Analyzing an Infinite Parallel Job Allocation Process

Micah Adler[1], Petra Berenbrink[2], and Klaus Schröder[3]

[1] Department of Computer Science, University of Toronto, Canada
micah@cs.toronto.edu †
[2] Department of Mathematics and Computer Science, Paderborn University, Germany
pebe@uni-paderborn.de ‡
[3] Heinz Nixdorf Institute and Department of Mathematics and Computer Science, Paderborn University, Germany
ellern@hni.uni-paderborn.de §

Abstract. In recent years the task of allocating jobs to servers has been studied with the "balls and bins" abstraction. Results in this area exploit the large decrease in maximum load that can be achieved by allowing each job (ball) a very small amount of choice in choosing its destination server (bin). The scenarios considered can be divided into two categories: *sequential*, where each job can be placed at a server before the next job arrives, and *parallel*, where the jobs arrive in large batches that must be dealt with simultaneously. Another, orthogonal, classification of load balancing scenarios is into *fixed time* and *infinite*. Fixed time processes are only analyzed for an interval of time that is known in advance, and for all such results thus far either the number of rounds or the total expected number of arrivals at each server is a constant. In the infinite case, there is an arrival process and a deletion process that are both defined over an infinite time line.

In this paper, we present an algorithm for allocating jobs arriving in parallel over an infinite time line. While there have been several results for the infinite sequential case, no analogous results exist for the infinite parallel case. We consider the process where m jobs arrive in each round to n servers, and each server is allowed to remove one job per round. We introduce a simple algorithm, where it is sufficient for each job to choose between 2 random servers, that obtains the following result: if $m \leq \frac{n}{6e}$, then for any given round, the probability that any job has to wait more than $\mathcal{O}(\log\log n)$ rounds before being processed is at most $1/n^{\alpha}$ for any constant α. Furthermore, we analyze the distribution on waiting times: with the same probability, the number of jobs of any given round that have to wait $t + c$ rounds to be processed is at most $\mathcal{O}(\frac{n}{2^{(2^t)}})$ for a small constant c. These results are comparable with existing results for the infinite sequential case.

† Supported by an operating grant from the Natural Sciences and Engineering Research Council of Canada, and by ITRC, an Ontario Centre of Excellence. This research was conducted in part while he was at the Heinz Nixdorf Institute Graduate College, Paderborn, Germany.

‡ Supported by DFG-SFB 376 "Massive Parallelität", and by EU ESPRIT Long Term Reseach Project 20244 (ALCOM-IT).

§ Supported by the DFG-Graduiertenkolleg "Parallele Rechnernetzwerke in der Produktionstechnik", by DFG-SFB 376 "Massive Parallelität", and by EU ESPRIT Long Term Reseach Project 20244 (ALCOM-IT).

1 Introduction

Dynamic resource allocation problems have seen much attention in recent years. In these problems, the objective is to distribute a set of jobs across a set of servers as evenly as possible, with a minimum of coordination between jobs and servers. Often, this problem is stated in terms of balls and bins; such occupancy results have several applications in load balancing, hashing, and PRAM simulation [KLM92], [ABKU94], [ACMR95]. A simple distributed algorithm for allocating jobs to servers is to place each job at a random server. This requires no coordination, but it is well known that if there are n jobs and n servers, it is likely that one of the servers receives $\mathcal{O}(\frac{\log n}{\log \log n})$ jobs. However, [ABKU94] demonstrated that a small amount of choice in possible servers for each job can lead to as much as an exponential decrease in the maximum imbalance, without prohibitively increasing coordination.

Some of the work in this area [ABKU94,CS97] can be described as *sequential*. In such a scenario, jobs arrive one at a time, and each is placed at a server prior to the arrival of the subsequent job. Results for such a scenario are rather limited in their applicability to parallel and distributed settings, and thus much of the work on allocation processes has concentrated on the *parallel* scenario ([ACMR95,Mit96,St96,BMS97], [Mit97]), where the jobs arrive in large batches that must be processed simultaneously.

The seminal paper on the sequential scenario [ABKU94] considered both a *fixed time* (which they call finite) process, and an *infinite* process. Fixed time processes are only analyzed for a fixed interval of time that is known in advance, where infinite processes consist of a job arrival process and a deletion process that are both defined over an infinite time line. Despite a large subsequent body of work on the parallel scenario as well as further work on infinite processes in the sequential scenario [CS97], up to now there has been no analysis of an infinite process for the parallel scenario. In fact, for all previous parallel results, either the number of rounds or the total expected number of arrivals at each server is a constant. However, most long running systems do in fact process a large number of jobs at each server.

In this paper, we present the first results for jobs arriving in parallel over an infinite time line. We consider a model where m jobs arrive in each synchronized round to n servers, and each server is allowed to complete one job per round. The *waiting time* of a job is the number of rounds the job stays in the system. Our goal is to assign the jobs to the servers in such a way that we minimize both the expected waiting time, as well as the maximum waiting time of the jobs that arrive during any given round.

