Balls-into-Bins: A Paradigm for Job Allocation, Data Distribution Processes, and Routing.

Dissertation
von
Klaus Schröder

Acknowledgments

I like to thank my advisor Friedhelm Meyer auf der Heide for all the things he taught me. He was a great resource of knowledge, ideas, and common effort. This work also benefited from many other people. Many ideas rose in intensive discussions with Petra Berenbrink. Berthold Vöcking was a great source of stimulation. I always admired his endurance and accuracy to think the things to their very end. This thesis further profited from Christof Krick, Harald Räcke, and Matthias Westermann. They helped me to wipe out many errors and typos in this thesis.

Contents

1 Introduction

One of the most important problems in the efficient use of high-performance parallel systems is to distribute workload among servers. In general this problem requires to determine where a job shall be executed, but also when execution takes place — a scheduling problem has to be solved. Scheduling is a very general problem and hard to solve, therefore it is natural to make some simplifying assumptions if it is important to obtain solutions very fast.

If one relaxes the relations between the jobs and assumes that the execution time of a job is the same on all servers, a scheduling problem becomes an easier *Load-Balancing Problem*. In particular precedence relations between jobs are neglected here, thus a solution of a Load-Balancing Problem mainly specifies where to execute a job. Load-Balancing Problems are further divided into *static problems* where jobs allocated to a server do not move to another server during execution and *dynamic problems* where such movements are allowed. In this thesis we consider static allocations of *independent jobs*. Independent jobs are jobs with no or negligible dependencies: they may be executed in any order and they do not communicate with each other. However they may interfere with each other in the struggle for access to the limited resources of the servers. The allocation problem models some central aspects of the more general Load-Balancing Problem and offers an access for theoretical investigations of them. On the other hand the problem of allocating independent jobs itself is of practical and theoretical interest as it has a lot of applications in theory and use of parallel systems.

Even though the concept of independent jobs seems to be very restrictive, it has many applications in the use of parallel systems of many kinds. The most obvious application is to place jobs issued from client workstations for execution on compute servers. In this case a possible objective in assigning the jobs to the servers is to assign approximately the same number of jobs to every server.

Another application arises in the design of so called *Video on Demand (VoD)* Servers: a set of disks storing MPEG coded movies has to fulfill requests issued from user ports for small parts of the stored movies under real time restrictions. The requests have to be allocated to the disks in a manner that the data can be delivered by the disks in time. Moreover the allocation has to obey that a request can only be allocated to a disk which stores the desired data. Here the objective is to upper bound the maximum time used by the disks to answer the requests.

One can also model some routing problems by using independent jobs. This holds in particular for *circuit switching*. In a circuit-switched network messages arrive, each requesting a path from its source to its destination. To fulfill the request a path con-

necting source and destination has to be chosen. As the capacity of links in a network is bounded it is important that the number of paths using a certain link, i.e. the congestion, is not too large.

The allocation problem further arises in the theory of parallel systems, as it has a natural relation to Hashing where data items are stored in buckets and — in order to keep the load of the buckets small — the load has to be spread evenly among the buckets. A kind of hashing has been used by Karp et al. [KLM92] to simulate a PRAM on a Distributed Memory Machine (DMM). The main problem of the simulation is the distribution of the PRAM memory cells to the memory modules of the DMM in a way that allows fast access to the contents of the PRAM memory cells during the simulation. In a step of the simulation the N processors of the DMM simulating the N processors of the PRAM request access to N memory cells. Each request has to be answered by a memory module storing a copy of the requested cell. The number of answers made by a memory module is obviously a lower bound for the time required to simulate a step of the PRAM. The tight relation between Hashing and the allocation problem further shows that allocation problems also arise in sequential contexts.

The Video on Demand and the PRAM simulation example differ in an important aspect from the other mentioned examples. The Video on Demand and the PRAM simulation deal with data distribution and they require that a request can only be fulfilled by a subset of the servers. Each server in the subset has to store a copy of the data-item. As in both cases the storage overhead is a critical measure, the subset of servers capable to execute a job is quite small, typically it contains only a constant number of servers. The servers capable to execute a particular job are called the *possible servers of the job*. The choice of possible servers has great impact on the result an allocation algorithm can achieve. For sake of theoretical analysis, there are three natural scenarios for the choice of the possible servers.

Worst case (adversarial) possible servers
: It is easy to see that the distribution of jobs may become poor, if the possible servers of all jobs are in the same small set of servers. This restricts the possibilities of classical worst case analysis. A more fruitful approach uses *competitive analysis* to compare the performance of an allocation algorithm with the optimal result (see for example [ST85, BE98]). We give a short discussion of related results in Section 1.3.4.

All servers are possible servers
: This scenario corresponds to the workload distribution examples presented above. No exterior restrictions apply to the allocation algorithm.

Random possible servers
: This model being somewhere between the other two models is investigated in this thesis. It assumes that the possible servers of a job are chosen at random according to some distribution functions. The distribution functions are assumed to be the same for all jobs.

