Geographically Distributed Load Balancing with (Almost) Arbitrary Load Functions

Piotr Skowron
University of Warsaw
Poland
Email: p.skowron@mimuw.edu.pl

Krzysztof Rzadca
University of Warsaw
Poland
Email: krzadca@mimuw.edu.pl

***Abstract*—In geographically-distributed systems, communication latencies are non-negligible. The perceived processing time of a request is thus composed of the time needed to route the request to the server and the true processing time. Once a request reaches a target server, the processing time depends on the total load of that server; this dependency is described by a load function. We consider a broad class of load functions; we just require that they are convex and two times differentiable. In particular our model can be applied to heterogeneous systems in which every server has a different load function. We present optimization centralized and a decentralized algorithms for load balancing. We prove bounds on the algorithms' convergence. To the best of our knowledge these bounds were not known even for the special cases studied previously (queuing theory and batches of requests). Both algorithms are any-time and self-stabilizing algorithms.**

***Keywords*-load balancing, geographic distribution**

I. Introduction

We are impatient. An "immediate" reaction must take less than 100 ms [9]; a Google user is less willing to continue searching if the result page is slowed by just 100-400 ms [8]; and a web page loading faster by just 250 ms attracts more users than the competitor's [38]. Few of us are thus willing to accept the 100-200ms Europe-US round-trip time; even fewer, 300-400ms Europe-Asia. Internet companies targeting a global audience must thus serve it locally. Google builds data centers all over the world; a company that doesn't have Google scale uses a generic content delivery network (CDN) [19], [43], such as Akamai [39], [41], [48]; or spreads its content on multiple Amazon's Web Service regions.

A geographically-distributed system is an abstract model of world-spanning networks. It is a network of interconnected servers processing requests. The system considers both communication (request routing) and computation (request handling). E.g., apart from the communication latencies, a CDN handling complex content can no longer ignore the load imposed by requests on servers. As another example consider computational clouds, which are often distributed across multiple physical locations, thus must consider the network latency in addition to servers' processing times.

Normally, each server handles only the requests issued by local users. For instance, a CDN node responds to queries incoming from the sub-network it is directly connected to (e.g., DNS redirections in Akamai [31], [39], [48]). However, load varies considerably. Typically, a service is more popular during the day than during the night (the daily usage cycle). Load also spikes during historic events, ranging from football finals to natural disasters. If a local server is overloaded, some requests might be handled faster on a remote, non-overloaded server. The users will not notice the redirection if the remote server is "close" (the communication latency is small); but if the remote server is on another continent, the round-trip time may dominate the response time.

In this paper we address the problem of balancing servers' load taking into account the communication latency. We model the response time of a single server by a *load function*, i.e., a function that for a given load on a server (the number of requests handled by a server) returns the average processing time of requests. In particular, we continue the work of Liu et al. [35] and Tantawi and Towsley [50]. Liu et al. [35] showed the convergence of the algorithms for the particular load function that describes requests' handling time in the queuing model [22]. They considered only the particular load function that describes requests' handling time in the queuing model [22]. We use a broad class of functions that are continuous, convex and twice-differentiable (Section II-A), which allows us to model not only queuing theory-based systems, but also a particular application, where the servers' response time is measured empirically in a stress-test. Tantawi and Towsley [50] (who originally proposed the model), on the other hand, showed the algorithm for the case when the communication delay between each pair of nodes is the same.

We assume that the servers are connected by links with high bandwidth. Although some models (e.g., routing games [40]) consider limited bandwidth, our aim is to model servers connected by a dense network (such as the Internet), in which there are multiple cost-comparable routing paths between the servers. The communication time is thus dominated by the latency: a request is sent over a long distance with a finite speed. We assume that the latencies are known, as monitoring pairwise latencies is a well-studied problem [11], [49]; if the latencies change due to, e.g., network problems, our optimization algorithms can be run again. On each link, the latency is constant, i.e., it does not vary with the number of sent requests [46]. This assumption

is consistent with the previous works on geographically distributed load balancing [4], [10], [23], [25], [35], [44], [46].

Individual requests are small; rather than an hour-long batch job, a request models, e.g., a single web page hit. Such assumption is often used [5], [18], [20], [25], [35], [44], [46], [50], [51]. In particular the continuous allocation of requests to servers in our model is analogous to the divisible load model with constant-cost communication (a special case of the affine cost model [5]) and multiple sources (multiple loads to be handled, [18], [51]).

The problem of load balancing in geographically distributed systems has been already addressed, however it received limited attention. Liu et al. [35] show the convergence of the algorithms for a particular load function from the queuing theory. Cardellini et al. [10] analyze only simple redirection policies, like round robin, or redirection to least loaded server. Colajanni et al. [14] presents experimental evaluation of a round-robin-based algorithm for a similar problem. Minimizing the cost of energy due to the load balancing in geographically distributed systems is a similar problem considered in the literature [34], [36], [37]. A different load function is used by Skowron and Rzadca [46] to model the flow time of batch requests.

Some papers analyze the game-theoretical aspects of load balancing in geographically distributed systems [2], [4], [23], [25], [44]. These works use a similar model, but focus on capturing the economic relation between the participating entities.

The majority of the literature on distributed load balancing ignores the communication costs [12], [16], [17], [24], [26], [27], [30], [32], [33] (some literature considers that during the communication delay the states of the servers may change, but do not consider communication delay as a cost that user perceive [17], [27]). Our distributed algorithm is the extension of the diffusive load balancing [1], [3], [6]; it incorporates communication latencies into the classical diffusive load balancing algorithms.

Additionally to the problem of effective load balancing we can optimize the choice of the locations for the servers [15], [29], [45]. The generic formulation of the placement problem, facility location problem [13] and k-median problem [28] have been extensively studied in the literature.

The contributions of this paper are the following. (i) We construct a centralized load-balancing algorithm that optimizes the response time up to a given (arbitrary small) distance to the optimal solution (Section IV). The algorithm is polynomial in the unary representation of total load of the system and in the upper bounds of the derivatives of the load function. (ii) We show a decentralized load-balancing algorithm (Section V) in which pairs of servers balance their loads. We prove that the algorithm is optimal (there is no better algorithm that uses only a single pair of servers at each step). We also bound the number of pair-wise exchanges required for convergence. (iii) We do not use a particular load function; instead, we only require the load function to be continuous and twice-differentiable (Section II-A); thus we are able to model empirical response times of a particular application on a particular machine, but also to generalize (Section II-B) Liu et al. [35]'s results on queuing model and the results on flow time of batch requests [46].

Our algorithms are suitable for real applications. Both are any-time algorithms which means that we can stop them at any time and get a complete, yet suboptimal, solution. They are self-stabilizing—if the parameters change, the algorithms do not have to be restarted. The distributed algorithm is particularly suitable for distributed systems. It performs only pairwise optimizations (only two servers need to be available to perform a single optimization phase), which means that it is highly resilient to failures. It is also very simple and does not require additional complex protocols.