1.1 New Results

In this paper we introduce and analyze a simple infinite process allocating jobs to servers. We assume that the system starts empty. During each round, every job arriving to the system sends requests to $d \geq 2$ independently and uniformly at random (i.u.r.) chosen servers. Each of the servers stores all of its requesting jobs in a First-in-First-out (FIFO) queue. During each round, the server performs the first job of its queue and deletes the other requests of that job.

We provide bounds on the performance of this process. It is not hard to show that, when $m < \frac{n}{2de}$, the expected waiting time of any job is a constant number of rounds.

Here e is the base of the natural logarithm. It is much more difficult to analyze the entire distribution on the waiting time. The main results of this paper are the following two theorems:

Theorem 1. *Let c be $\frac{2\alpha+1}{\log(6em/n)} + 1$ and $m < \frac{n}{3de}$. For any constant $\alpha > 0$:*
Each of the jobs generated in any fixed round T_j has waiting time smaller than $\frac{\log\log n}{\log d} + 2\alpha + c$ with probability at least $1 - \frac{1}{n^\alpha}$.

Note that this result implies the stability of the considered process: the number of jobs in the system does not grow to infinity.

Theorem 2. *For $c \geq 6\alpha/\log\left(\frac{n}{6e \cdot m}\right) + 1$, $\max\{5, \log(12\alpha), c\} \leq t' \leq \log\log n - 3\alpha$, and $m \leq \frac{n}{6de^{1+1/e}}$ it holds for any constant $\alpha > 0$:*
At most $\frac{n}{2^{(d^{t'})}}$ of the m jobs generated in any fixed round T_j have waiting time $t \geq t' + c$ with probability at least $1 - \frac{1}{n^\alpha}$.

Here, and throughout the paper, we say *with high probability* (w.h.p.) to denote with probability at least $1 - \frac{1}{n^\alpha}$ for any constant $\alpha > 0$. It is easy to show that the same result holds if the jobs are generated by n generators which can be arbitrarily distributed over the processors (see [SV96]). Each generator is allowed to produce a job with a probability smaller than $\frac{1}{2de}$ per round. It is also possible to use generators with different generation probabilities if the expected overall generated load is smaller than $\frac{1}{2de}$. Our results also hold if the jobs are generated asynchronously, for instance by a Poisson Process.

The analysis of the process is performed using a type of delay sequence argument, using a structure known as a witness tree [MSS95]. To use this type of argument, we first show that every time the process fails a witness tree must exist. We then show that that it is very unlikely for a witness tree to occur. Usually the latter involves enumerating the set of possible witness trees and then bounding the probability that a given witness tree exists. In the infinite case analyzed here, this simple enumeration technique is impossible, since an unbounded number of jobs may appear, and the size of the witness trees that can exist cannot be bounded. We develop a method for using a witness tree argument in the analysis of an infinite process; this is the main technical contribution of our paper.

1.2 Previous Work

Azar et al. [ABKU94] examine a sequential protocol called *greedy process* to place n balls into n bins. For each ball they choose d bins i.u.r. and put the ball into the bin with minimum load at the time of placement. They show that after placing n balls the maximum load is $\Theta(\log\log(n)/\log(d) + 1)$, w.h.p. Furthermore, they provide a result for an infinite version of their sequential process: in the stationary distribution, the fullest bin contains less than $\log\log(n)/\log(d) + O(1)$ balls, w.h.p., where n is both the number of balls and bins in the system. The simple sequential game has many applications and is also used as an online algorithm for competitive Load Balancing (see [ABK94], [AKP+93], and [PW93]). Recently, Czumaj et al. ([CS97]) extended

the results in several directions. They present an adaptive process where the number of choices made in order to place a ball depends on the load of the previously chosen bins, and an off-line allocation process knowing the random choices in advance. Then, they consider a scenario allowing reassignments: During the allocation of a new ball one may ask for some number of possible locations and then arbitrarily distribute all balls (together with the new one) among the chosen bins. They show that a process with reassignments yields a maximum load that is never smaller by more than an constant factor than the maximum load of the process from [ABKU94].

