HALO: Heterogeneity-Aware Load Balancing

Anshul Gandhi, Xi Zhang, Naman Mittal
Stony Brook University, Stony Brook, NY, 11790
{anshul.gandhi,xi.zhang.1,naman.mittal}@stonybrook.edu

Abstract—**Load Balancers (LBs) play a critical role in managing the performance and resource utilization of distributed systems. However, developing efficient LBs for large, distributed clusters is challenging for several reasons: (i) large clusters require numerous scheduling decisions per second, (ii) such clusters typically consist of heterogeneous servers that widely differ in their computing power, and (iii) such clusters often experience significant changes in load. In this paper we propose HALO, a class of scalable, heterogeneity-aware LBs for cluster systems. HALO LBs are based on simple randomized algorithms that are analytically optimized for heterogeneity. We develop HALO for randomized, Round-Robin, and Power-of-D LBs. We illustrate the benefits of HALO and demonstrate its superiority over other comparable LBs using analytical, simulation, and (Apache-based) implementation results. Our results show that HALO LBs provide significantly lower response times without incurring additional overhead across a wide range of scenarios.**

I. Introduction

Large-scale cluster deployments are common in today's cloud-hosted application environments. Online service providers such as Amazon, Facebook and Google often employ clusters of thousands of nodes for serving web requests [1]–[3]. These online services often handle thousands of customer requests per second [1], [3]–[8]. Scheduling such large amounts of traffic on large-scale clusters requires Load Balancers (LBs) that are capable of making millions of scheduling decisions per second. Taking into account the fact that typical response times for online services such as Amazon and Facebook are on the order of tens of milliseconds [1], [6], a scheduling delay of even a few *milliseconds* can lead to a significant loss of revenue due to customer abandonment. Thus, LBs must quickly route requests to servers and should be inherently scalable.

Given the scalability limitations of complex, centralized LBs, prior work has strongly emphasized simple, decentralized LBs [9], [10]. Popular scalable LBs in use in today's production systems are typically based on Randomized (RND) or Round-Robin (RR) LBs. **RND** LBs split incoming requests among available back-end servers based on some probabilities [11]. A more powerful version of RND LBs are Power-Of-D choices (**POD**) LBs whereby a handful of servers are probed, and the one with the least load is selected for request forwarding [12], [13]. **RR** LBs are not randomized; they successively cycle incoming requests among all available back-end servers [14]. The aforementioned LBs can easily scale to internet-sized systems, and are thus often the default choice in LB products such as those offered by Apache [15] and those employed by AWS EC2 groups [16].

Unfortunately, the simple design of scalable LBs is what leads to their deficiencies. In particular, simple LBs are *not* heterogeneity-aware and are *not* adaptive to changes in load because of non-trivial factors, such as request rate and server speeds, and their non-linear effect on response times. However, today's cluster systems are inherently heterogeneous [17], [18] due to many factors including equipment and capacity upgrades and replacement of failed components. Cloud service providers, such as Amazon, also offer a diverse range of VM sizes to suit various user requirements; currently, AWS offers 28 different types of EC2 instances, grouped into 5 separate classes [19]. Likewise, online services often experience significant variations in demand [20]–[22]. As a result, it is not surprising that (load-balanced) clusters often experience low utilization [2], leading to energy and resource wastage. Thus, we claim that there is a *need for scalable and adaptive LBs that are suitable for heterogeneous environments*.

In this paper we propose **HALO**, a class of novel heterogeneity-aware load balancing algorithms. HALO can be employed in clusters where servers have diverse configurations including quantitative differences (for example, number of CPU cores) and qualitative differences (for example, operating frequencies). HALO algorithms are simple, hence scalable, yet heterogeneity-aware and adaptive. In developing HALO, we start with simple randomized LB algorithms; we then tune the request split, via queueing-theoretic analysis, for optimal performance in heterogeneous environments. Our findings are non-trivial, and suggest that the natural solution of proportional request splitting based on a server's computing power can be far from optimal. Our implementation, simulation, and analytical results show that HALO significantly reduces response time without incurring additional resource or energy overhead across various load and cluster configurations.

The contributions of this paper are the design, analysis, and implementation of HALO, a novel class of scalable, heterogeneity-aware LBs. In particular:

- We theoretically prove that, under certain restrictions, HALO provides optimal performance without consuming additional resources.
- We develop HALO algorithms for RND, RR, and POD LBs.
- Via numerical analysis and simulations, we thoroughly evaluate HALO LBs and show that they provide significantly lower response times when compared to other scalable LBs.
- We implement our simple algorithms in the Apache LB [23] and validate and demonstrate the superiority of HALO in physical and virtual clusters.

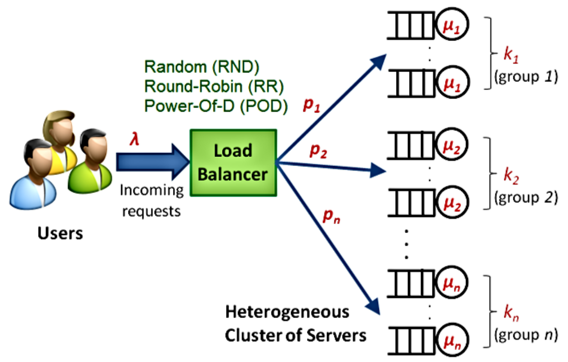


Fig. 1. Illustration of our load-balanced heterogeneous cluster environment.

The rest of the paper is organized as follows. We initially focus on RND LB and describe our cluster model and problem statement in Section II. We then present our analytical results in Section III that help optimize our HALO algorithms. Our analytical results also provide qualitative estimates for system performance under HALO. Based on our analysis, we develop HALO algorithms for RND, RR, and POD LBs in Section IV and present numerical and simulation results that demonstrate the superiority of HALO. We present our implementation results on small physical and virtual clusters in Section V. We discuss prior work in Section VI and conclude the paper in Section VII.

II. Model and Problem Statement

Fig. 1 illustrates the heterogeneous cluster environment that we consider. We model the cluster using a queueing theoretic abstraction. We assume that there are different "types" of servers. Each server type represents a distinct resource configuration. We logically partition the cluster into n groups of homogeneous servers as shown in the figure. We only consider the single dominant resource at a server, such as CPU, so the number of units or speed or bandwidth of that resource determines its type. Let the number of type i servers be k_i. We initially assume that all servers are single-core; we relax this assumption in Section III-B. Let the speed of a type i server be μ_i req/s (assuming a request takes $1/\mu_i$ seconds to complete, on average, on a type i server). Note that this definition of server speed takes request size into account. Thus, the total number of servers is $\sum_{i=1}^{n} k_i$ and the maximum system throughput is $\sum_{i=1}^{n} k_i \cdot \mu_i$ req/s. Let the incoming request rate into the entire system be λ req/s. We define load as $\rho = \lambda / \left(\sum_{i=1}^{n} k_i \cdot \mu_i\right)$. Load is a normalized measure of the request rate, and is representative of system utilization. Note that $0 \leq \rho \leq 1$. We typically report performance as a function of load in our results in this paper.

Our goal is to determine how best to split λ among the available servers to minimize average response time, T. We start with RND routing and then move to RR and POD in Section IV. Under RND LB, we split incoming requests among the available server groups. Let p_i denote the probability of sending a request to server group i. Within each group, we load balance requests among the servers. Thus, each server in group i receives p_i/k_i fraction of incoming requests. The RND LB's objective is to determine the **optimal load split** vector, p_i^*, to minimize T. Note that $\sum_{i=1}^{n} p_i = 1$. Intuitively, the load should be split proportionally, that is, $p_i = k_i \cdot \mu_i / \sum_{j=1}^{n} k_j \cdot \mu_j$. Surprisingly, as we show in the next section, the optimal load split can be far from proportional.