As we will see, the random possible servers scenario allows to obtain near optimal allocations using very simple algorithms. We therefore suggest to use random possible servers also in the second scenario, where actually all servers are possible. The restriction of the third scenario is then used as a paradigm in addressing the second one. The use of randomly chosen possible servers has inspired many people to model the allocation of independent jobs as a Balls-into-Bins game: each ball (representing a job) has to be allocated to one out of d randomly chosen bins (representing the possible servers). The objective in allocating balls is to minimize the number of balls allocated to the fullest bin. The case $d = 1$ has gained much attendance due to its importance for many hashing techniques and it is well-known that the fullest bin contains $\Theta(\log N / \log\log N)$[1] balls, with high probability, if there are N balls and the bins are chosen independently and uniform at random. More accurately Gonnet shows in [Gon81] that the number of balls in the fullest bin is $\Gamma^{-1}(N) - \frac{3}{2} + o(1)$ [2], with high probability. While the $d = 1$ case is well studied, the problem becomes more interesting if $d \geq 2$ is considered. Using $d \geq 2$ requires an algorithm to choose a server for each job, and — as we will see — allows to obtain much better allocations.

One of our applications differs in an important way from the others. The routing example requires that a whole set of links is used to serve a request, this is in contrast to all other examples where single servers suffice. While, as we will see, all other examples are tightly related, the routing example needs a completely different treatment. We therefore divide the remainder of this thesis into two parts. One part is formed by the remainder of this chapter and the subsequent chapters up to Chapter 5. Chapter 6 forms the, less extensive, second part. It contains all material for the routing application of the balls-into-bins paradigm including the introduction.

(The title of this thesis stresses the term "Balls-into-Bins". In this thesis, however, we speak about "jobs" and "servers" instead. The reason for this is quite simple: "Balls" and "Bins" have the same first letter while "job" and "server" have not. This simplifies to find suggestive variable names for the latter. [3])

1.1 The Allocation Problem

In most applications jobs are not all the same. They differ especially in the amount of resources they need. We therefore associate a *weight* with each job. This weight may correspond to the time it takes to perform a job on a parallel computer or to the amount of data needed to fulfill a request. These quantities are typically not all the same. There are, however, examples where the utilization of resources is the same for each job (consider for instance the PRAM application). We therefore consider the case

[1] Here and throughout this thesis we assume $\log = \log_2$.

[2] Here $\Gamma(x) = \int_0^\infty e^{-t} t^{x-t} dt$, $x > 0$ denotes the *Gamma Function*. For any $x \in \mathbb{N}$ the Gamma Function fulfills $\Gamma(x+1) = x!$.

[3] The author has also noticed that speaking about jobs and servers helps to convince people about the Balls-into-Bins game having applications.

where all weights are the same, too. As this case is much simpler to deal with than the general case, it allows to obtain more exact results.

We further distinguish between *finite* and *infinite* allocation problems, depending on whether the number of jobs to be allocated is finite or not. Many real world examples are better modeled using an infinite number of jobs. The disks in the Video on Demand example, for instance, are typically confronted with a very long stream of requests for movies data blocks. Fulfilled requests do not contribute to the load of the servers any more. Therefore we associate an *entry time* and a *deletion time* with each job.

Definition 1.1.1:

Allocation Problem

Let $d \in \mathbb{N}$ be the *number of copies*. Let $\mathcal{S}$ be a finite set of *servers* and let $\mathcal{J}$ be a countable set of *jobs*. For each job $J \in \mathcal{J}$ let $0 \leq W(J) \leq 1$ be the *weight of* J. For each $i = 1, \ldots, d$ let $\Xi(i)$ be a probability distribution on $\mathcal{S}$. The distribution $\Xi(i)$ is called the *i-th copy distribution*.

Then the sets $\mathcal{S}$ and $\mathcal{J}$, the weights $W(J)$ for $J \in \mathcal{J}$, and the i-th copy distributions $\Xi(i)$ for $i \in \{1, \ldots, d\}$ define a *weighted allocation problem*. An allocation problem is called an *ordinary allocation problem* if all jobs have weight 1.

An allocation problem is called *finite*, if $\mathcal{J}$ is finite, and *infinite* otherwise. The number of servers $|\mathcal{S}|$ of the allocation problem in consideration is always denoted by N. The number of jobs $|\mathcal{J}|$ (if applicable) is denoted by M.

Allocation

For each $J \in \mathcal{J}$ let $S^{(i)}(J)$ be a random variable distributed according to $\Xi(i)$. The server $S^{(i)}(J)$ is called *the i-th possible server of job J*. The servers $S^{(1)}(J), \ldots, S^{(d)}(J)$ are called the *possible servers of job J*.

An *allocation* is a mapping $S : \mathcal{J} \to \mathcal{S}$ with $S(J) \in \{S^{(1)}(J), \ldots, S^{(d)}(J)\}$ for each job $J \in \mathcal{J}$. We say that J *is allocated to* S if $S = S(J)$.

Load of an Allocation

For each job $J \in \mathcal{J}$ let $\delta(J) \geq 0$ be the *entry time* of J and let $\Delta(J) \geq \delta(J)$ be the *deletion time* of J. We allow $\Delta(J) = \infty$ to denote that J is never deleted. Δ is called the *deletion scheme*. The set $\mathcal{J}_\tau^{\delta,\Delta} = \{J \in \mathcal{J} \,|\, \delta(J) \leq \tau < \Delta(J)\}$ contains the jobs *being in the system at time τ*. (Note that neither δ nor Δ are part of the allocation problem. The reason for this is that we allow Δ to depend on the possible servers and on the allocation.)