Adler et al. [ACMR95] explore the problem in parallel and distributed settings for the case of placing n balls into n bins. They provide a lower bound for *non-adaptive* (possible destinations are chosen before any communication takes place) and *symmetric* algorithms (all balls and bins perform the same underlying algorithm, and all destinations are chosen i.u.r.). For any constant number $r \in \mathbb{N}$ of communication rounds, the maximum load is shown to be at least $\Omega\left(\sqrt[r]{\frac{\log n}{\log\log n}}\right)$. Additionally, they present parallelizations of the sequential strategy found in [ABKU94]. They give a two-round parallelization of the greedy process, matching the lower bound. Furthermore, they introduce a multiple-round strategy requiring $\log\log n + \mathcal{O}(1)$ rounds of communication and achieving maximum load $\log\log n + \mathcal{O}(1)$, w.h.p. Finally, they examine a strategy using a threshold T: In each of r communication rounds each non-allocated ball tries to access two bins chosen i.u.r. and each bin accepts up to T balls during each round. They show that with $T = \sqrt[r]{\frac{(2r+o(1))\cdot\log n}{\log\log n}}$ this algorithm terminates after r rounds with maximum load $r \cdot T$, w.h.p.

Stemann [St96] extends the results for the case where the number of balls m is larger than the number n of bins. For $m = n$, he analyzes a very simple class of algorithms achieving maximum load $\mathcal{O}\left(\sqrt[r]{\frac{\log n}{\log\log n}}\right)$ if r rounds of communication are allowed. This matches the lower bound presented in [ACMR95]. He generalizes the algorithm for $m > n$ balls and achieves the optimal load of $\mathcal{O}(\frac{m}{n})$ using $\frac{\log\log n}{\log(m/n)}$ rounds of communication, w.h.p., or load $\max\left\{\sqrt[r]{\log n}, \mathcal{O}\left(\frac{m}{n}\right)\right\}$ using r rounds of communication, w.h.p.

In [BMS97] the authors extend the lower bound of [ACMR95] to arbitrary $r \leq \log\log n$, implying that the result of Stemanns protocol is optimal for all r. Their main result is a generalization of Stemanns upper bound to weighted jobs: Let W^A (W^M) denote the average (maximum) weight of the balls and $\Delta = W^A/W^M$. The authors present a protocol which achieves maximum load of $\gamma \cdot \left(\frac{m}{n} \cdot W^A + W^M\right)$ using $\mathcal{O}\left(\frac{\log\log n}{\log(\gamma\cdot((m/n)\cdot\Delta+1))}\right)$ rounds of communication. In particular, for $\log\log n$ rounds the optimal load of $\mathcal{O}\left(\frac{m}{n} \cdot W^A + W^M\right)$ is achieved.

Mitzenmacher [Mit96] analyzes a dynamic but fixed time allocation strategy: Customers (balls) arrive as a Poisson stream of rate λn, $\lambda < 1$, at a collection of n servers (bins). Each customer chooses d servers i.u.r. and joins the server with the fewest customers. Customers are served according to the First-Come-First-Serve protocol, and the service time is exponentially distributed with mean one. He calls his model the supermarket model. For a time interval of fixed and constant length T, he shows the expected waiting time to be $\mathcal{O}(1)$ for $N \to \infty$, and the maximum queue length to be

$\mathcal{O}(\log\log n + o(1))$, w.h.p. His analysis makes use of deep results of Kurtz ([Kur81]) on so called density dependent Markov Chains. In [Mit97] the author extends his results to several different load generation and consumption schemes. For example, he analyzes the same process with constant service times, the customers having a different number of choices, and bounded queue lengths. However, the results of Mitzenmacher are only valid in the case of time intervals of constant length.

2 The Infinite Allocation Process

During each round, m jobs enter the system; the running time of the jobs is equal to the duration of one round. Each job J sends a request to d i.u.r. chosen servers $B_1(J), \ldots, B_d(J)$. We say these d servers *hold a request* of J, and $B_i(J)$ is called the i-th request of J. Each request of job J is marked with a unique job identifier, the number of the round J enters the system (this round is called *entry round* of J in the following), and a list of all d servers holding a request of J.

The server stores all incoming requests in a First-in-First-out (FIFO) queue, jobs belonging to the same round are stored in arbitrary order. During each round, each non-empty server deletes the first request of its FIFO-queue and performs the work for that job. Additionally, each server sends a *deletion message* to the other servers holding a request of the finished job, and each server deletes the requests corresponding to a deletion message from its queue. The infinite allocation process is given in Figure 1.

Infinite Allocation Process

For all jobs J_i generated during round t **do in parallel**
-Choose i.u.r. d servers $B_1(J_i), \ldots, B_d(J_i)$.
-Send a request (t, J_i, $B_1(J_i), \ldots, B_d(J_i)$) to each chosen server.
For all servers B_j **do in parallel**
-Insert incoming requests in an arbitrary order into the FIFO queue.
-Delete the first job J_{B_j} from the queue and send a deletion message to each server holding a request of J_{B_j}.
-Finish the work on the job.

Fig. 1. The Infinite Allocation Process

3 Bounding the Maximum Waiting Time

In this section we prove Theorem 1 estimating the waiting time of a job entering the system in any fixed round T_j. For ease of presentation we state the proof only for the case $d = 2$, the proof for $d \geq 2$ appears in a full version of this paper.