III. Theoretical Analysis

In this section we present our theoretical results that help optimize the load split for RND LBs in heterogeneous environments. These results will guide the development of our HALO algorithms presented in Section IV.

For RND LBs, our objective is to find the optimal load split among the servers, possibly as a function of the cluster configuration and the load (or request rate). We use queueing theory to find the optimal load split, p_i^*, under certain restrictions. We later show in Sections IV-A and V that the p_i^* vector results in near-optimal performance even when we relax the restrictions and consider more realistic settings.

For this section, we assume that the arrival process is Poisson, with mean λ req/s. There is no restriction on the distribution of request sizes. We assume that there is only one class of CPU-bound requests, and each request can be served by any of the available servers. Servers can process multiple requests simultaneously (processor sharing). We first consider, in Section III-A, clusters where heterogeneity is due to differences in server speed. We then consider, in Section III-B, clusters where heterogeneity is due to differences in the number of cores.

A. Heterogeneity due to server speeds

We will first express mean response time, T, as a function of the probabilistic load split vector, p_i. Based on the model described in Section II and Fig. 1, we can compute T using a network of $M/G/1/PS$ queues [24] as follows:

$$T = \sum_{i=1}^{n} p_i T_i = \sum_{i=1}^{n} \frac{p_i}{\mu_i - \lambda p_i / k_i} \qquad (1)$$

where T_i is the mean response time of a request in group i. Eq. (1) says that T is a weighted sum of the mean response times for each server type, $T_i = 1/(\mu_i - \lambda p_i / k_i)$. Here, $\lambda p_i / k_i$ is the request rate at each type i server (since we load balance the λp_i req/s among the k_i servers). Note that T is a function of the load split, p_i, in addition to other parameters.

The optimal probabilities, p_i^*, and the optimal response time, T^*, can be determined by optimizing Eq. (1), given λ and server speeds μ_i. We derive p_i^* in two steps: we first derive p_i^* for the case of $k_i = 1$ in Lemma 1, and then derive p_i^* for the general case of arbitrary k_i in Lemma 2. Due to lack of space, we only provide proof sketches (deferred to Appendix).

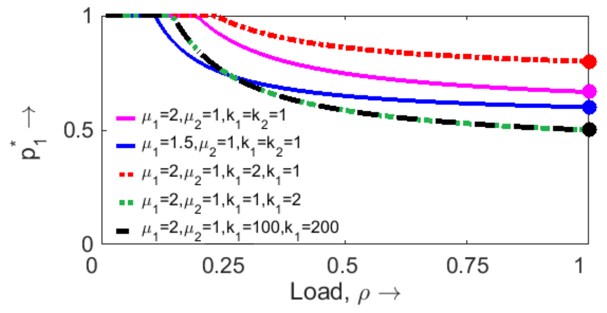


Fig. 2. Analytical results showing optimal load split, p_1^*, as a function of load for various cluster configurations with $n = 2$. The solid dots at the right of the figure show the value of p_1 under proportional load split.

Lemma 1. *For a heterogeneous cluster of n $M/G/1/PS$ servers with speeds μ_i ($i = 1, 2, \ldots n$) and total request rate λ, under RND LB:*

$$p_i^* = \frac{\mu_i \sum_{j=1}^{n} \sqrt{\mu_j} - \sqrt{\mu_i} \sum_{j=1}^{n} \mu_j + \lambda\sqrt{\mu_i}}{\lambda \sum_{j=1}^{n} \sqrt{\mu_j}} \tag{2}$$

$$T^* = \frac{2 \sum_{i=1}^{n} \sum_{j=i+1}^{n} \sqrt{\mu_i \mu_j} - (n-1) \sum_{i=1}^{n} \mu_i + n\lambda}{\lambda(\sum_{i=1}^{n} \mu_i - \lambda)} \tag{3}$$

Interestingly, p_i^* is not the same as (the intuitive) proportional split given by $p_i = k_i\mu_i/(\sum_{j=1}^{n} k_j\mu_j)$. In fact, p_i^* can be far from the proportional split. For the simple case of $n = 2$, and $k_1 = k_2 = 1$, we can show that optimal split sends more requests (when compared with proportional split) to the faster server (say, μ_1) when $\lambda < \mu_1 - \mu_2$.

To better understand Eq. (2), consider the simple case where $n = 2$. In this case, we have:

$$p_1^* = \frac{\mu_1\sqrt{\mu_2} - \mu_2\sqrt{\mu_1} + \lambda\sqrt{\mu_1}}{\lambda(\sqrt{\mu_1} + \sqrt{\mu_2})}, \quad p_2^* = \frac{\mu_2\sqrt{\mu_1} - \mu_1\sqrt{\mu_2} + \lambda\sqrt{\mu_2}}{\lambda(\sqrt{\mu_1} + \sqrt{\mu_2})}$$

When $\mu_1 > \mu_2$, we have $p_1^* > p_2^*$, as expected. Further, when $\lambda < \mu_1 - \sqrt{\mu_1\mu_2}$, we have $p_1^* > 1$, implying that all requests should be sent to server 1 when the request rate, λ, is low. While this sounds counter-intuitive, the key idea here is that when request rate is (very) low, it is best to send all requests to the faster server since there is no contention in the system. However, as request rate increases, we need both servers to handle the load. This also shows that the optimal load split depends on request rate (or load).

The solid lines (magenta and blue) in Fig. 2 illustrate our analytical results for $k_i = 1$ and $n = 2$. We only show p_1^* ($p_2^* = 1 - p_1^*$). We see that $p_1^* = 1$ when load is low, as expected. As load increases, p_1^* decreases, and approaches the proportional load split ($p_1 = \mu_1/(\mu_1 + \mu_2)$) as $\rho \rightarrow 1$. For the case of $\mu_1 = 2$, $\mu_2 = 1$ (magenta line), the improvement in response time afforded by the optimal load split over the proportional load split varies from 25% to 3% as load varies from 0.01 to 0.99. For the case of $\mu_1 = 1.5$, $\mu_2 = 1$ (blue line), there is lesser heterogeneity in the system, and the improvement in response time varies from 16% to 1%.

The above result for $k_i = 1$ can be easily extended to the case of arbitrary k_i, as given by Lemma 2 below. The proof proceeds along similar lines as that of Lemma 1 (in the Appendix) and is omitted due to lack of space.

Lemma 2. *For a heterogeneous cluster of n groups of k_i ($i = 1, 2, \ldots n$) $M/G/1/PS$ servers with speeds μ_i ($i = 1, 2, \ldots n$) and total request rate λ, under RND LB:*

$$p_i^* = \frac{k_i\mu_i \sum_{j=1}^{n} (k_j\sqrt{\mu_j}) - k_i\sqrt{\mu_i} \sum_{j=1}^{n} (k_j\mu_j) + k_i\lambda\sqrt{\mu_i}}{\lambda \sum_{j=1}^{n} (k_j\sqrt{\mu_j})} \tag{4}$$

$$T^* = \frac{2 \sum_{i=1}^{n} \sum_{j=i+1}^{n} (k_ik_j\sqrt{\mu_i\mu_j}) - \sum_{i=1}^{n} (k_i\mu_i \sum_{j=1, j\neq i}^{n} k_j) + \lambda \sum_{i=1}^{n} k_i}{\lambda \left(\sum_{i=1}^{n} (k_i\mu_i) - \lambda \right)} \tag{5}$$

The above lemmas tell us how to optimally split incoming traffic among available back-end servers for heterogeneous (w.r.t. server speed) clusters. Of course, we restrict p_i^* to be between 0 and 1. Note the dependence of p_i^* on request rate, λ. This shows that the optimal split depends on system load.

