable in practice, it puts the other techniques into perspective.

In contrast, before triggering the elastic controller, both of Kurma's models try to distribute the load amongst datacenters such that their corresponding objectives are achieved (i.e., minimizing global SLO violations and bounding SLO violations at 5% margin in each datacenter). When either model would exceed the SLO target, the elastic controller scales up the overloaded datacenter.

Today, the minimum billing period from third-party cloud providers is 1 minute [1, 42, 65]. However, depending on the type of service and the size of a VM's state, it might be impossible to turn on/off VMs at such a high frequency. Therefore, for completeness we performed evaluations using minimum billing periods of 1 and 60 minutes. For each configuration, we report average savings per day over a 30 day period.

Fig. 12 shows the results for 1 minute billing interval. This is considered a worst case scenario for Kurma's relative savings given that VM allocations can be more flexibly provisioned to accommodate changes in workload. KurmaCost is only 7% off from the absolute lower bound — that assumes that all datacenters are co-located. With a 60-minute billing interval (figure excluded for brevity), KurmaPerf can reduce costs by up to 15%, while KurmaCost can reduce costs by up to 17% and this is only 6% from maximum attainable savings.

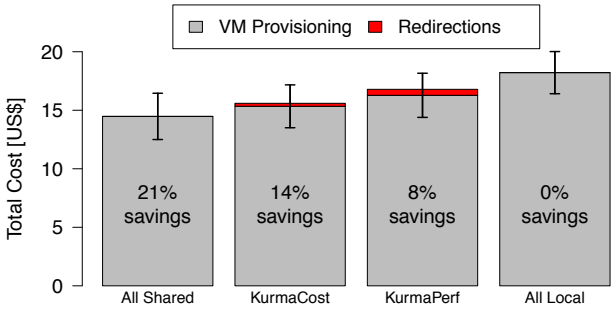


Figure 12: Total cost of provisioning VMs and redirecting requests for 24-hours (averaged over 30 days), minimum billing period is 1 minute. KurmaCost is only 7% off from the maximal attainable savings.

Currently, KurmaCost considers uniform cost for inter-datacenter WAN traffic and uniform cost of computation in each datacenter. However, these costs could vary depending on the datacenters' locations, the time of a day, and electricity sources currently available to each datacenter[61, 77, 81]. By considering these costs as a set of additional parameters, Kurma's model can easily be extended to cover such pricing cases.

8 LIMITATIONS

Predictable service time distribution. Kurma inherently assumes predictable, low variance service time of the target system, such that it is possible to establish a relationship between the rate of request arrival and the rate of SLO violations. If the variance of service time is too high, then the estimate of SLO curve will not be accurate, leading to suboptimal performance (i.e., redirecting

[10] Days 34-64 from [2], scaled up for a cluster of 5 VMs per datacenter.
[11] For this analysis, we ignore the cost of gossiping traffic as it is negligible.

too many or too few of requests). For example, the fan-out of a search query might be unknown *apriori* making it difficult to derive expectations about query completion time. For other setups, the variance could be a consequence of the system's software delays. In Cassandra, garbage collectors can cause unexpected stalls, resulting in a high service time variance. We address this issue by deploying Cassandra with Zing JVM and its pauseless garbage collector [91]. Our results show that even with Java we can get good results. We expect that systems that show lower variance in service times (e.g., RAM based caches and systems written in native programming languages), Kurma should be able to operate even better.

Common redundancy techniques such as request duplication [95, 98] and speculative retries [10], can also be used with storage systems to reduce latency variance. We argue that such methods can be layered on-top of Kurma to improve service time predictability, which goes hand in hand with Kurma's load balancing performance.

9 DISCUSSION

How general is Kurma's design? Even though this paper's focus was mainly on using key-value store deployed across geo-distributed datacenters, we believe that the mechanism for estimating the expected rate of SLO violations for redirected requests used in Kurma is fairly generic and can be used for other applications and deployments. Specifically, utilizing Kurma for load balancing can be advantageous for services whose communication delays and service times are of comparable orders of magnitudes.

How well can Kurma perform under strong consistency? Our current prototype assumes services with eventual consistency — which is a common use-case today [50, 92, 94]. Kurma performs load balancing by carefully choosing requests' split ratios among available datacenters. Strong consistency, typically, requires simultaneous communication with multiple replicas, which naturally limits available choices and could reduce the benefits of load balancing in general. Moreover, the effect of stragglers can obscure some of the benefits attained from accurate geographic load balancing.

