A Simple Approach for Adapting Continuous Load Balancing Processes to Discrete Settings[*][†]

Hoda Akbari[‡] and Petra Berenbrink[§] and Thomas Sauerwald[¶]

Abstract

We consider the neighbourhood load balancing problem. Given a network of processors and an arbitrary distribution of tasks over the network, the goal is to balance load by exchanging tasks between neighbours.

In the *continuous model*, tasks can be arbitrarily divided and perfectly balanced state can always be reached. This is not possible in the *discrete model* where tasks are non-divisible.

In this paper we consider the problem in a very general setting, where the tasks can have arbitrary weights and the nodes can have different speeds. Given a continuous load balancing algorithm that balances the load perfectly in T rounds, we convert the algorithm into a discrete version. This new algorithm is deterministic and balances the load in T rounds so that the difference between the average and the maximum load is at most $2d \cdot w_{\max}$, where d is the maximum degree of the network and $w_{\max}$ is the maximum weight of any task. For general graphs, these bounds are asymptotically lower compared to the previous results.

The proposed conversion scheme can be applied to a wide class of continuous processes, including first and second order diffusion, dimension exchange, and random matching processes.

For the case of identical tasks, we present a randomized version of our algorithm that balances the load up to a discrepancy of $\mathcal{O}(\sqrt{d \log n})$ provided that the initial load on every node is large enough.

1 Introduction

In this paper we consider the problem of neighbourhood load balancing in arbitrary networks. The networks are modelled by a graph with n nodes. The nodes of the graph represent processors or other resources and the edges of the graph communication links between them. We assume that in the beginning m tasks are arbitrarily distributed over the nodes of the network. The tasks can have arbitrary weights; the weight of

[*]This paper is an extended version of [6].

[†]The second author's work was supported by an NSERC Discovery Grant "Analysis of Randomized Algorithms".

[‡]Simon Fraser University, Burnaby, Canada

[§]Simon Fraser University, Burnaby, Canada

[¶]University of Cambridge, Cambridge, UK task i is w_i. The weights can be used to model the runtime of tasks, or their resource requirements. We also allow the processors to have different speeds to model situations in which the processors in the network have different processing rates. The speed of processor i is denoted by s_i.

In this general setting the *load* of a resource is the total weight of the tasks assigned to it. We define the *makespan* of a resource as its load divided by the speed of the resource. The goal of neighbourhood load balancing is to minimize the makespan difference over all nodes. Note that in a perfectly balanced state the load of each resource is proportional to its speed. In other words, the ratio of load to speed (makespan) should be equal for all the nodes.

Neighbourhood load balancing algorithms usually work in synchronous rounds. In every round each node is allowed to compare its load with all or a subset of its neighbours. Then it calculates for every neighbour the amount of load and the subset of its tasks that it will send to it. One distinguishes between *discrete* and *continuous* settings of the problem. In the discrete case the tasks are atomic and cannot be broken into smaller tasks. Hence, nodes can only forward whole tasks to their neighbours. Continuous load balancing relaxes the problem by assuming that tasks can be split into arbitrarily small pieces, and hence load can be balanced perfectly. Most continuous balancing algorithms are well-understood, there exist tight bounds on the number of rounds they need to balance the load completely [34]. Discrete load balancing algorithms are quite often not able to balance the load perfectly, especially in the case of weighted tasks and resources with speeds. The common approach to analyze a discrete algorithm is to consider a closely related continuous version of the algorithm and to bound the deviation between the two processes. For an overview we refer the reader to Section 2.2 where we elaborate on discrete load balancing algorithms that use randomized or deterministic rounding techniques.

One also distinguishes between *diffusion* load balancing where every node balances its load with every neighbour in every round, and *dimension exchange* models where nodes balance only with one neighbour per round. In the latter case it is often assumed that a partial matching is chosen randomly (*random matching model*), or that a set of perfect matchings that cover every edge of the graph is given. These matchings are then used periodically (*periodic matching model*).

As an example for neighbourhood load balancing let us consider the so-called standard diffusion load balancing algorithm with unweighted tasks and uniform resources. Here all the nodes balance their load with all their neighbours. Both in the discrete and in the continuous case every node u will calculate the load difference f_v between its own load and the load of each neighbours v. If the difference is positive, u will divide f_v by its own degree d. In the continuous case u will send f_v/d tokens to v. Since f_v/d is not necessarily an integer this might not be possible in the discrete case. Hence u has to round f_v/d to one of the nearest integers. It is easy to see that this rounding may lead to a load difference that is a function of the diameter and the degree of the network.

Neighbourhood load balancing strategies have many advantages compared to strategies where nodes are allowed to balance their load by routing it toward any node in the network. First, they do not need any global knowledge since the load balancing actions are determined by the load of direct neighbours. This keeps the balancing algorithm simple and scalable. Neighbourhood balancing algorithms also have the tendency to keep the tasks close to their initial location which is beneficial if the tasks originated on the same resource have to exchange information.

1.1 New Results

We present two results in this paper. In Section 4 we present a general framework (Algorithm 1) that translates a continuous load balancing process into a discrete version. In every round the discrete algorithm *imitates* the continuous algorithm as closely as possible by trying to send the same amount of load over every edge as the continuous algorithm. To be more detailed, let $f_e^c(t)$ and $f_e^d(t)$, respectively, be the total load sent over edge e during the first t rounds of the continuous process and its discrete version. Then the discrete version of the algorithm tries to send a load of $\lfloor f_e^c(t) - f_e^d(t-1) \rfloor$[1] over e in round t. Note that it might not be possible for the discrete algorithm to send the required amount of load over all the edges since the load of the nodes might not be sufficiently large. In that case the node will create some *dummy tokens* to fulfil all demands. This is done by assuming that each node has an *infinite source* of dummy tokens to be generated on demand. The dummy tokens will be regarded as "normal" tokens for the rest of the balancing process. At the end of the balancing process they will be eliminated, which reduces the makespan of the resources who were holding some of the dummy tokens at the end of the process. See Algorithm 1 for details. Note that in actual implementation, we do not need to create and transfer workload units and consume communication bandwidth for each dummy token. That would incur communication overhead proportional to the number of dummy tokens. Instead, we can communicate only the amount of dummy workload that should be forwarded to each neighbour.

Let d be the maximum degree and $w_{\max}$ the maximum task weight. Let the *max-min discrepancy* be defined as the difference between the maximum and minimum makespan. The *max-avg discrepancy* is defined as the difference between the maximum and the average makespan. Furthermore, let T be the time it takes for the continuous process to balance the load (more or less) completely (see Section 3 for details). Then we show that the following holds for the discrete version of the continuous process, using Algorithm 1 (see Theorem 3 for details).

(1) At time T the max-avg discrepancy is $\mathcal{O}(d \cdot w_{\max})$.

(2) If the initial makespan of any node is at least $d \cdot w_{\max}$, then the above bound holds on the final max-min discrepancy as well.

Our general framework can be applied to a wide and natural class of continuous algorithms that we called *additive terminating* algorithms (see Definition 2 and Definition 3).

An additive algorithm, starting with a load distribution $D = D_1 + D_2$, transmits the same amount of tasks over every edge as the sum of the amounts it would transmit in

[1]The discrete version of the algorithm has to know the continuous flow $f_e^c(t)$ for every edge $e = (u, v)$. This knowledge is easy to gather by simulating the continuous load balancing process in parallel on every node.

the case that it were started with D_1 and D_2. A terminating algorithm does not do any load balancing actions on a completely balanced load distribution. The class of supported algorithms includes first and second order diffusion algorithms [34], dimension exchange algorithms (where the nodes balance their load only with one neighbour chosen deterministically per round), and random matchings models [27] (where in each round the balancing actions are restricted to the edges of a random matching).

While being very simple, our work is the first to analyze a discretization scheme supporting such a wide class of continuous algorithms. The analysis holds for arbitrary graphs, weighted tasks and resources with speed, whereas most existing papers consider only discrete algorithms in the uniform task model. Also, except a few publications [2, 21] the majority of the previous results assume uniform resource speeds (see Section 2.2 for more details).

Tables 1 and 2 in the appendix compare our algorithms with previous results. For easier comparison, our results are translated to the case of uniform tasks and speeds. Table 1 compares our algorithms with other diffusion algorithms. Algorithm 1 achieves a final max-min discrepancy independent of n and graph expansion, and in particular, the only algorithm achieving constant max-min discrepancy for all constant-degree graphs. In the matching model (Table 2), Algorithm 1 is the only algorithm that achieves final max-min discrepancy independent of n for an arbitrary, possibly non-regular graph.

Our second result is a transformation based on randomized rounding (see Algorithm 2 in Section 5). Again, let $f_e^c(t)$ and $f_e^d(t)$, respectively, be the total load sent over edge e during the first t rounds of the continuous process and its discrete version. Then, according to Algorithm 2, the discrete version of the algorithm sends either $\lfloor f_e^c(t) - f_e^d(t-1) \rfloor$ or $\lceil f_e^c(t) - f_e^d(t-1) \rceil$ tasks over edge e in round t.

Algorithm 2 reaches a max-avg discrepancy of

$$\frac{d}{4} + \mathcal{O}(\sqrt{d \log n})$$

after T steps. (Theorem 8). If the initial makespan of every node is at least $d/4 + \mathcal{O}(\sqrt{d \log n})$, then the result improves to a max-min discrepancy of $\mathcal{O}(\sqrt{d \log n})$. For large values of d these bounds improve the results of the deterministic transformation presented above.

Algorithm 2 improves over diffusion algorithms of [9, 26, 37] by reaching a max-min discrepancy independent of graph expansion for arbitrary graphs (see Table 2). Similarly, in the matching model, Algorithm 2 achieves max-min discrepancy bounds independent of graph expansion, thus giving improved bounds compared to [24, 37, 38] for low-expansion graphs.

In comparison to the existing results on non-uniform speeds [2, 21], both Algorithm 1 and Algorithm 2 achieve max-min discrepancy bounds independent of global graph parameters while previous bounds depend on the expansion [2, 21] or the diameter [2].

It should be noted that in both our deterministic and randomized algorithms, the max-min discrepancy results hold under the condition that the initial load of each node is sufficient (see part (2) of Theorems 3 and 8). Otherwise, we only provide bounds on max-avg discrepancy. Though such conditions were not necessary in previous works, we do not consider this as a major drawback. The reason is in the load balancing context usually what matters is the maximum completion time of all the jobs. In other words, the main concern is to minimize the maximum load rather than minimizing the difference between the maximum and the minimum load. When this is the case, max-avg discrepancy bounds serve equally as well as max-min discrepancy bounds. On the other hand, when resource utilization is the concern max-min discrepancy bounds provide better guarantees.

2 Existing Algorithms and Techniques

There is a vast amount of literature about load balancing. In this section, we give an overview of the results on continuous (Section 2.1) and discrete neighbourhood load balancing (Section 2.2) only. There are many related models such as selfish load balancing [1, 3, 10, 11, 22, 23], deterministic random walks [13, 14, 16, 25, 32], periodic balancing circuits [7, 37], and token distribution [4, 28, 33]. We will not consider these models here any further.

When not stated otherwise, the results are for the uniform case without speeds and weights. In the following we will consider the results both in the discrete and the continuous settings.