An important result that can be immediately derived from the above lemmas is the *insensitivity* of the response time and optimal load split to scaling of the system size (scaling here refers to $k_i \rightarrow ck_i$ for some constant integer $c > 1$). In particular, recalling from Section II that load, $\rho = \lambda / \left(\sum_{i=1}^{n} k_i \cdot \mu_i\right)$, we have the following useful corollaries for scaled systems:

Corollary 1. *The optimal mean response time, T^*, for a heterogeneous cluster of $M/G/1/PS$ servers operating at load ρ does not change under scaling (of k_i).*

Corollary 2. *The optimal load split, p_i^*, for a heterogeneous cluster of $M/G/1/PS$ servers operating at load ρ does not change under scaling (of k_i).*

The dashed lines in Fig. 2 illustrate our analytical results for arbitrary k_i and $n = 2$ with $\mu_1 = 2$ and $\mu_1 = 1$. Again, we see that $p_1^* = 1$ when load is low. As load increases, p_1^* decreases, and approaches the proportional load split ($p_1 = k_1\mu_1/(k_1\mu_1 + k_2\mu_2)$) as $\rho \rightarrow 1$. In fact, it can be easily shown by substituting $\rho = \lambda / \left(\sum_{i=1}^{n} k_i \cdot \mu_i\right) = 1$ in Eq. (4) that:

Corollary 3. *For a heterogeneous cluster of $M/G/1/PS$ servers operating under RND LB, the optimal load split approaches the proportional as load (or request rate) increases. Mathematically, $p_i^* \rightarrow k_i\mu_i / \left(\sum_{j=1}^{n} k_j \cdot \mu_j\right)$ as $\rho \rightarrow 1$.*

The improvement in response time afforded by the optimal split over the proportional split varies from 33% (at low load) to 3% (at high load) when the number of slower

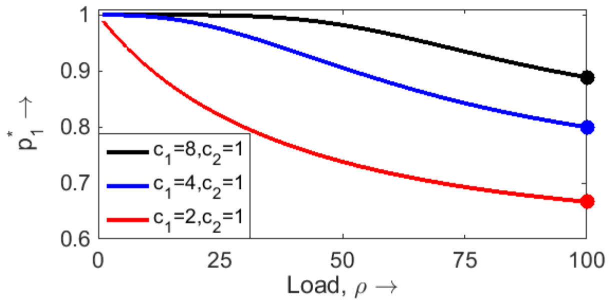


Fig. 3. Numerical results showing optimal load split, p_1^*, as a function of load for various multi-core cluster configurations with $n = 2$ and $k_i = 1$. The solid dots at the right show the value of p_1 under proportional load split.

servers is higher (green line) and from 16% to 2% when the number of faster servers is higher (red line). This reduction in improvement can be analytically proven based on Cor. 3. We omit the proof and present the final result:

Corollary 4. *For a heterogeneous cluster of $M/G/1/PS$ servers operating under RND LB, the improvement in mean response time afforded by the optimal load split over the proportional load split decreases with load (or request rate).*

Finally, observe the black dashed line in Fig. 2 that represents the case of k_i scaling with the cluster having 100× the number of servers as for the case of the green dashed line. The fact that the p_1^* value is the same for both these cases illustrates Cor. 2.

B. Heterogeneity due to server cores

We now consider the case where the server speeds (μ_i) are identical, but the number of cores in each server group may vary. This is typically the case when renting VMs from cloud service providers such as AWS [19]. In this scenario, we model each server as an $M/M/c$, where c is the number of cores. Thus, we represent the heterogeneous cluster as n groups of k_i $M/M/c_i$ systems. Assuming a request rate of λ and core speed of μ (load, $\rho = \lambda/(\mu \sum_{i=1}^{n} k_i c_i)$), we can express the mean response time of the cluster as:

$$T = \sum_{i=1}^{n} p_i \cdot \tau\left(\frac{\lambda p_i}{k_i}, \mu, c_i\right), \tag{6}$$

where $\tau(l,\mu,c) = \frac{1}{\mu} + \frac{l^k}{\mu^k(k\mu-l)k!} \cdot \left[\frac{k\mu-l}{k\mu}\sum_{i=0}^{k-1}\frac{l^i}{\mu^i i!} + \frac{l^k}{\mu^k k!}\right]^{-1}$ is the mean response time of an $M/M/c$ system with request rate l and service rate μ [24]. Unfortunately, the complex expression for τ makes it difficult to obtain closed-form expressions for p_i^* that minimize T in Eq. (6). We instead resort to numerical results to illustrate the optimal load split for this case.

Fig. 3 illustrates our numerical results for core heterogeneity with $n = 2$ and $k_i = 1$. We set $c_2 = 1$ and vary c_1. We see that initially $p_1^* = 1$ when load is very low. As load increases, p_1^* decreases, and approaches the proportional load split ($p_1 = c_1/(c_1 + c_2)$) as $\rho \rightarrow 1$. In this case, p_1^* is always higher

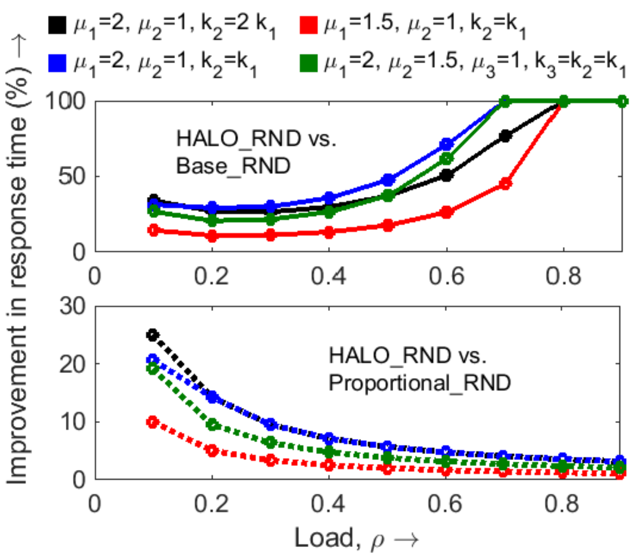


Fig. 4. Analytical results comparing HALO and Base (top) and HALO and Proportional (bottom) for randomized LB. In both cases, HALO provides significantly lower response times.

than the proportional load split because a multi-core server can handle bursts of load better than a single-core server with proportionally lower load. In fact, the improvement in response time afforded by the optimal load split over the proportional load split *increases* with load, unlike the case of heterogeneity due to server speeds (Cor. 4). The maximum improvement over the range of load considered in Fig. 3 is 18%, 10%, and 4% for $c_1 = 8$ (black line), $c_1 = 4$ (blue line), and $c_1 = 2$ (red line), respectively.

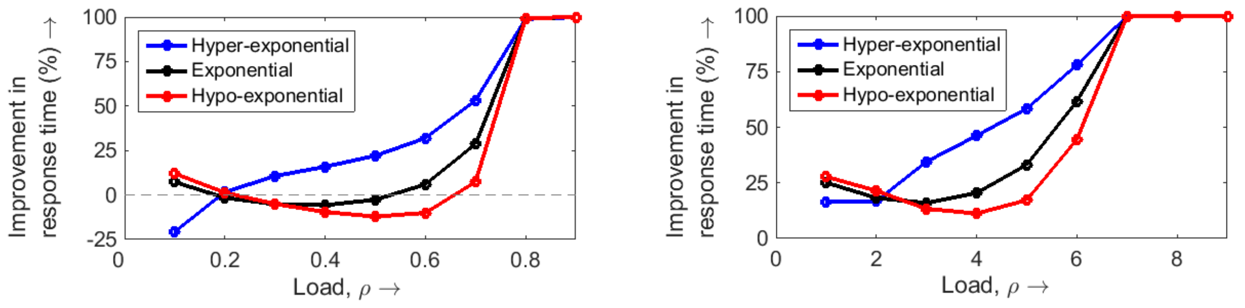
IV. HALO LBs

We now present our HALO LB algorithms. We develop HALO algorithms for RND (Section IV-A), RR (Section IV-B), and POD (Section IV-C) LBs. In each case, we also evaluate HALO and compare it with other algorithms.