Can Kurma work without full replication? Kurma is inherently aware of keyspaces' replication policies and assumes that all data is replicated at every datacenter. This allows it to direct requests towards datacenters that *can* serve clients' requests while at the same time adhere to the redirection rates provided by the model. However, having data (e.g., key or row) replicated only in a subset of datacenters would limit the number of requests that *can* be redirected between a source and destination datacenters depending on the mix of clients' requests arriving at the source datacenter.

While not currently implemented, Kurma's model can be further extended to support such replication policies via an additional constraint on the request redirection matrix λ_{ij}. Specifically, each value λ_{ij} would be bounded by the number of the arriving requests at i that can be served at j.

Can Kurma operate under failures? Kurma relies on the presence of an external mechanism to detect failures and inform distributed instances of Kurma.[12] At run-time, Kurma's model handles changes in the number of datacenters and changes to datacenters' capacities (i.e., the number of servers in each datacenter). For example, if a server fails, all distributed Kurma instances switch to an SLO curve that matches the reduced capacity of that datacenter. Alternatively, if an entire datacenter becomes unavailable or network partitioning occurs, Kurma excludes inaccessible datacenters when solving its model. Thus, no redirection rates are assigned to unavailable datacenters.

Can Kurma operate with different workload mixes? Service time distribution of a storage system can change non-negligibly depending on the workload's characteristics, e.g., request sizes and read/write ratio. We assume, that in practice these characteristics will be known for the most common workloads, making it possible to incorporate them into the initial offline profiling (see Section 4.1). At run-time, Kurma can then react to changes in workload mixes by simply switching between pre-computed SLO curves as necessary.

In §7.1.2 we showed that Kurma can operate well with two distinct types of request (namely reads and writes). We believe that the dimensionality of SLO curves could be expanded to cover request size distributions. Operating under variable request sizes is left as future work.

How can Kurma be deployed on large-scale systems? Kurma is designed to be logically centralized at a datacenter level. However, the current prototype does not address coordination among multiple intra-datacenter instances of Kurma, as would be necessary for large scale production deployments. To achieve this, multiple Kurma instances, deployed within one datacenter, would need to perform distributed rate limiting towards the backend tier. Existing works in this domain, such as [78, 88] could be integrated with Kurma. We leave integration and evaluation of these techniques for future work.

10 CONCLUSION

Kurma is the first system that takes into account the actual service time *and* inter-datacenter WAN latency distributions to accurately estimate the rate of SLO violations for requests redirected across geo-distributed datacenters. Kurma realizes a decentralized geo-distributed load balancing system that quickly performs accurate, global load balancing decisions within a few seconds, enabling it to rapidly react to changes in load. By operating at the granularity of seconds, Kurma can work in tandem with modern elastic controllers, thereby reducing over-provisioning and SLO violations incurred during provisioning delays. We demonstrated that our techniques can operate with both reads and writes and can optionally reduce the cost of running a service.

Acknowledgments. We thank Göran Andersson and Christian Schulte for their help with designing and reviewing Kurma's optimization model. We are grateful to Marco Chiesa for his comments on earlier drafts of this paper. This work is financially supported by the Swedish Foundation for Strategic Research. This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 770889). Waleed Reda was supported by a fellowship from the Erasmus Mundus Joint Doctorate in Distributed Computing (EMJD-DC) program funded by the European Commission (EACEA) (FPA 2012-0030).

[12] Current Kurma's implementation utilizes Datastax CQL driver that maintains a list of alive Cassandra nodes.

REFERENCES

[1] Amazon EC2. http://aws.amazon.com/ec2/ Accessed 28-Aug-2018.

[2] The Internet Traffic Archive. http://ita.ee.lbl.gov/html/contrib/WorldCup.html Accessed 01-Feb-2018.

[3] ADAN, I., AND RESING, J. *Queueing theory*. Eindhoven University of Technology Eindhoven, 2002.

[4] AGARWAL, S., DUNAGAN, J., JAIN, N., SAROIU, S., WOLMAN, A., AND BHOGAN, H. Volley: Automated Data Placement for Geo-Distributed Cloud Services. In *Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation* (Berkeley, CA, USA, 2010), NSDI'10, USENIX Association, pp. 2–2.