II. Preliminaries

In this section we first describe the mathematical model, and next we argue that our model is highly applicable; in particular it generalizes two problems considered in the literature (see Section II-B).

A. Model

Servers, requests, relay fractions, current loads. The system consists of a set of *m servers* (processors) connected to the Internet. The i-th server has its *local (own) load* of size n_i consisting of small *requests*. The local load can be the current number of requests, the average number of requests, or the rate of incoming requests in the queuing model.

The server can relay a part of its load to the other servers. We use a fractional model in which a *relay fraction* ρ_{ij} denotes the fraction of the i-th server's load that is sent (relayed) to the j-th server ($\forall_{i,j}\, \rho_{ij} \geq 0$ and $\forall_i \sum_{j=1}^{j=m} \rho_{ij} = 1$). Consequently, ρ_{ii} is the part of the i-th load that is kept on the i-th server. We consider two models. In the *single-hop model* the request can be sent over the network only once. In the *multiple-hop* model the requests can be routed between servers multiple times[1]. Let r_{ij} denote the size of the load that is sent from the server i to the server j. In the single-hop model the requests transferred from i to j come only from the local load of the server i, thus:

$$r_{ij} = \rho_{ij} n_i. \tag{1}$$

In the multiple-hop model the requests come both from the local load of the server i and from the loads of other servers

[1]We point the analogy between the multiple-hop model and the Markov chain with the servers corresponding to states and relay fractions ρ_{ij} corresponding to the probabilities of changing states.

that relay their requests to i, thus r_{ij} is a solution of:

$$r_{ij} = \rho_{ij}\left(n_i + \sum_{k \neq i} r_{ki}\right). \quad (2)$$

The *(current) load* of the server i is the size of the load sent to i by all other servers, including i itself. Thus, $l_i = \sum_{j=1}^{m} r_{ji}$ in the single-hop model, and $l_i = r_{ii}$ in the multiple-hop model.

Load functions. Let f_i be a *load function* describing the average request's processing time on a server i as a function of i's load l_i (e.g.: if there are $l_i = 10$ requests and $f_i(10) = 7$, then on average it takes 7 time units to process each request). We assume f_i is known from a model or experimental evaluation; but each server can have a different characteristics f_i (heterogeneous servers). The total processing time of the requests on a server i is equal to $h_i(l_i) = l_i f_i(l_i)$ (e.g., it takes 70 time units to process all requests). In most of our results we use f_i instead of h_i to be consistent with [35].

Instead of using a certain load function, we derive all our results for a broad class of load functions (see Section II-B on how to map existing results to our model). Let $l_{max,i}$ be the load that can be effectively handled on a server i (beyond $l_{max,i}$ the server fails due to, e.g., trashing). Let $l_{max} = \max_i l_{max,i}$. Let $l_{tot} = \sum_i n_i$ be the total load in the system. We assume that the total load can be effectively handled, $\sum_i l_{max,i} \geq l_{tot}$ (otherwise, the system is clearly overloaded). We assume that the values $l_{max,i}$ are chosen so that $f_i(l_{max,i})$ are equal to each other (equal to the maximal allowed processing time of the request).

We assume that the load function f_i is bounded on the interval $[0, l_{max,i}]$ (If $l > l_{max,i}$ then we follow the convention that $f_i(l) = \infty$). We assume f_i is non-decreasing as when the load increases, requests are not processed faster. We also assume that f_i is convex and twice-differentiable on the interval $[0; l_{max,i}]$ (continuous functions that are not twice-differentiable can be well approximated by twice-differentiable functions). We assume that the first derivatives f_i' of all f_i are upper bounded by U_1 ($U_1 = \max_{i,l} f_i'(l)$), and that the second derivatives f_i'' are upper bounded by U_2 ($U_2 = \max_{i,l} f_i''(l)$). These assumptions are technical—every continuous function that is defined on the closed interval can be upper-bounded by a constant (however the complexity of our algorithms depends on these constants).

Communication delays. If the request is sent over the network, the observed handling time is increased by the communication latency on the link. We denote the communication latency between i-th and j-th server as c_{ij} (with $c_{ii} = 0$). We assume that the requests are small, and so the communication delay of a single request does not depend on the amount of exchanged load (the same assumption was made in the previous works [5], [18], [20], [25], [35], [44], [46], [51] and it is confirmed by the experiments conducted on PlanetLab [46]). Thus, c_{ij} is just a constant instead of a function of the network load.

We assume *efficient ε-load processing*: for sufficiently small load $\varepsilon \to 0$ the processing time is lower than the communication latency, so it is not profitable to send the requests over the network. Thus, for any two servers i and j we have:

$$h_i(\varepsilon) < \varepsilon c_{ij} + h_j(\varepsilon). \quad (3)$$

We use an equivalent formulation of the above assumption (as $h_i(0) = h_j(0) = 0$):

$$\frac{h_i(\varepsilon) - h_i(0)}{\varepsilon} < c_{ij} + \frac{h_j(\varepsilon) - h_j(0)}{\varepsilon}. \quad (4)$$

Since the above must hold for every sufficiently small $\varepsilon \to 0$, we get:

$$h_i'(0) < c_{ij} + h_j'(0) \Leftrightarrow f_i(0) < c_{ij} + f_j(0). \quad (5)$$

Problem formulation: the total processing time. We consider a system in which all requests have the same importance. Thus, the optimization goal is to minimize the total processing time of all requests $\sum C_i$, considering both the communication latencies and the requests' handling times on all servers, i.e.,

$$\sum C_i = \sum_{i=1}^{m} l_i f_i(l_i) + \sum_{i=1}^{m}\sum_{j=1}^{m} c_{ij} r_{ij}. \quad (6)$$

We formalize our problem in the following definition:

Definition 1 (Load balancing)**.** *Given m servers with initial loads $\{n_i, 0 \leq i \leq m\}$, load functions $\{f_i\}$ and communication delays $\{c_{ij} : 0 \leq i, j \leq m\}$ find ρ, a vector of fractions, that minimizes the total processing time of the requests, $\sum C_i$.*

We denote the optimal relay fractions by ρ^* and the load of the server i in the optimal solution as l_i^*.

B. Motivation

Since the assumptions about the load functions are moderate, our analysis is applicable to many systems. In order to apply our solutions one only needs to find the load functions f_i. In particular, our model generalizes the following previous models.

