Distributed Algorithms for QoS Load Balancing*

Heiner Ackermann† Simon Fischer‡ Martin Hoefer§ Marcel Schöngens¶

Abstract

We consider a dynamic load balancing scenario in which users allocate resources in a non-cooperative and selfish fashion. The perceived performance of a resource for a user decreases with the number of users that allocate the resource. In our dynamic, concurrent model, users may reallocate resources in a round-based fashion. As opposed to various settings analyzed in the literature, we assume that users have quality of service (QoS) demands. A user has zero utility when falling short of a certain minimum performance threshold and having positive utility otherwise.

Whereas various load-balancing protocols have been proposed for the setting without quality of service requirements, we consider protocols that satisfy an additional locality constraint: The behavior of a user depends merely on the state of the resource it currently allocates. This property is particularly useful in scenarios where the state of other resources is not readily accessible. For instance, if resources represent channels in a mobile network, then accessing channel information may require time-intensive measurements.

We consider several variants of the model, where the quality of service demands may depend on the user, the resource, or both. For all cases we present protocols for which the dynamics converge to a state in which all users are satisfied. More importantly, the time to reach such a state scales nicely. It is only logarithmic in the number of users, which makes our protocols applicable in large-scale systems.

*A preliminary extended abstract of this paper has appeared in the proceedings of SPAA 2009 [2]. Supported by DFG through UMIC Research Center at RWTH Aachen University and by grant Ho 3831/3-1.

†Fraunhofer Institute for Industrial Mathematics ITWM, Kaiserslautern, Germany. Email: heiner.ackermann@itwm.fraunhofer.de

‡Dept. of Computer Science, RWTH Aachen University, Germany. Email: fischer@rapid-i.com

§Dept. of Computer Science, RWTH Aachen University, Germany. Email: mhoefer@cs.rwth-aachen.de

¶Institute of Theoretical Computer Science, ETH Zürich, Switzerland. Email: schoengens@inf.ethz.ch

1 Introduction

The recent decade has witnessed the advent of large computational systems, which work massively parallel or are composed of a multitude of decentralized units. In such systems load balancing is an essential need to provide satisfactory functionality. In the literature classic load balancing problems are often modeled and analyzed using a centralized and rather static approach, in which an optimization algorithm is able to detect all relevant parameters and to centrally implement a solution. Such an approach, however, neglects important characteristics of current load balancing problems in distributed systems, such as, e.g., in the channel allocation process in wireless networks. Due to their size and decentralized nature, it becomes infeasible to even detect all relevant parameters that a global optimization would require. Instead, recent work has focused on distributed protocols, in which decisions about task allocation are transfered from a central authority to independent users, namely, the tasks themselves. Exploring the possibilities of this more realistic paradigm has attracted a lot of attention in algorithmic game theory on the incentives for rational users in load balancing scenarios [23]. Ideally, the population of users converges rapidly to a balanced state, while each user independently decides about the allocation of his task based only on local information, and without cooperation or global coordination. As a side product of the rationality assumption, users still need to have some global knowledge about loads and latencies to make fully informed decisions. This makes them much less applicable in a decentralized setting. In contrast, in practice users are often satisfied if they obtain a certain minimum performance, e.g., a maximum response time or latency – independent of the fact whether the performance on a different resource is slightly better or not. In this paper, we incorporate this idea and study resource allocation problems with users having such Quality of Service (QoS) demands. This will allow us to provide algorithms that do not require global coordination or cooperation and yield rapid convergence to balanced states, while significantly strengthening the locality constraint.

1.1 Model and Results

Let us at first formally introduce our model and state our results before we proceed with a discussion and motivation of the modeling choices. We consider a distributed load-balancing problem with a set $\mathcal{R}$ of m resources and a set $\mathcal{N}$ of n users. For a fixed assignment of users to resources, let x denote the congestion profile of the resources, i. e., x_r denotes the number of users allocating resource r in this assignment. For every user $i \in \mathcal{N}$ and every resource $r \in \mathcal{R}$, let $T_r^i \in \mathbb{N}$ be the threshold of user i on resource r. That is, if user i allocates resource r, and if $x_r \leq T_r^i$, then user i is *satisfied* with resource r. Otherwise, a user is *unsatisfied* with resource r. An assignment is called *balanced* or in a *stable state* if all users are satisfied with their currently allocated resource.

We say that an instance of the load balancing problem is *user-independent* if for every resource $r \in \mathcal{R}$ there exists T_r such that $T_r^i = T_r$ for all $i \in \mathcal{N}$, and *user-specific* otherwise. Similarly, we say that an instance is *resource-independent* if for every user $i \in \mathcal{N}$ there exists T^i such that $T_r^i = T^i$ for all $r \in \mathcal{R}$, and *resource-specific* otherwise. Finally, an instance is *uniform* if it is both resource-independent and user-independent.

In our model, we assume that each user controls a similar piece of unsplittable load that needs to be allocated to one of the resources. The performance of a resource degrades with the total number of users allocating their load to it (or, equivalently, with the total load assigned to the resource). We assume that the assignment of users to resources is controlled by the users themselves, and that they act in a non-cooperative manner, i. e. they independently form decisions about allocation of their tasks. We assume that users are satisfied if the performance of their chosen resource exceeds a certain threshold. Only unsatisfied users try to move over time and improve by exchanging their chosen resource. In our scenario we assume that there is no central authority that dictates or manages the allocation. Instead, we are interested in obtaining distributed algorithms that (1) can be executed by the users locally and (2) yield rapid convergence to balanced states. We would like to avoid that users must have a global view of the system, they should only use information that is available locally. This includes, for example, that we do not require users to coordinate their actions, that the algorithms are stateless (i.e., users do not need to have memory), and that users base their migration decisions only on the quality of their currently allocated resource. Furthermore, the convergence time should be sublinear in the number of users, which is essential for rapid convergence in large-scale systems. Note

that such a goal can only be obtained by algorithms that allow concurrent migration.

In Section 2 we consider the uniform case that each resource r yields a threshold T_r for all users. If every T_r is at least by a constant (exceeding unity by an arbitrarily small amount) larger than an average load n/m, the convergence time of our algorithm is only $\mathcal{O}(\log n)$. In case the above average property is absent, but thresholds T_r admit a stable state, we can adjust the algorithm to yield a convergence time logarithmic in n, but polynomial in m. It is not surprising that m must occur polynomially in this bound since it might be that there exists a single good resource whereas all other resources are poor. If all users must eventually move to the same resource, it takes expected time $\Omega(m)$ for a single user running our algorithm to find this resource.