[5] AMAZON. Amazon EC2 Dedicated Instances. https://aws.amazon.com/ec2/purchasing-options/dedicated-instances/ Accessed 28-Aug-2018.

[6] AMAZON. Amazon Elastic Block Store. https://aws.amazon.com/ebs/details/ Accessed 28-Aug-2018.

[7] AMAZON. EC2 Auto Scaling. https://docs.aws.amazon.com/autoscaling/ec2/userguide/what-is-amazon-ec2-auto-scaling.html Accessed 01-Jun-2017.

[8] AMAZON. Target tracking scaling policies. http://goo.gl/fzjZ92 Accessed 01-Jun-2017.

[9] ANGEL, S., BALLANI, H., KARAGIANNIS, T., O'SHEA, G., AND THERESKA, E. End-to-end Performance Isolation Through Virtual Datacenters. In *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)* (2014), pp. 233–248.

[10] APACHE. Cassandra. http://cassandra.apache.org/ Accessed 28-Aug-2018.

[11] APACHE. Cassandra, Dynamic Snitching. https://docs.datastax.com/en/cassandra/3.0/cassandra/architecture/archSnitchDynamic.html?hl=dynamic%2Csnitch Accessed 20-Jan-2018.

[12] ARDAGNA, D., CASOLARI, S., COLAJANNI, M., AND PANICUCCI, B. Dual time-scale distributed capacity allocation and load redirect algorithms for cloud systems. *Journal of Parallel and Distributed Computing 72*, 6 (2012), 796–808.

[13] ARDEKANI, M. S., AND TERRY, D. B. A Self-Configurable Geo-Replicated Cloud Storage System. In *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)* (2014), pp. 367–381.

[14] ARMBRUST, M., FOX, A., GRIFFITH, R., JOSEPH, A. D., KATZ, R. H., KONWINSKI, A., LEE, G., PATTERSON, D. A., RABKIN, A., STOICA, I., ET AL. Above the Clouds: A Berkeley View of Cloud Computing. Tech. rep., Technical Report UCB/EECS-2009-28, EECS Department, University of California, Berkeley, 2009.

[15] ATIKOGLU, B., XU, Y., FRACHTENBERG, E., JIANG, S., AND PALECZNY, M. Workload Analysis of a Large-Scale Key-Value Store. In *ACM SIGMETRICS Performance Evaluation Review* (2012), vol. 40, ACM, pp. 53–64.

[16] BARB DARROW. Amazon and Google Continue Cloud Arms Race With New Data Centers. 30-Sep-2016. Fortune.com. http://fortune.com/2016/09/30/amazon-google-add-data-centers/ Accessed 01-May-2018.

[17] BESSANI, A., CORREIA, M., QUARESMA, B., ANDRÉ, F., AND SOUSA, P. DepSky: Dependable and Secure Storage in a Cloud-of-Clouds. *ACM Transactions on Storage (TOS) 9*, 4 (2013), 12.

[18] BOGDANOV, K. *Reducing Long Tail Latencies in Geo-Distributed Systems*. Licentiate Thesis, KTH Royal Institute of Technology, 2016. ISBN: 978-91-7729-160-2 URN: urn:nbn:se:kth:diva-194729.

[19] BOGDANOV, K. *Enabling Fast and Accurate Run-Time Decisions in Geo-Distributed Systems: Better Achieving Service Level Objectives*. Doctoral Dissertation, KTH Royal Institute of Technology, 2018. Planned for the Fall 2018.

[20] BOGDANOV, K., REDA, W., MAGUIRE JR, G. Q., KOSTIĆ, D., AND CANINI, M. *Kurma: Fast and Efficient Load Balancing for Geo-Distributed Storage Systems (Technical Report)*. Technical report, KTH Royal Institute of Technology, 2018. http://urn.kb.se/resolve?urn=urn%3Anbn%3Ase%3Akth%3Adiva-222289.

[21] BRUTLAG, J. Speed matters for Google web search. *Google. June* (2009). https://services.google.com/fh/files/blogs/google$_{d}$elayexp.pdf Accessed 28-Aug-2018.