Queuing model. Our results generalize the results of Liu et al. [35] for the queuing model. In the queuing model, the initial load n_i corresponds to the rate of local requests at the i-th server. Every server i has a processing rate μ_i. According to the queuing theory the dependency between the load l (which is the effective rate of incoming requests) and the service time of the requests is described by $f_i(l) = \frac{1}{\mu_i - l}$ [22]. Its derivative, $f_i'(l) = \frac{1}{(\mu_i - l)^2}$ is upper

bounded by $U_1 = \max_i \frac{1}{(\mu_i - l_{max,i})^2}$, and its second derivative $f_i''(l) = \frac{2}{(l-\mu_i)^3}$ is upper bounded by $U_2 = \max_i \frac{2}{(l_{max,i}-\mu_i)^3}$.

Batch model. Skowron and Rzadca [46] consider a model inspired by batch processing in high-performance computing. The goal is to minimize the flow time of jobs in a single batch. In this case the function f_i linearly depends on load $f_i(l) = \frac{l}{2s_i}$ (where s_i is the speed of the i-th server). Its derivative is constant, and thus upper bounded by $\frac{1}{2s_i}$. The second derivative is equal to 0.

III. Problem Characterization

In this section, we show various results that characterize the solutions in both the single-hop and the multiple-hop models. We will use these results in performance proofs in the next sections.

The relation between the single-hop model and the multiple-hop model is given by the proposition followed by the following lemma.

Lemma 1. *If communication delays satisfy the triangle inequality (i.e., for every i, j, and k we have $c_{ij} < c_{ik} + c_{kj}$), then in the optimal solution there is no server i that both sends and a receives the load, i.e. there is no server i such that $\exists_{j\neq i, k\neq i}((\rho_{ij} > 0) \wedge (\rho_{ki} > 0))$.*

Proof: Proofs omitted in main text are provided in the full version of this paper [47]. ∎

Proposition 2. *If communication delays satisfy the triangle inequality then the single-hop model and the multiple-hop model are equivalent.*

We will also use the following simple observation.

Corollary 3. *The total processing time in the multiple-hop model is not higher than in the single-hop model.*

In the next two statements we recall two results given by Liu et al. [35] (these results were formulated for the general load functions). First, there exists an optimal solution in which only $(2m-1)$ relay fractions ρ_{ij} are positive. This theorem makes our analysis more practical: the optimal load balancing can be achieved with sparse routing tables. However, we note that most of our results are also applicable to the case when every server is allowed to relay its requests only to a (small) subset of the servers; in such case we need to set the communication delays between the disallowed pairs of servers to infinity.

Theorem 4 (Liu et al. [35])**.** *In a single-hop model there exists an optimal solution in which at most $(2m-1)$ relay fractions ρ_{ij} have no-zero values.*

Second, all optimal solutions are equivalent:

Theorem 5 (Liu et al. [35])**.** *Every server i has in all optimal solutions the same load l_i^*.*

Finally, in the next series of lemmas we characterize the optimal solution, by linear equations. We will use these characterization in the analysis of the central algorithm.

Lemma 6. *In the multiple hop model, the optimal solution $\langle \rho_{ij}^* \rangle$ satisfies the following constraints:*

$$\forall_i \quad l_i^* \leq l_{max,i} \tag{7}$$

$$\forall_{i,j} \quad \rho_{ij}^* \geq 0 \tag{8}$$

$$\forall_i \quad \sum_{j=1}^{m} \rho_{ij}^* = 1. \tag{9}$$

Proof: Inequality 7 ensures that the completion time of the requests is finite. Inequalities 8 and 9 state that the values of ρ_{ij}^* are valid relay fractions. ∎

Lemma 7. *In the multiple hop model, the optimal solution $\langle \rho_{ij}^* \rangle$ satisfies the following constraint:*

$$\forall_{i,j} \;\; f_j(l_j^*) + l_j^* f_j'(l_j^*) + c_{ij} \geq f_i(l_i^*) + l_i^* f_i'(l_i^*) \tag{10}$$

Lemma 8. *In the multiple hop model, the optimal solution $\langle \rho_{ij}^* \rangle$ satisfies the following constraint:*

$$\forall_{i,j} \;\; \text{if } \rho_{ij}^* > 0 \text{ then } f_j(l_j^*) + l_j^* f_j'(l_j^*) + c_{ij} \leq f_i(l_i^*) + l_i^* f_i'(l_i^*) \tag{11}$$

Lemma 9. *If some solution $\langle \rho_{ij} \rangle$ satisfies Inequalities 7, 8, 9, 10, and 11 then every server i under $\langle \rho_{ij} \rangle$ has the same load as in the optimal solution $\langle \rho_{ij}^* \rangle$.*

IV. Centralized Algorithm

In this section we show a centralized algorithm for the multiple-hop model. As a consequence of Proposition 2 the results presented in this section also apply to the single-hop model with the communication delays satisfying the triangle inequality.

For the further analysis we introduce the notion of optimal network flow.

Definition 2 (Optimal network flow)**.** *The vector of relay fractions $\rho = \langle \rho_{ij} \rangle$ has an optimal network flow if and only if there is no $\rho' = \langle \rho'_{ij} \rangle$ such that every server in ρ' has the same load as in ρ and such that the total communication delay of the requests $\sum_{i,j} c_{ij} r'_{ij}$ in ρ' is lower than the total communication delay $\sum_{i,j} c_{ij} r_{ij}$ in ρ.*

The problem of finding the optimal network flow reduces to finding a minimum cost flow in uncapacitated network. Indeed, in the problem of finding a minimum cost flow in uncapacitated network we are given a graph with the cost of the arcs and demands (supplies) of the vertices. For each vertex i, b_i denotes the demand (if positive) or supply (if negative) of i. We look for the flow that satisfies demands and supplies and minimizes the total cost. To transform our problem of finding the optimal network flow to the above form it suffices to set $b_i = l_i - n_i$. Thus our problem can be solved in time $O(m^3 \log m)$ [42]. Other distributed algorithms

include the one of Goldberg et al. [21], and the asynchronous auction-based algorithms [7], with e.g., the complexity of $O(m^3 \log(m) \log(\max_{i,j} c_{ij}))$.

The following theorem estimates how far is the current solution from the optimal based on the degree to which Inequality 10 is not satisfied. We use the theorem to prove approximation ratio of the load balancing algorithm.

Theorem 10. *Let ρ be the vector of relay fractions satisfying Inequalities 7, 8, 9 and 11, and having an optimal network flow. Let Δ_{ij} quantify the extent to which Inequality 10 is not satisfied:*

$$\Delta_{ij} = \max(0, f_i(l_i) + l_i f_i'(l_i) - f_j(l_j) - l_j f_j'(l_j) - c_{ij}).$$

Let $\Delta = \max_{i,j} \Delta_{ij}$. Let e be the absolute error—the difference between $\sum C_i$ for solution ρ and for ρ^*, $e = \sum C_i(\rho) - \sum C_i(\rho^*)$. For the multiple-hop model and for the single-hop model satisfying the triangle inequality we get the following estimation:

$$e \leq l_{tot} m \Delta.$$

Proof: Let I be the problem instance. Let $\tilde{I}$ be a following instance: initial loads n_i in $\tilde{I}$ are the same as in I; communication delays c_{ij} are increased by Δ_{ij} ($\tilde{c_{ij}} := c_{ij} + \Delta_{ij}$). Let $\tilde{\rho^*}$ be the optimal solutions for $\tilde{I}$ in the multiple-hop model.