Job J' *weakly-blocks* job J in round T, if J and J' send requests to a server B, and B performs J' during round T. If J' weakly-blocks J and J is performed in round $T+1$ (maybe by another server than B), J' is said to *block* J. We make use of a *delay tree* built of several *delay sequences*. A delay sequence consists of a sequence of jobs, each job (weakly-) blocks his predecessor in the sequence. A delay tree is a (not necessarily

complete) 2-ary tree whose nodes represent jobs. The jobs on any branch from the root to a leaf form a delay sequence. We show that a delay tree with depth $t+1$ has to exist if there is a job J with waiting time at least t. The root of the tree represents J and the branches of the tree represent delay sequences, i.e. a sequence of jobs which have blocked their predecessors respectively. We show that it is very unlikely that such a delay tree exists.

Let J be a job generated during round T_j which is not performed after round $T = T_j + t$ with $t = \log\log n + 2\alpha + c$. In round T, J is weakly-blocked by two jobs J_1 and J_2. J_1 and J_2 are generated not later than round T_j and wait at least $t-1$ steps to be performed (according to the FIFO-rule used by the servers to choose one of their jobs). In turn, J_1 (J_2) is blocked by two jobs J_1^1 and J_1^2 (J_2^1 and J_2^2) during round $T-1$. (Since J_1 and J_2 are performed in round T they are *blocked* rather than being only *weakly-blocked*.) Clearly, J_1^1, J_1^2, J_2^1 and J_2^2 wait at least $t-2$ steps to be performed. Thus, the construction can be carried on until jobs without waiting time are reached, these jobs are performed in the same round as they have been generated. Note that a blocked job J performed at server B is in general blocked by two jobs J_1, J_2 (J_1 at server $B_1(J)$ and J_2 at $B_2(J)$). Clearly, we have $B_1(J) = B$ or $B_2(J) = B$. To simplify our further discussion we assume in the following that the left child of v represents $B_1(J)$ if $B_1(J) = B$, and $B_2(J)$, otherwise.

The construction above yields a delay tree D_t of height at least $t+1$, the root V of the tree represents the job J, the two children of the root represent J_1 and J_2. The children of J_1 (J_2) represent J_1^1 and J_1^2 (J_2^1 and J_2^2), and so on. Thus, the nodes of D_t represent jobs blocking the job represented by their predecessor. Each branch of a delay tree D_t starting at the root has length at least t. This holds as each server with a non-empty queue performs a request and J has waiting time at least t.
As a consequence of the construction above, we state Lemma 1.

Lemma 1. *If a job J has waiting time at least t rounds, there exists a delay tree D_t with root J.*

In the sequel we use a recursive decomposition of D_t into delay sequences different from the delay sequences used to define the delay tree. The first delay sequence S is defined by the branch P from the root to the leftmost leaf of D_t. Remove P's edges from D_t. Then D_t becomes a forest of trees, cf. Figure 2. The root of each resulting tree has degree one and is a node of P. We apply the construction recursively to these trees: The branch P' from the root of a new tree to the leftmost leaf of that tree defines a delay sequence (see dotted lines in Figure 2). The edges of P' are removed generating a new forest to which the construction is applied again. This decomposition of the delay tree into delay sequences is called *regular decomposition*. The delay sequences of a regular decomposition are called *regular delay sequences*. In each regular delay sequence except the one starting at the root, the first edge is a right edge of D_t, and the other ones are left edges. Thus, the jobs of one regular delay sequence are all blocked in the same server.

Note that the regular delay sequences share some nodes as a result of the construction. Let the up-most node v of the branch defining a regular delay sequence S be called *top node* of S, and let the first job of S be the *top job* of S.

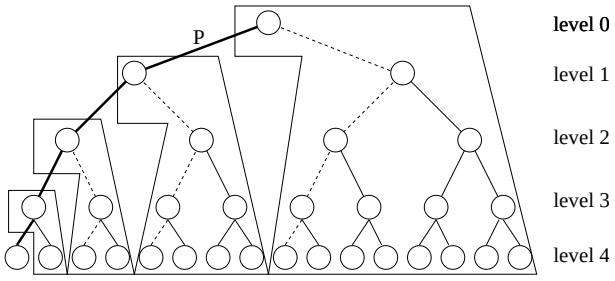

Fig. 2. recursive decomposition into delay trees and sequences

Let the length of a delay sequence S be the number of jobs in S minus one – i.e. we do not count the top job of S. Consider the set V of level $\log\log n + 2\alpha$ nodes of D_t. Each node $v \in V$ is on a path defining a regular delay sequence S_v of length at least c (recall that D_t has at least $\log\log n + 2\alpha + c$ complete levels). We call the regular delay sequences S_v with $v \in V$ *long delay sequences*. In the following we only examine the tree consisting of these long delay sequences.