[22] BRYANT, R., TUMANOV, A., IRZAK, O., SCANNELL, A., JOSHI, K., HILTUNEN, M., LAGAR-CAVILLA, A., AND DE LARA, E. Kaleidoscope: Cloud Micro-Elasticity via VM state Coloring. In *Proceedings of the sixth conference on Computer systems* (2011), ACM, pp. 273–286.

[23] BUYYA, R., RANJAN, R., AND CALHEIROS, R. N. Intercloud: Utility-Oriented Federation of Cloud Computing Environments for Scaling of Application Services. In *International Conference on Algorithms and Architectures for Parallel Processing* (2010), Springer, pp. 13–31.

[24] CALLAHAN, T., ALLMAN, M., AND RABINOVICH, M. On modern DNS behavior and properties. *ACM SIGCOMM Computer Communication Review 43*, 3 (2013), 7–15.

[25] CARDELLINI, V., COLAJANNI, M., AND YU, P. S. Geographic Load Balancing for Scalable Distributed Web Systems. In *Modeling, Analysis and Simulation of Computer and Telecommunication Systems, 2000. Proceedings. 8th International Symposium on* (2000), IEEE, pp. 20–27.

[26] CARDELLINI, V., COLAJANNI, M., AND YU, P. S. Request Redirection Algorithms for Distributed Web Systems. *IEEE Transactions on Parallel and Distributed Systems 14*, 4 (2003), 355–368.

[27] CHANDRA, A., GONG, W., AND SHENOY, P. Dynamic Resource Allocation for Shared Data Centers Using Online Measurements. *ACM SIGMETRICS Performance Evaluation Review 31*, 1 (2003), 300–301.

[28] CHANDRASEKARAN, B., SMARAGDAKIS, G., BERGER, A., LUCKIE, M., AND NG, K.-C. A Server-to-Server View of the Internet. In *Proceedings of the 11th International Conference on emerging Networking EXperiments and Technologies* (Heidelberg, Germany, December 2015), CoNEXT '15, ACM, p. 40.

[29] COCKCROFT, A., AND SHEAHAN, D. Benchmarking Cassandra Scalability on AWS over a million writes per second. 1-Nov-2011. Netflix Technology Blog. https://goo.gl/Gtn2XH Accessed 28-Aug-2018.

[30] COLAJANNI, M., YU, P. S., AND CARDELLINI, V. Dynamic Load Balancing in Geographically Distributed Heterogeneous Web Servers. In *Distributed Computing Systems, 1998. Proceedings. 18th International Conference on* (1998), IEEE, pp. 295–302.

[31] COOPER, B. F., SILBERSTEIN, A., TAM, E., RAMAKRISHNAN, R., AND SEARS, R. Benchmarking cloud serving systems with YCSB. In *Proceedings of the 1st ACM symposium on Cloud computing* (2010), ACM, pp. 143–154.

[32] CORMODE, G., SHKAPENYUK, V., SRIVASTAVA, D., AND XU, B. Forward Decay: A Practical Time Decay Model for Streaming Systems. In *Data Engineering, 2009. ICDE'09. IEEE 25th International Conference on* (2009), IEEE, pp. 138–149.

[33] DATASTAX. Apache Cassandra Drivers. https://academy.datastax.com/download-drivers Accessed 1-Oct-2017.

[34] DEAN, J., AND BARROSO, L. A. The Tail at Scale. *Communications of the ACM 56*, 2 (2013), 74–80.

[35] DILLEY, J., MAGGS, B., PARIKH, J., PROKOP, H., SITARAMAN, R., AND WEIHL, B. Globally Distributed Content Delivery. *IEEE Internet Computing 6*, 5 (2002), 50–58.

[36] EISENBUD, D. E., YI, C., CONTAVALLI, C., SMITH, C., KONONOV, R., MANN-HIELSCHER, E., CILINGIROGLU, A., CHEYNEY, B., SHANG, W., AND HOSEIN, J. D. Maglev: A Fast and Reliable Software Network Load Balancer. In *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)* (2016), USENIX Association, pp. 523–535.

[37] EUGSTER, P. T., GUERRAOUI, R., KERMARREC, A.-M., AND MASSOULIÉ, L. Epidemic Information Dissemination in Distributed Systems. *Computer 37*, 5 (2004), 60–67.