By Lemma 9, loads of servers in ρ are the same as in $\tilde{\rho^*}$, as ρ satisfies all inequalities for $\tilde{I}$. Let c^* and c denote the total communication delay of $\tilde{\rho^*}$ in $\tilde{I}$ and ρ in I, respectively. First, we show that $c^* \geq c$.

For the sake of contradiction, assume that $c^* < c$. We take the solution $\tilde{\rho^*}$ in $\tilde{I}$ and modify $\tilde{I}$ by decreasing each latency $\tilde{c_{ij}}$ by Δ_{ij}. We obtain instance I. During the process, we decreased (or did not change) communication delay over every link, and so we decreased (or did not change) the total communication delay. Thus, in I, $\tilde{\rho^*}$ has smaller communication delay than ρ. This contradicts the thesis assumption that ρ had in I the optimal network flow.

As $\tilde{I}$ has the same initial loads and not greater communication delay,

$$\sum C_i(\rho, I) \leq \sum C_i(\tilde{\rho^*}, \tilde{I}).$$

Based on Proposition 2, the same result holds if ρ is the solution in the single-hop model satisfying the triangle inequality.

We use a similar analysis to bound the processing time. In the multiple-hop model, if the network flow is optimal, then every request can be relayed at most m times. Thus, any solution transfers at most $l_{tot} m$ load. Thus, by increasing latencies from I to $\tilde{I}$ we increase the total communication delay of a solution by at most $l_{tot} m \Delta$. Taking the optimal solution ρ^*, we get:

$$\sum C_i(\rho^*, \tilde{I}) \leq l_{tot} m \Delta + \sum C_i(\rho^*, I).$$

As $\sum C_i(\tilde{\rho^*}, \tilde{I}) \leq \sum C_i(\rho^*, \tilde{I})$, by combining the two inequalities we get:

$$\sum C_i(\rho, I) \leq \sum C_i(\tilde{\rho^*}, \tilde{I}) \leq \sum C_i(\rho^*, \tilde{I}) \leq l_{tot} m \Delta + \sum C_i(\rho^*, I).$$

■

The above estimations allow us to construct an approximation algorithm (Algorithm 1). The lines 15 to 19 initialize the variables. In line 20 we build any finite solution (any solution for which the load l_i on the i-th server does not exceed $l_{max,i}$). Next in the while loop in line 23 we iteratively improve the solution. In each iteration we find the pair (i, j) with the maximal value of Δ_{ij}. Next we balance the servers i and j in line 8. Afterwards, it might be possible that the current solution does not satisfy Inequality 11. In the lines 9 to 13 we fix the solution so that Inequality 11 holds.

Algorithm 1: The approximation algorithm for multiple-hop model.

Notation:
e — the required absolute error of the algorithm.
$\boldsymbol{c_{ij}}$ — the communication delay between i-th and j-th server.
$\boldsymbol{l}[i]$ — the load of the i-th server in a current solution.
$\boldsymbol{r}[i,j]$ — the number of requests relayed between i-th and j-th server in a current solution.
`OptimizeNetworkFlow`$(\rho, \langle c_{ij} \rangle)$ — builds an optimal network flow using algorithm of Orlin [42].

```
1
2  Adjust (i, j):
3      Δr ← argmin_Δr ((l_i − Δr) f_i(l_i − Δr) + (l_j + Δr) f_j(l_j + Δr) + Δr c_ij);
4      l[i] ← l[i] − Δr;
5      l[j] ← l[j] + Δr;
6      r[i, j] ← r[i, j] + Δr;
7  Improve (i, j):
8      Adjust (i, j);
9      servers ← sort servers topologically according to the order ≺:
           i ≺ j ⟺ ρ_ij > 0;
10     for ℓ in servers do
11         for k ← 1 to m do
12             if r[k, ℓ] > 0 and
                   f_ℓ(l[ℓ]) + l[ℓ] f_ℓ'(l[ℓ]) + c_kℓ > f_k(l[k]) + l[k] f_k'(l[k]) then
13                 Adjust (ℓ, k);
14 Main(⟨c_ij⟩, ⟨n_i⟩, ⟨s_i⟩):
15     for i ← 1 to m do
16         l[i] ← n_i;
17         for j ← 1 to m do
18             r[i, j] ← 0;
19         r[i, i] ← n_i;
20     BuildAnyFiniteSolution() ;
21     OptimizeNetworkFlow(r, ⟨c_ij⟩);
22     (i, j) ← argmax_(i,j) Δ_ij;
23     while Δ_ij > e / (l_tot m) do
24         (i, j) ← argmax_(i,j) Δ_ij;
25         Improve(i, j);
26     OptimizeNetworkFlow(r, ⟨c_ij⟩);
```

The following Theorem shows that Algorithm 1 achieves an arbitrary small absolute error e.

Theorem 11. *Let e_d be the desired absolute error for Algorithm 1, and let e_i be the initial error. In the multiple-hop model Algorithm 1 decreases the absolute error from e_i*

Algorithm 2: CALCBESTTRANSFER(i, j)

input: (i, j) – the identifiers of the two servers
Data: $\forall_k\ r_{ki}$ – initialized to the number of requests owned by k and relayed to i ($\forall_k\ r_{kj}$ is defined analogously)
Result: The new values of r_{ki} and r_{kj}

1 **foreach** k **do**
2 $\quad r_{ki} \leftarrow r_{ki} + r_{kj}$; $r_{kj} \leftarrow 0$;
3 $l_i \leftarrow \sum_k r_{ki}$; $l_j \leftarrow 0$;
4 *servers* $\leftarrow$ sort $[k]$ so that $c_{kj} - c_{ki} < c_{k'j} - c_{k'i} \implies k$ is before k';
5 **foreach** $k \in$ *servers* **do**
6 $\quad \Delta_{opt} r_{ikj} \leftarrow$ $\operatorname{argmin}_{\Delta r} \left(h_i(l_i - \Delta r) + h_j(l_j + \Delta r) - \Delta r c_{ki} + \Delta r c_{kj}\right)$;
7 $\quad \Delta r_{ikj} \leftarrow \min\left(\Delta_{opt} r_{ikj}, r_{ki}\right)$;
8 $\quad$ **if** $\Delta r_{ikj} > 0$ **then**
9 $\quad\quad r_{ki} \leftarrow r_{ki} - \Delta r_{ikj}$; $r_{kj} \leftarrow r_{kj} + \Delta r_{ikj}$;
10 $\quad\quad l_i \leftarrow l_i - \Delta r_{ikj}$; $l_j \leftarrow l_j + \Delta r_{ikj}$;
11 **return** *for each k:* r_{ki} *and* r_{kj}

Algorithm 3: Min-Error (MinE) algorithm performed by server id.