The above average property expresses an intuitive condition, e.g., if all users and resources have similar requirements and performance properties, respectively. In this case, all thresholds should be similar, and therefore be approximately n/m to guarantee the existence of a stable state. In addition, our results indicate that above average thresholds represent "nice" cases of the load balancing problem, where we can obtain convergence times independent of m.

In Section 3 we consider the case that the threshold does not only depend on the resource, but is also user-specific. We then consider a protocol in which overloaded resources deallocate some of their users which subsequently reallocate resources chosen uniformly at random. Again, if thresholds have the above average property, the convergence time only depends in a logarithmic fashion on the number of users. In this case, the above average property is also necessary, as there are simple examples in which we fail to achieve convergence at all.

Finally, Section 4 shows how our protocols can be applied if even the number of users is unknown. We conclude the paper in Section 5 with some open problems for future work.

1.2 User Thresholds

We assume satisfaction of users depends on whether the current performance of the resource is above a threshold. The idea of such threshold functions has recently found interest in different areas of distributed systems. An example, which is close to our focus, are recent advances in the modeling of interference and channel allocation in wireless networks. Traditionally, this problem has been studied using geometric intersection graphs, where a node can successfully access a channel whenever all neighboring nodes are inactive. This gives rise to independent set and coloring problems, which have been treated extensively. More recently, the focus has shifted to the more realistic *physical model of interference* [18], which is widely used in engineering. In this scenario, interferences do not come as binary conflicts, but they are quantifiable and accumulate. The central criterion of success is that the so-called signal-to-interference-plus-noise ratio (SINR) satisfies a threshold. This bounds the maximum allowable amount of interference (and noise) such that successful communication on a channel is still possible. In this case, channel allocation becomes essentially a graph-based load balancing problem with user thresholds, where resources are channels, "loads" are interferences, and the threshold of satisfaction is given by the bound on the SINR ratio. In this way, the model and the algorithms studied in this paper can be used as a (rather coarse) approach to distributed and rapid minimization of interference (see [22]). However, our focus here is not to treat this specific application in detail. Instead, our interest is more fundamental – to understand the properties and advantages of thresholds in the context of distributed load balancing algorithms.

A simple but important advantage of user preferences based on thresholds is their locality. In the literature various concurrent protocols for distributed load balancing are proposed. In general, these protocols operate by activating users in parallel allowing them to improve their currently perceived performance. For example, a user currently assigned to a resource may sample another resource according to a probability distribution and migrate to the new resource with a certain probability. Whereas being based on local information in principle, most of the protocols presented in the literature also rely on some amount of global information, e.g. the set of underloaded resources or the current performance of the sampled resource. In contrast, the user thresholds allow us to design algorithms, in which the actions performed by a user depend only on information about the performance of the resource it is currently assigned to.

1.3 Game Theory

In principle, our paper connects to related work in algorithmic game theory mentioned above and surveyed in more detail below. We can formulate thresholds and users being satisfied or not as a simple game. In this case, the set of resources is the set of strategies for all users. Each user is a player that has a 0/1 utility function. The utility is 1 if and only if the resource has a total load of less than or equal to the threshold and 0 otherwise. Then a stable state as defined above is obviously a Nash equilibrium of the game, as all players obtain a maximum utility of 1. In this sense, the algorithms presented in this paper guarantee rapid convergence to Nash equilibrium.

Nevertheless, we believe that the connection to game theory in this work is only loose. One might argue that when users are allowed to act strategically, their inherent preferences are often much more complicated than the resulting 0/1 utility functions. Rational acting upon more complicated preferences, however, requires users to compare resources, and then non-local algorithms and protocols studied in the literature seem more appropriate. Alternatively, a user with a threshold utility function for the load might be interested in frequent migration to arrive at a favorable resource as quickly as possible – or he might be interested in being reassigned a minimum number of times. Such natural strategic objectives have neither been considered in this paper nor in any related work the authors are aware of. Finally, in later sections we assume that in each round users report their thresholds to the resources, which then decide about evacuation of users. In this process, users can obviously have a strategic incentive to misreport their thresholds in order to avoid or promote migration. Such a repeated mechanism design scenario is beyond the scope of this paper and represents an interesting avenue for future work.

In conclusion, our algorithms seem more appropriate in a non-strategic case, when we strive to design interaction protocols to arrive at states that meet performance thresholds imposed by the users of the system.

1.4 Related Work

Various protocols for dynamic load balancing are proposed in the literature. These protocols usually model the load balancing process as a weighted congestion game [21] with non-cooperative selfish users. In this setting there are n weighted users that assign load to m resources or parallel links, and the latency equals (an affine function of) the load of a resource. A stable state in this context is a Nash equilibrium, in which no user unilaterally can decrease his latency by reassignment to a different resource. An overview about existence, cost, and complexity of Nash equilibria in selfish load balancing is given by Vöcking [23].

While most work concentrates on centralized computation of Nash equilibria and worst-case sequential best-response dynamics [13], Goldberg [17] considers a randomized sequential process for the case, in which the latency equals the load of a link. Starting from an initial state randomly selected users move sequentially. They migrate to a randomly selected resource if this improves their latency. The expected time to reach a Nash equilibrium is pseudopolynomial. Sequential dynamics for scenarios with latency functions are also considered in [11].