[38] GANDHI, A., HARCHOL-BALTER, M., RAGHUNATHAN, R., AND KOZUCH, M. A. Autoscale: Dynamic, Robust Capacity Management for Multi-Tier Data Centers. *ACM Transactions on Computer Systems (TOCS) 30*, 4 (2012), 14.

[39] GANDHI, R., LIU, H. H., HU, Y. C., LU, G., PADHYE, J., YUAN, L., AND ZHANG, M. Duet: Cloud Scale Load Balancing with Hardware and Software. *ACM SIGCOMM Computer Communication Review 44*, 4 (2015), 27–38.

[40] GMACH, D., ROLIA, J., CHERKASOVA, L., AND KEMPER, A. Workload Analysis and Demand Prediction of Enterprise Data Center Applications. In *Workload Characterization, 2007. IISWC 2007. IEEE 10th International Symposium on* (2007), IEEE, pp. 171–180.

[41] GONG, Z., GU, X., AND WILKES, J. PRESS: PRedictive Elastic ReSource Scaling for cloud systems. In *2010 International Conference on Network and Service Management* (2010), IEEE, pp. 9–16.

[42] GOOGLE. Google compute engine pricing. https://cloud.google.com/compute/pricing Accessed 28-Aug-2018.

[43] GRAY, W. D., AND BOEHM-DAVIS, D. A. Milliseconds matter: An introduction to microstrategies and to their use in describing and predicting interactive behavior. *Journal of Experimental Psychology: Applied 6*, 4 (2000), 322.

[44] GUO, T., SHENOY, P., AND HACIGÜMÜS, H. H. Geoscale: Providing Geo-Elasticity in Distributed Clouds. In *Cloud Engineering (IC2E), 2016 IEEE International Conference on* (2016), IEEE, pp. 123–126.

[45] HAJJAT, M., SHANKARANARAYANAN, P., MALTZ, D., RAO, S., AND SRIPANIDKULCHAI, K. Dynamic Request Splitting for Interactive Cloud Applications. *IEEE Journal on Selected Areas in Communications 31*, 12 (2013), 2722–2737.

[46] HERODOTOU, H., DONG, F., AND BABU, S. No One (Cluster) Size Fits All: Automatic Cluster Sizing for Data-intensive Analytics. In *Proceedings of the 2nd ACM Symposium on Cloud Computing* (2011), ACM, p. 18.

[47] HØILAND-JØRGENSEN, T., AHLGREN, B., HURTIG, P., AND BRUNSTROM, A. Measuring Latency Variation in the Internet. In *Proceedings of the 12th International on Conference on emerging Networking EXperiments and Technologies* (2016), ACM, pp. 473–480.

[48] IAN PAUL, P. Jackson's Death a Blow to the Internet. http://www.pcworld.com/article/167435/jackson$_{d}$eath$_{b}$low$_{t}$o$_{i}$nternet.html Accessed 28-Aug-2018.

[49] JUNG, J., KRISHNAMURTHY, B., AND RABINOVICH, M. Flash crowds and denial of service attacks: Characterization and implications for CDNs and web sites. In *Proceedings of the 11th international conference on World Wide Web* (2002), ACM, pp. 293–304.

[50] KALANTZIS, C. Eventual Consistency != Hopeful Consistency. Talk at Cassandra Summit, 2013. https://www.youtube.com/watch?v=A6qzx$_{H}$E3EU.

[51] KALYVIANAKI, E., CHARALAMBOUS, T., AND HAND, S. Self-adaptive and self-configured CPU resource provisioning for virtualized servers using Kalman filters. In *Proceedings of the 6th international conference on Autonomic computing* (2009), ACM, pp. 117–126.

[52] KANIZO, Y., RAZ, D., AND ZLOTNIK, A. Efficient Use of Geographically Spread Cloud Resources. In *Cluster, Cloud and Grid Computing (CCGrid), 2013 13th IEEE/ACM International Symposium on* (2013), IEEE, pp. 450–457.

[53] KEMPE, D., DOBRA, A., AND GEHRKE, J. Gossip-based computation of aggregate information. In *Foundations of Computer Science, 2003. Proceedings. 44th Annual IEEE Symposium on* (2003), IEEE, pp. 482–491.

[54] KRIOUKOV, A., MOHAN, P., ALSPAUGH, S., KEYS, L., CULLER, D., AND KATZ, R. NapSAC: Design and Implementation of a Power-Proportional Web Cluster. *ACM SIGCOMM computer communication review 41*, 1 (2011), 102–108.