1 partner $\leftarrow$ `random(`m`)`;
2 `relay` (id, partner, `calcBestTransfer(`*id, partner*`)`);

to e_d *in time* $O(\frac{{l_{tot}}^2 m^4 e_i (U_1 + l_{max} U_2)}{e_d^2})$.

Using a bound from Theorem 10 corresponding to the single-hop model we get the following analogous results.

Corollary 12. *If the communication delays satisfy the triangle inequality then Algorithm 1 for the single-hop model decreases the absolute error from* e_i *to* e_d *in time* $O(\frac{{l_{tot}}^2 m^4 e_i (U_1 + l_{max} U_2)}{e_d^2})$.

For the relative (to the total load) errors $e_{i,r} = \frac{e_i}{l_{tot}}$, and $e_{d,r} = \frac{e_d}{l_{tot}}$, Algorithm 1 decreases $e_{i,r}$ to $e_{d,r}$ in time $O(\frac{l_{tot}(U_1 + l_{max} U_2) e_{i,r}}{e_{d,r}^2} m^4)$. Thus, we get the shortest runtime if l_{tot} is large and $e_{i,r}$ is small. If the initial error $e_{i,r}$ is large we can use a modified algorithm that performs `OptimizeNetworkFlow` in every iteration of the last "while" loop (line 23). Using a similar analysis as before we get the following bound.

Theorem 13. *The modified Algorithm 1 that performs* `OptimizeNetworkFlow` *in every iteration of the last "while" loop (line 23) decreases the relative error* $e_{i,r}$ *by a multiplicative constant factor in time* $O(\frac{l_{tot} m^5 \log m (U_1 + l_{max} U_2)}{e_{i,r}})$.

Algorithm 1 is any-time algorithm. We can stop it at any time and get a so-far optimized solution. This algorithm is also self-stabilizing.

The complexity of our algorithms requires some discussion. The quality of the results seem to be weaken by the fact that the complexity of decreasing relative error depends on the size of unary representation of total load l_{tot}. However, this is more an artifact of the problem formulation rather than of the quality of algorithms. This can be made clear by the following observation. Consider what happens if every request is divided into $x > 1$ new requests. On one hand, l_{tot} and l_{max} increase by the factor of x. On the other hand, since the requests are smaller, each load function changes as well: the new load function f_i^{new} would have the following form $f_i^{\text{new}}(\ell) = f_i(\frac{\ell}{x})$. Consequently, $U_1^{\text{new}} = \max_{i,\ell}(f_i^{\text{new}})'(\ell) = \max_{i,\ell} \frac{1}{x} f_i'(\frac{\ell}{x}) = \frac{U_1}{x}$. Similarly, $U_2^{\text{new}} = \frac{U_2}{x^2}$. Thus, the complexity of decreasing relative error by a constant factor would remain unchanged. This is what we would, intuitively, expect: if there were no l_{tot} in the formulation of Theorem 13 (and other theorems in the next section) we could obtain better complexity by changing load granularity.

Summarizing, the total load is required in the formulas on convergence time, because it is related to the bounds on derivatives (U_1, U_2). If such total load would not be included in our formulas, it would mean that the complexity of the algorithm could be improved just by dividing requests into smaller pieces (which can be always performed)—and therefore increasing the number of requests in the system—which would be unexpected.

V. DISTRIBUTED ALGORITHM

The centralized algorithm requires the information about the whole network. The size of the input data is $O(m^2)$. A centralized algorithm has thus the following drawbacks: (i) collecting information about the whole network is time-consuming; moreover, loads and latencies may frequently change; (ii) the central algorithm is more vulnerable to failures. Motivated by these limitations we introduce a distributed algorithm for optimizing the query processing time. This decentralized algorithm and its analysis is our core contribution.

Our algorithm uses different ideas than the centralized algorithm and the algorithms of Liu et al. [35] and of Tantawi and Towsley [50]. In our algorithm, each server, i, keeps for each server, k, information about the number of requests that were relayed to i by k. The algorithm iteratively improves the solution – the i-th server in each step communicates with a random partner server – j (Algorithm 3). The pair (i, j) locally optimizes the current solution by adjusting, for each k, r_{ki} and r_{kj} (Algorithm 2). In the first loop of the Algorithm 2, one of the servers i, takes all the requests that were previously assigned to i and to j. Next, all the servers $[k]$ are sorted according to the ascending order of $(c_{kj} - c_{ki})$. The lower the value of $(c_{kj} - c_{ki})$, the less

communication delay we need to pay for running requests of k on j rather than on i. Then, for each k, the loads are balanced between servers i and j. Theorem 14 shows that Algorithm 2 optimally balances the loads on the servers i and j.

The idea of the algorithm is similar to the diffusive load balancing [1], [3], [6]; however there are substantial differences related to the fact that the machines are geographically distributed: (i) In each step no real requests are transferred between the servers; this process can be viewed as a simulation run to calculate the relay fractions ρ_{ij}. Once the fractions are calculated the requests are transferred and executed at the appropriate server. (ii) Each pair (i, j) of servers exchanges not only its own requests but the requests of all servers that relayed their requests either to i or to j. Since different servers may have different communication delays to i and j the local balancing requires more care (Algorithms 2 and 3).

Algorithm 3 has the following properties: (i) The size of the input data is $O(m)$ for each server—communication latencies from a server to all other servers (and not for all pairs of servers). It is easy to measure these pair-wise latencies (Section I). The algorithm is also applicable to the case when we allow the server to relay its requests only to the certain subset of servers (we set the latencies to the servers outside of this subset to infinity). (ii) A single optimization step requires only two servers to be available (thus, it is very robust to failures). (iii) Any algorithm that in a single step involves only two servers cannot perform better (Theorem 14). (iv) The algorithm does not require any requests to be unnecessarily delegated – once the relay fractions are calculated the requests are sent over the network. (v) In each step of the algorithm we are able to estimate the distance between the current solution and the optimal one (Theorem 18).

The following theorem shows the optimality of Algorithm 2.

Theorem 14. *After execution of Algorithm 2 for the pair of servers i and j, $\sum C_i$ cannot be further improved by sending the load of any servers between i and j (by adjusting r_{ki} and r_{kj} for any k).*

A. Convergence

The following analysis bounds the error of the distributed algorithm as a function of the servers' loads. When running the algorithm, this result can be used to assess whether it is still profitable to continue. As the corollary of our analysis we will show the convergence of the distributed algorithm.