Concurrent protocols for load balancing have received some attention only recently. Even-Dar and Mansour [12] consider concurrent protocols in a setting where links have speeds. However, their protocols require global knowledge in the sense that the users must be able to determine the set of underloaded and overloaded links. The convergence time is doubly logarithmic in the number of users. In [4] Berenbrink *et al* consider a distributed protocol for the case that the latency equals the load that does not rely on this knowledge. Their bounds on the convergence time are also doubly logarithmic in the number of users but polynomial in the number of links. In [5] the results are generalized to the case of weighted users. In this case, the convergence time is only pseudopolynomial, i. e., polynomial in the number of users, links, and in the maximum weight. More recently, the last two protocols have been generalized to load balancing over networks [6], for more recent work on atomic network load balancing scenarios see [8, 9, 10]. Fotakis *et al* [16] consider a scenario with latency functions for every resource. Their protocol involves local coordination among the users sharing a resource. If the number of users asymptotically equals the number of resources, it yields fast convergence to almost Nash equilibria. Intuitively, an almost Nash equilibrium is a state in which there are not too many too expensive and too cheap resources. In [1], Ackermann *et al* propose an imitation protocol for general

atomic congestion games. In this protocol each user samples another user and imitates its strategy if it yields latency improvement. It yields rapid convergence to approximately balanced states, but also relies on knowledge about the resources in the sampled strategy. Finally, the performance of no-regret learning dynamics has been analyzed in atomic load balancing, however, these dynamics usually only converge in the history of play and only to a stable *distribution* over states, such as mixed Nash or correlated equilibria [20].

A protocol similar to the one presented here is considered in [15] in the fluid limit, i.e., for the non-atomic case with a finite demand of infinitely many, infinitesimally small users. In finite time this model only allows convergence to a state that approximately satisfies all the user thresholds. There are numerous additional results on such non-atomic [14] or atomic-splittable cases [3], where in the latter each of a finite number of users can split his demand arbitrarily.

2 User-Independent Instances

We first consider the case that on each resource all users have the same threshold. Our fully distributed load balancing algorithm proceeds as follows. Any user allocating a resource with the congestion x_r that exceeds the resource's threshold T_r leaves this resource with a probability of $\alpha \cdot \frac{x_r - T_r}{x_r}$ and reassigns itself to a resource chosen uniformly at random. The algorithm is given in pseudocode as Algorithm 1. It uses a parameter α that depends on the thresholds and is defined below.

Algorithm 1 Threshold-Balancing Protocol

```
for all users i in parallel do
  Let r(i) be the resource currently allocated by user i.
  if x_{r(i)} > T_{r(i)} then
    With probability α · (x_{r(i)} − T_{r(i)}) / x_{r(i)} migrate to a resource chosen uniformly at random.
  end if
end for
```

We assume all users repeat the execution this algorithm infinitely often in a round-based fashion. To analyze convergence properties of our protocol we introduce a potential function. This potential simply sums up the load by which the thresholds are exceeded. More precisely, let

$$\Phi_r(x) = \max\{0, x_r - T_r\}$$

and

$$\Phi(x) = \sum_{r \in \mathcal{R}} \Phi_r(x) \ .$$

For subsequent states x and x', the potential difference is

$$\Delta\Phi(x, x') = \Phi(x') - \Phi(x) \ .$$

The contribution of resource r to the potential difference is defined as

$$\Delta\Phi_r(x, x') = \begin{cases} \max\{0, T_r - x'_r\} & \text{if } x_r > T_r \\ -(\min\{x'_r, T_r\} - x_r) & \text{if } x_r \leq T_r. \end{cases}$$

We call a resource *overloaded* if $x_r > T_r$ and *underloaded* if $x_r < T_r$. Note that $\Delta\Phi_r(x, x') \neq \Phi_r(x') - \Phi_r(x)$. The first term accounts for users that leave an overloaded resource r' and decrease its load to below $T_{r'}$. These users do not create a potential gain since they did not contribute to Φ in x. Thus, they are counted as increase in potential. Intuitively, the second term counts the number of users migrating into "holes", thus reducing the potential if they come from an overloaded resource. Any user migrating to a resource r and increasing its load to above T_r does not reduce the potential any further, so the gain is cut off at $T_r - x_r$. For an intuitive understanding of Δ_r see Figure 1.

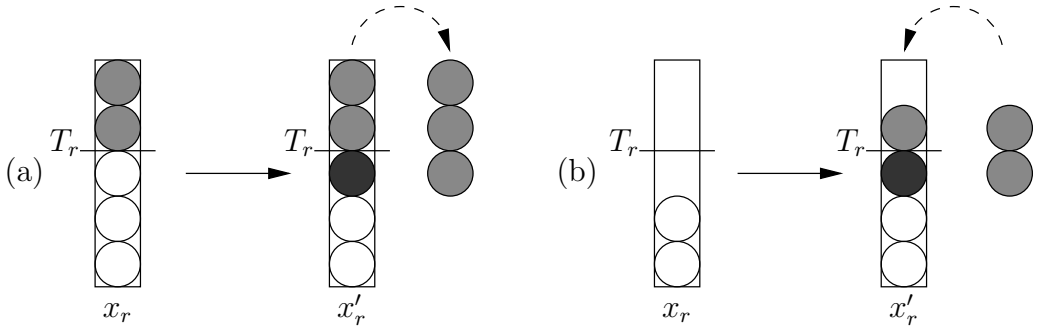

Figure 1: Potential Difference. We artificially split the migration process in two stages. (a) Stage 1: Evacuation of overloaded resources. Users counted in Φ (gray) that migrate remain counted. If more than the excess users migrate (dark gray), these additional users get added to the potential. All migrating users are temporarily counted in Φ. (b) Stage 2: Arrival on all resources. If users arrive and the load remains below the threshold, these users are subtracted from the potential (dark gray). If more users arrive above the threshold, they remain counted towards Φ (gray). $\Delta\Phi_r$ is given by the number of dark gray users, which are counted positive in (a) and negative in (b). In total, this yields $\Delta\Phi$.