We compare allocations with respect to the load they obtain. The *load of a server $S \in \mathcal{S}$ at time τ* is defined as

$$load_\tau(S) = \sum_{\substack{J \in \mathcal{J}_\tau^{\delta,\Delta} \\ S(J)=S}} W(J)$$

and the *load of an allocation at time* τ is defined as

$$load_\tau = \max_{S \in \mathcal{S}}(load_\tau(S)).$$

If the set of jobs is finite, we assume that $\delta(J) = 1$ and $\Delta(J) = \infty$ for each job $J \in \mathcal{J}$. In the sequel we assume that there is an ordering '$<$' on $\mathcal{S}$ and $\mathcal{J}$.

We will further always assume that $\sum_{J \in \mathcal{J}} W(J) = \mathrm{O}(N \cdot \log N)$. Otherwise a simple application of a Chernoff Bound as found in Lemma 2.2.2 shows that allocating each job to a server chosen randomly from the possible servers yields a near optimal result.

1.2 Criteria for the Evaluation of Allocation Algorithms

An allocation algorithm is an algorithm allocating each job to one of its possible servers. We assume that each sensible allocation algorithm works distributed; each job and each server is assumed to have an agent performing the respective part of the algorithm. The decision where to allocate a job is made by the jobs agent. Each job agent can only communicate with the agents of the possible servers of its job. The server agents are restricted to answer to messages from job agents, they cannot initiate communication with a job which never sent them a message before. Further we assume that communication is organized in *rounds*. In a round each job agent can send a message to each of the jobs possible servers, and each server agent can reply to each job it received a message from.

The most important distinction we make is between *sequential* and *parallel* algorithms. A sequential allocation algorithm deals with the jobs one by one. This corresponds to an allocation problem in the so-called *sequential setting*. In the sequential setting in each round $\tau \leq |\mathcal{J}|$ there is exactly one job with entry time τ. This job has to be allocated by the allocation algorithm in an On-Line fashion. In round τ the job J with entry time τ has to be allocated without any information on the possible locations or the size of any job entering the system after round τ. The server job J is allocated to cannot be changed in future rounds. The only important criterion for the evaluation of a sequential allocation algorithm is therefore the load it obtains. In the *parallel setting* there may be several jobs with the same entry time. We use $\vec{M}$ to upper bound the weight of the jobs entering the system in a round.

Different from the sequential case, a parallel allocation algorithm is allowed to defer the allocation of a job to future rounds. Besides the obtained load, the number of rounds used to allocate a job has a severe impact on the quality of the algorithm. If the allocation problem is infinite, another important measure in the maximum value of $\vec{M}$ for which the performance of the system can be guaranteed.

1.3 Previous Results and Related Work

The balls-into-bins paradigm is simple, quite general, and has a lot of applications. It is therefore hard to tell the very first origin of the idea. Up to our knowledge the first use of

the balls-into-bins paradigm and the first analytical demonstration of its power is due to Karp et al. [KLM92]. Karp et al. deal with PRAM simulations on Distributed Memory Machines (DMMs). Their results on PRAM simulations are of particular impact on solutions for the allocation problem in the parallel setting. Most results from PRAM simulations use i-th copy distributions derived from hash-functions chosen at random from suitable classes. Thus the possible servers of a job are not chosen independently as we required in Definition 1.1.1. The results are not affected, if we assume that the servers chosen by the jobs are chosen independently. Moreover the PRAM simulations frequently require a job to be allocated to several servers. It should be clear that an allocation in our meaning is easy to obtain from such allocations. The sequential setting of the allocation problem was considered by Azar et al. in [ABKU94]. This paper is, up to our knowledge, the first paper which considers the allocation problem on its own.

Most research on the allocation problem concentrated on relatively few algorithms. We therefore organize our discussion of previous results with respect to the different settings and within the settings according to the similarity of the considered algorithms. Throughout this section the allocation problem in consideration is assumed to have N servers, $d \geq 2$ copies, and uniform i-th copy distributions, if not stated otherwise. If the considered allocation problem is finite, we denote the number of jobs by M.

1.3.1 The sequential setting

The sequential setting is ruled by the *greedy algorithm* described in Section 1.7.2. It is introduced by Azar et al. in [ABKU94]. (In fact a similar algorithm has been proposed much earlier by Eager et al. [ELZ86] for load balancing purposes. Eager et al. however are not interested in the asymptotic performance of the algorithms they consider. They note that in their setting the greedy algorithm performed slightly *worse* than a Threshold algorithm similar to the one considered in [ACMR95].) Azar et al. examine the greedy algorithm in the finite and in the infinite setting. In the finite setting they show that the greedy algorithm obtains load $\ln\ln N/(((1+\mathrm{o}(1))\ln d) + \Theta(M/N)$, with high probability. In the infinite setting they assume that the servers initially contain N jobs allocated to arbitrary locations. For each new job entering the system, a job is chosen independently uniform at random (i.u.r.) and deleted from the system. Using this deletion scheme they show that at the end of an arbitrary round $\tau > N^3$ the load is at most $\ln\ln N/\ln d + \mathrm{O}(1)$, with high probability. The greedy algorithm is the most natural allocation algorithm in the sequential setting — and for uniform i-th copy distributions it is also the algorithm which performs best. Surprisingly enough this is not true, if non-uniform i-th copy distributions are used. As shown in [Vöc99a], it is possible to obtain load $\ln\ln N/(d\cdot\ln\phi_d) + \mathrm{O}(1)$. The numbers $\phi_2, \phi_3, \ldots$ correspond to generalized Fibonacci sequences; in particular $\phi_2 \approx 1.6180\ldots$ which is known as the golden ratio, and $\phi_d > 2^{\frac{d-1}{d}}$. Vöcking shows that, up to an additive constant, no algorithm and no choice of i-th copy distributions allows to obtain better load. This also holds, if the choice of the j-th server $S^{(j)}(J)$ of a job J may depend on the choices