In proofs, we will use an *error graph* that quantifies the difference of loads between the current and the optimal solution.

Definition 3 (Error graph)**.** *Let ρ be the snapshot (the current solution) at some moment of execution of the distributed algorithm. Let ρ^* be the optimal solution (if there are multiple optimal solutions with the same $\sum C_i$, ρ^* is the closest solution to ρ in the Manhattan metric). $(P, \Delta\rho)$ is a weighted, directed* error graph *with multiple edges. The vertices in the error graph correspond to the servers; $\Delta\rho[i][j][k]$ is a weight of the edge $i \to j$ with a label k. The weight indicates the number of requests owned by k that should be executed on j instead of i in order to reach ρ^* from ρ.*

The error graphs are not unique. For instance, to move x requests owned by k from i to j we can move them directly, or through some other server ℓ. In our analysis we will assume that the total weight of the edges in the error graph $\sum_{i,j,k} \Delta\rho[i][j][k]$ is minimal, that is that there is no i, j, k, and ℓ, such that $\Delta\rho[i][\ell][k] > 0$ and $\Delta\rho[\ell][j][k] > 0$.

Let $succ(i) = \{j : \exists_k \Delta\rho[i][j][k] > 0\}$ denote the set of (immediate) successors of server i in the error graph; $prec(i) = \{j : \exists_k \Delta\rho[j][i][k] > 0\}$ denotes the set of (immediate) predecessors of i.

We will also use a notion of *negative cycle*: a sequence of servers in the error graph that essentially redirect some of their requests to one another.

Definition 4 (Negative cycle)**.** *In the error graph, a* negative cycle *is a sequence of servers $i_1, i_2, \ldots, i_n$ and labels $k_1, k_2, \ldots, k_n$ such that:*

1) *$i_1 = i_n$; (the sequence is a cycle)*
2) *$\forall_{j \in \{1,\ldots n-1\}} \Delta\rho[i_j][i_{j+1}][k_j] > 0$; (for each pair there is an edge in the error graph)*
3) *$\sum_{j=1}^{n-1} c_{k_j i_{j+1}} < \sum_{j=1}^{n-1} c_{k_j i_j}$ (the transfer in the circle $i_j \xrightarrow{k_j} i_{j+1}$ decreases communication delay).*

A current solution that results in an error graph without negative cycles has smaller processing time: after dismantling a negative cycle, loads on servers remain the same, but the communication time is reduced. Thus, if the current solution has an optimal network flow, then there are no negative cycles in the error graph.

Analogously we define *positive cycles*. The only difference is that instead of the third inequality we require $\sum_{j=1}^{n-1} c_{k_j i_{j+1}} \geq \sum_{j=1}^{n-1} c_{k_j i_j}$. Thus, when an error graph has a positive cycle, the current solution is better than if the cycle would be dismantled.

We start by bounding the load imbalance when there are no negative cycles. The proof of Lemma 15 is somehow involved; for details, we refer the reader to the full version of this paper [47].

Lemma 15. *Let $impr_{pq}$ be the improvement of the total processing time $\sum C_i$ after balancing servers p and q by Algorithm 2. Let l_i be the load of a server i in the current state; and l_i^* be the optimal load. If the error graph $\Delta\rho$ has no negative cycles, then for every positive ε the following*

estimation holds:

$$f_i(l_i) - f_i(l_i^*) \leq \frac{6U_1 + 3l_{max}U_2}{\varepsilon} \max_{pq} impr_{pq} + m\varepsilon.$$

As the result we get the following corollary.

Corollary 16. *If the network flow in the current solution ρ is optimal, then the absolute error e is bounded:*

$$e \leq l_{tot} \frac{6U_1 + 3l_{max}U_2}{\varepsilon} \max_{pq} impr_{pq} + ml_{tot}\varepsilon$$

Proof: The value $f_i(l_i)$ denotes the average processing time of a request on the i-th server. For every server i the average processing time of every request on i in ρ is by at most $\frac{6U_1+3l_{max}U_2}{\varepsilon} \max_{pq} impr_{pq} + m\varepsilon$ greater than in ρ^*. Thus, since there are l_{tot} requests in total, we get the thesis. ∎

We can use Lemma 15 directly to estimate the error during the optimization if we run a distributed negative cycle removal algorithm (e.g. [7], [21]). However, this result is even more powerful when applied together with the lemmas below, as it will allow to bound the speed of the convergence of the algorithm (even without additional protocols optimizing the network flow). Now, we show how to bound the impact of the negative cycles.

Lemma 17. *For every $\varepsilon > 0$, removing the negative cycles improves the total processing time $\sum C_i$ of the solution by at most:*

$$\varepsilon l_{tot} + 2m \sum_{ij} impr_{ij} + \frac{16U_1 + 8l_{max}U_2}{\varepsilon} \max_{ij} impr_{ij} l_{tot}.$$

Finally we get the following estimation.

Theorem 18. *Let $impr_{ij}$ be the improvement of the total processing time $\sum C_i$ after balancing servers i and j through Algorithm 2. Let e be the absolute error in $\sum C_i$ (the difference between $\sum C_i$ in the current and the optimal state). For every $\varepsilon > 0$, we have:*

$$e \leq 2m \sum_{ij} impr_{ij} + \max_{ij} impr_{ij} \frac{22U_1 + 11l_{max}U_2}{\varepsilon} l_{tot} + (m+1) l_{tot} \varepsilon.$$

Proof: The error coming from the negative cycles is bounded by Lemma 17 by:

$$\varepsilon l_{tot} + 2m \sum_{ij} impr_{ij} + \frac{16U_1 + 8l_{max}U_2}{\varepsilon} \max_{ij} impr_{ij} l_{tot}. \quad (12)$$

The error coming from the processing times is, according to Lemma 15 bounded by:

$$l_{tot} \frac{6U_1 + 3l_{max}U_2}{\varepsilon} \max_{ij} impr_{ij} + ml_{tot}\varepsilon$$

The sum of the above errors leads to the thesis. ∎

And the following theorem.