A. HALO_RND

HALO_RND is directly based on the optimal load split derived in Lemma 2. In particular, **HALO_RND** routes each incoming request to a server in group i with probability p_i^* given by Eq. (4). This is in contrast to **Base_RND** that routes requests to servers with equal probability ($p_i = k_i/\sum_{j=1}^{n} k_j$) and **Proportional_RND** that (intuitively) routes requests to servers based on their speed ($p_i = k_i\mu_i/\sum_{j=1}^{n} k_j\mu_j$). Note that the algorithms only differ in their routing probabilities, and are thus comparable in terms of overhead.

Fig. 4 shows our analytical results comparing HALO_RND with Base_RND (top) and Proportional_RND (bottom). In order to show multiple cases, we plot the percentage improvement in mean response time afforded by HALO over others as a function of load, ρ (defined in Section II). We use load as a proxy for request rate in the graphs. Note that, by definition, load is not affected by the LB policy since it depends on the



(a) $\mu_1 = 1.5$, $\mu_2 = 1$.

(b) $\mu_1 = 2$, $\mu_2 = 1$.

Fig. 5. Simulation results comparing the improvement in mean response time afforded by HALO_RR over Base_RR for various request size distributions. The hyper-exponential distribution is more variable that the exponential, and the hypo-exponential is less variable. Here, $n = 2$ and $k_2 = k_1 = 1$.

workload and cluster configuration. Also note that we only report the relative values of k_i due to the insensitivity of response time to scaling under RND LB (see Cor. 1).

We see that HALO_RND provides great improvements (almost *as much as 100%*) in response time when compared to Base_RND, especially at high loads when the slower servers are overwhelmed by traffic under Base_RND. HALO_RND also provides significant improvements (*as much as 26%*) when compared to Proportional_RND, especially at low loads when some of the requests under Proportional_RND are routed to slower servers. This is in agreement with Cor. 4. Observe that, in both cases (top and bottom plots), the improvement increases with heterogeneity (difference between speeds of servers). Clearly, Base and Proportional are not as good as expected. We further validate this result via implementation in Section V-A.

B. HALO_RR

Round-Robin (RR) is easier to implement than RND routing (no random number generation is required), and provides more fine-grained load balancing due to its non-random routing. RR is the default LB algorithm for many systems, including Amazon's Elastic Load Balancer [16]. Unfortunately, RR does not take heterogeneity into account. This can result in high response times. In particular, **Base_RR** simply cycles successive requests among all available servers regardless of server speeds. We specifically take heterogeneity into account for our **HALO_RR** which first routes requests to server groups based on Eq. (4), and then uses RR to cycle successive requests within each group.

Fig. 5 shows our results comparing HALO_RR with Base_RR for the simple case of $n = 2$ and $k_2 = k_1 = 1$. Due to the complexity involved in analyzing RR routing, we employ simulations to evaluate our results. We again see that HALO_RR provides great improvements (almost *as much as 100%*) in response time when compared to Base_RR, especially at high loads. Base_RR treats all servers as equal when assigning requests to servers. Thus, the faster servers are under-utilized, whereas the slower servers are overloaded, and this imbalance is more pronounced at higher loads. By contrast, HALO_RR sends more requests to faster servers, thus providing significantly lower response times.

Interestingly, HALO_RR provides qualitatively similar improvements even for the case of non-exponential distributions, as shown in Fig. 5 (blue and red lines). This result, along with the fact that HALO_RND's optimality does not depend on the request size distribution (since $M/G/1/PS$), suggests that *HALO is robust*, to some extent, to the request size distribution.

Observe, from Figs. 5(a) and 5(b), that the improvement in response time increases with heterogeneity for all distributions. In fact, when the heterogeneity is weak (Fig. 5(a)), the improvement is *negative* for the case of moderate load when compared to Base_RR. This is because Base_RR deterministically maximizes the time between requests (inter-arrival time) at a server as opposed to HALO_RR which (only) probabilistically maximizes this time. Further, at low loads, HALO_RR benefits from sending requests almost exclusively to faster servers. As load increases, HALO_RR must use slower servers as well to handle the increased load, similar to Base_RR. Thus, as load increases, the improvement over Base_RR decreases. At very high loads, HALO_RR again benefits from preferentially sending *more* traffic (compared to Base_RR) to faster servers.

Fig. 6 shows our simulation results comparing HALO_RR with Base_RR (top) and Proportional_RR (bottom) across various cluster configurations. Here, **Proportional_RR** is the LB that first proportionately routes requests to server groups based on their speed and then uses RR to cycle successive requests within each group. We see that HALO_RR provides significant improvements (*as much as 25%*) when compared to Proportional_RR, especially at low loads when a greater fraction (compared to HALO_RR) of the requests under Proportional_RR are routed to slower servers. This observation is in agreement with Cor. 4 despite the fact that Cor. 4 deals with RND routing.

Note that the improvement in response time increases with heterogeneity, and is negative for the case of weak heterogeneity (black line) and moderate load as discussed above. Finally, note the difference between the scaled systems (black and red lines) in Fig. 6. This indicates that RR LB *is* sensitive to scaling as opposed to RND LB (see Cors. 1 and 2).

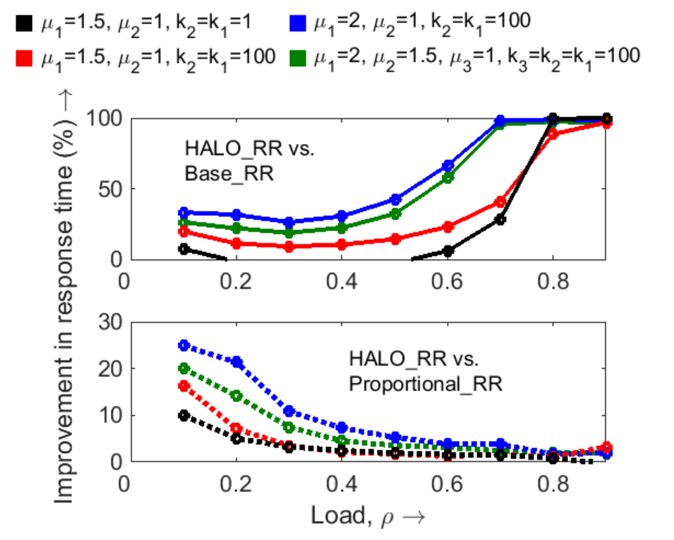


Fig. 6. Simulation results comparing HALO and Base (top) and HALO and Proportional (bottom) for Round-Robin LB. In both cases, HALO provides significantly lower response times.

C. HALO_POD

Power-Of-D (POD) routing is known [12], [13] to improve response times, especially at high loads, over RND and RR LBs. This is because POD randomly samples the current load at a few (D) servers, and routes the incoming request to the least loaded of the sampled servers. In terms of overhead, POD is more expensive than RND or RR due to the need for sampling the load at multiple servers. The value of D (number of servers sampled) can be used as a proxy for the cost of POD.