To show that the expressions $\Delta\Phi_r$ are useful in capturing progress in Φ, we define $\mathcal{R}_o(x)$ to be the set of overloaded resources in x and $\mathcal{R}_{\overline{o}}(x) = \mathcal{R} - \mathcal{R}_o(x)$. In addition, let

$$\begin{aligned} A &= \mathcal{R}_o(x) \cap \mathcal{R}_o(x') \\ B &= \mathcal{R}_o(x) \cap \mathcal{R}_{\overline{o}}(x') \\ C &= \mathcal{R}_{\overline{o}}(x) \cap \mathcal{R}_o(x') \\ D &= \mathcal{R}_{\overline{o}}(x) \cap \mathcal{R}_{\overline{o}}(x') \end{aligned}$$

be the sets of resources that become or remain overloaded/non-overloaded in x and x'. As indicated in Figure 1, we can intuitively imagine that users are stacked on resources. The change in potential $\Delta\Phi(x, x')$ measures the change in the number of total users on all resources stacked above the height of the thresholds, i.e.,

$$\Delta\Phi(x, x') = \sum_{r \in A}(x'_r - x_r) + \sum_{r \in B}(T_r - x_r) + \sum_{r \in C}(x'_r - T_r) + \sum_{r \in D} 0 \ .$$

If the number of users stacked above the thresholds changes, then obviously the total number of users stacked below the thresholds changes by the same amount. In particular, the change in the number of users stacked below the thresholds is

$$\sum_{r \in A} 0 + \sum_{r \in B}(x'_r - T_r) + \sum_{r \in C}(T_r - x_r) + \sum_{r \in D}(x'_r - x_r) = -\Delta\Phi(x, x') \ .$$

Note that

$$\Delta\Phi_r(x, x') = \begin{cases} 0 & \text{for } r \in A, \\ T_r - x'_r & \text{for } r \in B, \\ x_r - T_r & \text{for } r \in C, \\ x_r - x'_r & \text{for } r \in D, \end{cases}$$

and therefore

$$\sum_{r \in R} \Delta_r(x, x') = \Delta\Phi(x, x') \ .$$

Hence, in total the terms $\Delta\Phi_r$ capture the improvement in Φ correctly.

2.1 Above Average Thresholds

In this section we consider convergence times of our algorithm in instances, in which the thresholds T_r are above average, i.e., in which $T_r > n/m$. We will show below that with a suitable choice of α the potential decreases in expectation in every round. This allows us to bound the expected convergence time to a stable state.

Whether or not the potential actually decreases in expectation depends on whether the terms $\Delta\Phi_r$ dominate for the underloaded or for the overloaded resources. In particular, if the current state is close to balanced, it might become more likely that on an overloaded resource more than $x_r - T_r$ users leave whereas only a small fraction of them actually chooses to migrate to an underloaded resource. In this case the potential may even increase in expectation. To avoid this behavior, we need the damping factor α in our protocol. The closer the thresholds T_r are to n/m, the more difficult it becomes to balance the system. Hence, we define

$$\varepsilon_r = \varepsilon(T_r) = \frac{m\,T_r}{n} - 1 \qquad \text{and} \qquad \varepsilon_{min} = \min_{r\in\mathcal{R}}\{\varepsilon_r\} \ .$$

Then, $(1+\varepsilon_{min})$ is the factor by which all the thresholds exceed the fractionally balanced assignment in which every resource receives n/m users. With this parameter the main result of this section reads as follows.

Theorem 2.1. *If $\alpha \leq \mathcal{O}(\varepsilon_{min})$ small enough, the system converges to a balanced state with $x_r \leq T_r$ for all $r \in \mathcal{R}$ in time*

$$\mathcal{O}\left(\frac{1}{\alpha\,\varepsilon_{min}} \cdot \log(n)\right) \ .$$

For example, if $T_r = (1+\varepsilon)(n/m)$ for all r and α and ε are constants, then the convergence time is only $O(\log n)$.

For the proof we monitor the potential change as long as the system is not balanced. In particular, we consider the change between two rounds. Throughout, we fix an unbalanced state x and omit the argument x whenever possible. We differentiate between overloaded and underloaded resources. The first lemma bounds the number of significantly underloaded resources. This later serves to bound the probability that a migrating user hits a "hole" and decreases the potential. We denote by $k(x) = |\{r : x_r < T_r - \Phi(x)/m\}|$ the number of significantly underloaded resources.

Lemma 2.2. *If $\Phi > 0$, it holds that $k \geq \Omega(\varepsilon_{min}\, m)$.*

Proof. We denote by $T_{min} = \min_{r\in\mathcal{R}}\{T_r\}$ the smallest threshold and $k_{min} = |\{r : x_r < T_{min} - \Phi/m\}|$. Note that by definition we have $k \geq k_{min}$ and $n \geq (m - k_{min}) \cdot (T_{min} - \Phi/m) + \Phi$. Solving for k_{min} yields

$$k_{min} \geq \frac{m^2\,T_{min} - n\,m}{T_{min}\,m - \Phi} \ .$$

Clearly, this expression is strictly increasing in Φ, so we may substitute $\Phi \geq 0$ to obtain

$$k_{min} \geq \frac{m^2\,T_{min} - n\,m}{T_{min}\,m} = \frac{m\,T_{min} - n}{T_{min}} \ .$$

Now, by choice of T_{min},

$$k_{min} \geq \frac{(1+\varepsilon_{min})\,n - n}{(1+\varepsilon_{min}) \cdot (n/m)} = m \cdot \frac{\varepsilon_{min}}{1+\varepsilon_{min}} \geq \frac{\varepsilon_{min}}{2}\,m \ ,$$

if $\varepsilon_{min} \leq 1$. □

The next lemma shows that on each of these resources there is a significant expected decrease of the potential function.

Lemma 2.3. *For any $r \in \mathcal{R}$ with $x_r \leq T_r - \Phi/m$ it holds that $\mathbb{E}[\Delta\Phi_r] \leq -\Omega(\alpha\,\Phi/m)$.*

Proof. Let X_r be the random variable that denotes the number of users arriving at resource r. It holds that

$$\mathbb{E}[X_r] = \sum_{r' \in \mathcal{R}: x_{r'} > T_{r'}} x_{r'} \cdot \alpha \cdot \frac{x_{r'} - T_{r'}}{x_{r'}} \cdot \frac{1}{m} = \frac{\alpha \Phi}{m} .$$

Note that X_r counts the number of players migrating to resource r. It is a sum of independent Bernoulli variables that indicate whether a player wants to migrate to resource r or not. Thus, we can use Chernoff bounds to lower bound the probability that X_r is at least half its expectation by a constant. Note that if $X_r \geq \tau$ for $1 \leq \tau \leq T_r - x_r$, this causes a strict decrease $\Delta\Phi_r \leq -\tau$.