Theorem 19. *Let e_i and e_d be the initial and the desired absolute errors. The distributed algorithm reaches the e_d in expected time complexity (the random element of the algorithm is the process of selecting pair peers):*

$$O\left(\frac{{l_{tot}}^2 (U_1 + l_{max}U_2) e_i m^3}{e_d^2}\right).$$

Proof: In the estimation from Theorem 18 we set $\varepsilon = \frac{e_d}{2(m+1)l_{tot}}$ and relax the upper bound by replacing $\max_{i,j} impr_{ij}$ with $\sum_{i,j} impr_{ij}$:

$$e \leq (2m+2) \sum_{i,j} impr_{ij} \left(1 + \frac{22U_1 + 11l_{max}U_2}{e_d} {l_{tot}}^2\right) + \frac{e_d}{2}$$
$$\approx 2m \sum_{i,j} impr_{ij} \frac{22U_1 + 11l_{max}U_2}{e_d} {l_{tot}}^2 + \frac{e_d}{2}.$$

Thus, either:

$$2m \sum_{ij} impr_{ij} \frac{22U_1 + 11l_{max}U_2}{e_d} {l_{tot}}^2 \leq \frac{e_d}{2},$$

and the algorithm has already reached the error e_d; or in every execution step we have:

$$\sum_{i,j} impr_{ij} \geq \frac{e_d^2}{4m(22U_1 + 11l_{max}U_2){l_{tot}}^2}.$$

The expected improvement of the distributed algorithm during every pairwise communication is $\frac{1}{m^2} \sum_{i,j} impr_{ij}$, and thus it is lower bounded by:

$$\frac{e_d^2}{4m^3(22U_1 + 11l_{max}U_2){l_{tot}}^2}.$$

Thus, after, in expectation, $O(\frac{{l_{tot}}^2 (U_1 + l_{max}U_2) e_i m^3}{e_d^2})$ steps the initial error drops to 0. This completes the proof. ∎

For the relative errors $e_{i,r} = \frac{e_i}{l_{tot}}$, and $e_{d,r} = \frac{e_d}{l_{tot}}$, the complexity of the algorithm is equal to $O(\frac{l_{tot}(U_1 + l_{max}U_2) e_{i,r} m^3}{e_{d,r}^2})$.

VI. Conclusions

In this paper we considered the problem of balancing the load between geographically distributed servers. In this problem the completion time of a request is the sum of the communication latency needed to send the request to a server and the servers' processing time. The processing time on a server is described by a load function and depends on the total load on the server. Throughout the paper we considered a broad class of load functions with the mild assumptions that they are convex and two times differentiable.

We presented two algorithms—the centralized one and the distributed one. Both algorithms are any-time and self-stabilizing algorithms that continuously optimize the current solution. We showed that both algorithms converge for almost arbitrary load function. We also presented bounds on the speed of their convergence that depend (apart from the standard parameters) on bounds on the first and second

derivatives of the load functions. The centralized algorithm decreases an initial relative error $e_{i,r}$ to a desired value $e_{d,r}$ in time $O(\frac{l_{tot}(U_1+l_{max}U_2)e_{i,r}}{e_{d,r}^2}m^4)$. The distributed algorithm decreases $e_{i,r}$ to $e_{d,r}$ in $O(\frac{l_{tot}(U_1+l_{max}U_2)e_{i,r}}{e_{d,r}^2}m^3)$ steps. Also, for the large values of initial error $e_{i,r}$ the centralized algorithm decreases the error by half in time $O(\frac{l_{tot}(U_1+l_{max}U_2)}{e_{i,r}}m^5\log m)$.

While the running time of our algorithms depends on the total load l_{tot}, in the two last paragraphs of Section IV we explain that this is not an indicator of the algorithms' slowness. Including l_{tot} in the formulas is required to compensate the bounds on derivatives (U_1, U_2), which depend on granularity of the load.

The run-times of our algorithms depend non-linearly on the number of servers, however we argue that these algorithms are applicable in practice: (i) The "server" as used in our model can correspond to e.g., a datacenter, rather than a single node: the number of distinct servers is the number of entities between which there is non-negligible latency (latency resulting from geographic distance). Even in large commercial systems, there are dozens, rather than thousands of datacenters. (ii) Our algorithms can be used as a reference for comparing other faster heuristics through simulations. (iii) Our load balancing optimization can be run in the background (in an on-line system, continuously); the solution—the resulting load fractions—would drive the actual load balancing decisions, which are instantaneous.

The distributed algorithm is based on the idea of gossiping. To perform a single optimization step, the algorithm requires just two servers to be available. Thus, the algorithm is robust to transient failures. It also does not require additional protocols. In some sense it is also optimal: we proved that the local optimization step performed by this algorithm cannot be improved. Finally, at any time moment, during the execution of the distributed algorithm, we are able to assess the current error.

Experimental results were shown for a different algorithm applied to the queuing model [35]; and for a version of our distributed algorithm specialized to the batch model [46]. In our future work we plan to experimentally assess the performance of our algorithms on real workloads and several load functions, including the queuing model.

Acknowledgments. This work was supported by Polish National Science Center grants: Sonata 2012/07/D/ST6/02440 and Preludium 2013/09/N/ST6/03661.

References

[1] H. Ackermann, S. Fischer, M. Hoefer, and M. Schöngens. Distributed algorithms for qos load balancing. In *Proceedings of the twenty-first annual symposium on Parallelism in algorithms and architectures*, SPAA '09, pages 197–203, 2009.

[2] C. P. J. Adolphs and P. Berenbrink. Distributed selfish load balancing with weights and speeds. In *PODC*, pages 135–144, 2012.

[3] C. P. J. Adolphs and P. Berenbrink. Improved bounds for discrete diffusive load balancing. In *IPDPS*, pages 820–826. IEEE Computer Society, 2012.

[4] S. S. Aote and M. U. Kharat. A game-theoretic model for dynamic load balancing in distributed systems. In *ICAC3*, pages 235–238, 2009.

[5] O. Beaumont, H. Casanova, A. Legrand, Y. Robert, and Y. Yang. Scheduling divisible loads on star and tree networks: results and open problems. *Parallel and Distributed Systems, IEEE Transactions on*, 16(3):207–218, 2005.

[6] P. Berenbrink, M. Hoefer, and T. Sauerwald. Distributed selfish load balancing on networks. In *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '11, pages 1487–1497, 2011.

[7] D. P. Bertsekas. Auction algorithms for network flow problems: A tutorial introduction. *Computational Optimization and Applications*, 1:7–66, 1992.

[8] J. Brutlag. Speed matters. http://googleresearch.blogspot.com/2009/06/speed-matters.html, 2009.

[9] S. K. Card, T. P. Moran, and A. Newell. *The psychology of human computer interaction*. Routledge, 1983.

[10] V. Cardellini. Geographic load balancing for scalable distributed web systems. In *MASCOTS*, pages 20–27, 2000.

[11] E. Chan-Tin and N. Hopper. Accurate and provably secure latency estimation with treeple. In *NDSS*. The Internet Society, 2011.

[12] T. C. K. Chou and J. A. Abraham. Load balancing in distributed systems. *IEEE Transactions on Software Engineering*, 8(4):401–412, 1982.

[13] F. Chudak and D. P. Williamson. Improved approximation algorithms for capacitated facility location problems. *Math. Program.*, 102(2):207–222, Mar. 2005.

[14] M. Colajanni, P. S. Yu, and V. Cardellini. Dynamic load balancing in geographically distributed heterogeneous web servers. In *In International Conference on Distributed Computing Systems*, pages 295–302, 1998.