Fig. 7 shows the improvement in mean response time afforded by POD over RR as a function of D for various values of load for a homogeneous cluster ($n = 1$) of $k_1 = 10$ servers. Note that, for homogeneous clusters, HALO = Base = Proportional. As expected, the improvement in response time increases with D. Also, as expected, response time improvement is greater for higher loads. Note that D= 1 is the same as RND, and thus results in higher response time than RR. D= $k_1 = 10$ is popularly known as Join-the-Shortest-Queue (**JSQ**) routing [24], and provides the lowest response time under POD, but at a high cost (of D= k_1 units). Typically,

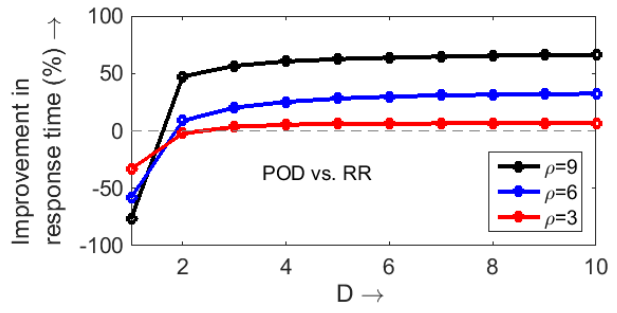


Fig. 7. Simulation results illustrating the improvement in response time afforded by POD over RR as a function of D for various values of load. Here, $n = 1$, $k_1 = 10$, and $\mu = 1$.

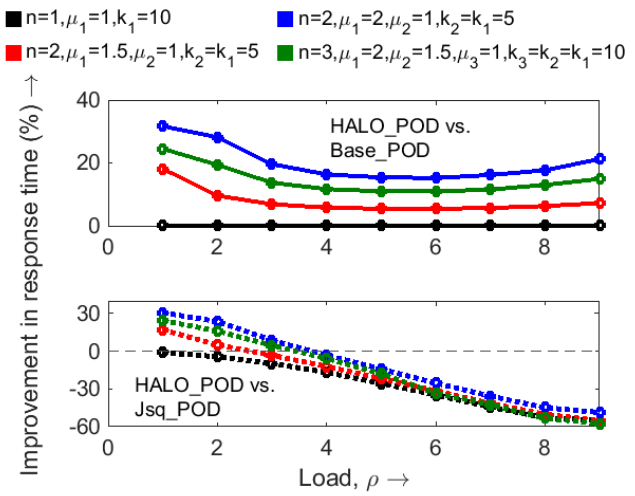


Fig. 8. Simulation results comparing HALO and Base (top) and HALO and Jsq (bottom) for Power-Of-D LB. HALO provides significantly lower response times when compared to Base. When compared to the costly Jsq, HALO provides significantly lower response times at low loads, but results in relatively higher response times at high loads.

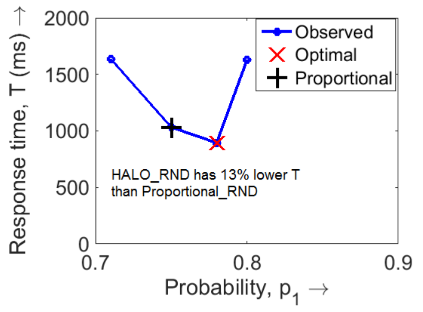
D= 2 provides the best tradeoff between response time and cost. We refer to POD with D= 2 as **Base_POD**, and POD with D= $\sum_{i=1}^{n} k_i$ as **Jsq_POD**.

We adapt Base_POD for heterogeneous environments by proposing **HALO_POD** which samples servers according to the optimal group probability given by Eq. (4). In particular, HALO_POD samples a server in group i with probability p_i^*/k_i. We also use these probabilities when breaking ties between server groups. We retain the value of D (= 2).

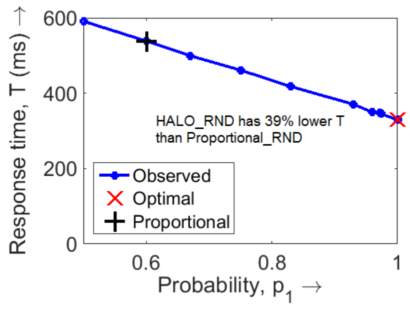
Fig. 8 shows our simulations results comparing HALO_POD with Base_POD (top) and Jsq_POD (bottom) for various cluster configurations. Here, the black line represents the scenario in Fig. 7. We see that HALO_POD provides significant improvement (*as much as 32%*) over Base_POD by simply modifying the sampling probabilities to adapt to heterogeneity. While HALO_POD does provide significant improvement (*as much as 30%*) over Jsq_POD at low loads, Jsq_POD is better at higher loads. This is because Jsq_POD with D= $\sum k_i$ is intrinsically heterogeneity-aware as it monitors the load at *all* servers before routing requests. However, Jsq_POD is expensive. For example, for the green line, Jsq_POD samples 30 servers, whereas HALO_POD only samples 2 servers (a *15×* difference in overhead). Further, Jsq_POD does not directly take server speeds into account. In fact, if we set D= $\sum k_i$ for HALO_POD, we can still obtain some improvement (as much as 10%) over Jsq_POD at high loads. Note that, as usual, the improvement in response time increases with heterogeneity.

V. Implementation Results

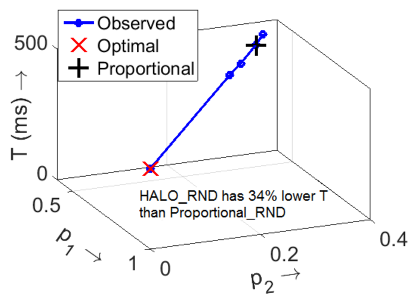
In this section we present our implementation results for HALO; we use Apache [23] as our LB. We modify Apache's `mod_proxy*` modules to enable our HALO algorithms. We first present our results on a small heterogeneous (due to server



(a) $\mu_1 = 2.4$GHz, $\mu_2 = 0.8$GHz. $\lambda = 8$ req/s.



(b) $\mu_1 = 2.4$GHz, $\mu_2 = \mu_3 = 0.8$GHz. $\lambda = 2$ req/s.



(c) $\mu_1 = 2.4$GHz, $\mu_2 = 1.6$GHz, $\mu_3 = 0.8$GHz. $\lambda = 0.5$ req/s.

Fig. 9. Implementation results for various RND LB policies. In all cases, HALO_RND provides the lowest response time, and is about 10-40% better than Proportional_RND.

speeds) physical cluster in Section V-A, and then present our results on a small heterogeneous (due to server cores) virtual cluster in Section V-B. We only present results for RND.

A. Physical cluster

Our physical testbed consists of five dual-core Intel servers with DVFS technology. We leverage DVFS to set different operating frequencies of 0.8GHz, 1.6GHz, and 2.4GHz, for the servers. We disable turbo-boost for our experiments. We employ three of these servers as the back-end worker servers running PHP. Each back-end server is connected to an inline power meter (WattsUp Pro ES) to measure its instantaneous power draw. We employ the fourth server as the LB running Apache. We employ the fifth server as our dedicated load generator running httperf [25]. We generate PHP requests from this load generator, and each request stresses the CPU at the back-end servers. At the maximum frequency of 2.4Ghz, each request takes about 300ms to complete. Thus, the server speed at 2.4GHz is $2 \cdot \frac{1}{0.3} \approx 6.7$ req/s (since dual-core). Likewise, we can compute the speeds at different operating frequencies. We experiment with various server clusters and request rates. Note that the physical cluster experiments represent the case of heterogeneity due to differences in server speeds, as analyzed in Section III-A.

Fig. 9 illustrates a subset of our experimental results. Here, μ_i is the operating frequency of server i, and λ is the total request rate into the cluster. We set $i = 1$ for the faster server type. For each graph, we experiment with various choices of request routing probabilities, p_i, and plot the average response time over multiple runs as a function of these probabilities, highlighting the HALO LB split (red cross) and Proportional LB split (black plus).