2.1 Continuous Load Balancing

The first diffusion algorithm (also called *first order schedule*, FOS) was independently introduced by Cybenko [15] and Boillat [12]. Their results were later generalized to the case of non-uniform speeds in [20]. To introduce the FOS process we first need some additional notation. Let $x_i(t)$ be the load of resource i at the beginning of round $t \geqslant 0$ of the process. We define $x(t) = (x_1(t), \ldots, x_n(t))$ as the *load vector* in the beginning of round t. For an arbitrary node i let $N(i)$ be the set of neighbours of i. Furthermore, let d_i be the degree of node i and recall that s_i is the speed of resource i. For $j \in N(i)$ let $y_{i,j}(t)$ be the (positive) amount of load transferred from node i to node j in round t. For $j \notin N(i)$, we let $y_{i,j}(t)$ to be zero. Then the FOS process is defined as follows.

$$y_{i,j}(t) = \frac{\alpha_{i,j}}{s_i} \cdot x_i(t), \tag{1}$$

$$x_i(t+1) = x_i(t) - \sum_{j \in N(i)} \alpha_{i,j} \left(\frac{x_i(t)}{s_i} - \frac{x_j(t)}{s_j} \right), \tag{2}$$

where $\alpha_{i,j} = \alpha_{j,i}$ are parameters to be chosen with the restriction that for all i, we must have $\sum_{j \in N(i)} \alpha_{i,j} < s_i$. Common choices for $\alpha_{i,j}$ are $1/(2\max(d_i, d_j))$ or $1/(\max(d_i, d_j)+1)$. The process can also be defined using a so-called *diffusion matrix* P, where for all i we have $P_{i,i} = 1 - \sum_j \alpha_{i,j}/s_i$ and for $j \in N(i)$, we have $P_{i,j} = \alpha_{i,j}/s_i$. Other entries of P are set to zero. Then

$$x(t+1) = x(t) \cdot P. \tag{3}$$

P is a stochastic matrix that can be viewed as the transition matrix of an ergodic Markov chain with a unique steady-state distribution $(s_1/S, \ldots, s_i/S, \ldots, s_n/S)$. Hence, repeatedly applying Equation (3) leads to the perfectly balanced state. Let K denote the initial discrepancy and λ the second-largest eigenvalue in absolute value of the diffusion matrix. Here and throughout, we use the variable T to denote the balancing time of various continuous processes. When not clear from the context, we redefine T explicitly. Then [20, 34, 37] use the above approach to show that

$$T = \mathcal{O}\left(\frac{\log(Kn)}{1-\lambda}\right),$$

where the result of [20] is for the case of non-uniform speeds.

Muthukrishnan et al. [34] introduced the *second order schedule* (SOS). Later, the SOS was generalized by Elsässer et al. [20] to the case of non-uniform speeds. The SOS method is inspired by a numerical iterative method called successive over-relaxation. In SOS, the amount of load transmitted over each edge depends on the current state of the network as well as the load transferred in the previous round. The first round equation is similar to FOS Equations (1) and (2), and subsequent rounds are defined by

$$y_{i,j}(t) = (\beta - 1)\cdot y_{i,j}(t-1) + \beta \cdot \frac{\alpha_{i,j}}{s_i} \cdot x_i(t), \tag{4}$$

where $\alpha_{i,j}$'s are as in Equation (1) and $0 < \beta \leqslant 2$. This leads to the following round equation with diffusion matrix P defined as in FOS[2]:

$$x(t+1) = \beta \cdot x(t) \cdot P + (1-\beta)\cdot x(t-1).$$

For some choices of β SOS converges faster than FOS [34]. The optimal choice for β is known to be $2/(1+\sqrt{1-\lambda^2})$ [20, 34], which leads to

$$T = \mathcal{O}\left(\frac{\log(Kn)}{\sqrt{1-\lambda}}\right),$$

[2]Derivation steps can be found below (see also [34, Lemma 3]). We use $[n]$ to denote $\{1,\dots,n\}$:

$$\begin{aligned}
x_i(t+1) &= x_i(t) + \sum_{j\in[n]} (y_{j,i}(t) - y_{i,j}(t)) \\
&= (1-\beta+\beta)\cdot x_i(t) + \sum_{j\in[n]} \big((\beta-1)\cdot y_{j,i}(t-1) + \beta\cdot P_{j,i}\cdot x_j(t)\big) \\
&\quad - \sum_{j\in[n]} \big((\beta-1)\cdot y_{i,j}(t-1) + \beta\cdot P_{i,j}\cdot x_i(t)\big) \\
&= (1-\beta)\cdot\left(x_i(t) - \sum_{j\in[n]} (y_{j,i}(t-1) - y_{i,j}(t-1))\right) + \beta\cdot x_i(t) - \beta\cdot\sum_{j\in[n]} x_i(t)\cdot P_{i,j} \\
&\quad + \beta\cdot\sum_{j\in[n]} x_j(t)\cdot P_{j,i} \\
&= (1-\beta)\cdot x_i(t-1) + \beta\cdot x_i(t) - \beta\cdot x_i(t)\cdot\sum_{j\in[n]} P_{i,j} + \beta\cdot\sum_{j\in[n]} x_j(t)\cdot P_{j,i} \\
&= (1-\beta)\cdot x_i(t-1) + \beta\cdot\sum_{j\in[n]} x_j(t)\cdot P_{i,j}
\end{aligned}$$

where the last equation holds since we have $\sum_{j\in[n]} P_{i,j} = 1$. Translating to the matrix form, this yields the desired round equation.

where the result of [20] is for the case of non-uniform speeds.

For SOS it can happen that the total outgoing demand $\sum_{j\in N(i)} y_{i,j}$ exceeds the load x_i, which results in so-called *negative load*.

The *dimension exchange* model is motivated by single-port architectures as opposed to the diffusion model which necessitates multi-port communication [15, 27, 28, 37]. In the matching based models every node balances its load with only one neighbour. More formally, the load transfer in each round is restricted to a – not necessarily perfect – matching of the underlying graph. Let (i, j) be an edge in the matching of round t. Then resource i and resource j calculate $y_{i,j}$ and $y_{j,i}$ such that their makespans are equalized. This can be done by the following equations, which can be regarded as a special case of the Equations (1) and (2)).

$$y_{i,j}(t) = \frac{\alpha_{i,j}}{s_i} \cdot x_i(t), \qquad (5)$$
$$x_i(t+1) = \frac{s_i}{s_i+s_j} \cdot \left(x_i(t) + x_j(t)\right),$$

where $\alpha_{i,j} = \alpha_{j,i} = s_i s_j/(s_i + s_j)$.

Similar to the diffusion load balancing, the process can be defined using a sequence of matrices $\{P(0), P(1), \ldots\}$, where $P(t)$ represents a modification of the adjacency matrix of the matching that is used in step t, as follows. If (i, j) is in the matching of the round t then $P_{i,j}(t) = \alpha_{i,j}/s_i$ and $P_{i,i}(t) = 1 - P_{i,j}(t)$. If i is not matched then $P_{i,i}(t) = 1$. All other entries are zero.

Several publications assume that a fixed set of matchings (usually roughly maximum degree many) is given and the matchings are used periodically. Hence, for all t we have $P(t) = P(t \bmod \tilde{d})$ where $\tilde{d}$ is the length of the period. The model was originally introduced in [30], together with a distributed edge-colouring algorithm (see also [35, 36]) that can be used to construct the matchings. As far as we know, the algorithms in the matching model have been analyzed only for the case of uniform tasks and speeds. The analysis for the algorithms using periodic matchings is very similar to the analysis of FOS. The convergence was first analyzed in [15] for hypercubes and in [31] for arbitrary graphs. Define

$$\mathscr{P} := P(0) \cdot P(1) \cdot \ldots \cdot P(\tilde{d} - 1)$$

and let λ be the second-largest eigenvalue in absolute value of the matrix $\mathscr{P}$ [3]. If we consider a group of $\tilde{d}$ consecutive rounds together, we have $x((t+1)\cdot\tilde{d}) = x(t\cdot\tilde{d})\cdot\mathscr{P}$, which is similar to the Equation (3) of the first order diffusion. Consequently, we have [37]

$$T = \mathscr{O}\left(\frac{\tilde{d}\cdot\log(Kn)}{1-\lambda}\right).$$

Another approach is to use in every step t a randomly generated matching. In [27],

[3]For the case where the matrix $\mathscr{P}$ is not symmetric, we need to define λ as the second-largest eigenvalue in absolute value of the matrix $\mathscr{P}\cdot\mathscr{P}^T$ (see the discussion in [37] for more details).

it is shown that w.h.p.[4]

$$T = \mathcal{O}\left(\frac{d \cdot \log(Kn)}{\gamma}\right),$$

where $\gamma \leqslant d$ is the second-smallest eigenvalue of the Laplacian matrix of the original graph G.

2.2 Discrete Load Balancing

As far as we know, existing papers consider only discrete algorithms in the uniform task model. Many publications consider uniform processors [8, 9, 18, 19, 24, 26, 27, 34, 37, 38], while a few others incorporate processor speeds into the model [2, 21]. There are two main approaches for analyzing discrete neighbourhood load balancing processes. The first one is to use potential functions and the second one is to compare the performance of a discrete process with that of a continuous version of that process.

The first approach is used in [27] for the random matching model and in [34] for the diffusion model. In both papers the discrete process calculates the amount of load to be transmitted over every edge as in the continuous process (as in Equations (1) and (2)), which is then rounded down. The potential function used in [27, 34] is defined as follows,

$$\Phi(t) := \sum_{i \in V} \left(x_i(t) - \frac{m}{n}\right)^2 \tag{6}$$

It is shown in [34] that in every round of the continuous FOS, $\Phi(t)$ drops at least by a factor of λ^2. Furthermore, in the discrete process, for any choice of parameter $\varepsilon < 1$, if $\Phi(t) \geqslant 16d^2n^2/\varepsilon^2$ then $\Phi(t+1) \leqslant (1+\varepsilon) \cdot \lambda^2 \cdot \Phi(t)$. Thus, the discrete process for most part behaves similarly to the continuous process as long as the potential is large enough. More precisely, the authors of [34] show that the potential is reduced to

$$\mathcal{O}\left(\frac{d^2n^2}{\varepsilon^2}\right) \text{ within } \mathcal{O}\left(\frac{\log \Phi(0)}{1-(1+\varepsilon)\lambda^2}\right) \text{ rounds.} \tag{7}$$

For FOS schemes, [34] left it as an open question to analyze the potential drop when the potential is smaller than $\mathcal{O}(d^2n^2)$. Ghosh and Muthukrishnan [27] consider the random matching model with the same potential function as defined in Equation (6). They show that as long as $\Phi(t) = \Omega(dn)$, the potential drops by at least a multiplicative factor of $\Omega(\gamma/d)$ per round, where γ is the second smallest eigenvalue of the Laplacian of the graph. For a smaller potential they were still able to show an additive drop of $\Omega(1/d)$ per round. Using this fact, they prove that the max-min discrepancy is reduced to

$$\mathcal{O}(\operatorname{diam}(G)) \text{ within } \mathcal{O}\left(\frac{d \log \Phi(0) + d^2 n}{\gamma}\right) \text{ rounds,}$$

w.h.p. Note that analyzing algorithms in the matching model is easier since in this model the potential never increases.

[4]We say an event occurs *with high probability (w.h.p)* if its probability is at least $1 - \mathcal{O}(n^{-\alpha})$ for some constant $\alpha > 0$.

The discrete SOS process was first analyzed in [18]. The authors of that paper measure the distance to the balanced state by the second norm of the difference between the load vector and the balanced vector. This measure is equal to $\sqrt{\Phi}$, where Φ is defined in Equation (6). The authors show that in sufficiently many rounds $\sqrt{\Phi(t)}$ is reduced to $\mathcal{O}((d \cdot \sqrt{n})/(1-\lambda))$.

The above results for FOS and SOS are generalized in [21] to the case of non-uniform speeds. Note that for non-uniform speeds, the balanced allocation is $(W/S) \cdot (s_1, \ldots, s_n)$, where $W := w_1 + \ldots + w_m$ is the total task weight and $S := s_1 + \ldots + s_n$ is the total speed of the processors. Hence, the optimal load of processor i is $s_i \cdot W/S$, and the potential function defined in Equation (6) becomes

$$\Phi(t) := \sum_{i \in V} (x_i(t) - s_i \cdot W/S)^2 .$$

In [21] the authors measure the distance to the balanced state by the second norm of the difference between the load vector and the balanced load vector $(s_1, \ldots, s_n)W/S$. They show that after a sufficient number of rounds t, $\sqrt{\Phi(t)}$ is reduced to $\mathcal{O}((d \cdot \sqrt{n \cdot s_{\max}})(/1-\lambda))$ in an FOS or SOS process.

The second analysis technique was introduced by Rabani et al. [37] and was later used in [9, 24, 26, 38]. In [37], the authors introduce a framework to analyze a large class of discrete neighbourhood load balancing algorithms. They transform a continuous algorithm A_c into a discrete version A_d that tries to stay as close to A_c as possible. Assume we apply both algorithms on a load vector x. Then, for every edge e, both A_c and A_d calculate the amount of load y_e that A_c would send over e. A_c will send exactly y_e tokens over e and A_d will *round down* y_e to an integer y'_e. The difference between y_e and the amount of load y'_e that is transferred in reality is called *rounding error*, which we denote by

$$\Delta y_e := y_e - y'_e \tag{8}$$

They show that the max-min discrepancy at time T is bounded by

$$\mathcal{O}\left(\frac{d \log n}{1-\lambda}\right).$$

See Tables 1 and 2 for the results of [37] for different graph classes. Their analysis framework applies to both FOS and dimension-exchange processes. The results hold for a wide class of rounding schemes, as long as the rounding errors are bounded by a constant.