[15] E. Cronin, S. Jamin, C. Jin, A. R. Kurc, D. Raz, Y. Shavitt, and S. Member. Constrained mirror placement on the internet. In *JSAC*, pages 31–40, 2002.

[16] G. Cybenko. Dynamic load balancing for distributed memory multiprocessors. *J. Parallel Distrib. Comput.*, 7(2):279–301, Oct. 1989.

[17] S. Dhakal, M. M. Hayat, J. E. Pezoa, C. Yang, and D. A. Bader. Dynamic load balancing in distributed systems in the presence of delays: A regeneration-theory approach. *IEEE Trans. Parallel Distrib. Syst.*, 18(4):485–497, Apr. 2007.

[18] M. Drozdowski and M. Lawenda. Scheduling multiple divisible loads in homogeneous star systems. *Journal of Scheduling*, 11(5):347–356, 2008.

[19] M. Freedman. Experiences with coralcdn: A five-year operational view. In *NSDI USENIX, Proceedings*, 2010.

[20] M. Gallet, Y. Robert, and F. Vivien. Divisible load scheduling. In Y. Robert and F. Vivien, editors, *Introduction to Scheduling*. CRC Press, Inc., 2009.

[21] A. V. Goldberg and R. E. Tarjan. Finding minimum-cost circulations by successive approximation. *Math. Oper. Res.*, 15:430–466, July 1990.

[22] D. Gross, J. F. Shortle, J. M. Thompson, and C. M. Harris. *Fundamentals of Queueing Theory*. Wiley-Interscience, New York, NY, USA, 4th edition, 2008.

[23] D. Grosu and A. T. Chronopoulos. Algorithmic mechanism design for load balancing in distributed systems. In *CLUSTER*, pages 445–, Washington, DC, USA, 2002. IEEE Computer Society.

[24] D. Grosu and A. T. Chronopoulos. Noncooperative load balancing in distributed systems. *Journal of Parallel and Distributed Computing*, 65(9):1022 – 1034, 2005.

[25] D. Grosu, A. T. Chronopoulos, and M. Y. Leung. Cooperative load balancing in distributed systems. *Concurr. Comput. : Pract. Exper.*, 20(16):1953–1976, Nov. 2008.

[26] A. Hać. Load balancing in distributed systems: A summary. *SIGMETRICS Perform. Eval. Rev.*, 16(2-4):17–19, 1989.

[27] M. M. Hayat, S. Dhakal, C. T. Abdallah, J. Douglas, and J. Chiasson. Dynamic time delay models for load balancing, part ii: A stochastic analysis of the effect of delay uncertainty. In *CNRS-NSF Workshop: Advances in Control of Time-Delay Systems*, 2003.

[28] K. Jain and V. Vazirani. Approximation algorithms for metric facility location and k-median problems using the primal-dual schema and lagrangian relaxation. *J. ACM*, 48(2):274–296, Mar. 2001.

[29] X. Jia, D. Li, X. Hu, and D. Du. Optimal placement of web proxies for replicated web servers in the internet. *Comput. J.*, 44(5):329–339, 2001.

[30] H. Kameda, J. Li, C. Kim, and Y. Zhang. *Optimal Load Balancing in Distributed Computer Systems*. Springer Publishing Company, Incorporated, 1st edition, 2011.

[31] F. Leighton and D. Lewin. Global hosting system. US Patent No. 6,108,703.

[32] Y. Li and Z. Lan. A survey of load balancing in grid computing. In *CIS'04*, pages 280–285, 2004.

[33] J. T. Lim and S. M. Meerkov. Distributed load balancing. In *IEEE Conference on Decision and Control*, volume 2, page 1487, 1988.

[34] M. Lin, Z. Liu, A. Wierman, and A. L. H. Lachlan. Online algorithms for geographical load balancing. In *IGCC*, pages 1–10, 2012.

[35] Z. Liu, M. Lin, A. Wierman, S. H. Low, and L. L. Andrew. Greening geographical load balancing. In *SIGMETRICS*, pages 233–244, 2011.

[36] Z. Liu, M. Lin, A. Wierman, S. H. Low, and A. L. H. Lachlan. Geographical load balancing with renewables. *SIGMETRICS Perform. Eval. Rev.*, 39(3):62–66, Dec. 2011.

[37] Z. Liu, A. Wierman, Y. Chen, B. Razon, and N. Chen. Data center demand response: Avoiding the coincident peak via workload shifting and local generation. *SIGMETRICS Perform. Eval. Rev.*, 41(1):341–342, June 2013.

[38] S. Lohr. For impatient web users, an eye blink is just too long to wait. *New York Times*, 2012.

[39] R. Mahajan. How akamai works? http://research.microsoft.com/en-us/um/people/ratul/akamai.html.

[40] N. Nisan, T. Roughgarden, E. Tardos, and V. V. Vazirani. *Algorithmic Game Theory*, chapter Routing Games. Cambridge University Press, 2007.

[41] E. Nygren, R. K. Sitaraman, and J. Sun. The Akamai network: a platform for high-performance internet applications. *SIGOPS Oper. Syst. Rev.*, 44:2–19, August 2010.

[42] J. B. Orlin. A faster strongly polynomial minimum cost flow algorithm. In *OPERATIONS RESEARCH*, pages 377–387, 1988.

[43] G. Pallis and A. Vakali. Content delivery networks. *Communications of the ACM*, 49(1):101, 2006.

[44] S. Penmatsa and A. T. Chronopoulos. Cooperative load balancing for a network of heterogeneous computers. In *IPDPS*, pages 162–162, 2006.

[45] L. Qiu, V. N. Padmanabhan, and G. M. Voelker. On the placement of web server replicas. In *IEEE INFOCOM, Proceedings*, pages 1587–1596, 2001.

[46] P. Skowron and K. Rzadca. Network delay-aware load balancing in selfish and cooperative distributed systems. In *IPDPSW*, pages 7–18, 2013.

[47] P. Skowron and K. Rzadca. We are impatient: Algorithms for geographically distributed load balancing with (almost) arbitrary load functions. *CoRR*, abs/1402.2090, 2014.

[48] A.-J. Su, D. R. Choffnes, A. Kuzmanovic, and F. E. Bustamante. Drafting behind Akamai. *SIGCOMM Comput. Commun. Rev.*, 36:435–446, August 2006.

[49] M. Szymaniak, G. Pierre, and M. Steen. Scalable cooperative latency estimation. In *ICPADS*, 2004.

[50] A. Tantawi and D. Towsley. Optimal static load balancing in distributed computer systems. *J. ACM*, 32(2):445–465, 1985.

[51] B. Veeravalli and G. Barlas. Efficient scheduling strategies for processing multiple divisible loads on bus networks. *Journal of Parallel and Distributed Computing*, 62(1):132–151, 2002.