for $S^{(1)}(J), \ldots, S^{(j-1)}(J)$. The difference between the greedy algorithm and Vöckings *Always-go-Left* algorithm is, however, quite small: in case of a tie, when the greedy algorithm makes an arbitrary choice, the Always-go-Left algorithm chooses the server with the smallest index. Vöcking also analyzes the performance of the Always-go-Left algorithm in the infinite setting. If the system contains at most M jobs at any point of time, he shows that the load is at most $\ln\ln N/(d \cdot \phi_d) + \mathrm{O}(M/N)$, with high probability, in an arbitrary round τ. The analysis is able to deal with any *oblivious* deletion scheme, i.e. the sequence of insertions and deletions of jobs is arbitrary, but has to be independent of the possible servers of a job. His analysis allows to transfer the result of Azar et al. on the performance of the greedy algorithm in finite settings to infinite settings with arbitrary oblivious deletion schemes. Vöckings analysis applies the so-called *witness tree* argument, introduced by Meyer auf der Heide, Scheideler and Stemann in [MSS95]. Compared to the method of *Layered Induction* applied by Azar et al. both techniques achieve good constants, but the witness tree analysis allows to deduce more general results.

In [CS97] Czumaj and Stemann address several questions related to the allocation problem in the sequential setting and the greedy algorithm. One point they are interested in, is the *allocation time* of the algorithms, which is defined to be the number of servers examined by a job before it is allocated. In an allocation problem with number of copies d, the maximum allocation time is d. To minimize the average allocation time, they assume that the requests sent by a job are made and answered sequentially. If a job agent receives a message from a server with load at most c, the job is allocated to that server and no further requests are made by that job. The load obtained by this algorithm is at most $\ln\ln N/\ln d + \mathrm{O}(c)$, with high probability. For any constant $\varepsilon > 0$ there is a c such that their allocation algorithm has average allocation time at most $1 + \varepsilon$. Czumaj and Stemann also consider the rate of convergence of the greedy algorithm (see also [Czu97]). They show that if N jobs are initially allocated to arbitrary servers, $(1 + \mathrm{o}(1)) \cdot N \cdot \ln N$ rounds suffice to assure that the load is $\ln\ln N/\ln d + \mathrm{O}(1)$, with high probability. Finally they generalize the lower bound on the load of [ABKU94] to algorithms which are allowed to reallocate the jobs currently allocated to the possible servers of a job J among those servers when allocating job J. For the sake of reallocation the algorithms can make use of complete information about the distribution of the current load of all servers in the system. Even such algorithms cannot obtain load better than $\Omega(\ln\ln N/\ln d + M/N)$, with high probability. If $M = N$ the load is at least $\ln\ln N + \Omega(1)$, with high probability, thus the possibility of reallocations does not improve the possible performance by more than an additive constant.

A completely different analysis is used by Mitzenmacher for his supermarket model ([Mit96a, Mit96b, Mit97]) which is an allocation problem in the infinite setting. In the supermarket model new jobs arrive as a Poisson stream of rate $\lambda \cdot N$, upon arrival they are allocated to a server using the greedy algorithm. If allocated to a server the job joins a FIFO-queue attached to this server. Each server serves the jobs in its queue with exponentially distributed service time with mean 1, served jobs are deleted. Mitzenmachers particular interest is the expected time a customer spends in the system,

if the system is in equilibrium. Using a powerful result from [Kur81] he shows that in the equilibrium state of a corresponding system with an infinite number of servers, the service time decreases doubly exponentially. Moreover he shows that for a polynomial time interval the latter system is a good approximation for the real supermarket model. This allows him to show that for a polynomial time interval the length of the longest queue in an initially empty supermarket model is $\frac{\log\log N}{\log d} + \mathrm{O}(1)$, with probability at least $1 - \mathrm{O}(1/N)$. In [Mit97] the results for the corresponding infinite systems are shown to hold for systems with various other service times, too. The technique of Mitzenmacher allows to obtain very sharp results on the load. Its major drawback is that it only allows to obtain results on the expected load of a corresponding infinite system. Independently a technique similar to the one used by Mitzenmacher has been developed by Vvedenskaya et al with a stronger mathematical background. In [VDK96] Vvedenskaya et al. apply their technique to analyze the performance of the greedy algorithm in a queuing system which is essentially the supermarket model.