Another technique using the potential function of Equation (6) is proposed in [8], where the potential drop is estimated using a *sequentialization* technique. In this technique, all the edges are assigned weights proportional to their scheduled load transfer. To estimate the potential drop, they consider a sequentialized version of the process in which edges are activated sequentially in increasing order of weight. Berenbrink et al. [8] show that under certain conditions the potential drop of the sequentialized and the original FOS process differ only by a constant factor. They show that

$$\Phi(t) = \mathcal{O}(d^3 \cdot n/\gamma) \text{ within } \mathcal{O}\left(\frac{d}{\gamma} \cdot \log\left(\frac{\Phi(0) \cdot \gamma}{d^3 \cdot n}\right)\right) \text{ rounds.}$$

Adolphs and Berenbrink [2] present a potential function based analysis of FOS for resources with non-uniform speeds. The analysis has two steps. First the authors show that

$$\Phi(t) = \mathcal{O}\left(d^3 \cdot s_{\max}^3 \cdot \frac{n}{\gamma}\right)$$

$$\text{after } \mathcal{O}\left(\frac{d \cdot s_{\max}^2 \log(m/n)}{\gamma}\right) \text{ rounds.}$$

In comparison with the result of [34] (see Equation (7)), they use fewer rounds when m is small compared to n and the potential bound is better for graphs with good expansion. In the second step, they use a different potential function to show that the max-min discrepancy is reduced to

$$\mathcal{O}(\text{diam}(G) \cdot d \cdot s_{\max})$$

$$\text{after } \mathcal{O}\left(\frac{n \cdot d^3 \cdot s_{\max}^3}{\gamma}\right) \text{ additional rounds.}$$

Both [8] and [2] employ techniques from spectral graph theory by representing the potential drop as a function of a quadratic from.

There are also several results showing lower bounds. When the continuous flow is rounded down, the final discrepancy is $\Omega(d \cdot \text{diam}(G))$ for a discrete FOS process [26, 27] and $\Omega(\text{diam}(G))$ for a discrete process in the matching model [27]. Friedrich and Sauerwald [24] consider the matching model and show that for any graph the discrepancy is at least $\Omega(\log n / \log \log n)$ after $\mathcal{O}(\log n)$ rounds if the rounding decisions and the initial load distribution are determined by an adversary.

2.3 Improved Processes for Discrete Load Balancing

The next three subsections discuss three different approaches that were used in order to reduce the difference (caused by the rounding error) in the load distribution between discrete and continuous balancing processes.

Random Walk Approach. Algorithms that use this approach [18, 19, 21] have two phases. The first phase uses the discrete diffusion approach of [37]. In the second phase it is assumed that the nodes know the average load m/n (which can be easily obtained by simulating the continuous diffusion algorithm on any node in Phase 1).

The second so-called fine balancing phase is not a simple neighbourhood load balancing strategy. Every token above $\alpha = m/n + c$ is marked as a *positive token*, and nodes with fewer than α tokens generate a *negative token* for every *hole* they have. The negative and positive tokens then perform one random walk step in each round. In reality, the movement of a negative token from node i to node j is translated as a token movement from node j to node i. Hence, whenever a negative token hits a positive token, both are eliminated. Note that this method can create negative loads when in one round too many negative tokens move to the same node. The tightest analysis of the algorithm is due to Elsässer and Sauerwald [19] where the authors achieve constant max-min discrepancy in $\mathcal{O}(T)$ rounds.

Randomized Rounding. This technique applies randomized rounding on $y_{i,j}$ in order to improve the bounds on the discrepancy for discrete neighbourhood load balancing. It was suggested in [39] and first analyzed by [24]. In the latter paper the authors consider a discrete dimension exchange algorithm for the matching model. Every node i that is connected to a matching edge (i,j) first calculates $y_{i,j}$ as in Equation (5). If that value is positive, it rounds it up or down, each with a probability one half. The authors combine the approach of [37] with analysis techniques for randomized algorithms to show improved discrepancy bounds for general graphs. For expanders, they provide a separate analysis that shows constant discrepancy is achieved by slightly increasing the running time. See Table 2 for a detailed statement of their results.

For arbitrary regular graphs the results of [24] were further improved in [38]. One of the main ideas of this paper is to show *negative dependence* [17] among the token locations, which enables them to use simple Chernoff bounds for their proofs. They show that a constant final discrepancy can be achieved within $\mathcal{O}(T)$ rounds for regular graphs in the random matching model, and constant-degree regular graphs in the periodic matching model. Note that the constant hidden in the asymptotic notation is large and the bounds can be achieved with probability $1-\exp(-\log^c n)$ for some constant $c<1$. See Table 2 for more detailed results.

A randomized rounding FOS process is considered in [26]. Similar to [24, 38] the process rounds $y_{i,j}$ up or down, with a probability such that the expected load forwarded over edge (i,j) is exactly $y_{i,j}$. Again, the analysis uses the framework of [37] together with analysis techniques for randomized algorithms. See Table 1 for a detailed statement of the results. Note that, if a node rounds up for too many edges, it may not have sufficiently many tokens. This can lead to so-called *negative load* on some of the nodes. In contrast, Berenbrink et al. [9] propose an FOS process that uses randomized rounding but avoids this problem. Again, every node i calculates for every edge (i,j) the amount of load $y_{i,j}$ that would be transferred in the continuous case (see Equation (1)) and rounds it down. The remaining $\sum_{j\in N(i)\cup\{i\}}(y_{i,j}-\lfloor y_{i,j}\rfloor)$ so-called *excess tokens* are then distributed as follows. Instead of rounding $y_{i,j}$ for every edge, node i forwards its excess tokens to randomly chosen neighbours (without replacement).

Note that it is possible to get similar results if the excess tokens are sent to neighbours chosen randomly with replacement or if the neighbours are chosen in a round-robin fashion with a random starting point [5]. The max-min discrepancy results of the randomized algorithms in [9, 26] are further improved by applying the results from [38], where tighter bounds are obtained for certain graph parameters used in discrepancy bounds of [9, 26]. See Table 1 for more detailed statement of the results.

Deterministic Rounding. Friedrich et al. [26] follow up on the rounding idea suggested in [39], raising the question whether this randomized algorithm can be derandomized without sacrificing its performance. Instead of randomized rounding, they use a deterministic rounding scheme. For each edge (i,j) they define the *accumulated rounding error* at the end of round t as

$$\widehat{\Delta y}_{i,j}(t) := \sum_{\ell=1}^{t} \Delta y_{i,j}(\ell),$$

where the error $\Delta y_{i,j}(\ell)$ is defined in Equation (8). Then, in round $t+1$ each node i calculates

$$\min\left\{\left|\widehat{\Delta y}_{i,j}(t)+y_{i,j}-\lfloor y_{i,j}\rfloor\right|,\left|\widehat{\Delta y}_{i,j}(t)+y_{i,j}-\lceil y_{i,j}\rceil\right|\right\}.$$

If the first term is the minimum, node i sends $\lfloor y_{i,j}\rfloor$ many tokens over (i,j), otherwise it sends $\lceil y_{i,j}\rceil$ many tokens. First the authors show that their process has the property that for every edge and in every round the accumulated rounding error is bounded by a constant, which they call the *bounded-error property*. Then they show that for hypercubes and tori, any process with bounded-error property achieves final discrepancy of $\mathcal{O}(\log^{3/2} n)$ and $\mathcal{O}(1)$, respectively. Note that this algorithm might also create negative load on some of the nodes.

3 Notation and Basic Facts

We model the network by an undirected graph $G=(V,E)$, where $V=\{1,\dots,n\}$. $N(i)$ is the set of direct neighbours of node i, and d is the maximum degree.

Initially there are in total m tasks which are assigned arbitrarily to the n nodes of the graph G. Tasks may be of different integer weights and the maximum task weight is denoted by $w_{\max}$. W is the total weight of the tasks, i.e., the sum of the weights of all tasks. When tasks are identical, they are called tokens.

$s_i \geqslant 1$ is an integer denoting the speed of the resource i. Note that we can always assume (by means of proper scaling) that the minimum speed is 1. We define $S=s_1+s_2+\ldots+s_n$ as the *capacity* of the network. The *load* x_i of resource i is defined as the total weight of its tasks. Recall that the *makespan* of a resource i is defined as the total weight of its tasks divided by its speed, i.e., x_i/s_i. The makespan of an assignment $(x_1,\dots,x_n)$ is the maximum makespan of any resource. The *max-min discrepancy* of an assignment is defined as the difference between the minimum and maximum makespan. The *max-avg discrepancy* is defined as the difference between the maximum makespan and W/S, which is the makespan of the balanced allocation.

Recall that for a fixed process, $x(t)=(x_1(t),\dots,x_n(t))$ denotes the load vector in the beginning of round $t\geqslant 0$ of the process, so $x(0)$ is the load vector that describes the initial distribution of the m tasks. Recall that $y_{i,j}(t)$ represents the non-negative load transferred from node i to node j at round $t\geqslant 0$, and that each node has an *infinite source* of dummy tokens to be generated on demand when the outgoing demand is more than the available load. If i does not use its infinite source in round t, then we have:

$$x_i(t+1)=x_i(t)-\sum_{j\in N(i)}(y_{i,j}(t)-y_{j,i}(t)). \tag{9}$$

We define $f_{i,j}(t)$ as the total load transferred from i to j by the end of the round t, which is

$$f_{i,j}(t)=\sum_{\tau=0}^{t}(y_{i,j}(\tau)-y_{j,i}(\tau)).$$

We assume $f_{i,j}(-1)=0$. To distinguish between the processes we will use the names of these processes as superscript in the above definitions. We will use A for a continuous algorithm and $\mathfrak{D}(A)$ for its discrete counterpart (following the transformation introduced by Algorithm 1 or Algorithm 2, depending on the context). We define

$$e_{i,j}(t) = f^{A}_{i,j}(t) - f^{\mathfrak{D}(A)}_{i,j}(t)$$

as the difference in the flow forwarded over the edge (i,j) by A and $\mathfrak{D}(A)$ at the end of the round t. Note that

$$e_{i,j}(t) = -e_{j,i}(t)$$

and

$$f_{i,j}(t) = -f_{j,i}(t).$$

The *balancing time* of a continuous algorithm A is the time it takes for the algorithm until every node has a load that is very close to its load in the ideally balanced state; that is

$$T^{A} = T^{A}(x(0)) = \min\{t : \forall i, |x_i(t) - W \cdot s_i/S| \leqslant 1\}.$$

In the following, we follow up with a few definitions to be used in the statement of our new results.

Consider a continuous process A. For the transformations introduced by Algorithm 1 and Algorithm 2, we require initial load vectors that do not lead to *negative load* in the continuous case; that is, we need to ensure that when executing A, the outgoing demand of a node never exceeds its available load. The definition below formalizes this property:

Definition 1. *We say A* does not induce negative load *on* $\mathbf{x}$ *when the following holds: If $x^{A}(0) = \mathbf{x}$, then for all $i \in V$ and $t \geqslant 0$ we have:*[5]

$$x^{A}_{i}(t) - \sum_{j \in N(i)} y^{A}_{i,j}(t) \geqslant 0.$$

Note that among the processes mentioned in Section 2, only SOS may induce negative load, depending on the given graph structure and load distribution.

Our framework works for *additive* and *terminating* processes, as defined below. For brevity, we will not repeat these requirements throughout in the statement of the results. A balancing process is *terminating* when starting with a balanced load vector, the net load transfer over each edge is always zero. One can see that this is a very natural property of a load balancing process.