Observation *There are* $2^{\log\log n + 2\alpha} = 2^{2\alpha} \cdot \log n$ *long delay sequences.*

The next Lemma summarizes some results concerning the delay trees and sequences built by the decomposition of a delay tree.

Lemma 2.

1. *A job occurring twice in a delay tree is represented by nodes of the same level of the tree, and jobs represented by nodes of a single delay sequence are distinct from each other.*
2. *The entry rounds of the jobs represented by a delay sequence are non-increasing, and the last job of a long delay sequence is performed in the same round it enters the system.*

Proof. 1. The nodes of level $i > 0$ of the delay tree represent jobs which are performed by a server during round $T - i + 1$, $1 \leq i \leq T$. If a job is performed by a server, its other requests are deleted in the same round. Thus, it is not possible that a job is performed twice during distinct rounds. This entails the fact that jobs belonging to a delay sequence are distinct from each other.

2. The entry rounds of jobs represented by a delay sequence are non-increasing because each job blocks the job represented by its predecessor, and the jobs are performed following the FIFO-rule. The construction of a branch in the delay tree ends reaching a job J' which is not blocked by another job. Thus, J' enters an empty FIFO queue and is performed in the same round as it enters the system.

We call two delay sequences S_1 and S_2 *pairwise distinct*, if every job of S_1 is either the top job of S_1 or S_2, or does not appear in S_2. The rest of the proof is divided into several Lemmas. We know that there exists a delay tree D_t with root J if a job J has

waiting time larger than t rounds. Then, either job J is the root of a delay tree with $\log n$ pairwise distinct long delay sequences, or it is the root of a delay tree with less than $\log n$ pairwise distinct long delay sequences. We show that both events are very unlikely. In order to estimate the probability that D_t consists of less than $\log n$ pairwise distinct long delay sequences, we have to upper bound the number of nodes of D_t. The next lemma upper bounds the probability that there exists a delay sequence S of length l with a fixed top job. The fixed top job will help us to estimate an upper bound for the range of the possible entry rounds of the jobs involved in S.

Lemma 3. *Let J_0 be the top job of a delay sequence S and T_0 be the entry round of J_0.*

Then it holds: The expected number of delay sequences with top job J_0 and length l is at most $\left(\frac{6e \cdot m}{n}\right)^l$.

Proof. Let $J_0, J_1, J_2, \ldots, J_l$ be the sequence of jobs of S, i.e. J_i is blocked by job J_{i+1}. Let T_i be the entry round of the i-th job, let $\bar{T}_i \geq T_i$ be the round where J_i is performed, and let $\Delta = T_0 - T_l$. There are $\binom{\Delta+l-1}{l-1}$ possible ways to choose the $l-1$ entry rounds of the jobs in a way that they are not increasing. If the entry rounds of the jobs are fixed, there are at most m^l ways to choose l jobs of the delay sequence. Thus, the number of possible delay sequences is bounded by

$$\binom{\Delta+l-1}{l-1} \cdot m^l \leq \binom{\Delta+l}{l} \cdot m^l \leq \left(\frac{(\Delta+l)\cdot e}{l}\right)^l \cdot m^l.$$

Next, we bound Δ, the range of the entry rounds of the considered jobs. J_i is performed in round $\bar{T}_0 - i$, and thus $\bar{T}_l = T_l = \bar{T}_0 - l$. Hence $\Delta = T_0 - T_l \leq \bar{T}_0 - T_l = l$, and it follows that

$$\left(\frac{(\Delta+l)\cdot e}{l}\right)^l \cdot m^l \leq \left(\frac{2e\cdot l}{l}\right)^l \cdot m^l = (2e\cdot m)^l.$$

To bound the probability that there exists such a delay sequence we first choose whether the first or the second request of a job leads to the blocking. This yields 2^l possibilities. The probability that the chosen requests of J_i and J_{i+1}, $1 \leq i < l$ go to the same server is bounded by $\left(\frac{1}{n - r\cdot log\log n}\right)^l$ for a constant r. This holds because the number of servers involved in a delay sequence can be bounded by $r \cdot \log\log n$ (note: the servers which are involved in the delay sequences can only change in the first $\log\log n + 2\alpha$ levels of the tree). Thus, the probability that there exists a delay sequence of length l can be bounded by

$$(2e\cdot m)^l \cdot \left(\frac{2}{n - r\cdot\log\log n}\right)^l = \left(\frac{6e\cdot m}{n}\right)^l.$$

The following Corollary is a consequence of Lemma 3.

Corollary 1. *Let $\left(\frac{6e\cdot m}{n}\right) < 1$; let J_0 be a job. Then the expected number of delay sequences S with top job J_0 and length at least l is at most $\left(\frac{6e\cdot m}{n}\right)^{l-1}$.*

Note that in Lemma 3 and Corollary 1 the expected number of delay sequences is also an upper bound for the probability that such a sequence does exist. In the next lemma we prove that the number of nodes in the long delay sequences is bounded by $\mathcal{O}(\log^2 n)$, w.h.p. This result will be used to bound the number of jobs occurring several times in the tree.