We now bound the decrease in $\Delta\,\Phi_r$. We have $T_r - x_r \geq \Phi/m$ by assumption and set $\tau = \alpha\,\Phi/2m$. Then $\mathbb{P}[X_r \geq \alpha\,\Phi/2m]$ is a constant, and by the integrality of X_r, so is $\mathbb{P}[X_r \geq \lceil \alpha\,\Phi/2m \rceil]$. Using our observations above it follows that $\mathbb{P}[\Delta\Phi_r \leq -\lceil \alpha\,\Phi/2m \rceil]$ is a constant as well. Note that $\alpha\,\Phi/2m = 0$ if and only if we are in a balanced state. Hence, we can bound $\mathbb{E}[\Delta\,\Phi_r]$ by

$$\mathbb{E}[\Delta\Phi_r] \leq -\left\lceil \frac{\alpha\,\Phi}{2\,m} \right\rceil \cdot \mathbb{P}\left[\Delta\Phi_r \leq -\left\lceil \frac{\alpha\,\Phi}{2\,m} \right\rceil\right] = -\Omega(\alpha\,\Phi/m) .$$

□

The decrease of Φ shown in the last lemma is leveled off when too many users migrate away from an overloaded resource. The following lemma bounds the expected impact of this effect.

Lemma 2.4. *If $\alpha \leq 1/(2\,\mathrm{e})$, then for any $r \in \mathcal{R}$ with $x_r > T_r$ it holds that $\mathbb{E}[\Delta\Phi_r] \leq \mathcal{O}(\alpha^2)$.*

Proof. Note that $\Delta\Phi_r$ is the number of users leaving resource r exceeding Φ_r. Since the number of users leaving resource r is binomially distributed, we have

$$\begin{aligned}
\mathbb{E}[\Delta\Phi_r] &= \sum_{l=1}^{x_r} l \cdot \binom{x_r}{\Phi_r + l} \cdot \left(\alpha \cdot \frac{\Phi_r}{x_r}\right)^{\Phi_r + l} \cdot \left(1 - \alpha \cdot \frac{\Phi_r}{x_r}\right)^{x_r - (\Phi_r + l)} \\
&\leq \sum_{l=1}^{x_r} l \cdot \left(\frac{\mathrm{e}\,x_r}{\Phi_r + l} \cdot \alpha \cdot \frac{\Phi_r}{x_r}\right)^{\Phi_r + l} \\
&= \sum_{l=1}^{x_r} l \cdot \left(\frac{\mathrm{e}\,\alpha\,\Phi_r}{\Phi_r + l}\right)^{\Phi_r + l} \leq \sum_{l=1}^{x_r} l \cdot (\mathrm{e}\,\alpha)^{\Phi_r + l} \\
&\leq \sum_{l=1}^{\infty} l \cdot (\mathrm{e}\,\alpha)^{\Phi_r + l} = (\mathrm{e}\,\alpha)^{\Phi_r} \sum_{l=1}^{\infty} l \cdot (\mathrm{e}\,\alpha)^{l} \\
&= (\mathrm{e}\,\alpha)^{\Phi_r} \frac{\mathrm{e}\,\alpha}{(\mathrm{e}\,\alpha - 1)^2} = (\mathrm{e}\,\alpha)^{\Phi_r + 1} \frac{1}{(\mathrm{e}\,\alpha - 1)^2} \\
&\leq 4\,(\mathrm{e}\,\alpha)^{\Phi_r + 1} \leq 4\,\mathrm{e}^2\,\alpha^2 \\
&\leq 30\,\alpha^2 .
\end{aligned}$$

□

Assembling these insights, the next lemma bounds the expected decrease of Φ in one round of execution.

Lemma 2.5. *If $\alpha = \mathcal{O}(\varepsilon_{min})$ small enough, then $\mathbb{E}[\Delta\Phi] \leq -\Omega(\alpha\,\varepsilon_{min}\,\Phi)$.*

Proof. Summing up over all resources,

$$\begin{aligned}
\mathbb{E}[\Delta\Phi] &\leq \sum_{r:x_r<T_r-\Phi/m} \mathbb{E}[\Delta\Phi_r] + \sum_{r:T_r-\Phi/m\leq x_r\leq T_r} \mathbb{E}[\Delta\Phi_r] + \sum_{r:x_r>T_r} \mathbb{E}[\Delta\Phi_r] \\
&\leq -k(x)\cdot\Omega\left(\frac{\alpha\,\Phi}{m}\right) + 0 + |\{r : x_r > T_r\}|\cdot 30\,\alpha^2 \\
&\leq -\frac{\varepsilon_{min}\,m}{2}\cdot\Omega\left(\frac{\alpha\,\Phi}{m}\right) + |\{r : x_r > T_r\}|\cdot 30\,\alpha^2 \\
&\leq -\Omega(\alpha\,\varepsilon_{min}\,\Phi) + \Phi\cdot 30\,\alpha^2 \\
&\leq -\Omega(\alpha\,\varepsilon_{min}\,\Phi)\ ,
\end{aligned}$$

if $\alpha \leq c\,\varepsilon_{min}$ for some constant c small enough. □

Proof. (of Theorem 2.1) Lemma 2.5 reveals that in expectation the potential decreases by a fraction of $O(\alpha\varepsilon_{min})$ in every round. For the proof of Theorem 2.1 we have to bound the expected time until the potential drops to 0. Such multiplicative drift results are somewhat folklore in randomized algorithms since the early work of Hajek [19]. For a recent and elegant proof of the following result see, e.g., [7].

Lemma 2.6. *Let Ω be a finite set of states, $\Omega^* \subseteq \Omega$ a set of desired states, and $\Phi : \Omega \to \mathbb{N}$ a function that satisfies $\Phi(x) = 0$ if and only if $x \in \Omega^*$. Consider a sequence of random variables $X_0, X_1, \ldots$ that take values in Ω. Denote by $T^* = \min\{t \geq 0 \mid X_t \in \Omega^*\}$ the time that a state from Ω^* is reached for the first time. If*

$$\mathbb{E}[\Phi(X_{t+1}) \mid X_t = x] \leq (1-\delta)\Phi(x)$$

for all $x \in \Omega \setminus \Omega^*$ and some constant $0 < \delta < 1$, then

$$\mathbb{E}[T^*] \leq \frac{1+\ln\Phi_{\max}}{\delta}\ ,$$

where $\Phi_{\max} = \max_{x\in\Omega}\Phi(x)$. Also, for any $c > 0$ we have that $\mathbb{P}[T^* > (\ln\Phi_{\max} + c\ln n)/\delta] \leq n^{-c}$.