Definition 2 (Terminating)**.** *We say A is* terminating *when for any $\ell \geqslant 0$, if $x^{A}(0) = \ell \cdot (s_1, \ldots, s_n)$ then $y^{A}_{i,j}(t) = 0$ for all nodes i, j and round t.*

Definition 3 (Additive)**.** *Consider a load balancing process A. Let $\mathbf{x}'$, and $\mathbf{x}''$ be non-negative load vectors. Let $\mathbf{x} = \mathbf{x}' + \mathbf{x}''$, and suppose we start three instances of A with $\mathbf{x}, \mathbf{x}'$, and $\mathbf{x}''$, and that A does not induce negative load on $\mathbf{x}$, $\mathbf{x}'$, or $\mathbf{x}''$. Let $\mathbf{y}_{i,j}(t)$, $\mathbf{y}'_{i,j}(t)$, and $\mathbf{y}''_{i,j}(t)$, respectively, be the amount of load transferred from i to j in the round t of the three instances of the process.*

Then A is said to be additive *if the equation $\mathbf{y}_{i,j}(t) = \mathbf{y}'_{i,j}(t) + \mathbf{y}''_{i,j}(t)$ holds for all nodes i, j and round $t \geqslant 0$ regardless of the choice of $\mathbf{x}'$, $\mathbf{x}''$.*[6]

[5]For randomized processes such as the random matchings model, the statement should hold w.h.p. over the probability space.

[6]For randomized processes, the three runs are coupled with the same sequence of outcomes.

The next lemma shows that the class of additive terminating processes includes several well known existing processes.

Lemma 1. *The following processes as defined in Section 2 are both additive and terminating:*

- *First order diffusion*
- *Second order diffusion*
- *Matching-based processes (using periodic/ random matchings)*

Proof. Consider a sequence of matrices $\langle P(0), P(1), \ldots \rangle$ and $0 \leqslant \beta \leqslant 2$. Observe that all the processes listed in the observation can be described by the following general equations (see Equations (1), (4), and (5)):

$$y_{i,j}(0) = P_{i,j}(0) \cdot x_i(0) \tag{10}$$

$$y_{i,j}(t) = (\beta - 1) \cdot y_{i,j}(t-1) + \beta \cdot P_{i,j}(t) \cdot x_i(t) \qquad \text{for } t \geqslant 1, \tag{11}$$

where in diffusion cases, for all $t \geqslant 0$ we have $P(t) = P$.

Proof of Additivity. Let $\mathbf{x}'$, $\mathbf{x}''$ be arbitrary load distributions. Let $\mathbf{x}(t)$, $\mathbf{x}'(t)$, and $\mathbf{x}''(t)$ denote the load vector at round t starting $\mathscr{A}$ from $\mathbf{x} = \mathbf{x}' + \mathbf{x}''$, $\mathbf{x}'$, and $\mathbf{x}''$ respectively. Let $\mathbf{y}$, $\mathbf{y}'$, and $\mathbf{y}''$ denote their corresponding load transfer variables.

By induction on t, we prove that the following hold for arbitrary nodes i, j and round t:

(a) $\mathbf{x}_i(t) = \mathbf{x}'_i(t) + \mathbf{x}''_i(t)$.

(b) $\mathbf{y}_{i,j}(t) = \mathbf{y}'_{i,j}(t) + \mathbf{y}''_{i,j}(t)$

For $t = 0$, (a) holds because of the assumptions, and consequently (b) holds by Equation (10).

Suppose that for some $t \geqslant 0$ both $\mathbf{x}(t) = \mathbf{x}'(t) + \mathbf{x}''(t)$ and $\mathbf{y}(t) = \mathbf{y}'(t) + \mathbf{y}''(t)$ hold for every round $\leqslant t$. We show in the following that (a) and (b) must be true for $t+1$ as well. To prove (a), we use the fact that A does not induce negative load on $\mathbf{x}$, $\mathbf{x}'$, or $\mathbf{x}''$ to apply Equation (9) and write for arbitrary i:

$$\begin{aligned}
\mathbf{x}_i(t+1) &= \mathbf{x}_i(t) - \sum_{j \in N(i)} (\mathbf{y}_{i,j}(t) - \mathbf{y}_{j,i}(t)) \\
&= \mathbf{x}'(t) + \mathbf{x}''(t) - \sum_{j \in N(i)} \big(\mathbf{y}'_{i,j}(t) + \mathbf{y}''_{i,j}(t) - \mathbf{y}'_{j,i}(t) - \mathbf{y}''_{j,i}(t)\big) \\
&= \mathbf{x}'_i(t) - \sum_{j \in N(i)} \big(\mathbf{y}'_{i,j}(t) - \mathbf{y}'_{j,i}(t)\big) + \mathbf{x}''_i(t) - \sum_{j \in N(i)} \big(\mathbf{y}''_{i,j}(t) - \mathbf{y}''_{j,i}(t)\big) \\
&= \mathbf{x}'_i(t+1) + \mathbf{x}''_i(t+1).
\end{aligned} \tag{12}$$

Now, we proceed to prove (b). For arbitrary nodes i, j we have:

$$\begin{aligned}
\mathbf{y}_{i,j}(t+1) &= (\beta-1)\cdot\mathbf{y}_{i,j}(t)+\beta\cdot P_{i,j}(t)\cdot\mathbf{x}_i(t+1)\\
&= (\beta-1)\cdot\left(\mathbf{y}'_{i,j}(t)+\mathbf{y}''_{i,j}(t)\right)+\beta\cdot P_{i,j}(t)\cdot\left(\mathbf{x}'_i(t+1)+\mathbf{x}''_i(t+1)\right) \qquad (13)\\
&= (\beta-1)\cdot\mathbf{y}'_{i,j}(t)+\beta\cdot P_{i,j}(t)\cdot\mathbf{x}'_i(t+1)+(\beta-1)\cdot\mathbf{y}''_{i,j}(t)+\beta\cdot P_{i,j}(t)\cdot\mathbf{x}''_i(t+1)\\
&= \mathbf{y}'_{i,j}(t+1)+\mathbf{y}''_{i,j}(t+1)
\end{aligned}$$

where Equation (13) follows from Equation (12) and the induction hypothesis.

Proof of being Terminating. Suppose for some ℓ we have $x_i(0) = s_i\cdot\ell$ for every node i. We prove inductively that $y_{i,j}(t) = y_{j,i}(t)$ and $x(t+1) = x(0)$ for all i, j, and t. In case of the matching model, we only need to consider the matching edges. For the remaining of the proof, $\alpha_{i,j}$ is defined as in Equation (1), Equation (4) and Equation (5), depending on the process. The only property of $\alpha_{i,j}$'s we rely on here is that $\alpha_{i,j} = \alpha_{j,i}$ which holds in all three cases.

For $t = 0$, from Equation (10) we get

$$\begin{aligned}
y_{i,j}(0) &= (\alpha_{i,j}/s_i)\cdot s_i\cdot\ell\\
&= \alpha_{j,i}\cdot\ell\\
&= (\alpha_{j,i}/s_j)\cdot s_j\cdot\ell\\
&= y_{j,i}(0).
\end{aligned}$$

Therefore, the load vector remains unchanged and we have $x(1) = x(0)$.

Now suppose that for all $1\leqslant\tau\leqslant t-1$, we have $x(\tau+1) = x(0)$ and $y_{i,j}(\tau) = y_{j,i}(\tau)$ for all neighbours i, j. We prove that this yields $y_{i,j}(t) = y_{j,i}(t)$ for all i, j, from which it follows that $x(t+1) = x(0)$. From Equation (11), for arbitrary neighbours i, j we get:

$$\begin{aligned}
y_{i,j}(t) &= (\beta-1)\cdot y_{i,j}(t-1)+\beta\cdot P_{i,j}(t)\cdot x_i(t)\\
&= (\beta-1)\cdot y_{j,i}(t-1)+\beta\cdot(\alpha_{i,j}/s_i)\cdot s_i\cdot\ell\\
&= (\beta-1)\cdot y_{j,i}(t-1)+\beta\cdot(\alpha_{j,i}/s_j)\cdot s_j\cdot\ell\\
&= y_{j,i}(t).
\end{aligned}$$

Hence $x(t+1) = x(t) = x(0)$ and the proof follows. □

The next lemma establishes a fact which we will later need to show that under certain conditions our discrete algorithms do not induce negative load.

Lemma 2. *Let $\mathbf{x}'$ be an arbitrary load vector on which A does not induce negative load. Suppose $x^A(0) = \mathbf{x}'+\mathbf{x}''$ such that $\mathbf{x}'' = \ell\cdot(s_1,\ldots,s_n)$ for some $\ell\geqslant 0$. Then for $i\in V$, $t\geqslant 0$, and any $L\subseteq N(i)$ we have:*

$$x_i^A(t)-\sum_{j\in L}\left(y^A_{i,j}(t)-y^A_{j,i}(t)\right)\geqslant s_i\cdot\ell.$$

Proof. By additivity, we have $x_i^A(t) = x_i'(t) + x_i''(t)$ and $y_{i,j}^A(t) = y_{i,j}'(t) + y_{i,j}''(t)$. Thus,

$$\begin{aligned}
x_i^A(t) - \sum_{j \in L} \left(y_{i,j}^A(t) - y_{j,i}^A(t) \right) &= x_i'(t) - \sum_{j \in L} \left(y_{i,j}'(t) - y_{j,i}'(t) \right) + x_i''(t) - \sum_{j \in L} \left(y_{i,j}''(t) - y_{j,i}''(t) \right) \\
&= x_i'(t) - \sum_{j \in L} \left(y_{i,j}'(t) - y_{j,i}'(t) \right) + x_i''(t) \qquad (A \text{ is terminating}) \\
&= x_i'(t) - \sum_{j \in L} \left(y_{i,j}'(t) - y_{j,i}'(t) \right) + s_i \cdot \ell \\
&\geqslant x_i'(t) - \sum_{j \in L} y_{i,j}'(t) + s_i \cdot \ell \\
&\geqslant x_i'(t) - \sum_{j \in N(i)} y_{i,j}'(t) + s_i \cdot \ell \\
&\geqslant s_i \cdot \ell,
\end{aligned}$$

where the last equation follows from the fact that A does not induce negative load on $\mathbf{x}'$. □

4 Deterministic Flow Imitation

In this section, we present and analyze an algorithm that transforms a continuous process A into its discrete counterpart which we call $\mathfrak{D}(A)$. To distinguish between the two processes we will use A and $\mathfrak{D}(A)$ as superscripts in the definitions of Section 3.

The algorithm (see Algorithm 1) keeps track of the total flow $f_{i,j}^A(t)$ that is sent over the edge (i,j) by the continuous algorithm. It calculates the difference in the flow forwarded over the edge by the continuous and the discrete algorithm which we define as

$$\widehat{y}_{i,j}^A(t) := f_{i,j}^A(t) - f_{i,j}^{\mathfrak{D}(A)}(t-1).$$

It then *tries to* find a set $\mathscr{S}_{ij}$ of tasks with a total weight $|\mathscr{S}_{ij}|$ of that difference. These tokens will be forwarded over the edge (i,j). In the case of identical tasks, the amount of load sent from i to its neighbour j is

$$y_{i,j}^{\mathfrak{D}(A)}(t) = \left\lfloor f_{i,j}^A(t) - f_{i,j}^{\mathfrak{D}(A)}(t-1) \right\rfloor.$$

In the general case of weighted tasks, $\mathscr{S}_{ij}$ is chosen in a way that $f_{i,j}^A(t) - f_{i,j}^{\mathfrak{D}(A)}(t) \leqslant w_{\max}$.

It might happen that the node i does not contain enough load. In that case the algorithm will create new, artificial tasks and send them to the corresponding neighbours (or equivalently, we may think of an attached infinite source of tokens from which the node gets some tokens). Later we will show that this never happens if the initial load of the resources is large enough. We also note that in actual implementation we do not need to create and transfer workload units and consume communication bandwidth for each dummy token. That would incur communication overhead proportional to the number of dummy tokens. Instead, we can communicate only a number indicating the amount of dummy workload that should be forwarded to each neighbour.

The following theorem states our result about the transformation introduced by Algorithm 1.

Theorem 3. [7] *Suppose the discrete process $\mathfrak{D}(A)$ is obtained by transforming a continuous process A using Algorithm 1. Then the following hold:*

(1) If A does not induce negative load on $x^A(0)$, then for all $t \geqslant T^A$, the max-avg discrepancy of $x^{\mathfrak{D}(A)}(t)$ is at most $2d \cdot w_{\max} + 2$.

(2) If $x^A(0) = \mathbf{x}' + \mathbf{x}''$ such that $\mathbf{x}'' = d \cdot w_{\max} \cdot (s_1, \ldots, s_n)$, and A does not induce negative load on $\mathbf{x}'$, then the max-min discrepancy of $x^{\mathfrak{D}(A)}(t)$ is at most $2d \cdot w_{\max} + 2$.