Lemma 4. *Let $r \in \mathbb{N}$ be large enough. The total number of jobs contained in long delay sequences of D_t is at most $r \cdot \log^2 n$, w.h.p.*

Proof. First we show that each long delay sequence of D_t has length at most $r' \log n$ for a constant r', w.h.p. (r and r' are dependent on α). If one of the long delay sequences S of the delay tree has length at least $r' \log n$, there exists a lengthened delay sequence consisting of S and the path from the top node of S to the root. This delay sequence S' also has length at least $r' \log n$.

By using Corollary 1 we show that the probability that S' has length at least $r' \log n$ is bounded by $\left(\frac{6e \cdot m}{n}\right)^{r' \log n - 1}$. Thus, the probability that any of the lengthened delay sequence and, consequently, any of the long delay sequences has length larger than $r' \log n$ can be bounded by

$$2^{\log\log n + 2\alpha} \cdot \left(\frac{6e \cdot m}{n}\right)^{r' \log n - 1} \leq n^2 \cdot \left(\frac{6e \cdot m}{n}\right)^{r' \log n - 1} \leq \frac{1}{n^\alpha} \quad .$$

for suitably chosen but constant r'. Since there are $2^{\log\log n + 2\alpha}$ long delay sequences, the total number of jobs contained in long delay sequences is at most $2^{2\alpha} \cdot \log n \cdot r' \log n \leq r \log^2 n$, for a constant $r \in \mathbb{N}$.

Now we estimate the probability that the long delay sequences of D_t are not pairwise distinct.

Lemma 5. *The delay tree D_t that consists of at most $r \cdot \log^2 n$ nodes contains at least $\log n$ pairwise distinct long delay sequences, w.h.p.*

Proof. Consider a long delay sequence S whose top node is not the root of D_t. Let $\mathcal{S}$ be a subset of the long delay sequences of D_t, with $S \notin \mathcal{S}$. Let $\mathcal{J}$ be the set of jobs contained in a long delay sequence $S' \in \mathcal{S}$, let $\mathcal{B}$ be the set of servers defined by the long delay sequences S' – i.e. if $J'_i \in S'$ is blocked by $J'_{i+1} \in S'$ at server B'_{i+1}, then $B'_{i+1} \in \mathcal{B}$. Let v be the top node of S. Node v represents a job J_0, and we assume that v is the only node of D_t representing J_0 served at a server B_0. The right child of v represents job J_1 blocking J_0 at a server B_1. Due to the construction of the delay tree, B_1 is a server chosen i.u.r. Thus the probability that $B_1 \in \mathcal{B}$ is at most $\frac{r \log^2 n}{n}$. If $B_1 \notin \mathcal{B}$, the probability that any $J \in \mathcal{J}$ sends a request to B_1 is at most $\frac{2r \log^2 n}{n}$. On account of the construction, S contains only requests sent to B_1. Thus, the total probability that $J_i \in \mathcal{J}$ for at least one $J_i \in S$, $J_i \neq J_0$ is at most $\frac{3r \log^2 n}{n}$.

For each node v of level $0, \ldots, \log n + 2\alpha$ of D_t let S_v be defined as the long delay sequence with top node v. If P_v is the path from v to the root of D_t, then the delay sequences S_w with $w \in P_v$, are called the *ancestors* of S_v. Now consider a

breadth first ordering $\{v_1, v_2, \dots\}$ of the nodes of D_t. This ordering defines an ordering $S_1 = S_{v_1}, S_2 = S_{v_2}, \dots$ of the long delay sequences. Let $\mathcal{S}_1 = \{S_1\}$, and let $\mathcal{S}_i = \mathcal{S}_{i-1} \cup \{S_i\}$, if each delay sequence in $\mathcal{S}_{i-1}$ is pairwise distinct of S_i, and all ancestors of S_i are in $\mathcal{S}_{i-1}$. Otherwise let $\mathcal{S}_i = \mathcal{S}_{i-1}$. If S_i is *not* pairwise distinct of all delay sequences in $\mathcal{S}_{i-1}$ but all ancestors of S_i are in $\mathcal{S}_{i-1}$, we call S_i a *first order delete*. As stated above, the probability for S_i being a first order delete is at most $\frac{3r\log^2 n}{n}$. There are $k = 2^{2\alpha} \cdot \log n$ long delay sequences in D_t. Thus, for n large enough, the probability that at least 2α long delay sequences S_i are first order deletes is at most

$$\binom{k}{2\alpha} \cdot \left(\frac{3r\log^2 n}{n}\right)^{2\alpha} \leq \left(2^{2\alpha} \cdot \log n\right)^{2\alpha} \cdot \left(\frac{3r\log^2 n}{n}\right)^{2\alpha} \leq \frac{1}{n^{\alpha}} \quad .$$