1.3.2 The parallel setting

As stated above, the allocation problem in the parallel finite setting is closely related to PRAM simulations on DMMs. For reasons arising from the different DMM models, the results from the PRAM simulation area mainly concentrate on two algorithms: the *c-arbitrary algorithm* — which is up to our knowledge introduced in [KLM92] — and the *c-collision algorithm* — which up to our knowledge appeared first as *Process_3* in [DM93]. The c-collision algorithm is described in Figure 1.1 on Page 22, the c-arbitrary algorithm is given as a special case of the c-priority algorithm in Figure 5.1, Page 79.

The c-priority algorithm In [KLM92] Karp et al. consider the idea of using several hash functions in randomized PRAM simulations. Their approach requires to solve an allocation problem with $M = N$ in the parallel finite setting. The 1-arbitrary algorithm (in the separated version as described on Page 79), called *Process_3*, is shown to obtain load at most $\mathrm{O}(\log\log N)$ and to require at most $\mathrm{O}(\log\log N)$ rounds, with high probability. To prove this result they make use of a graph whose edges represent jobs and whose nodes represent servers. An edge is incident to a node if the server represented by the node is one of the possible servers of the job represented by the edge. This graph, called *access graph*, is shown to have some properties which then imply the result.

Another version of the c-priority algorithm is considered by Adler et al. in [ACMR95]. We present their *MPgreedy algorithm* in our formulation in Figure 5.1 on Page 79. As a priority rule (cf. to Step (3) in Figure 5.1) the MPgreedy algorithm uses the ordering '$<$' on $\mathcal{J}$. The analysis relates the allocation obtained by MPgreedy to the allocation obtained by a version of the greedy algorithm, where in case of a tie in Step (1) (see Fig. 1.2) the job is allocated to each server with minimum load.

If N jobs are to be allocated to N servers, the MPgreedy algorithm finishes within $\log\log N + O(1)$ rounds, with high probability. Adler et al. also consider an algorithm they call *Threshold(c) Algorithm* which looks like a version of the c-arbitrary algorithm at a first glance. But the Threshold(c) algorithm chooses a new server each time a job fails to be allocated, thus the number of copies is unbounded. Unfortunately the analysis for the Threshold(c) algorithm heavily depends on this fact.

The c-collision algorithm The separated version of the c-collision algorithm as given in Section 5.2.2 is found in [DM93]. Similar to the c-arbitrary algorithm it is motivated by an access conflict rule used to resolve concurrent accesses to a memory module. Dietzfelbinger and Meyer auf der Heide show that the c-collision algorithm allocates $M = \Omega(N)$ jobs to N servers using 3 copies with uniform i-th copy distributions in the parallel finite setting. The c-collision algorithm obtains load $O(1)$ and requires at most $O(\log\log N)$ rounds. The analysis utilizes 3-uniform hyper-graphs similar to the graph used in [KLM92]. In [MSS95, MSS96] Meyer auf der Heide, Scheideler and Stemann anticipate the results of [KLM92, DM93]. Besides several improvements of the PRAM simulations, they generalize the Access Schedule 2 to a more general version of the c-collision algorithm, the so-called (N, ϵ, a, b, c)-process. The analysis of the performance of the (N, ϵ, a, b, c)-process makes use of an argument, called the *witness tree*, which is also the core of our proof (cf. to Chapter 3). As a result of the new approach Meyer auf der Heide et al. are able to obtain much more general results for the performance of the c-collision algorithm than the ones obtained in [DM93]. The latter result is restricted to 3 copies, $c \geq 3$ and $M \leq 3/(2e) \cdot N$, while the result from [MSS95] allows any $0 < \epsilon \leq 1$, $2 \leq d \leq \sqrt[3]{\log N}$, $M \leq \epsilon \cdot N$, and c large enough to ensure that $(c^2 \cdot (d-1))/(c+1) > 1$ and $\epsilon \cdot (1/c!)^{d-1} \leq \frac{1}{2}$. MacKenzie, Plaxton, and Rajaraman achieve similar results in [MPR94] using a different approach. In particular they show that the 1-collision algorithm terminates within $\Theta(\log\log N)$ rounds, with high probability, if N jobs are allocated to N servers using $d = 3$ copies and uniform i-th copy distributions. This results holds for each $c \geq 2$ and $d \geq 2$. Moreover they show that the 1-collision algorithm takes $\Theta(\log N)$ rounds,with high probability, if only 2 copies are used.

The relation between PRAM simulations on DMMs and the balls-into-bins paradigm was deployed by Stemann in [Ste96]. Stemann applies the c-collision algorithm to the allocation problem with 2 copies and shows that an allocation with load 32 can be obtained in $0.17 \cdot \log\log N$ rounds of the c-collision algorithm, with high probability. He also presents an asynchronous version of the c-collision algorithm, compare Section 5.2.2.1, which achieves similar performance.

1.3.3 Lower bounds for the parallel setting

In the past a bunch of lower bounds has been shown for the allocation problem in the parallel setting, compare [MSS95, Ste95, Mit96b, BMS97, BMS99]. They all assume

that the i-th copy distributions are uniform, and in principle their underlying concept is always the same. The lower bounds vary in the class of allocation algorithms and range of parameters for which the lower bound holds. We give a short sketch of the underlying ideas, and define some concepts, which will help us to discuss the lower bounds.