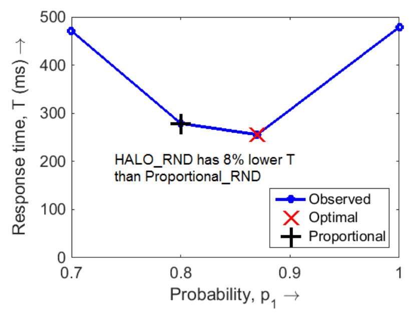
Fig. 9(a) shows our results for a 2-server experiment with $\mu_1 = 2.4$GHz and $\mu_2 = 0.8$GHz. The request rate (Poisson distributed) is $\lambda = 8$ req/s. The optimal load split probability for HALO_RND in this case is $p_1^* = 0.78$ according to Eq. (4). The Proportional_RND load split probability is $p_1 = \mu_1/(\mu_1 + \mu_2) = 0.75$. As shown in Fig. 9(a), HALO_RND results in *13% lower* response time than Proportional_RND. Note that we only show p_1 since $p_2 = 1 - p_1$. The average power consumption of the back-end (two servers), as measured by our inline power meters, for HALO_RND and Proportional_RND is similar. This is because the total work (request processing) done by the servers under both LB policies is the same; the difference is in the workload *split* between the servers.

Fig. 9(b) shows our results for a 3-server experiment with $\mu_1 = 2.4$GHz, $\mu_2 = \mu_3 = 0.8$GHz, and $\lambda = 2$ req/s. Here, the optimal load split probability for HALO_RND is $p_1^* = 1$ (send all load to the fastest server) whereas Proportional_RND probability is $p_1 = 0.6$. In this case, HALO_RND results in *39% lower* response time. Finally, Fig. 9(c) shows our results for a 3-server experiment with $\mu_1 = 2.4$GHz, $\mu_2 = 1.6$Ghz, $\mu_3 = 0.8$GHz, and $\lambda = 0.5$ req/s. We use a 3-d plot here to show the dependence of response time on p_1 and p_2 (p_3=1-p_1-p_2). In this case, the optimal load split probability for HALO_RND is $p_1^* = 1$ and $p_2^* = p_3^* = 0$, whereas the Proportional_RND load split probability is $p_1 = 0.5$, $p_2 = 0.33$, and $p_3 = 0.17$. We see that HALO_RND results in *34% lower* response time than Proportional_RND. Again, in both of the above 3-server experiments, the average power consumption of the back-end (three servers) under HALO_RND and Proportional_RND is similar.

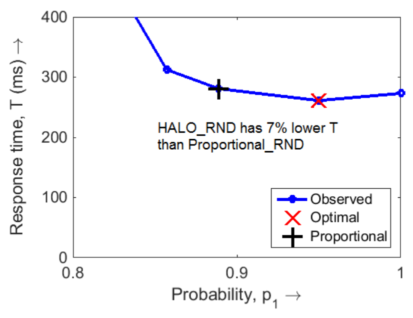
The above experimental results validate our HALO_RND LB (Eq. (4)) and highlight our superiority over Proportional_RND. These results are also in agreement with Cors. 3 and 4 which state that the optimal load split approaches the proportional load split at high request rates (Fig. 9(a)) and that the improvement afforded by the optimal load split over the proportional load split decreases with request rate, respectively (Figs. 9(b) and 9(c) have lower request rate but greater response time improvement than Fig. 9(a)).

B. Virtual cluster

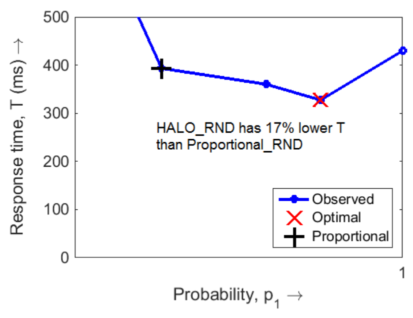
For our virtual cluster, we rent various VMs from Amazon EC2 [26] to act as our back-end servers. In particular, we experiment with m3.large (1 core), m3.xlarge (2 cores), m3.2xlarge (4 cores), c3.large (1 core), c3.4xlarge (8 cores) and c3.8xlarge (16 cores) instance types. We turn off hyper-



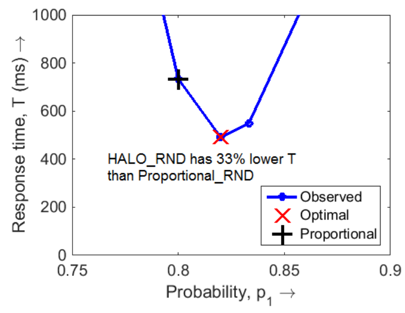
(a) m3.Large:m3.2XLarge, $\lambda = 15$ req/s.



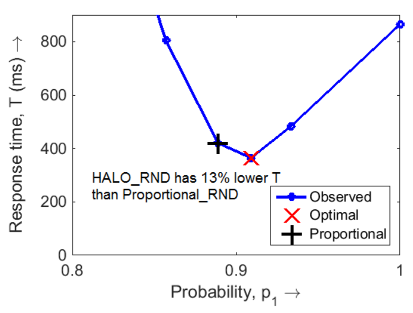
(b) c3.Large:c3.4XLarge, $\lambda = 31$ req/s.



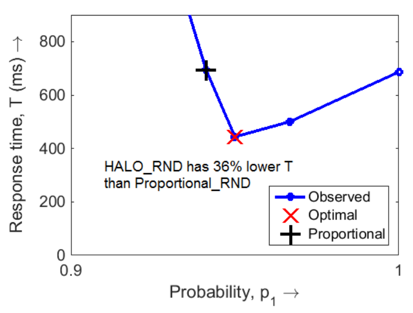
(c) c3.Large:c3.8XLarge, $\lambda = 64$ req/s.



(d) m3.Large:m3.2XLarge, $\lambda = 20$ req/s.



(e) c3.Large:c3.4XLarge, $\lambda = 40$ req/s.



(f) c3.Large:c3.8XLarge, $\lambda = 80$ req/s.

Fig. 10. Implementation results for various RND LB policies on EC2. The top row shows experiments with lower request rates, and the bottom row shows experiments with higher request rates. In all cases, our HALO_RND LB provides the lowest response time, and is about 7-36% better than Proportional_RND.

threading for our experiments. We only consider 2-server experiments. For the m3 instances, each request takes about 217ms to complete on each core, thus $\mu = \frac{1}{.217} \approx 4.6$ req/s. For the c3 instances, each request takes about 200ms to complete on each core, thus $\mu = \frac{1}{.2} = 5$ req/s. We continue using our physical servers for running the load generator and the modified Apache LB. Note that the virtual cluster experiments represent the case of heterogeneity due to differences in the number of cores, as analyzed in Section III-B.

Fig. 10 illustrates a subset of our results. We set $i = 1$ for the server with more cores. Each column represents the same configuration, but with lower request rate on top and higher request rate below. We see that higher request rate results in a greater improvement in mean response time afforded by HALO_RND over Proportional_RND, similar to our observations in Section III-B. In general, we find that the percentage improvement for the virtual cluster experiments is lower than that for the physical cluster experiments. Nevertheless, we see that HALO_RND significantly lowers response time, by *as much as 36%*, when compared to Proportional_RND LB.

VI. Prior Work

A. Scalable LBs

Prior work has strongly emphasized the need for simple, scalable LBs [9], [10], [27]. Popular examples of such LBs are those based on RND routing [11], RR routing [15], [16], and POD routing [27]–[29]. Simple RND based LBs, such as Base_RND and Proportional_RND, are scalable but do not perform well in heterogeneous environments, as we show in Figs. 4, 9, and 10. Likewise, RR LBs are simple and scalable, but cannot be easily extended to heterogeneous clusters. As shown in Figs. 5 and 6, Base_RR performs very poorly in heterogeneous environments, especially at high loads, since it treats all servers as equal. While POD LBs are typically superior to RND and RR LBs [12], [13], they are not well suited to heterogeneous environments [27], [28], and typically incur some overhead (proportional to D) associated with sampling [29].