Let each level of D_t containing a top node of a first order delete be called *faulty level*. As we have already shown, with high probability there are at most 2α faulty levels in D_t. In each faulty level, there is at least one node who is top node of a first order delete. In each non-faulty level, the number of delay sequences in $\mathcal{S}_k$ doubles. In the sequel we assume that $\mathcal{S}_k$ does not contain delay sequences with top nodes in a faulty level, or heaving an ancestor with a top node in a faulty level. Since D_t contains at least $\log\log n$ non-faulty levels, w.h.p., there are at least $\log n$ long delay sequences in $\mathcal{S}_k$ and thus, there are at least $\log n$ pairwise distinct long delay sequences.

Lemma 6. *Let J be a job entering during round T_j. There exists no delay tree D_t with root representing job J containing at least $\log n$ pairwise distinct long delay sequences, w.h.p.*

Proof. We bound the expected number of delay trees. According to Lemma 1 there exists a delay tree D_t if job J entering in round T_j has waiting time at least $t = \log\log n + 2\alpha + c$ rounds. Lemma 4 ensures that D_t contains at least $\log n$ pairwise different delay sequences of length at least c, w.h.p. Furthermore, the number of servers involved in the whole tree can be bounded by $r \cdot \log^2 n$. The expected number of delay sequences with fixed top jobs and length at least c can be bounded by

$$\left(\frac{2}{n - r\cdot\log^2 n}\right)^{c-1} \leq \left(\frac{6em}{n}\right)^{c-1} \quad .$$

Thus we first have to "fix" the top job of each delay sequence.

If v is the root of D_t, the top job of S_v is J, which is already fixed. If v' is a node of the branch from the root of D_t to the leftmost leaf of D_t, then the top job of $S_{v'}$ is fixed once S_v has been fixed. In general, the top job of S_i (cf. proof of Lemma 5) is fixed, when the jobs of the delay sequences in $\mathcal{S}_{i-1}$ are fixed, or: the expected number of ways to fill the delay sequences in $\mathcal{S}_{i-1}$ upper bounds the expected number of possible top jobs in S_i. Since $\mathcal{S}_k$ does not contain all long delay sequences, we first have to choose $\mathcal{S}_k$ among the set of all long delay sequences. There are at most $\log\log n + 2\alpha$ levels in D_t possibly containing the top node of a first order delete. Thus, there are at most $\binom{\log\log n + 2\alpha}{2\alpha}$ ways to choose the faulty levels. The expected number of delay trees containing at least $\log n$ pairwise distinct long delay sequences is at most

$$\binom{\log\log n + 2\alpha}{2\alpha} \cdot \left[\left(\frac{6em}{n}\right)^{c-1}\right]^{\log n} \leq (2\log\log n)^{2\alpha} \cdot n^{-2\alpha-1} \leq n^{-\alpha-1}$$

if $c \geq \frac{2\alpha+1}{\log\left(\frac{6em}{n}\right)} + 1$.

Since $m \leq n$ jobs enter the system per round, with probability at least $\frac{1}{n^\alpha}$ there is no delay tree D_t whose root represents a job entering the system in a given round. This ends the proof of Theorem 1.

4 Distribution of the Waiting Time

In this section, we sketch the proof of Theorem 2 specifying the distribution of the waiting time of the jobs generated in any fixed round T_j.

We assume the worst case $m = \frac{n}{4e}$. As in Section 3, we can show that a set $\mathcal{D}_t = \{D_t^1 \ldots D_t^k\}$ of $k = \frac{n}{2^{2^{t'}}}$ delay trees of depth at least $t = t' + c$ per branch, and roots representing jobs generated in round T_j have to exist if k of the jobs generated in round T_j have a waiting time of at least t. We can easily show a Lemma that is related to Lemma 1. We state it without any explicit proof:

Lemma 7. *If $n/2^{2^{t'}}$ jobs generated in round T_j have a waiting time of a length greater than $t = t' + c$ rounds, there exists $n/2^{2^{t'}}$ delay trees with roots representing jobs generated during round T_j.*

As already described in Section 3, every tree $D_t^i \in \mathcal{D}_t$ can be regularly decomposed into regular delay sequences. Furthermore, each node of level t' defines a regular delay sequence of length at least c. There are $2^{t'}$ of these long delay sequences per tree. In the following we call the first t' levels of the delay trees their *upper parts*. The next Lemma shows that at most half of the upper parts of the delay trees share a job with one of the other upper parts of the trees, i.e. at least half of the upper parts of the trees represent completely different jobs. We say a job J' *occurs only once* in $\mathcal{D}_t$, if there is at most one node in one of the trees $\{D_t^1 \ldots D_t^k\}$ representing J'. The proof can be done according to the proof of Lemma 5.