For node v (an edge e) of a graph $G = (V, E)$ let the *T-neighborhood of v (of e)* be the subgraph of G induced by the nodes $v' \in V$ with $\text{dist}(v, v') \leq T$ ($\text{dist}(v', e) \leq T$). Consider the access graph $G = (V, E)$. We consider allocation algorithms using at most T rounds of communication. Thus a job agent represented by an edge e can only communicate with jobs represented by edges in the T-neighborhood of e in G. The idea of the lower bounds mentioned above is to assume that a job J cannot do better than selecting a server from $S^{(1)}(J), \ldots, S^{(d)}(J)$ at random, if the neighborhood of each node incident to the edge representing J *looks the same*. We call those jobs (and edges representing such jobs) *clueless*. It suffices now to find a node v with c incident clueless edges. The jobs represented by these edges are each allocated to v with probability $\frac{1}{d}$, thus the expected load of the server represented by v is c/d. The main differences between the different lower bounds lie in the different meanings of "looks the same" and thus the implied different definitions of clueless.

Adler et al. consider lower bounds for constant numbers of rounds T and $d = 2$ in [ACMR95]. In their lower bound an edge is clueless if the neighborhood of each incident node is a complete d-uniform tree G_{Tree} of height T whose inner nodes have degree $c = \Omega(\sqrt[T]{\frac{\log N}{\log\log N}})$. They show that with constant probability the access graph contains a connected component being such a complete d-uniform tree of height $T + 1$. In this tree each child edge of the root is a clueless node, thus with constant probability $\Omega(\sqrt[T]{\frac{\log N}{\log\log N}})$ jobs are allocated to the server represented by the root. Adler et al. also present an allocation algorithm whose performance matches the lower bound up to a constant factor, if $T = 2$. The major drawback of their lower bound is of course the restriction to constant values of T. The authors mention that their analysis allows T to grow slightly with N, but the bound cannot capture the parameters of, for instance, the c-collision algorithm. Mitzenmacher gives a more general version of this result in his PhD thesis [Mit96b]. He extends the bound to arbitrary constant d and notes that the load is at least $\Omega(\frac{\log\log N}{\log\log\log N})$, with probability at least $\text{O}(1/\log^c N)$, if at most $\text{O}(\frac{\log\log N}{\log\log\log N})$ rounds are used.

In [BMS97, BMS99] Berenbrink et al. use a result from Stemann [Ste95] to extend the lower bound to larger values of T. Assuming $d = 2$, Stemann shows that with constant probability an access graph for N servers and at least $N/\log\log N$ jobs contains any tree $G = (V, E)$ with $|V| \leq \frac{\log N}{(9\cdot\log\log N)}$ as a subgraph. In particular this ensures that the access graph contains a complete tree of height $T + 1$ whose inner nodes have degree $c = \frac{\log N}{(18\cdot\log\log N)}$. Berenbrink et al. assume that an edge is clueless if the T-neighborhood of each incident node *contains* a complete tree of height T whose inner nodes have degree c (in contrast to the bound of Adler et al. who assumed that the T-neighborhood *is*

such a tree). Consequently the root of the tree has load $\Omega(\frac{\log N}{(18 \cdot \log\log N)})$, with constant probability. The major drawback of this lower bound is that the criterion for a job being clueless seems to be unfounded. The neighborhood of one node incident to a clueless edge can contain much more edges than the neighborhood of the other nodes and nevertheless the job represented by that edge is assumed to commit a random decision. Berenbrink et al. base their definition of "clueless" on the following assumptions:

1. The allocation algorithm is a direct algorithm.

2. The decisions of the allocation algorithm depend only on the topological structure of the access graph. In particular no whatsoever label of a server or a job is regarded.

3. If the neighborhoods of the nodes incident to an edge are isomorphic, the job represented by that edge commits a random decision.

4. If an arbitrary subset of the jobs is deleted, the load of the allocation obtained by the allocation algorithm is not increased. The choice of the jobs being deleted may depend on the random variables $S^{(i)}(J)$, $i \in \{1, \ldots, d\}$, $J \in \mathcal{J}$.

The properties (1) to (3) are also required by the lower bound of Adler et al. Property (4) allows to ensure that Property (3) is fulfilled if the requirements of the bound of Berenbrink et al. for an edge being clueless are fulfilled. To make the different neighborhoods isomorphic, simply delete the edges which do not belong to the tree. The intuition behind Property (4) is that solving smaller problems should not be harder than solving a larger one. On the other hand deleting edges also reduces an algorithms possibilities for communication. Up to our knowledge there is no lower bound for allocation algorithms which gets by without similar assumptions. Property (2) is a severe restriction, too. For instance the MPgreedy algorithm does not fulfill Property (2), as it makes use of the jobs ID numbers. For the class of algorithms which fulfill the requirements, the lower bound of Berenbrink et al. matches the upper bound by Scheideler et al. up to a constant factor.