Note that among the algorithms mentioned in Section 2, only SOS may induce negative load. For other algorithms, the result in part (1) of the above theorem automatically holds, and the condition in part (2) can be translated as having sufficient initial load.

For the special case of identical tasks, the above theorem gives a bound of $2d+1$ on the max-avg discrepancy. Compared to the deterministic algorithm of [26] that has been analyzed only for hypercubes and tori, we obtain improved bounds for hypercubes while our analysis is more general with respect to network topology, and heterogeneity of tasks and resources. Besides, it also works for the matching-based models and the second order diffusion. See Tables 1 and 2 for a more detailed comparison of discrepancy bounds.

Algorithm 1 $\mathfrak{D}(A)$: Discretized A using *flow imitation*: the process on node i at round t

```
for each neighbour $j$ of $i$
    Compute $f^A_{i,j}(t)$
    $\widehat{y}^A_{i,j}(t) \leftarrow f^A_{i,j}(t) - f^{\mathfrak{D}(A)}_{i,j}(t-1)$
    while $\widehat{y}^A_{i,j}(t) - |\mathscr{S}_{ij}| \geqslant w_{\max}$
        if $i$ has no unallocated tasks then
            $q \leftarrow$ a unit weight task generated by the attached infinite source
        else
            $q \leftarrow$ arbitrary task removed from the set of unallocated tasks of $i$
        Add $q$ to $\mathscr{S}_{ij}$
    $y^{\mathfrak{D}(A)}_{i,j}(t) \leftarrow |\mathscr{S}_{ij}|$
```

Before proving the theorem, we need to establish some preliminary results.

Observation 4. *As long as the Algorithm 1 is allowed to access the infinite source we have $|e_{i,j}(t)| < w_{\max}$ for every $(i,j) \in E$ and $t \geqslant 0$.*

[7] An earlier version of this result appeared in [6, Theorem 1.1]

Proof. Recall that $e_{i,j}(t) = f^A_{i,j}(t) - f^{\mathfrak{D}(A)}_{i,j}(t)$. We observe that $e_{i,j}(t) = -e_{j,i}(t)$. Thus, it suffices to prove the inequality holds for an arbitrary edge direction. In the following, we prove that $f^A_{i,j}(t) - f^{\mathfrak{D}(A)}_{i,j}(t) < w_{\max}$.

Fix an edge (i,j) and observe that

$$f^A_{i,j}(t) - f^{\mathfrak{D}(A)}_{i,j}(t-1) = (-1)\cdot\left(f^A_{j,i}(t) - f^{\mathfrak{D}(A)}_{j,i}(t-1)\right).$$

Assume $f^A_{i,j}(t) - f^{\mathfrak{D}(A)}_{i,j}(t-1) \geqslant 0$ (otherwise we switch i and j). From the definition of Algorithm 1, it follows that $y^{\mathfrak{D}(A)}_{j,i}(t) = 0$ since $f^A_{j,i}(t) - f^{\mathfrak{D}(A)}_{j,i}(t-1) \leqslant 0$. Therefore,

$$f^{\mathfrak{D}(A)}_{i,j}(t) \ = \ f^{\mathfrak{D}(A)}_{i,j}(t-1) + y^{\mathfrak{D}(A)}_{i,j}(t).$$

After exiting the loop for node i, we have:

$$f^A_{i,j}(t) - f^{\mathfrak{D}(A)}_{i,j}(t-1) - \|\mathscr{S}_{ij}\| < w_{\max},$$

or equivalently,

$$f^A_{i,j}(t) - f^{\mathfrak{D}(A)}_{i,j}(t-1) - y^{\mathfrak{D}(A)}_{i,j}(t) < w_{\max},$$

which yields:

$$f^A_{i,j}(t) - f^{\mathfrak{D}(A)}_{i,j}(t) < w_{\max}. \tag{14}$$

Let w be the weight of the last task added to $\mathscr{S}_{ij}$. Before adding this task, the loop condition was fulfilled. Hence we have:

$$f^A_{i,j}(t) - f^{\mathfrak{D}(A)}_{i,j}(t) + w \geqslant w_{\max},$$

and thus:

$$f^A_{i,j}(t) - f^{\mathfrak{D}(A)}_{i,j}(t) \geqslant w_{\max} - w \geqslant 0. \tag{15}$$

Combining Equations (14) and (15), we get

$$\left| f^A_{i,j}(t) - f^{\mathfrak{D}(A)}_{i,j}(t) \right| < w_{\max},$$

as needed. ☐

Observation 5. *For every $(i,j) \in E$ and $t \geqslant 0$ the following holds:*
If $y^{\mathfrak{D}(A)}_{i,j}(t) > 0$, then we have

$$y^{\mathfrak{D}(A)}_{i,j}(t) \leqslant y^A_{i,j}(t) - y^A_{j,i}(t) + e_{i,j}(t-1).$$

Proof. First observe that:

$$\begin{aligned} y^A_{i,j}(t) - y^A_{j,i}(t) + e_{i,j}(t-1) &= y^A_{i,j}(t) - y^A_{j,i}(t) + f^A_{i,j}(t-1) - f^{\mathfrak{D}(A)}_{i,j}(t-1) \\ &= f^A_{i,j}(t) - f^{\mathfrak{D}(A)}_{i,j}(t-1) \\ &= \widehat{y}^A_{i,j}(t). \end{aligned}$$

It remains to prove $y_{i,j}^{\mathfrak{D}(A)}(t) \leqslant \widehat{y}_{i,j}^{A}(t)$. Let w be the weight of the last task added to $\mathscr{S}_{ij}$ in round t. Before adding this task, the loop condition of Algorithm 1was fulfilled and we had

$$\widehat{y}_{i,j}^{A}(t) - (|\mathscr{S}_{ij}| - w) \;\geqslant\; w_{\max}.$$

After the loop, we have:

$$y_{i,j}^{\mathfrak{D}(A)}(t) = |\mathscr{S}_{ij}|.$$

Consequently,

$$\widehat{y}_{i,j}^{A}(t) - y_{i,j}^{\mathfrak{D}(A)}(t) \geqslant w_{\max} - w \geqslant 0.$$

□

The next lemma is used to prove the discrepancy bound assuming no token is borrowed from the infinite source.

Lemma 6. *For $i \in V$ and $\tau \geqslant 0$, suppose i does not use its infinite source by the end of the round $\tau - 1$. Then for any $t \leqslant \tau$ the following hold:*

(1) $x_i^{\mathfrak{D}(A)}(t) = x_i^A(t) + \sum_{j \in N(i)} e_{i,j}(t-1)$.

(2) $\left| x_i^{\mathfrak{D}(A)}(t) - x_i^A(t) \right| < d \cdot w_{\max}$.

Proof. The proof of (1) is by induction on t. For $t = 0$, we have $x_i^{\mathfrak{D}(A)}(0) = x_i^A(0)$ and for all i, j, $E_{i,j}(-1) = 0$. Therefore the equation holds.

Suppose for some $t \geqslant 0$ we have

$$x_i^{\mathfrak{D}(A)}(t) = x_i^A(t) + \sum_{j \in N(i)} e_{i,j}(t-1).$$

It remains to prove that the statement holds for $t+1$ as well. As long as $t+1 \leqslant \tau$, we can apply Equation (9) to get:

$$\begin{aligned}
x_i^{\mathfrak{D}(A)}(t+1) &= x_i^{\mathfrak{D}(A)}(t) + \sum_{j \in N(i)} (y_{j,i}^{\mathfrak{D}(A)}(t) - y_{i,j}^{\mathfrak{D}(A)}(t)) \\
&= x_i^A(t) + \sum_{j \in N(i)} e_{i,j}(t-1) + \sum_{j \in N(i)} \left(y_{j,i}^{\mathfrak{D}(A)}(t) - y_{i,j}^{\mathfrak{D}(A)}(t) \right) \\
&= x_i^A(t) + \sum_{j \in N(i)} \left(f_{i,j}^A(t-1) - f_{i,j}^{\mathfrak{D}(A)}(t-1) + y_{j,i}^{\mathfrak{D}(A)}(t) - y_{i,j}^{\mathfrak{D}(A)}(t) \right) \\
&= \left(x_i^A(t+1) - \sum_{j \in N(i)} (y_{j,i}^A(t) - y_{i,j}^A(t)) \right) + \sum_{j \in N(i)} \left(f_{i,j}^A(t-1) - f_{i,j}^{\mathfrak{D}(A)}(t) \right) \\
&= x_i^A(t+1) + \sum_{j \in N(i)} \left(f_{i,j}^A(t-1) + y_{i,j}^A(t) - y_{j,i}^A(t) - f_{i,j}^{\mathfrak{D}(A)}(t) \right) \\
&= x_i^A(t+1) + \sum_{j \in N(i)} \left(f_{i,j}^A(t) - f_{i,j}^{\mathfrak{D}(A)}(t) \right) \\
&= x_i^A(t+1) + \sum_{j \in N(i)} e_{i,j}(t).
\end{aligned}$$

This finishes the proof of (1). For (2), we apply (1) to get

$$\left|x_i^{\mathfrak{D}(A)}(t) - x_i^A(t)\right| = \left|\sum_{j\in N(i)} e_{i,j}(t-1)\right| \leqslant \sum_{j\in N(i)} |e_{i,j}(t-1)|.$$

Now, the result can be obtained using Observation 4. □

The following lemma shows that if there is initially sufficient amount of load on each node, then $D(A)$ does not induce negative load on $x^{\mathfrak{D}(A)}(t)$.

Lemma 7. *Suppose $x(0) = \mathbf{x}' + \mathbf{x}''$ such that $\mathbf{x}'' = d \cdot w_{\max} \cdot (s_1, \ldots, s_n)$, and A does not induce negative load on $\mathbf{x}'$.*

Then for all $i \in V$ and $t \geqslant 0$, the following holds:

$$x_i^{\mathfrak{D}(A)}(t) - \sum_{j\in N(i)} y_{i,j}^{\mathfrak{D}(A)}(t) \geqslant 0.$$

Proof. For the sake of contradiction, let us assume there is some round t_1 in which we use the infinite source for the first time. Let i be an arbitrary node with insufficient load, so that we have $x_i^{\mathfrak{D}(A)}(t_1) - \sum_{j\in N(i)} y_{i,j}^{\mathfrak{D}(A)}(t_1) < 0$. Define

$$L := \{j \in N(i) : y_{i,j}^{\mathfrak{D}(A)}(t_1) > 0\}$$

to be the set of neighbours of node i to which i transfers load in the round t_1. We get:

$$\begin{aligned}
x_i^{\mathfrak{D}(A)}(t_1) - \sum_{j\in N(i)} y_{i,j}^{\mathfrak{D}(A)}(t_1) &= x_i^{\mathfrak{D}(A)}(t_1) - \sum_{j\in L} y_{i,j}^{\mathfrak{D}(A)}(t_1) \\
&\geqslant x_i^A(t_1) + \sum_{j\in N(i)} e_{i,j}(t_1-1) - \sum_{j\in L}\left(y_{i,j}^A(t_1) - y_{j,i}^A(t_1) + e_{i,j}(t_1-1)\right) \qquad (16)\\
&= x_i^A(t_1) - \sum_{j\in L}\left(y_{i,j}^A(t_1) - y_{j,i}^A(t_1)\right) + \sum_{j\in N(i)-L} e_{i,j}(t_1-1) \\
&= d\cdot s_i \cdot w_{\max} + \sum_{j\in N(i)-L} e_{i,j}(t_1-1) \qquad (17)\\
&\geqslant d\cdot s_i \cdot w_{\max} - |N(i)-L|\cdot w_{\max} \qquad (18)\\
&\geqslant d\cdot s_i \cdot w_{\max} - d\cdot w_{\max} \geqslant 0, \qquad (\text{since } s_i \geqslant 1)
\end{aligned}$$

where in Equation (16) we use Observation 5 and Lemma 6, and the fact that no infinite source has been used before the round t_1. Equation (17) follows from the Lemma 2 using $x(0) = \mathbf{x}' + \mathbf{x}''$ and the conditions on $\mathbf{x}'$ and $\mathbf{x}''$ mentioned in the statement of the lemma, and by setting $\ell = d \cdot w_{\max}$. Also, Equation (18) results from Observation 4.

This contradicts our initial assumption that

$$x_i^{\mathfrak{D}(A)}(t_1) - \sum_{j\in N(i)} y_{i,j}^{\mathfrak{D}(A)}(t_1) < 0,$$

and the proof follows. □

We are now ready to prove Theorem 3.