We can apply this lemma using $\delta \in O(\alpha\varepsilon_{min})$ and $\Phi_{\max} \leq n$, and the theorem follows. In particular, we have the same asymptotic time bound to achieve convergence in expectation and with high probability. □

2.2 Feasible Thresholds

In contrast to the previous results this section treats instances, in which we do not require that thresholds are above average $T_r > n/m$. Instead, we say that a set $(T_r)_{r\in\mathcal{R}}$ of thresholds is *feasible* if $\sum_{r\in\mathcal{R}} T_r \geq n$. The following Theorem 2.7 is the main result in this section. It bounds the convergence for *every* set of feasible thresholds.

Theorem 2.7. *If $\alpha = \mathcal{O}(1/m)$ small enough, then the system converges to a balanced state with $x_r \leq T_r$ for all $r \in \mathcal{R}$ in time*

$$\mathcal{O}\left(\frac{m}{\alpha}\log(n)\right)\ .$$

The proof is very similar to the proof of Theorem 2.1. We again fix a state x and show that the expected potential decrease is lower bounded by a fraction of α/m. The result then follows from known facts.

In our presentation we drop the argument x whenever possible. Due to the diversity of thresholds there is no similar lower bound on the number of significantly underloaded resources as in Lemma 2.2. However, we can derive a similar bound on the expected potential decrease on each underloaded resource.

Lemma 2.8. *For any $r \in \mathcal{R}$ with $x_r \leq T - \Phi/m$ it holds that $\mathbb{E}[\Delta\Phi_r] \leq -\Omega(\alpha\,\Phi/m)$.*

Proof. Again, let X_r be the random variable that denotes the number of users arriving on r. This time,

$$\mathbb{E}[X_r] = \sum_{r' \in \mathcal{R}: x_{r'} > T} x_{r'} \cdot \alpha \cdot \frac{x_{r'} - T_{r'}}{x_{r'}} \cdot \frac{1}{m} = \frac{\alpha \Phi}{m} \ .$$

The rest of the proof follows from concentration of measure arguments given in the proof of Lemma 2.3. □

For the potential increase introduced at overloaded resources a generalized version of Lemma 2.4 holds.

Lemma 2.9. *If $\alpha \leq 1/(2\,\mathrm{e})$, then for any $r \in \mathcal{R}$ with $x_r > T_r$ it holds that $\mathbb{E}[\Delta\Phi_r] \leq \mathcal{O}(\alpha^2)$.*

The proof is identical to the one given for Lemma 2.4. It follows from the fact that the migration probability is bounded by $\alpha\,\Phi_r/x_r = \alpha \cdot (x_r - T_r)/x_r$.

Finally, we assemble a bound on the expected potential decrease. However, as there is no bound on the number of underloaded resources in terms of ε_r, the damping factor α is characterized by the number m of resources. As a consequence, the proof of Theorem 2.7 follows for *every* set of feasible thresholds.

Lemma 2.10. *If $\alpha = \mathcal{O}(1/m)$ small enough, then $\mathbb{E}[\Delta\Phi] \leq -\Omega(\alpha \cdot \Phi/m)$.*

Proof. Summing over all resources we obtain

$$\begin{aligned}
\mathbb{E}[\Delta\Phi] &\leq \sum_{r: x_r < T_r - \Phi/m} \mathbb{E}[\Delta\Phi_r] + \sum_{r: T_r - \Phi/m \leq x_r \leq T_r} \mathbb{E}[\Delta\Phi_r] + \sum_{r: x_r > T_r} \mathbb{E}[\Delta\Phi_r] \\
&\leq -1 \cdot \Omega\left(\frac{\alpha\,\Phi}{m}\right) + 0 + |\{r : x_r > T\}| \cdot 30\,\alpha^2 \\
&\leq -\Omega\left(\alpha \cdot \frac{\Phi}{m}\right) + \Phi \cdot 30\,\alpha^2 \\
&\leq -\Omega\left(\alpha \cdot \frac{\Phi}{m}\right) \ .
\end{aligned}$$

□

3 User-Specific Instances

When thresholds are user-specific, it is easy to construct an instance where the algorithm presented in the preceding section requires a convergence time that is polynomial in n. We can construct a situation in which there is only one unsatisfied user assigned to a particular resource which holds k users in total. In the user-independent setting, if one user is unsatisfied, so are all the others on its resource. Consequently, Algorithm 1 will remove each of the k users with a probability of at least α/k, so in expectation at least α in total. In the user-specific setting, however, if only one user is unsatisfied and its probability to move is only $1/k$, it will take expected time $\Omega(k)$ to leave the resource.

Rapid convergence cannot be achieved without knowledge about the thresholds of users sharing the same resource: Increasing the migration probability for unsatisfied users is possible only if we are sure that the others are satisfied, otherwise we risk increasing the potential by overshooting effects. Thus, we move control to the resources, which concurrently run the following algorithm. First, it sorts the users on resource r in decreasing order of threshold. Then it considers the set of users whose level *within resource r* exceeds their threshold. Note that this will be a contiguous section at the top of the ordering since their level within the resource is increasing and their threshold is decreasing. This set of users is removed from the resource and each of them is reassigned to a resource chosen uniformly at random. The algorithm is given in pseudocode as Algorithm 2.

Algorithm 2 Kick-The-Complainers

for all resources r in parallel **do**
  Sort users on resource r in decreasing order of T_r^i.
  Let $U_r = \{i : T_r^i > i\}$ denote the set of excess users.
  Remove U_r from r and reassign them uniformly at random.
end for

We first concentrate on resource-uniform instances. It is straightforward to observe that for an arbitrary feasible set of thresholds T^i, which allows a balanced assignment, Algorithm 2 cannot guarantee convergence[1]. Hence, throughout the section we assume that all thresholds are *above average*, i.e. that $T^i > n/m$ for all $i \in \mathcal{N}$. In addition, we define $\varepsilon_i = \varepsilon(T^i) = \frac{m\,T^i}{n} - 1$ as above, and let $\varepsilon_{\min} = \min_i\{\varepsilon_i\}$. For our convergence proof, we use the number excess users U_r as defined in Algorithm 2 as a potential. Let $n_u = \sum |U_r|$.