[55] LANG, W., SHANKAR, S., PATEL, J. M., AND KALHAN, A. Towards Multi-Tenant Performance SLOs. *IEEE Transactions on Knowledge and Data Engineering 26*, 6 (2014), 1447–1463.

[56] LI, C., PORTO, D., CLEMENT, A., GEHRKE, J., PREGUIÇA, N. M., AND RODRIGUES, R. Making Geo-Replicated Systems Fast as Possible, Consistent when Necessary. In *OSDI* (2012), vol. 12, pp. 265–278.

[57] LIM, H. C., BABU, S., AND CHASE, J. S. Automated control for elastic storage. In *Proceedings of the 7th international conference on Autonomic computing* (2010), ACM, pp. 1–10.

[58] LIN, M., WIERMAN, A., ANDREW, L. L., AND THERESKA, E. Dynamic right-sizing for power-proportional data centers. *IEEE/ACM Transactions on Networking (TON) 21*, 5 (2013), 1378–1391.

[59] LINDEN, G. Make Data Useful. http://goo.gl/DGKkzv. Slides for a talk for the course Data Mining (CS345) at Stanford University. Accessed 28-Aug-2018.

[60] LIU, H. H., VISWANATHAN, R., CALDER, M., AKELLA, A., MAHAJAN, R., PADHYE, J., AND ZHANG, M. Efficiently Delivering Online Services over Integrated Infrastructure. In *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)* (2016), pp. 77–90.

[61] LIU, Z., LIN, M., WIERMAN, A., LOW, S. H., AND ANDREW, L. L. Greening Geographical Load Balancing. In *Proceedings of the ACM SIGMETRICS joint international conference on Measurement and modeling of computer systems* (2011), ACM, pp. 233–244.

[62] LLOYD, W., FREEDMAN, M. J., KAMINSKY, M., AND ANDERSEN, D. G. Don't Settle for Eventual: Scalable Causal Consistency for Wide-Area Storage with COPS. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles* (2011), ACM, pp. 401–416.

[63] MACE, J., BODIK, P., FONSECA, R., AND MUSUVATHI, M. Retro: Targeted Resource Management in Multi-tenant Distributed Systems. In *NSDI* (2015), pp. 589–603.

[64] MADHYASTHA, H. V., ISDAL, T., PIATEK, M., DIXON, C., ANDERSON, T., KRISHNAMURTHY, A., AND VENKATARAMANI, A. iPlane: An Information Plane for Distributed Services. In *Proceedings of the 7th symposium on Operating systems design and implementation* (2006), USENIX Association, pp. 367–380.

[65] MICROSOFT. Microsoft Azure - Linux Virtual Machines Pricing. https://azure.microsoft.com/en-us/pricing/details/virtual-machines/linux/ Accessed 28-Aug-2018.

[66] MILLS, D. L. Internet time synchronization: The network time protocol. *Communications, IEEE Transactions on 39*, 10 (1991), 1482–1493.

[67] MITZENMACHER, M. How Useful Is Old Information? *IEEE Transactions on Parallel and Distributed Systems 11*, 1 (2000), 6–20.

[68] NETHERCOTE, N., STUCKEY, P. J., BECKET, R., BRAND, S., DUCK, G. J., AND TACK, G. MiniZinc: Towards a standard CP modelling language. In *International Conference on Principles and Practice of Constraint Programming* (2007), Springer, pp. 529–543.

[69] NGUYEN, H., SHEN, Z., GU, X., SUBBIAH, S., AND WILKES, J. AGILE: Elastic Distributed Resource Scaling for Infrastructure-as-a-Service. In *Proceedings of the 10th International Conference on Autonomic Computing (ICAC 13)* (2013), pp. 69–82.

[70] NOVAKOVIC, D., VASIC, N., NOVAKOVIC, S., KOSTIC, D., AND BIANCHINI, R. Deepdive: Transparently Identifying and Managing Performance Interference in Virtualized Environments. In *Proceedings of the 2013 USENIX Annual Technical Conference* (2013), no. EPFL-CONF-185984.

[71] OLTEANU, V., AGACHE, A., VOINESCU, A., AND RAICIU, C. Stateless datacenter load-balancing with beamer. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI)* (2018), vol. 18, pp. 125–139.