Proof. First we prove part (2). Suppose $x^A(0) = \mathbf{x}' + \mathbf{x}''$ such that $\mathbf{x}'' = d \cdot w_{\max} \cdot (s_1, \dots, s_n)$, and A does not induce negative load on $\mathbf{x}'$. Then by Lemma 7 we know that negative load never occurs. Therefore, no infinite source is used and the total load remains intact. Hence, by part (2) of the Lemma 6, we have:

$$\left| x_i^{\mathfrak{D}(A)}(t) - x_i^A(t) \right| < d \cdot w_{\max}.$$

This fact together with the fact that after the balancing time we have $\left| x_i^A(t) - W \cdot s_i / S \right| \leqslant 1$, yield

$$\left| x_i^{\mathfrak{D}(A)}(t) - W \cdot s_i / S \right| < d \cdot w_{\max} + 1.$$

Since $s_i \geqslant 1$, we have $\left| x_i^{\mathfrak{D}(A)}(t)/s_i - W/S \right| < d \cdot w_{\max} + 1$, which holds for arbitrary node i. Hence, for any pair of nodes i, j we get $\left| x_i^{\mathfrak{D}(A)}(t)/s_i - x_j^{\mathfrak{D}(A)}(t)/s_j \right| < 2d \cdot w_{\max} + 2$ which gives the desired max-min discrepancy bound.

In the general case, to get the bound of part (1) the algorithm first adds $d \cdot s_i \cdot w_{\max}$ *dummy* unit weight tasks to each resource i before the process begins. Note that this does not affect the convergence time of the continuous process, because the extra load is completely balanced. In the rest of the proof, we use x to refer to the new load vectors. Let W' and W denote the original and the new total load, respectively. We have:

$$\begin{aligned} W &= W' + \sum_i d \cdot s_i \cdot w_{\max} \\ &= W' + d \cdot S \cdot w_{\max}. \end{aligned}$$

Hence,

$$W/S \leqslant W'/S + d \cdot w_{\max}$$

At the end, the dummy tokens can be simply ignored. Nevertheless we can still use $x_i^{\mathfrak{D}(A)}(t)$ as an upper bound on the final load of the node i excluding the dummy tokens. Following steps similar to the max-min discrepancy case, we get

$$x_i^{\mathfrak{D}(A)}(t)/s_i - W/S < d \cdot w_{\max} + 1,$$

as required. □

5 Randomized Flow Imitation

In this section we analyze a randomized version of Algorithm 1 that can be applied for balancing identical tasks. Instead of always rounding down the flow that has to be sent over an edge, Algorithm 2 uses randomized rounding. The notation we use in this section is the same as defined in Section 4. We use uppercase letters to express random variables. As an example $f_{i,j}^A(t)$ is the flow sent over edge (i, j) in round t by the continuous process, while $F_{i,j}^{\mathfrak{D}(A)}(t)$ is the corresponding random variable for the discrete process.

Algorithm 2 calculates the flow

$$\widehat{Y}_{i,j}(t) := f_{i,j}^{A}(t) - F_{i,j}^{\mathfrak{D}(A)}(t-1)$$

that has to be sent over edge (i,j) as before. To calculate the flow that is actually sent, $\widehat{Y}_{i,j}(t)$ is randomly rounded up or down, with a probability depending on the value of its fractional part. Suppose $\widehat{Y}_{i,j}(t) > 0$ in some round t. Then the discrete flow $Y_{i,j}^{\mathfrak{D}(A)}(t)$ that is sent over the edge is a random variable determined by the following randomized rounding scheme. For a real x, we use $\{x\} = x - \lfloor x \rfloor$ to denote the fractional part of x. Then

$$Y_{i,j}^{\mathfrak{D}(A)}(t) = \begin{cases} \lfloor \widehat{Y}_{i,j}(t) \rfloor + 1 & \text{with probability } \{\widehat{Y}_{i,j}(t)\}, \\ \lfloor \widehat{Y}_{i,j}(t) \rfloor & \text{otherwise.} \end{cases}$$

In Algorithm 2 we use $Z_{i,j}(t)$, which is a zero-one random variable indicating whether we should round up. Once we know all the random choices in round t, we can calculate the load of processor i as before using

$$X_i^{\mathfrak{D}(A)}(t+1) = X_i^{\mathfrak{D}(A)}(t) - \sum_{j \in N(i)} (Y_{i,j}^{\mathfrak{D}(A)}(t) - Y_{j,i}^{\mathfrak{D}(A)}(t)).$$

Algorithm 2 $\mathfrak{D}$(A): Discretized A using randomized *flow imitation*: the process on node i at round t

```
for each neighbour $j$ of $i$ in parallel
  Compute $f_{i,j}^{A}(t)$
  $\widehat{Y}_{i,j}(t) \leftarrow f_{i,j}^{A}(t) - F_{i,j}^{\mathfrak{D}(A)}(t-1)$
  if $\widehat{Y}_{i,j}(t) > 0$ then
      Toss a coin with head probability $\{\widehat{Y}_{i,j}(t)\}$
      $Z_{i,j}(t) \leftarrow \begin{cases} 1 & \text{if head comes up;} \\ 0 & \text{otherwise.} \end{cases}$
      $Y_{i,j}^{\mathfrak{D}(A)}(t) \leftarrow \lfloor \widehat{Y}_{i,j}(t) \rfloor + Z_{i,j}(t)$
      Send $Y_{i,j}^{\mathfrak{D}(A)}(t)$ tokens to $j$
      if there are not enough tokens then
          generate the required amount using the attached infinite source
```

We will show that with high probability the roundings errors sum up to a small value. The following theorem states our result about the transformation introduced by Algorithm 2.

Theorem 8. [8] *Suppose the discrete process $\mathfrak{D}(A)$ is obtained by transforming a continuous process A using Algorithm 2 and that $T^A \leqslant n^{\kappa}$ for some constant κ. Then the following hold:*

[8]An earlier version of this result appeared in [6, Theorem 1.2]

(1) If A does not induce negative load on $x^A(0)$, then the max-avg discrepancy of $X^{\mathfrak{D}(A)}(T^A)$ is at most $d/4+\mathcal{O}(\sqrt{d\log n})$ w.h.p.

(2) If $x^A(0)=\mathbf{x}'+\mathbf{x}''$ such that $\mathbf{x}''=(d/4+2c\cdot\sqrt{d\log n})\cdot(s_1,\ldots,s_n)$ for a properly chosen constant $c>0$ and A does not induce negative load on $\mathbf{x}'$, then w.h.p. the max-min discrepancy of $X^{\mathfrak{D}(A)}(T^A)$ is $\mathcal{O}(\sqrt{d\log n})$.

Note that among the algorithms mentioned in Section 2, only SOS may induce negative load. For other algorithms, the result in part (1) of the above theorem automatically holds, and the condition in part (2) can be translated as having sufficient initial load.

The next observation provides useful tools for proving the above theorem.

Observation 9. *For $i\in V$, $j\in N(i)$ and $t\geqslant 0$ the following hold:*

(1) $E_{i,j}(t)=y^A_{i,j}(t)-y^A_{j,i}(t)+E_{i,j}(t-1)-(Y^{\mathfrak{D}(A)}_{i,j}(t)-Y^{\mathfrak{D}(A)}_{j,i}(t))$.

(2) If $Y^{\mathfrak{D}(A)}_{i,j}(t)>0$ then $Y^{\mathfrak{D}(A)}_{j,i}(t)=0$.

(3) If $\widehat{Y}_{i,j}(t)>0$, then we have[9]:

$$E_{i,j}(t)=\begin{cases}\{\widehat{Y}_{i,j}(t)\}-1 & \textit{if } Z_{i,j}(t)=1\\ \{\widehat{Y}_{i,j}(t)\} & \textit{otherwise.}\end{cases}$$

Proof. To prove the first statement, recall that $E_{i,j}(t)=f^A_{i,j}(t)-F^{\mathfrak{D}(A)}_{i,j}(t)$. The right side of the equation can be simplified as below:

$$\begin{aligned}&y^A_{i,j}(t)-y^A_{j,i}(t)+E_{i,j}(t-1)-\left(Y^{\mathfrak{D}(A)}_{i,j}(t)-Y^{\mathfrak{D}(A)}_{j,i}(t)\right)\\&=\left(y^A_{i,j}(t)-y^A_{j,i}(t)+f^A_{i,j}(t-1)\right)-\left(F^{\mathfrak{D}(A)}_{i,j}(t-1)+Y^{\mathfrak{D}(A)}_{i,j}(t)-Y^{\mathfrak{D}(A)}_{j,i}(t)\right)\\&=f^A_{i,j}(t)-F^{\mathfrak{D}(A)}_{i,j}(t)\\&=E_{i,j}(t).\end{aligned}$$

For the second statement, note that if $Y^{\mathfrak{D}(A)}_{i,j}(t)>0$ then according to Algorithm 2, $\widehat{Y}_{i,j}(t)>0$ must be satisfied; therefore $f^A_{j,i}(t)-F^{\mathfrak{D}(A)}_{j,i}(t-1)<0$ which yields $Y^{\mathfrak{D}(A)}_{j,i}(t)=0$.

Now we prove the third statement. Since $\widehat{Y}_{i,j}(t)>0$, we have $\widehat{Y}_{j,i}(t)=-\widehat{Y}_{i,j}(t)<0$ and therefore $Y^{\mathfrak{D}(A)}_{j,i}(t)=0$ by part (2) of the observation. Thus, using part (1) we get

$$\begin{aligned}E_{i,j}(t)&=y^A_{i,j}(t)-y^A_{j,i}(t)+E_{i,j}(t-1)-Y^{\mathfrak{D}(A)}_{i,j}(t)\\&=\widehat{Y}_{j,i}(t)-Y^{\mathfrak{D}(A)}_{i,j}(t)\\&=\widehat{Y}_{j,i}(t)-\lfloor\widehat{Y}_{i,j}(t)\rfloor-Z_{i,j}(t),\end{aligned}$$

where the last equation follows from the way $Y^{\mathfrak{D}(A)}_{i,j}(t)$ is obtained in Algorithm 2, and the proof follows from the fact that $Z_{i,j}(t)$ can be either zero or one. □

[9]Recall that for a real number a, $\{a\}:=a-\lfloor a\rfloor$

Note that part (3) of the Observation 9 shows that $\mathbf{Ex}[e_{i,j}(t)] = 0$.

Now we show that w.h.p. the discrete process does not deviate much from the continuous process. We identify some *undesirable* events as listed in the Lemma 10 and show in that each of these events happens only with a small probability. In our proofs, we make use of Lemma 12 in the appendix which is a simple adaptation of the Hoeffding's bound [29] for sums of randomized rounding errors.

Before stating the lemma, we introduce some notation. Define

$$H_i(t) := \{j \in N(i) : y^A_{i,j}(t) - y^A_{j,i}(t) + E_{i,j}(t-1) > 0\}$$

as the set of neighbours of i to which i may send tokens in round t. Let

$$L_i(t) := N(i) - H_i(t)$$

be the rest of i's neighbours.

Lemma 10. *Suppose there is a constant $\kappa > 0$ so that $T^A \leqslant n^\kappa$. Then for any constant $\alpha > 0$ there is a constant $c(\kappa, \alpha) > 0$ such that for any node i and round $t \leqslant T^A$, each of the following events occurs with probability at most $(n^{\alpha+1} \cdot T^A)^{-1}$:*

(1) $|\sum_{j \in N(i)} E_{i,j}(t)| \geqslant c \cdot \sqrt{d \log n}$,

(2) $|\sum_{j \in H_i(t)} E_{i,j}(t)| \geqslant c \cdot \sqrt{d \log n}$,

(3) $\sum_{j \in L_i(t+1)} E_{i,j}(t) \leqslant -\frac{d}{4} - c \cdot \sqrt{d \log n}$.

Proof. First we prove statement (1). Define $\Delta := \sum_{j \in N(i)} E_{i,j}(t)$.