Lemma 3.1. *After removing U_r from r (but before reassigning any users) all remaining users are satisfied.*

Proof. Consider the topmost user after removing U_r. Its level now equals the load x_r of the resource and the user was not contained in U_r. So $T_i \geq x_r$, and the user is satisfied. By our ordering the remaining users have an even larger threshold, so they are also satisfied. □

Theorem 3.2. *The system using Algorithm 2 converges to a stable state in time*

$$\mathcal{O}\left(\frac{1}{\varepsilon_{\min}}\log(n)\right) \ .$$

Proof. Lemma 3.1 shows that after removing the sets U_r of users from their resources, the potential is temporarily reduced to 0. We now reassign n_u users to resources sequentially. Consider a user i assigned to resource r. There are two cases:

1. When i is assigned to r with $x_r < T_{\min}$. Then, after adding user i, we still have $|U_r| = 0$.
2. For any other resource r, the user may contribute to U_r in the next round or may displace other users in the ordering. In any case, the additional contribution to U_r in the subsequent round caused by this insertion can be no more than 1.

Users falling into case 2 do not change the potential, they contribute in the preceding and in the subsequent round. Users falling into case 1 reduce the potential n_u by 1.

As in the user-independent setting, the fraction of resources with load at most $T_{\min} - n_u/m$ is again at least $\Omega(\varepsilon_{\min} m)$ (Lemma 2.2), the probability for one of the n_u reassigned users to fall into case 1 is at least $\varepsilon_{\min} \cdot m/m$, and the expected potential decrease is $-\Omega(\varepsilon_{\min} n_u)$. The theorem follows. □

Note that we can bound the time independently for each single user when it stops being reassigned. Thus, more tolerant users "stabilize" at an earlier point in time.

Corollary 3.3. *Fix $\tilde{\varepsilon} > 0$ and consider the set of users with $\varepsilon_i \geq \tilde{\varepsilon}$. Let τ denote the round at which any of these users is moved for the last time. Then,*

$$\tau = \mathcal{O}\left(\frac{1}{\tilde{\varepsilon}}\log(n)\right) \ .$$

[1] Consider two resources r_1 and r_2 and three users i_1, i_2, i_3. Two users have $T^{i_1} = T^{i_2} = 2$, one user has $T^{i_3} = 1$. If i_1 and i_2 are on different resources, the algorithm will reassign i_3 infinitely. If, however, i_1 and i_2 are both located on one resource, a balanced state evolves.

Proof. The decision about the migration of a user with $\varepsilon_i \geq \tilde{\varepsilon}$ depends only on the inclusion in U_r. Whether or not a user with $\varepsilon_i \geq \tilde{\varepsilon}$ belongs to U_r is independent of the users with $\varepsilon_i < \tilde{\varepsilon}$. Thus, to bound the convergence time for user i we can disregard all users with smaller ε_i from the system. □

Finally, note that Theorem 3.2 can be generalized to hold for the case of resource-specific thresholds that are above average with $T_r^i > n/m$, for all $i \in \mathcal{N}$ and $r \in \mathcal{R}$. In this case we define $\varepsilon_{min} = \min_{i\in\mathcal{N}, r\in\mathcal{R}}(mT_r^i/n) - 1$.

Corollary 3.4. *For user-specific and resource-specific instances the system using Algorithm 2 converges to a stable state in time*

$$\mathcal{O}\left(\frac{1}{\varepsilon_{\min}}\log(n)\right) \ .$$

4 Unknown Number of Users

In previous sections we were mainly concerned with a scenario, in which thresholds are intrinsic to users. In this section we consider the problem of explicitly designing an acceptance threshold of the users to provide fast convergence. Clearly, our design can be guided by the insights from the previous sections, which yields fast convergence for T sufficiently large. However, such a choice requires to know the total number of users in the system. In a distributed setting this value might not be readily available, but it can be estimated by performing an exponential search technique. While such an approach can be seen as a standard in the theory of algorithms, we describe it here (Algorithm 3) for completeness.

Note that in our algorithm we design thresholds to be uniform. One can certainly think of reasons as to why it might be profitable or required to design thresholds in a non-uniform (e.g., user- or resource-specific) way, but this issue is left for future work.

Algorithm 3 Exponential search

```
for all users i in parallel do
    set ñ ← 1
    set t ← 1
    repeat
        Execute Algorithm 1 for c · 1/(ε α) · log ñ rounds
        t ← t + 1
        ñ ← (1 + ε)^t
        T ← (1 + ε) ñ/m
    until indefinitely
end for
```

Theorem 4.1. *Let τ denote the time at which Algorithm 3 reaches a stable state for the first time and let T_τ denote the value of T at this point of time. Then,*

$$\mathbb{E}[\tau] = \mathcal{O}\left(\frac{1}{\epsilon^2\,\alpha}\log^2 n\right) \qquad \textit{and} \qquad \mathbb{E}[T_\tau] = \frac{n}{m}\cdot(1+\mathcal{O}(\epsilon)).$$

Proof. Denote by t the first round, in which we have $\tilde{n} \geq n$ and note that $t = \mathcal{O}\left(\log_{1+\epsilon} n\right) = \mathcal{O}(\log n/\epsilon)$. By Theorem 2.1 the expected convergence time of Algorithm 1 is then $\mathcal{O}\left(\frac{1}{\alpha\epsilon}\log n\right) = \mathcal{O}\left(\frac{1}{\alpha\epsilon}\log\tilde{n}\right)$. By Markov's inequality the probability to fail to converge after $c\cdot\frac{1}{\epsilon\,\alpha}\cdot\log\tilde{n}$ rounds can be at most a constant p if c is a constant chosen large enough. Since this also holds for any subsequent round,

$$\mathbb{E}[\tau] = t\cdot c\cdot\frac{1}{\epsilon\,\alpha}\cdot\log n + c\cdot\frac{1}{\epsilon\,\alpha}\cdot\sum_{i=1}^{\infty} p^i\cdot\log((1+\epsilon)^i n) \ .$$

Substituting the bound for t and using that the sum is bounded by $\mathcal{O}(\log n)$ yields the claimed bound on $\mathbb{E}[\tau]$. Similarly, for $\mathbb{E}[T_\tau]$ we have

$$\mathbb{E}[T_\tau] = T_t \cdot \sum_{0=1}^{\infty} p^i \cdot (1-p) \cdot (1+\epsilon)^i = T_t \cdot \frac{1-p}{1-p-p\epsilon},$$

which proves the bound on $\mathbb{E}[T_\tau]$. □

5 Conclusions

In this paper we have studied a load balancing model based on QoS demands by users. In our model users formulate minimum performance thresholds that must be satisfied by the system. We have presented distributed load balancing algorithms that guarantee rapid convergence to balanced states when executed locally by each user. The advantage of this approach is that the threshold formulation allows to significantly strengthen the locality constraint by using only information about the currently allocated resource.