[72] PADALA, P., HOU, K.-Y., SHIN, K. G., ZHU, X., UYSAL, M., WANG, Z., SINGHAL, S., AND MERCHANT, A. Automated Control of Multiple Virtualized Resources. In *Proceedings of the 4th ACM European conference on Computer systems* (2009), ACM, pp. 13–26.

[73] PADALA, P., SHIN, K. G., ZHU, X., UYSAL, M., WANG, Z., SINGHAL, S., MERCHANT, A., AND SALEM, K. Adaptive control of virtualized resources in utility computing environments. In *ACM SIGOPS Operating Systems Review* (2007), vol. 41, ACM, pp. 289–302.

[74] PANG, J., AKELLA, A., SHAIKH, A., KRISHNAMURTHY, B., AND SESHAN, S. On the Responsiveness of DNS-based Network Control. In *Proceedings of the 4th ACM SIGCOMM conference on Internet measurement* (2004), ACM, pp. 21–26.

[75] PATEL, P., BANSAL, D., YUAN, L., MURTHY, A., GREENBERG, A., MALTZ, D. A., KERN, R., KUMAR, H., ZIKOS, M., WU, H., ET AL. Ananta: Cloud Scale Load Balancing. In *ACM SIGCOMM Computer Communication Review* (2013), vol. 43, ACM, pp. 207–218.

[76] PUCHA, H., ZHANG, Y., MAO, Z. M., AND HU, Y. C. Understanding Network Delay Changes Caused by Routing Events. In *ACM SIGMETRICS Performance Evaluation Review* (2007), vol. 35, ACM, pp. 73–84.

[77] QURESHI, A., WEBER, R., BALAKRISHNAN, H., GUTTAG, J., AND MAGGS, B. Cutting the electric bill for internet-scale systems. In *ACM SIGCOMM computer communication review* (2009), vol. 39, ACM, pp. 123–134.

[78] RAGHAVAN, B., VISHWANATH, K., RAMABHADRAN, S., YOCUM, K., AND SNOEREN, A. C. Cloud Control with Distributed Rate Limiting. *ACM SIGCOMM Computer Communication Review 37*, 4 (2007), 337–348.

[79] RANJAN, S. Request redirection for dynamic content. In *Content Delivery Networks*. Springer, 2008, pp. 155–179.

[80] RANJAN, S., KARRER, R., AND KNIGHTLY, E. Wide Area Redirection of Dynamic Content by Internet Data Centers. In *INFOCOM 2004. Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies* (2004), vol. 2, IEEE, pp. 816–826.

[81] RAO, L., LIU, X., XIE, L., AND LIU, W. Minimizing Electricity Cost: Optimization of Distributed Internet Data Centers in a Multi-Electricity-Market Environment. In *INFOCOM, 2010 Proceedings IEEE* (2010), IEEE, pp. 1–9.

[82] REDA, W., AND BOGDANOV, K. L. Open Loop YCSB source code. https://github.com/kirillsc/ycsb/tree/openloop.

[83] ROUSSOPOULOS, M., AND BAKER, M. Practical load balancing for content requests in peer-to-peer networks. *Distributed Computing 18*, 6 (2006), 421–434.

[84] SCHULTE, C., TACK, G., AND LAGERKVIST, M. Z. Modeling and programming with gecode, 2010. http://www.gecode.org/doc-latest/MPG.pdf Accessed 17-Jan-2017.

[85] SHANKARANARAYANAN, P., SIVAKUMAR, A., RAO, S., AND TAWARMALANI, M. Performance Sensitive Replication in Geo-Distributed Cloud Datastores. In *2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks* (2014), IEEE, pp. 240–251.

[86] SHEN, Z., SUBBIAH, S., GU, X., AND WILKES, J. CloudScale: Elastic Resource Scaling for Multi-Tenant Cloud Systems. In *Proceedings of the 2nd ACM Symposium on Cloud Computing* (2011), ACM, p. 5.

[87] SOUDERS, S. Velocity and the Bottom Line. http://radar.oreilly.com/2009/07/velocity-making-your-site-fast.html Accessed 1-May-2018.

[88] STANOJEVI, R., AND SHORTEN, R. Fully Decentralized Emulation of Best-Effort and Processor Sharing Queues. *ACM SIGMETRICS Performance Evaluation Review 36*, 1 (2008), 383–394.