Assume $E_{i,j}(t-1)$ is fixed for all the edges (i,j). Then each of the random variables $E_{i,j}(t)$ can assume at most two different values and rounding up or down is independent of other edges (see part (3) of Observation 9). Let the random variable $\mathbf{e}_i(t)$ be a vector denoting the error values of the edges connected to i at the end of the round t. By the law of total probability we have:

$$\mathbf{Pr}\left[\, |\Delta| \geqslant \delta \,\right] = \sum_{\mathscr{E}_i} \mathbf{Pr}\left[\, |\Delta| \geqslant \delta \mid \mathbf{e}_i(t-1) = \mathscr{E}_i \,\right] \cdot \mathbf{Pr}\left[\, \mathbf{e}_i(t-1) = \mathscr{E}_i \,\right].$$

Note that each $E_{i,j}(t)$ is the random variable indicating the error in the randomized rounding of $\widehat{Y}_{i,j}(t)$ (part (3) of Observation 9). We can apply Lemma 12 in the appendix to bound Δ, which yields

$$\mathbf{Pr}[|\Delta| \geqslant \delta \mid \mathbf{e}_i(t-1) = \mathscr{E}_i] \leqslant 2\exp(-2\delta^2/d).$$

Hence,

$$\begin{aligned}\mathbf{Pr}\left[\, |\Delta| \geqslant \delta \,\right] &= \sum_{\mathscr{E}_i} \mathbf{Pr}\left[\, |\Delta| \geqslant \delta \mid \mathbf{e}_i(t-1) = \mathscr{E}_i \,\right] \cdot \mathbf{Pr}\left[\, \mathbf{e}_i(t-1) = \mathscr{E}_i \,\right] \\ &\leqslant \sum_{\mathscr{E}_i} 2\exp(-2\delta^2/d) \cdot \mathbf{Pr}\left[\, \mathbf{e}_i(t-1) = \mathscr{E}_i \,\right] \\ &= 2\exp(-2\delta^2/d).\end{aligned}$$

As $T^A \leqslant n^\kappa$, setting $\delta = c \cdot \sqrt{d \log n} \geqslant \sqrt{d \log(2n^2 T^A)/2}$ for some constant c yields the desired bound.

The proof of statement (2) is similar to the proof of (1). Here we define

$$\Delta := \sum_{j \in H_i(t)} E_{i,j}(t).$$

Recall the definition of $H_i(t) = \{j \in N(i) : y^A_{i,j}(t) - y^A_{j,i}(t) + E_{i,j}(t-1) > 0\}$. Observe that $|H_i(t)| \leqslant d$. Conditioned on $\mathbf{e}_i(t-1) = \mathscr{E}_i$, the set $H_i(t)$ is fixed. Hence we can apply Lemma 12 to obtain:

$$\mathbf{Pr}\left[|\Delta| \geqslant \delta \mid \mathbf{e}_i(t-1) = \mathscr{E}_i\right] \leqslant 2\exp(-2\delta^2/d).$$

Following the same steps as in (1) we get the desired result.

Now we proceed with the proof of statement (3). Recall that $L_i(t+1) = \{j \in N(i) : y^A_{i,j}(t+1) - y^A_{j,i}(t+1) + E_{i,j}(t) \leqslant 0\}$, so intuitively the set is biased toward containing lower values of $E_{i,j}(t)$. However, we can change the summation and use different random variables so that we can still apply Hoeffding's bounds. Define $E^-_{i,j}(t) := \min\{E_{i,j}(t), 0\}$. We have:

$$\sum_{j \in N(i)} E^-_{i,j}(t) \leqslant \sum_{j \in L_i(t+1)} E_{i,j}(t). \tag{19}$$

Fix an arbitrary node i, let $\Delta = \sum_{j \in N(i)} E^-_{i,j}(t)$ and $p_{ij} = \{\widehat{Y}_{i,j}(t)\}$. We have:

$$\mathbf{Ex}\left[E^-_{i,j}(t) \;\middle|\; \mathbf{e}_i(t-1) = \mathscr{E}_i\right] = -p_{ij} \cdot (1 - p_{ij}) + 0 \cdot p_{ij} \geqslant -1/4.$$

The last step follows from a simple minimization of $f(p_{ij}) = (1 - p_{ij}) + 0 \cdot p_{ij}$. Conditioned on a fixed $\mathbf{e}_i(t-1)$, the random variables $E^-_{i,j}(t)$ are independent ranging over intervals of length no more than one. Again, we can apply Hoeffding's bounds using $\delta = c \cdot \sqrt{d \log n} \geqslant \sqrt{d \log(2n^{\alpha+1} T^A)/2}$ with the same constant c as in the previous parts. Hence,

$$\mathbf{Pr}\left[\Delta < -\frac{d}{4} - c \cdot \sqrt{d \log n} \;\middle|\; \mathbf{e}_i(t-1) = \mathscr{E}_i\right] \leqslant \frac{1}{2n^{\alpha+1} \cdot T^A}. \tag{20}$$

By Equation (19), we can replace Δ with $\sum_{j \in L_i(t+1)} E_{i,j}(t)$ in the Equation (20) to obtain the desired bound. □

The next lemma provides the two main ingredients for proving the Theorem 8. First, the discrete process stays close to the continuous process; and second, that this happens without accessing the infinite sources.

Lemma 11. *Suppose there is a constant $\kappa > 0$ so that $T^A \leqslant n^\kappa$. Further suppose $x^A(0) = \mathbf{x}' + \mathbf{x}''$ such that $\mathbf{x}'' = (d/4 + 2c \cdot \sqrt{d \log n}) \cdot (s_1, \ldots, s_n)$, and A does not induce negative load on $\mathbf{x}'$. Then for any constant $\alpha > 0$ there is a constant $c(\kappa, \alpha) > 0$ such that the following holds:*

(1) $\mathbf{Pr}\Big[\ \big|\sum_{j\in N(i)} E_{i,j}(t)\big| \leqslant c\cdot\sqrt{d\log n}$ *holds for all* $i\in V$ *and* $t\leqslant T^A\ \Big] \geqslant 1-n^{-\alpha}$,

(2) No infinite source is used, i.e.,
$\mathbf{Pr}\Big[\ X_i^{\mathfrak{D}(A)}(t) - \sum_{j\in H_i(t)} Y_{i,j}^{\mathfrak{D}(A)}(t) \geqslant 0$ *holds for all* $t\leqslant T^A$ *and* $i\in V\ \Big] \geqslant 1-2n^{-\alpha}$.

Proof. To prove statement (1), we choose the value of c as computed by Lemma 10 for the same value of α and $t_1 = T^A$. The proof of the first statement follows by applying the union bound to part (1) of Lemma 10.

To prove statement (2), we need to show that the load of every node i at the beginning of every round $t\leqslant T^A$ is large enough to satisfy their outgoing demands. We prove by contradiction, assuming there is a first round $t'\leqslant T^A$ in which some node i has insufficient load. Recall that $H_i(t)$ is defined so that no load is transferred from i to any of its neighbours not in $H_i(t)$. In the following we prove that $X_i^{\mathfrak{D}(A)}(t') - \sum_{j\in H_i(t')} Y_{i,j}^{\mathfrak{D}(A)}(t') \geqslant 0$, contradicting the initial assumption that i does not have sufficient load in round t'.

$$\begin{aligned}
& X_i^{\mathfrak{D}(A)}(t') - \sum_{j\in H_i(t')} Y_{i,j}^{\mathfrak{D}(A)}(t') \\
&= X_i^{\mathfrak{D}(A)}(t') - \sum_{j\in H_i(t')} \big(y_{i,j}^A(t') - y_{j,i}^A(t') + E_{i,j}(t'-1) - E_{i,j}(t')\big) && (21)\\
&= x_i^A(t') + \sum_{j\in N(i)} E_{i,j}(t'-1) - \sum_{j\in H_i(t')} \big(y_{i,j}^A(t') - y_{j,i}^A(t') + E_{i,j}(t'-1) - E_{i,j}(t')\big) && (22)\\
&= x_i^A(t') - \sum_{j\in H_i(t')} \big(y_{i,j}^A(t') - y_{j,i}^A(t')\big) + \sum_{j\in L_i(t')} E_{i,j}(t'-1) + \sum_{j\in H_i(t')} E_{i,j}(t') \\
&\geqslant s_i\cdot(d/4 + 2c\cdot\sqrt{d\log n}) + \sum_{j\in L_i(t')} E_{i,j}(t'-1) + \sum_{j\in H_i(t')} E_{i,j}(t'), && (23)
\end{aligned}$$

where in Equation (21) we use parts (1) and (2) of the Observation 9, Equation (22) follows from the part (1) of the Lemma 6 using the fact that no infinite source is used before the round t'. Also, Equation (23) is obtained by applying Lemma 2 using $\ell = (d/4 + 2c\cdot\sqrt{d\log n})$.

To complete the proof, it suffices to consider $s_i \geqslant 1$ and apply the parts (2) and (3) of the Lemma 10 to the above equation using the union bound. $\square$

We are now ready to prove the Theorem 8.

Proof. First we prove part (2). Suppose $x^A(0) = \mathbf{x}' + \mathbf{x}''$ such that $\mathbf{x}'' = (d/4 + 2c\cdot\sqrt{d\log n})\cdot(s_1,\ldots,s_n)$, and A does not induce negative load on $\mathbf{x}'$. Consider an arbitrary constant $\alpha > 0$, and let $c = 2c'$ where c' is the constant computed in Lemma 11 using the same α. Applying the union bound to combine both parts of the Lemma 11 we get with probability of at least $1-3n^{-\alpha}$ that no infinite source is ever used and that $|\sum_{j\in N(i)} E_{i,j}(t)| < c'\cdot\sqrt{d\log n}$. As no infinite source is used, by part (1) of the Lemma 6 [10] we also get $|X_i^{\mathfrak{D}(A)}(t) - x_i^A(t)| < c'\cdot\sqrt{d\log n}$.

[10]It is easy to see that Lemma 6 also holds for the randomized scheme.

On the other hand, using the definition of the balancing time we get $|x_i^A(t) - W \cdot s_i/S| \leqslant 1$. Hence, we can conclude that

$$|X_i^{\mathfrak{D}(A)}(t) - W \cdot s_i/S| < c' \cdot \sqrt{d \log n} + 1.$$

Since $s_i \geqslant 1$, we have $|X_i^{\mathfrak{D}(A)}(t)/s_i - W/S| < c' \cdot \sqrt{d \log n} + 1$ which holds for every node i. Hence, for any pair of nodes i, j we get

$$|X_i^{\mathfrak{D}(A)}(t)/s_i - X_j^{\mathfrak{D}(A)}(t)/s_j| < 2c' \cdot \sqrt{d \log n} + 2,$$

yielding the desired max-min discrepancy bound.

To get the bound of part (1), the algorithm first adds $(d/4 + 2c \cdot \sqrt{d \log n}) \cdot s_i$ *dummy* unit weight tasks to each resource i before the process begins. Note that this does not affect the convergence time of the continuous process, because the extra load is completely balanced. In the rest of the proof, we use x to refer to the new load vectors. Let W' and W denote the original and the new total load, respectively. We have: $W = W' + (d/4 + 2c \cdot \sqrt{d \log n}) \cdot S$. Hence,

$$W/S \leqslant W'/S + d/4 + 2c \cdot \sqrt{d \log n}.$$

At the end, the dummy tokens can be simply ignored. Though, we can still use $X_i^{\mathfrak{D}(A)}(t)$ as an upper bound on the final load of the node i excluding the dummy tokens. Following steps similar to the max-min discrepancy case, we get $X_i^{\mathfrak{D}(A)}(t)/s_i - W/S < d/4 + \mathscr{O}(\sqrt{d \log n})$, which yields the desired max-avg discrepancy bound. □

References

[1] H. Ackermann, P. Berenbrink, S. Fischer, and M. Hoefer. Concurrent imitation dynamics in congestion games. In *PODC*, pages 63–72. ACM, 2009.

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

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

[4] W. Aiello, B. Awerbuch, B. Maggs, and S. Rao. Approximate load balancing on dynamic and asynchronous networks. In *STOC*, pages 632–641. ACM, 1993.

[5] H. Akbari and P. Berenbrink. Parallel rotor walks on finite graphs and applications in discrete load balancing. In *SPAA*, page to appear. ACM, 2013.

[6] H. Akbari, P. Berenbrink, and T. Sauerwald. A simple approach for adapting continuous load balancing processes to discrete settings. In *PODC*, pages 271–280, 2012.

[7] J. Aspnes, M. Herlihy, and N. Shavit. Counting networks. *J. ACM*, 41(5):1020–1048, 1994.

[8] P. Berenbrink, T. Friedetzky, and Z. Hu. A new analytical method for parallel, diffusion-type load balancing. *J. Parallel Distrib. Comput.*, 69(1):54–61, 2009.