Naturally, a variety of open problems remain. It is not clear whether the bounds presented in this paper are optimal. For instance, it might be the case that a different, more elaborate protocol achieves even doubly logarithmic convergence time. This was obtained, e.g., in [4] for a protocol that steers the migration of each user with information about several resources. Improved convergence times can only be obtained using concurrent or synchronized dynamics, as non-synchronized or sequential migration requires at least n rounds for convergence in the worst case. The main challenge in acceleration of convergence lies in avoiding oscillation effects. Furthermore, there are a variety of different aspects, especially on modeling various strategic incentives in the convergence process, which have not been addressed appropriately in existing work on distributed load balancing.

References

[1] Heiner Ackermann, Petra Berenbrink, Simon Fischer, and Martin Hoefer. Concurrent imitation dynamics in congestion games. In *Proc. 28th Symp. Principles of Distributed Computing (PODC)*, pages 63–72, 2009.

[2] Heiner Ackermann, Simon Fischer, and Martin Hoefer. Distributed algorithms for QoS load balancing. In *Proc. 21st Symp. Parallelism in Algorithms and Architectures (SPAA)*, pages 197–203, 2009.

[3] Baruch Awerbuch, Yossi Azar, and Rohit Khandekar. Fast load balancing via bounded best response. In *Proc. 19th Symp. Discrete Algorithms (SODA)*, pages 314–322, 2008.

[4] Petra Berenbrink, Tom Friedetzky, Leslie Ann Goldberg, Paul Goldberg, Zengjian Hu, and Russel Martin. Distributed selfish load balancing. *SIAM J. Comput.*, 37(4):1163–1181, 2007.

[5] Petra Berenbrink, Tom Friedetzky, Iman Hajirasouliha, and Zengjian Hu. Convergence to equilibria in distributed, selfish reallocation processes with weighted tasks. In *Proc. 15th European Symposium on Algorithms (ESA)*, pages 41–52, 2007.

[6] Petra Berenbrink, Martin Hoefer, and Thomas Sauerwald. Distributed selfish load balancing on networks. In *Proc. 22nd Symp. Discrete Algorithms (SODA)*, 2011. To appear.

[7] Benjamin Doerr and Leslie Goldberg. Drift analysis with tail bounds. In *Proc. 11th Intl. Conf. Parallel Problem Solving From Nature (PPSN)*, volume 1, pages 174–183, 2010.

[8] Robert Elsässer and Burkhard Monien. Load balancing of unit size tokens and expansion properties of graphs. In *Proc. 15th Symp. Parallelism in Algorithms and Architectures (SPAA)*, pages 266–273, 2003.

[9] Robert Elsässer, Burkhard Monien, and Stefan Schamberger. Distributing unit size workload packages in heterogeneous networks. *J. Graph Alg. Appl.*, 10(1):51–68, 2006.

[10] Robert Elsässer and Thomas Sauerwald. Discrete Load Balancing is (Almost) as Easy as Continuous Load Balancing. In *Proc. 29th Symp. Principles of Distributed Computing (PODC)*, pages 346–354, 2010.

[11] Eyal Even-Dar, Alexander Kesselman, and Yishay Mansour. Convergence time to Nash equilibria in load balancing. *ACM Trans. Algorithms*, 3(3), 2007.

[12] Eyal Even-Dar and Yishay Mansour. Fast convergence of selfish rerouting. In *Proc. 16th Symp. Discrete Algorithms (SODA)*, pages 772–781, 2005.

[13] Rainer Feldmann, Martin Gairing, Thomas Lücking, Burkhard Monien, and Manuel Rode. Nashification and the coordination ratio for a selfish routing game. In *Proc. 30th Intl. Coll. Automata, Languages and Programming (ICALP)*, pages 514–526, 2003.

[14] Simon Fischer. *Dynamic Selfish Routing.* PhD thesis, Lehrstuhl für Algorithmen und Komplexität, RWTH Aachen, 2007.

[15] Simon Fischer, Petri Mähönen, Marcel Schöngens, and Berthold Vöcking. Load balancing for dynamic spectrum assignment with local information for secondary users. In *Proc. Symp. Dynamic Spectrum Access Networks (DySPAN)*, 2008.

[16] Dimitris Fotakis, Alexis Kaporis, and Paul Spirakis. Atomic congestion games: Fast, myopic and concurrent. *Theory Comput. Syst.*, 47(1):38–49, 2010.

[17] Paul Goldberg. Bounds for the convergence rate of randomized local search in a multiplayer load-balancing game. In *Proc. 23rd Symp. Principles of Distributed Computing (PODC)*, pages 131–140, 2004.

[18] Piyush Gupta and P. R. Kumar. The capacity of wireless networks. *IEEE Trans. Inf. Theory*, 46(2):388–404, 2000.

[19] Bruce Hajek. Hitting-time and occupation-time bounds implied by drift analysis with applications. *Advances Appl. Prob.*, 13:502–525, 1982.

[20] Robert Kleinberg, Georgios Piliouras, and Éva Tardos. Load balancing without regret in the bulletin board model. In *Proc. 28th Symp. Principles of Distributed Computing (PODC)*, pages 56–62, 2009.

[21] Elias Koutsoupias and Christos Papadimitriou. Worst-case equilibria. *Computer Science Review*, 3(2):65–69, 2009.

[22] Marina Petrova, Natalia Olano, and Petri Mähönen. Balls and bins distributed load balancing algorithm for channel allocation. In *Proc. 7th Conf. Wireless On demand Network Systems and Services (WONS)*, 2010.

[23] Berthold Vöcking. Selfish load balancing. In Noam Nisan, Éva Tardos, Tim Roughgarden, and Vijay Vazirani, editors, *Algorithmic Game Theory*, chapter 20. Cambridge University Press, 2007.