As mentioned by Berenbrink et al. their lower bound can be used to show that the performance of the c-load collision algorithm as presented in Section 4.1 of this thesis is optimal up to a constant factor among all algorithms fulfilling the above conditions, as far as only the number of jobs and their average weight are considered. This becomes obvious for the following set of weighted jobs with average weight ϱ: $\varrho \cdot M$ jobs get weight 1 and $(1 - \varrho)$ jobs get weight 0. Under the assumption that adding balls with weights equal to zero does not change the complexity of an allocation problem, we may conclude that the performance of the c-load collision algorithm is optimal up to a constant factor. It is, however, possible to improve the upper bound for particular choices of the jobs weights.

1.3.4 Related work

In the area of PRAM simulations on DMMs it is reasonable to consider algorithms which are not direct (see eg. [MSS95, CMS95b] or [Ste95]). The key point in these non-direct algorithms is that an edge can explore more than its T-neighborhood in T rounds. This allows to compute allocations with constant load using $O(\log^* N \cdot \log\log\log N)$ rounds — an almost exponential improvement in performance. The algorithms, however, make use of sophisticated $\log^*$-techniques and might perform badly in practice. Even better performance bounds are possible if one considers quite exotic kinds of reconfigurable architectures (cf. to [LS91]) which are able to do *leader finding* in constant time. Here algorithms achieving $\log^* N$ load using $\log^* N$ rounds, with high probability, are possible (cf. to [CMS95a, CMS95c, CMS97]).

A variant of the c-collision algorithm is used in [BFM98] for load balancing in a model where load in generated and consumed by the servers, rather than coming from outside as in our model. The results are improved in [BFS99]. In particular it is shown that the the c-collision algorithm can help to distribute load in a system with random load generation. The maximum load is shown to be at most $O(\log\log N)$ in a system with N servers and expected load $O(N)$. Similar results are shown to hold for an adversarial model of load generation.

One can behold *Competitive on-line Load Balancing of permanent tasks* as an allocation problem in the sequential setting, where the random variables $S^{(1)}(J), \ldots, S^{(d)}(J)$, $i \in \{1, \ldots, d\}$, $J \in \mathcal{J}$ are not random variables but chosen arbitrarily by an adversary. To evaluate algorithms in such settings the load obtained by the allocation algorithm is compared to the load of an optimal allocation; the ratio of these two values is called *competitive ratio*. A survey on competitive on-line Load Balancing is found in [Aza98]. According to Azars definition, the allocation problem applies to the restricted assignment case of competitive on-line Load Balancing. As shown by Azar et al. in [ANR92, ANR95], the greedy algorithm as presented in Section 1.7.2 achieves a competitive ratio of $\lceil \log N \rceil + 1$. Further they show that no deterministic algorithm can achieve a competitive ration better than $\lceil \log(N+1) \rceil$, even in the case where each job has weight 1. The lower bound does not improve much, if randomized allocation algorithms are considered: no algorithm can obtain a competitive ratio better than $\ln N$. In [KVV90] Karp, Vazirani, and Vazirani show that the greedy algorithm with a random tie breaking mechanism obtains a competitive ratio of $\ln N + 1$, if jobs with weight 1 are considered and the optimal load is 1. In [BFL+95] Broder et al. consider a slight modification of the above problem: they assume that the possible destinations of a job are chosen by an adversary which has to ensure that the optimal load is 1, but the jobs are presented in random order. They show that the greedy algorithm obtains expected load $O(\log N / \log\log N)$, and there are inputs causing expected load $\Omega(\log N / \log\log N)$. For a comprehensive overview on the area of competitive on-line Load Balancing we refer to the survey by Azar [Aza98] mentioned above. In [Reh99] the allocation problem with adversarial possible servers is considered in the parallel

setting. It is shown that the c-collision algorithm is able to obtain competitive ratio 2 using at most $\Omega(\log N)$ rounds.

1.4 The Aim of this Thesis

As seen in the previous section, a lot of research has been conducted on the allocation problem in many settings using a bunch of different methods. Most results are, however, restricted to certain distributions, certain deletion schemes, or severe restrictions on the number of jobs. Moreover most results are restricted to the ordinary allocation problem and do not allow statements about the more important weighted case. Our first aim is to overcome these restrictions. We provide results on a wide scope of algorithms in various settings. This requires to use a general approach. As an approach we use the witness tree argument introduced in [MSS95], showing that this argument is able to provide sharp results for all allocation algorithms mentioned above. As we will see, there is a certain structure — a *representation* on a *witness forest* — occuring whenever an allocation algorithm performs bad. Showing that this structure does not occur, with high probability, we give performance bounds for the various algorithms.

As seen in the previous subsection the previous results on the performance of allocation algorithms typically involve terms like $\log\log N/\text{something} + \Omega(\text{optimum})$. For real world applications $\log\log(\cdot)$ is a function increasing incredibly slow. We therefore emphasize the importance of (at least) reasonable constants in the obtained results. Typically this requirement gets in conflict with the aim for a general result. In our analysis we try to obey the following priorities (given in declining importance)

- minimum number of copies required by algorithm/analysis being as small as possible,
- generality of analysis,
- quite sharp result on obtained load/number of rounds used to compute allocation,
- ability to state a single lemma encapsulating the main part of the analysis,
- reasonable additive constants in bounds on running time/obtained load,
- probability for performance bound being fulfilled (choice of α in "with high probability"),
- simplicity of analysis,
- ability to deal with for weighted allocation problems,
- minimum number of servers and jobs required for the analysis to work.