[9] P. Berenbrink, C. Cooper, T. Friedetzky, T. Friedrich, and T. Sauerwald. Randomized diffusion for indivisible loads. In *SODA*, pages 429–439. SIAM, 2011.

[10] P. Berenbrink, M. Hoefer, and T. Sauerwald. Distributed selfish load balancing on networks. In *SODA*, pages 1487–1497. SIAM, 2011.

[11] P. Berenbrink, T. Friedetzky, I. Hajirasouliha, and Z. Hu. Convergence to equilibria in distributed, selfish reallocation processes with weighted tasks. *Algorithmica*, 62(3–4):767–786, 2012.

[12] J. E. Boillat. Load balancing and poisson equation in a graph. *Concurrency and Computation: Practice and Experience*, 2:289–314, 1990.

[13] J. N. Cooper, B. Doerr, J. Spencer, and G. Tardos. Deterministic random walks on the integers. volume 28, pages 2072–2090. Academic Press Ltd., 2007.

[14] J. N. Cooper, B. Doerr, T. Friedrich, and J. Spencer. Deterministic random walks on regular trees. *Random Struct. Algorithms*, 37(3):353–366, 2010.

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

[16] B. Doerr and T. Friedrich. Deterministic random walks on the two-dimensional grid. *Comb. Probab. Comput.*, 18(1-2):123–144, 2009.

[17] D. Dubhashi and D. Ranjan. Balls and bins: A study in negative dependence. *Random Structures & Algorithms*, 13:99–124, 1996.

[18] R. Elsässer and B. Monien. Load balancing of unit size tokens and expansion properties of graphs. In *SPAA*, pages 266–273, 2003.

[19] R. Elsässer and T. Sauerwald. Discrete load balancing is (almost) as easy as continuous load balancing. In *PODC*, pages 346–354, 2010.

[20] R. Elsässer, B. Monien, and R. Preis. Diffusion schemes for load balancing on heterogeneous networks. *Theory Comput. Syst.*, 35(3):305–320, 2002.

[21] R. Elsässer, B. Monien, and S. Schamberger. Distributing unit size workload packages in heterogeneous networks. *J. Graph Algorithms Appl.*, 10(1):51–68, 2006.

[22] E. Even-Dar and Y. Mansour. Fast convergence of selfish rerouting. In *SODA*, pages 772–781. SIAM, 2005.

[23] E. Even-Dar, A. Kesselman, and Y. Mansour. Convergence time to nash equilibrium in load balancing. *ACM Trans. Algorithms*, 3(3), 2007.

[24] T. Friedrich and T. Sauerwald. Near-perfect load balancing by randomized rounding. In *STOC*, pages 121–130, 2009.

[25] T. Friedrich and T. Sauerwald. The cover time of deterministic random walks. *Electr. J. Comb.*, 17(1), 2010.

[26] T. Friedrich, M. Gairing, and T. Sauerwald. Quasirandom load balancing. *SIAM J. Comput.*, 41(4):747–771, 2012.

[27] B. Ghosh and S. Muthukrishnan. Dynamic load balancing by random matchings. *J. Comput. Syst. Sci.*, 53:357–370, 1996.

[28] B. Ghosh, F. T. Leighton, B. M. Maggs, S. Muthukrishnan, C. G. Plaxton, R. Rajaraman, A. W. Richa, R. E. Tarjan, and D. Zuckerman. Tight analyses of two local load balancing algorithms. *SIAM J. Comput.*, 29(1):29–64, 1999.

[29] W. Hoeffding. Probability inequalities for sums of bounded random variables. *J. Parallel Distrib. Comput.*, 58(301):13–30, 1963.

[30] S. H. Hosseini, B. E. Litow, M. I. Malkawi, and K. Vairavan. Distributed algorithms for load balancing in very large homogeneous systems. In *Proc. of the 1987 Fall Joint Computer Conference on Exploring technology: today and tomorrow*, ACM '87, pages 397–404. IEEE Computer Society, 1987.

[31] S. H. Hosseini, B. Litow, M. Malkawi, J. McPherson, and K. Vairavan. Analysis of a graph coloring based distributed load balancing algorithm. *J. Parallel Distrib. Comput.*, 10(2):160–166, 1990.

[32] S. Kijima, K. Koga, and K. Makino. Deterministic random walks on finite graphs. In *ANALCO*, pages 16–25, 2012.

[33] F. Meyer auf der Heide, B. Oesterdiekhoff, and R. Wanka. Strongly adaptive token distribution. *Algorithmica*, 15(5):413–427, 1996.

[34] S. Muthukrishnan, B. Ghosh, and M. H. Schultz. First- and second-order diffusive methods for rapid, coarse, distributed load balancing. *Theory Comput. Syst.*, 31 (4):331–354, 1998.

[35] A. Panconesi and A. Srinivasan. Improved distributed algorithms for coloring and network decomposition problems. In *STOC*, pages 581–592. ACM, 1992.

[36] A. Panconesi and A. Srinivasan. Randomized distributed edge coloring via an extension of the chernoff–hoeffding bounds. *SIAM J. Comput.*, 26(2):350–368, 1997.

[37] Y. Rabani, A. Sinclair, and R. Wanka. Local divergence of markov chains and the analysis of iterative load-balancing schemes. In *FOCS*, pages 694–703. IEEE Computer Society, 1998.

[38] T. Sauerwald and H. Sun. Tight bounds for randomized load balancing on arbitrary network topologies. In *FOCS*, pages 341–350. IEEE Computer Society, 2012.

[39] R. Subramanian and I. D. Scherson. An analysis of diffusive load-balancing. In *SPAA*, pages 220–225. ACM, 1994.

A Hoeffding's Bound: Adaptation for Randomized Rounding

Lemma 12. *Let $X_1, \ldots, X_k$ be k independent random variables, and $p_1, \ldots, p_k$ be constants where for all $i, 0 < |p_i| < 1$. Suppose X_i is $p_i - 1$ with probability p_i and p_i otherwise. If we define the random variable $X = \sum_{1 \leqslant i \leqslant k} X_i$, then we have for any $\delta > 0$ that*

$$\mathbf{Pr}\left[\ |X| \geqslant \delta\ \right] \leqslant 2\exp\left(-2\delta^2/k\right).$$

Proof. We first note that for each i, we have

$$\mathbf{Ex}\left[X_i\right] = (p_i - 1) \cdot p_i + p_i \cdot (1 - p_i) = 0;$$

Hence, by the linearity of expectation we get $\mathbf{Ex}\left[X\right] = \sum_{1 \leqslant i \leqslant k} \mathbf{Ex}\left[X_i\right]$. Also, for each i, $p_i - 1 \leqslant X_i \leqslant p_i$. Therefore, we can apply the Hoeffding's bound [29] to get $\mathbf{Pr}\left[\ |X| \geqslant \delta\ \right] \leqslant 2\exp\left(-2\delta^2/k\right)$, as required. $\square$

B Comparison Tables

Table 1: **Final max-min discrepancy of our algorithms compared to other discrete diffusion processes for different graph classes.** The running time of each process is $T = \mathcal{O}(\frac{\log Kn}{1-\lambda})$.

Discrete Processes	**Arbitrary Graphs**	**Expanders with $d = \mathcal{O}(1)$**	**Hypercubes**	**r-dim tori $r = \mathcal{O}(1)$**
Deterministic Rounding				
Rabani et al. [37]	$\mathcal{O}\left(\frac{d \log n}{1-\lambda}\right)$	$\mathcal{O}(\log n)$	$\mathcal{O}(\log^2 n)$	$\mathcal{O}(n^{1/r})$
Friedrich et al. [26] (deterministic)	–	–	$\mathcal{O}(\log^{3/2} n)$	$\mathcal{O}(1)$
Alg. 1 (Theorem 3)	$\mathcal{O}(d)$	$\mathcal{O}(1)$	$\mathcal{O}(\log n)$	$\mathcal{O}(1)$
Randomized Rounding				
Friedrich et al. [26] (randomized)	$\mathcal{O}\left(\frac{d \log\log n}{1-\lambda}\right)$	$\mathcal{O}(\log\log n)$	–	–
Berenbrink et al. [9]	$\mathcal{O}\left(\frac{d \log\log n}{1-\lambda}\right)$, and $\mathcal{O}\left(d\sqrt{\log n} + \sqrt{\frac{\log n \log d}{1-\lambda}}\right)$	$\mathcal{O}(\log\log n)$	$\mathcal{O}(\log n)$	$\mathcal{O}(\sqrt{\log n})$
The algorithm of [9] analyzed using [38]	$\mathcal{O}(d^2\sqrt{\log n})$	$\mathcal{O}(\sqrt{\log n})$	$\mathcal{O}(\log^{3/2} n)$	$\mathcal{O}(\sqrt{\log n})$
The algorithm of [26] analyzed using [38]	$\mathcal{O}(\sqrt{d \log n})$	$\mathcal{O}(\sqrt{\log n})$	$\mathcal{O}(\log n)$	$\mathcal{O}(\sqrt{\log n})$
Alg. 2 (Theorem 8)	$\mathcal{O}(\sqrt{d \log n})$	$\mathcal{O}(\sqrt{\log n})$	$\mathcal{O}(\log n)$	$\mathcal{O}(\sqrt{\log n})$

Table 2: **Final max-min discrepancy of our algorithms compared to other discrete processes in the matching model.** The running time of each process is $t = T$ unless otherwise specified.

Discrete Processes	**Arbitrary Graphs**	**Expanders with $d = \mathcal{O}(1)$**	**Hypercubes**	**r-dim tori $r = \mathcal{O}(1)$**
Periodic Matchings				
Round-Down				
Rabani et al. [37]	$\mathcal{O}\left(\frac{d\log n}{1-\lambda}\right)$	$\mathcal{O}(\log n)$	$\mathcal{O}(\log^2 n)$	$\mathcal{O}(n^{1/r})$
Randomized Rounding				
Friedrich and Sauerwald [24]	$\mathcal{O}\left(\frac{d\log\log n}{1-\lambda}\right)$, and $\mathcal{O}\left(\sqrt{\frac{d\log n}{1-\lambda}}\right)$	$\mathcal{O}(\log\log n)$, and $\mathcal{O}(1)^{\dagger}$	$\mathcal{O}(\log^{3/2} n)$	$\mathcal{O}(n^{1/2r}\sqrt{\log n})$
Sauerwald and Sun [38]	$\mathcal{O}(\log^{\varepsilon} n)^{*}$, and $\mathcal{O}(\log\log n)^{\P}$	$\mathcal{O}(1)^{*}$	$\mathcal{O}(\log^{\varepsilon} n)^{*}$	$\mathcal{O}(1)^{*}$
Random Matchings				
Round-Down				
Rabani et al. [37]	$\mathcal{O}\left(\frac{d\log n}{1-\lambda}\right)$	$\mathcal{O}(\log n)$	$\mathcal{O}(\log^2 n)$	$\mathcal{O}(n^{1/r})$
Randomized Rounding				
Friedrich and Sauerwald [24]	$\mathcal{O}\left(\sqrt{\frac{\log^3 n}{1-\lambda}}\right)$	$\mathcal{O}(1)^{\dagger}$	$\mathcal{O}(\log^2 n)$	$\mathcal{O}(n^{1/2r}\log n)$
Sauerwald and Sun [38]	$\mathcal{O}(\log^{\varepsilon} n)^{*}$, and $\mathcal{O}(\log\log n)^{\P}$	$\mathcal{O}(1)^{*}$	$\mathcal{O}(1)^{*}$	$\mathcal{O}(1)^{*}$
Periodic/Random Matchings				
Alg. 1: Round-Down	$\mathcal{O}(d)$	$\mathcal{O}(1)$	$\mathcal{O}(\log n)$	$\mathcal{O}(1)$
Alg. 2: Randomized Rounding	$\mathcal{O}(\sqrt{d\log n})$	$\mathcal{O}(\sqrt{\log n})$	$\mathcal{O}(\log n)$	$\mathcal{O}(\sqrt{\log n})$

* Unlike other probabilistic bounds that hold with probability $1 - n^{-\Omega(1)}$, these bounds hold with probability $1 - \exp(-(\log n)^c)$, for some $c < 1$

† in $t = \mathcal{O}(T \cdot \log^3 \log n)$ rounds

¶ in $t = \mathcal{O}(T \cdot \log\log n)$ rounds