Lemma 8. *At least $n/2 \cdot 2^{2^{t'}}$ of the upper parts of the delay trees in ${\mathcal{D}'}_t$ represent jobs occurring only once in ${\mathcal{D}'}_t$, w.h.p.*

It remains to show that $n/2^{2^{t'}}$ delay trees of height $t' + c$ do not occur, w.h.p. Each of the above delay trees induces $2^{t'}$ long delay sequences with fixed top jobs as defined in Section 3. Furthermore, the servers belonging to the Jobs of level t' of the trees are pairwise different from each other. The next Lemma bounds the expected number of occurrences of $n \cdot 2^{t'} / 2 \cdot 2^{2^{t'}}$ long delay sequences with fixed top jobs. Due to space limitations we present it without proof.

Lemma 9. *The expected number of occurrences of $n \cdot 2^{t'}/(2 \cdot 2^{2^{t'}})$ long delay sequences with fixed top jobs is at most $\left(\left(\frac{4e \cdot m}{n}\right)^{c-1} + \frac{2^{t'+1}}{2^{2^{t'}}}\right)^{n \cdot 2^{t'}/2 \cdot 2^{2^{t'}}}$.*

The next Lemma bounds the the probability that $n/2^{2^{t'}}$ Delay trees of height $t'+c$ occur. The proof can be done according to Lemma 6.

Lemma 10. *There exists no set $\mathcal{D}_t$ = of $\frac{n}{2^{2^{t'}}}$ delay trees with roots representing jobs generated during round T_j, w.h.p.*

References

[ABK94] Yossi Azar, Andrei Z. Broder, and Anna R. Karlin. On-line load balancing. *Theoretical Computer Science*, 130:73–84, 1994. A preliminary version appeared in FOCS 92.

[ABKU94] Y. Azar, A. Z. Broder, A. R. Karlin, and E. Upfal. Balanced allocations. In *Proceedings of the 26th ACM Symposium on Theory of Computing*, pages 593–602, New York, 1994. ACM, ACM Press.

[ACMR95] Micah Adler, Soumen Chakrabarti, Michael Mitzenmacher, and Lars Rasmussen. Parallel randomized load balancing. In *Proceedings of the 27th ACM Symposium on Theory of Computing*, pages 238–247, New York, USA, 1995. ACM, ACM Press.

[AKP+93] Yossi Azar, Bala Kakyanasundaram, Serge Plotkin, Kirk R. Pruhs, and Orli Waarts. On-line load balancing of temporary tasks. In *Proceedings of the 3rd Workshop on Algorithms and Data Structures*, pages 119–130, 1993.

[BMS97] Petra Berenbrink, Friedhelm Meyer auf der Heide, and Klaus Schröder. Allocating weighted jobs in parallel. In *Proceedings of the 9th Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 302–310, 1997.

[CS97] Artur Czumaj and Volker Stemann. Randomized allocation processes. In *Proceedings of the 38th IEEE Symposium on Foundations of Computer Science*, pages 194–203, Miami Beach, FL, 1997. IEEE Computer Society Press, Los Alamitos.

[KLM92] Richard M. Karp, Michael Luby, and Friedhelm Meyer auf der Heide. Efficient PRAM simulation on a distributed memory machine. In *Proceedings of the 24th ACM Symposium on Theory of Computing*, 1992.

[Kur81] Thomas G. Kurtz. *Approximation of Population Processes*. Regional Conference Series in Applied Mathematics. CMBS-NSF, 1981.

[Mit96] Michael Mitzenmacher. Density dependent jump markov processes and applications to load balancing. In *Proceedings of the 37th IEEE Symposium on Foundations of Computer Science*, pages 213–223, 1996.

[Mit97] Michael Mitzenmacher. On the analysis of randomized load balancing schemes. In *Proceedings of the 9th Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 292–301, Newport, Rhode Island, 1997.

[MSS95] Friedhelm Meyer auf der Heide, Christian Scheideler, and Volker Stemann. Exploiting storage redundancy to speed up randomized shared memory simulations. In *Proceedings of the 12th Annual Symposium on Theoretical Aspects of Computer Science*, 1995.

[PW93] S. Phillips and J. Westbrook. Online load balancing and network flow. In *Proceedings of the 25th ACM Symposium on Theory of Computing*, pages 402–411. ACM, 1993.

[SV96] Christian Scheideler and Berthold Voecking. Continuous Routing Strategies. In *Proceedings of the 8th Annual ACM Symposium on Parallel Algorithms and Architectures*, 1996.

[St96] Volker Stemann. Parallel balanced allocations. In *Proceedings of the 8th Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 261–269, New York, USA, 1996. ACM.