1) RND LBs: Randomized heterogeneity-aware LBs typically route requests to servers in (static) proportion to their computing power [11]. While scalable, such approaches are not adaptive to changes in workload. Our HALO_RND LB, by contrast, is heterogeneity-aware, adaptive to load, and is easily scalable given its simple randomized nature. Our preliminary work [30] focused on heterogeneity-aware RND LBs in the context of different server speeds. This paper significantly builds on our prior work by providing theoretical analysis, addressing heterogeneity due to server cores, and developing heterogeneity-aware RR and POD LBs.

2) RR LBs: The most widely-employed LBs are those based on RR routing. There has been some prior work on improving the performance of RR in heterogeneous settings. Moonian et al. [31] modify the RR algorithm to take job deadlines into account by dynamically allocating CPU shares. Tiwari et al. [32] extend RR to heterogeneous environments by considering a server's processing power, queue length, and utilization, resulting in a Proportional_RR like scheme. Rong et al. [33] extend RR to heterogeneous environments and propose a hierarchical routing scheme for heavy traffic. As we show in Fig. 6, significant improvements over Base_RR can be achieved even at low loads. Katevenis et al. [14] propose the implementation of weighted RR for ATM switches, but assume that the weights are given. Shreedhar et al. [34] propose Deficit RR which aims to provide fair allocation of resources with low overhead. Our focus in this paper is on optimizing performance with minimal overhead; we do not focus on fairness.

3) POD LBs: Eager et al. [28] proposed a decentralized POD LB for request processing. However, the authors specifically focus only on homogeneous environments. Recently, Sparrow [27] proposed a POD scheduler for large clusters. However, as acknowledged by the authors, Sparrow does not perform well in heterogeneous environments. Dean et al. [29] also proposed sending requests to multiple servers simultaneously to reduce latency (for example, by fetching cached results). However, the proposed replication technique consumes additional resources.

B. Heterogeneity-aware LBs

Heterogeneity-aware LBs thoroughly analyze the state of all servers in the cluster to determine the best candidate server for each incoming request [17], [18], [35]. Heath et al. [17] consider a heterogeneous environment and solve for the optimal load distribution between servers using iterative algorithms such as simulated annealing. While effective, such iterative algorithms are slow to converge, requiring several minutes between successive scheduling decisions. Likewise, Chen et al. [35] propose greedy algorithms for load distribution in heterogeneous environments. While the greedy algorithms leverage dynamic programming for expediting their decision time, they still require several seconds to compute the load distribution. Lee at al. [36] propose a heterogeneity-aware scheduler that computes the impact of resources on the progress of the job, and uses this knowledge to guide placement. However, the authors focus on jobs that have large completion times, thus allowing for complex decision making. While the above LBs successfully address heterogeneity, they require complex computations, thus limiting scalability.

C. Hierarchical LBs

LBs can also be employed as part of multi-level schedulers for clusters that handle diverse applications [37], [38]. In such cases, requests are initially assigned to groups of workload-specific servers; intra-group LBs then assign requests to individual servers. Our HALO LBs can easily be employed as intra-class LBs for handling, for instance, web requests.

VII. Conclusion

In this paper we present HALO, a class of novel LBs that provide low response times without incurring any significant overhead. While most LBs today are either scalable or heterogeneity-aware, HALO is both. HALO bridges this gap between scalability and heterogeneity-awareness by building on simple scalable LBs, randomized LBs in particular, and then optimizing them for heterogeneous environments. We use theoretical analysis to optimize HALO LBs, providing many useful results about the performance of HALO LBs in the process. We easily extend our randomized HALO LB to Round-Robin and Power-Of-D choices LBs, and believe that the basic idea behind HALO can be adapted by other LBs.

As part of future work, we will examine how HALO LBs can be adapted to data analytics workloads where the heterogeneity is (also) due to proximity to data. In such environments, data-local or rack-local servers are selected with higher probability; HALO LBs can be employed in such cases to preferentially select servers that are "closer" to source data. We will also experiment with larger cluster sizes to evaluate the scalability of HALO LBs. Finally, we intentionally avoided enforcing fairness in this paper to facilitate scalability. In future work, we will explore how we can integrate fairness without significantly limiting scalability. Our end-goal in this research is to build a dynamic LB for distributed computing environments that is scalable and heterogeneity-aware. Our efforts in this paper represent the first steps in this direction.

References

[1] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels, "Dynamo: Amazon's Highly Available Key-value Store," in *Proceedings of SOSP 2007*, Stevenson, WA, USA, pp. 205–220.

[2] X. Fan, W.-D. Weber, and L. A. Barroso, "Power provisioning for a warehouse-sized computer," in *Proceedings of ISCA 2007*, San Diego, CA, USA, pp. 13–23.

[3] R. Nishtala, H. Fugal, S. Grimm, M. Kwiatkowski, H. Lee, H. C. Li, R. McElroy, M. Paleczny, D. Peek, P. Saab, D. Stafford, T. Tung, and V. Venkataramani, "Scaling Memcache at Facebook," in *Proceedings of NSDI 2013*, Lombard, IL, USA, pp. 385–398.

[4] M. Arlitt and T. Jin, "Workload Characterization of the 1998 World Cup Web Site," *IEEE Network*, vol. 14, pp. 30–37, 2000.

[5] C. Reiss, A. Tumanov, G. R. Ganger, R. H. Katz, and M. A. Kozuch, "Heterogeneity and Dynamicity of Clouds at Scale: Google Trace Analysis," in *Proceedings of SOCC 2012*, no. 7, San Jose, CA, USA.

[6] D. Meisner, C. M. Sadler, L. A. Barroso, W.-D. Weber, and T. F. Wenisch, "Power management of online data-intensive services," in *Proceedings of the 38th Annual International Symposium on Computer Architecture*, ser. ISCA '11, San Jose, CA, USA, 2011, pp. 319–330.

[7] A. Qureshi, R. Weber, H. Balakrishnan, J. Guttag, and B. Maggs, "Cutting the electric bill for internet-scale systems," in *Proceedings of the ACM SIGCOMM 2009 Conference on Data Communication*, ser. SIGCOMM '09. New York, NY, USA: ACM, 2009, pp. 123–134.

[8] G. Urdaneta, G. Pierre, and M. van Steen, "Wikipedia workload analysis for decentralized hosting," *Comput. Netw.*, vol. 53, no. 11, pp. 1830–1845, Jul. 2009.

[9] M. Schwarzkopf, A. Konwinski, M. Abd-El-Malek, and J. Wilkes, "Omega: Flexible, Scalable Schedulers for Large Compute Clusters," in *Proceedings of EuroSys 2013*, Prague, Czech Republic, pp. 351–364.

[10] K. Ousterhout, A. Panda, J. Rosen, S. Venkataraman, R. Xin, S. Ratnasamy, S. Shenker, and I. Stoica, "The Case for Tiny Tasks in Compute Clusters," in *Proceedings of HotOS 2013*, Santa Ana Pueblo, NM, USA.

[11] A. Gandhi, M. Harchol-Balter, R. Das, and C. Lefurgy, "Optimal power allocation in server farms," in *Proceedings of Sigmetrics 2009*, Seattle, WA, USA, pp. 157–168.

[12] M. Mitzenmacher, "The Power of Two Choices in Randomized Load Balancing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 12, no. 10, pp. 1094–1104, Oct. 2001.

[13] Y. Azar, A. Z. Broder, A. R. Karlin, and E. Upfal, “Balanced Allocations,” *SIAM Journal on Computing*, vol. 29, no. 1, pp. 180–200, 1999.

[14] M. Katevenis, S. Sidiropoulos, and C. Courcoubetis, “Weighted round-robin cell multiplexing in a general-purpose ATM switch chip,” *IEEE Journal on Selected Areas in Communications*, vol. 9, no. 8, pp. 1265–1279, 1991.

[15] “Simple Load Balancing with Apache,” http://www.rackspace.com/knowledge_center/article/simple-load-balancing-with-apache.