[89] SURESH, P. L., CANINI, M., SCHMID, S., AND FELDMANN, A. C3: Cutting Tail Latency in Cloud Data Stores via Adaptive Replica Selection. In *NSDI* (2015), pp. 513–527.

[90] SVERDLIK, Y. Google to build and lease data centers in big cloud expansion. 22-apr-2016. Data Center Knowledge. http://goo.gl/hkxmXQ Accessed 1-May-2018.

[91] TENE, G., IYENGAR, B., AND WOLF, M. C4: The Continuously Concurrent Compacting Collector. *ACM SIGPLAN Notices 46*, 11 (2011), 79–88.

[92] TERRY, D. B., PRABHAKARAN, V., KOTLA, R., BALAKRISHNAN, M., AGUILERA, M. K., AND ABU-LIBDEH, H. Consistency-Based Service Level Agreements for Cloud Storage. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles* (2013), ACM, pp. 309–324.

[93] THORSTEN VON EICKEN. Animoto's Facebook scale-up, 23-Apr-2008. Right Scale. http://goo.gl/UDNXS9 Accessed 28-Aug-2018.

[94] VENKATARAMANI, V., AMSDEN, Z., BRONSON, N., CABRERA III, G., CHAKKA, P., DIMOV, P., DING, H., FERRIS, J., GIARDULLO, A., HOON, J., ET AL. TAO: How Facebook Serves the Social Graph. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data* (2012), ACM, pp. 791–792.

[95] VULIMIRI, A., GODFREY, P. B., MITTAL, R., SHERRY, J., RATNASAMY, S., AND SHENKER, S. Low Latency via Redundancy. In *Proceedings of the ninth ACM conference on Emerging networking experiments and technologies* (2013), ACM, pp. 283–294.

[96] WENDELL, P., JIANG, J. W., FREEDMAN, M. J., AND REXFORD, J. Donar: Decentralized Server Selection for Cloud Services. *ACM SIGCOMM Computer Communication Review 41*, 4 (2011), 231–242.

[97] WU, Z., BUTKIEWICZ, M., PERKINS, D., KATZ-BASSETT, E., AND MADHYASTHA, H. V. SPANStore: Cost-Effective Geo-Replicated Storage Spanning Multiple Cloud Services. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles* (2013), ACM, pp. 292–308.

[98] WU, Z., YU, C., AND MADHYASTHA, H. V. CosTLO: Cost-Effective Redundancy for Lower Latency Variance on Cloud Storage Services. In *Proc. 12th USENIX Symposium on Networked Systems Design and Implementation (NSDI)* (2015).

[99] XU, W., ZHU, X., SINGHAL, S., AND WANG, Z. Predictive Control for Dynamic Resource Allocation in Enterprise Data Centers. In *Network Operations and Management Symposium, 2006. NOMS 2006. 10th IEEE/IFIP* (2006), IEEE, pp. 115–126.

[100] ZHANG, Q., CHERKASOVA, L., AND SMIRNI, E. A Regression-Based Analytic Model for Dynamic Resource Provisioning of Multi-Tier Applications. In *Autonomic Computing, 2007. ICAC'07. Fourth International Conference on* (2007), IEEE, pp. 27–27.

[101] ZHOU, Z., LIU, F., XU, Y., ZOU, R., XU, H., LUI, J. C., AND JIN, H. Carbon-Aware Load Balancing for Geo-Distributed Cloud Services. In *2013 IEEE 21st International Symposium on Modelling, Analysis and Simulation of Computer and Telecommunication Systems* (2013), IEEE, pp. 232–241.

[102] ZHU, T., KOZUCH, M. A., AND HARCHOL-BALTER, M. WorkloadCompactor: Reducing Datacenter Cost While Providing Tail Latency SLO Guarantees. In *Proceedings of the 2017 Symposium on Cloud Computing* (2017), ACM, pp. 598–610.

[103] ZHU, X., YOUNG, D., WATSON, B. J., WANG, Z., ROLIA, J., SINGHAL, S., MCKEE, B., HYSER, C., GMACH, D., GARDNER, R., ET AL. 1000 islands: an integrated approach to resource management for virtualized data centers. *Cluster Computing 12*, 1 (2009), 45–57.