[16] “Elastic Load Balancing Concepts,” http://docs.aws.amazon.com/ElasticLoadBalancing/latest/DeveloperGuide/TerminologyandKeyConcepts.html.

[17] T. Heath, B. Diniz, E. V. Carrera, W. Meira Jr., and R. Bianchini, “Energy Conservation in Heterogeneous Server Clusters,” in *Proceedings of PPoPP 2005*, Chicago, IL, USA, pp. 186–195.

[18] R. Nathuji, C. Isci, and E. Gorbatov, “Exploiting Platform Heterogeneity for Power Efficient Data Centers,” in *Proceedings of ICAC 2007*, no. 5, Jacksonville, FL, USA.

[19] “Amazon EC2 Instances,” http://aws.amazon.com/ec2/instance-types.

[20] A. Gandhi, Y. Chen, D. Gmach, M. Arlitt, and M. Marwah, “Minimizing Data Center SLA Violations and Power Consumption via Hybrid Resource Provisioning,” in *Proceedings of the 2011 International Green Computing Conference*, Orlando, FL, USA, 2011, pp. 49–56.

[21] A. Gandhi, M. Harchol-Balter, R. Raghunathan, and M. Kozuch, “AutoScale: Dynamic, Robust Capacity Management for Multi-Tier Data Centers,” *Transactions on Computer Systems*, vol. 30, 2012.

[22] A. Gandhi, Y. Chen, D. Gmach, M. Arlitt, and M. Marwah, “Hybrid Resource Provisioning for Minimizing Data Center SLA Violations and Power Consumption,” *Sustainable Computing: Informatics and Systems*, vol. 2, pp. 91–104, 2012.

[23] “The Apache HTTP Server Project,” See http://httpd.apache.org.

[24] M. Harchol-Balter, *Performance Modeling and Design of Computer Systems: Queueing Theory in Action*. Cambridge University Press.

[25] D. Mosberger and T. Jin, “httperf—A Tool for Measuring Web Server Performance,” *ACM Sigmetrics: Performance Evaluation Review*, vol. 26, no. 3, pp. 31–37, 1998.

[26] Amazon Inc., “Amazon Elastic Compute Cloud (Amazon EC2),” http://aws.amazon.com/ec2, 2008.

[27] K. Ousterhout, P. Wendell, M. Zaharia, and I. Stoica, “Sparrow: Distributed, Low Latency Scheduling,” in *Proceedings of SOSP 2013*, Farminton, PA, USA, pp. 69–84.

[28] D. L. Eager, E. D. Lazowska, and J. Zahorjan, “Adaptive Load Sharing in Homogeneous Distributed Systems,” *IEEE Transactions on Software Engineering*, vol. 12, no. 5, pp. 662–675, 1986.

[29] J. Dean and L. A. Barroso, “The Tail at Scale,” *Communications of ACM*, vol. 56, no. 2, pp. 74–80, 2013.

[30] A. Gandhi, N. Mittal, and X. Zhang, “Optimal Load-Balancing for Heterogeneous Clusters,” in *Proceedings of the 3rd Workshop on Distributed Cloud Computing*, Portland, OR, USA, 2015.

[31] O. Moonian and G. Coulson, “SHRED: A CPU scheduler for heterogeneous applications,” *Embedded Processors for Multimedia and Communications II*, vol. 5683, pp. 132–143, 2005.

[32] A. Tiwari and P. Kanungo, “Dynamic load balancing algorithm for scalable heterogeneous web server cluster with content awareness,” in *Proceedings of TISC 2010*, Chennai, India, pp. 143–148.

[33] R. Wu and D. G. Down, “Round robin scheduling of heterogeneous parallel servers in heavy traffic,” *European Journal of Operational Research*, vol. 195, no. 2, pp. 372–380, 2009.

[34] M. Shreedhar and G. Varghese, “Efficient Fair Queueing Using Deficit Round Robin,” in *Proceedings of SIGCOMM 1995*, Cambridge, MA, USA, pp. 231–242.

[35] J.-J. Chen, K. Huang, and L. Thiele, “Power Management Schemes for Heterogeneous Clusters Under Quality of Service Requirements,” in *Proceedings of SAC 2011*, TaiChung, Taiwan, pp. 546–553.

[36] G. Lee, B.-G. Chun, and H. Katz, “Heterogeneity-aware Resource Allocation and Scheduling in the Cloud,” in *Proceedings of HotCloud 2011*, Portland, OR, USA.

[37] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. D. Joseph, R. Katz, S. Shenker, and I. Stoica, “Mesos: A Platform for Fine-grained Resource Sharing in the Data Center,” in *Proceedings of NSDI 2011*, Boston, MA, USA, pp. 295–308.

[38] V. K. Vavilapalli, A. C. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, T. Graves, J. Lowe, H. Shah, S. Seth, B. Saha, C. Curino, O. O’Malley, S. Radia, B. Reed, and E. Baldeschwieler, “Apache Hadoop YARN: Yet Another Resource Negotiator,” in *Proceedings of SOCC 2013*, Santa Clara, CA, USA.

Appendix

Proof sketch for Lemma 1. We first consider the case of $n = 2$. For $n = 2$, let the probability of sending a request to server 1 (with speed μ_1) be p. Then,

$$T = \frac{p}{\mu_1 - p\lambda} + \frac{1-p}{\mu_2 - (1-p)\lambda}.$$

We now derive the optimal value of $0 \le p \le 1$ that minimizes T. After some algebra (taking first and second derivatives of T w.r.t. p), we get

$$p^* = \frac{\mu_1\sqrt{\mu_2} - \mu_2\sqrt{\mu_1} + \lambda\sqrt{\mu_1}}{\lambda(\sqrt{\mu_1} + \sqrt{\mu_2})}.$$

Substituting this p^* into the expression for T gives us

$$T^* = \frac{2\sqrt{\mu_1\mu_2} - (\mu_1 + \mu_2) + 2\lambda}{\lambda(\mu_1 + \mu_2 - \lambda)}.$$

Note that this agrees with Eqs. (2) and (3) for $n = 2$.

We now use induction to prove Lemma 1 for all n. Assume that Lemma 1 is true for $n = N$. Then, for $n = (N+1)$, we can partition the $(N+1)$ server system into a single server with request probability p_1 and an N-server system with request probability $(1 - p_1)$. For the N-server system (with primed variables) with request rate $\lambda' = \lambda(1 - p_1)$, by the inductive hypothesis, we know that

$$p_i'^* = \frac{\mu_i \sum_{j=1}^{N} \sqrt{\mu_j} - \sqrt{\mu_i} \sum_{j=1}^{N} \mu_j + \lambda' \sqrt{\mu_i}}{\lambda' \sum_{j=1}^{N} \sqrt{\mu_j}}, \quad \text{and}$$

$$T'^* = \frac{2 \sum_{i=1}^{N} \sum_{j=i+1}^{N} \sqrt{\mu_i\mu_j} - (N-1) \sum_{i=1}^{N} \mu_i + N\lambda}{\lambda(\sum_{i=1}^{N} \mu_i - \lambda)}.$$

The mean response time for the $(N+1)$-server system can then be written as (from Eq. (1)):

$$T = \frac{p_1}{\mu_1 - \lambda p_1} + (1 - p_1) \cdot T'^* \qquad (7)$$

Note that T' is itself a function of p_1 (since request rate for the N-server system is $\lambda' = \lambda(1 - p_1)$). We can now derive the optimal p_1^* by differentiating Eq. (7). The remaining N optimal probabilities (for request rate λ) can then be derived by noting that

$$p_{i+1}^* = (1 - p_1^*) \cdot p_i'^*.$$

Finally, T^* for the $(N+1)$-server system can be derived by substituting p_1^* into Eq. (7). The resulting expressions match those given by Eqs. (2) and (3) for $n = (N+1)$, thus completing the proof by induction. $\square$