Besides the exact results we further wish to demonstrate that the power of the witness tree analysis is superior to the possibilities of other techniques used in this area.

1.5 New Results

All results on the allocation problem mentioned in Section "Previous Results and Related Work" deal with particular algorithms suited for particular allocation problems. This thesis provides a single approach to results for various algorithms working on various allocation problems. Our approach is based on the fact that there is a combinatorial structure which occurs whenever the performance of an allocation algorithm is poor. It therefore suffices to upper bound the probability for this structure to exist, in order to upper bound the probability for poor performance of any allocation algorithm. Our technique improves and extends previous versions of the witness tree argument introduced in [MSS95, MSS96]. Extending the possibilities of the witness tree technique

- We refine the witness tree argument such that we can deal with
  - any sensible i-th copy distribution,
  - large numbers M of jobs, and
  - infinite settings with various deletion schemes.
- We present a single general technical lemma that encapsulates the heavy combinatorics of the genuine witness tree argument.
- We devise a technique that allows to obtain results for weighted allocation problems.

Arbitrary i-th copy distributions Each previous results is restricted to a single set of i-th copy distributions. In each case these distributions are either uniform distributions or they are uniform on a subset of the servers and zero outside this subset. Our technique allows to consider any sensible i-th copy distribution. To describe the impact of the i-th copy distributions onto the performance bounds, we introduce a measure, the *stupidity* σ_Ξ of the i-th copy distributions. If the expected number of jobs having server S as a possible server is denoted by $\sigma_\Xi(S)$, then $\sigma_\Xi = \frac{1}{M \cdot d} \cdot \max_{S \in \mathcal{S}} \sigma_\Xi(S)$. Roughly spoken the stupidity exploits how well the i-th copy distributions allow to use the whole set of servers. All i-th copy distributions considered in previous results have optimal stupidity $\sigma_\Xi = 1$.

Large numbers of jobs Many previous results are restricted to the case where the number of jobs does not exceed the number of servers. This holds in particular for previous results using the witness tree technique. Our argument overcomes this restriction.

Infinite settings and various deletion schemes Our technique easily extends results for the finite setting to results in infinite settings using so-called *oblivious deletion schemes*. A deletion scheme is called oblivious if the insertion scheme δ and the deletion scheme Δ do not depend on the possible servers nor on the decisions of the

allocation algorithm. With a little bit more effort we are also able to handle an important non-oblivious deletion scheme, the so-called *server-oriented deletion scheme*, which overcomes an unnatural phenomenon associated with oblivious deletion schemes. In an oblivious deletion scheme the expected number of jobs deleted from a server *increases* with the load of the server. This assumption is fairly unrealistic for many applications. The server oriented deletion scheme avoids the problem as it assumes that in each time unit each server is able to delete up to c jobs allocated to it. We investigate the impact of this deletion scheme on the performance of the c-priority algorithm in Section 5.2.1.3. A discussion of the result is given below.

Weighted allocation problems All allocation algorithms have a natural weighted counterpart. Actually all these algorithms are easily described for the weighed case such that they act on ordinary allocation problem in the same way as their ordinary counterpart. Our technique for weighted allocation problems shows that it does not matter whether an allocation problem requires to allocate M jobs or an arbitrary set of jobs with sum of weights M. The performance bound for the algorithm on ordinary allocation problems also holds on weighted problems — up to a small constant factor.

1.5.1 Example: the greedy algorithm using general i-th copy distributions

It is beyond the scope of this introduction to expose the implications of our techniques on every algorithm in every setting. We therefore restrict ourselves to three examples here. A comprehensive overview on the application of our techniques is subject of Chapter 5. For the sequential setting and in order to demonstrate the impact of i-th copy distributions with non-optimal stupidity we consider the greedy algorithm on an ordinary allocation problem in the infinite setting using oblivious deletion schemes. In this situation our upper bound on the load mainly depends on three parameters:

- the number N of servers,
- an upper bound $\overline{M}$ on the number of jobs in the system, and
- the stupidity σ_{Ξ} of the i-th copy distributions.

In Section 5.1.1 we show the following theorem.

> **(Theorem 5.1.4:)** Consider an ordinary allocation problem in the finite setting with $\overline{M}$ jobs or an allocation problem in the sequential setting using an oblivious deletion scheme which ensures that there are at most $\overline{M}$ jobs in the system. Let σ_{Ξ} be the *stupidity of the i-th copy distributions*. Then any

version of the greedy algorithm which never allocates a job to more than one server assures that less than m servers obtain load larger than

$$\overline{T} = \log_d \log_{\epsilon_c} \frac{N}{m} + \left(\min\{5.275 + \tfrac{1}{d}, d\} + \mathrm{o}(1)\right) \cdot \frac{\overline{M}}{N} \cdot \sigma_\Xi + k_{\mathrm{Gr}} + 2 + \mathrm{o}(1),$$

with probability at least $1 - N^{-\alpha}$, for

$$\begin{aligned} \epsilon_c &= 3^{d-1} \\ k_{\mathrm{Gr}} \cdot (d-1) &\geq 2, \\ \alpha &\leq \frac{1}{12}\left[1 + k_{\mathrm{Gr}} \cdot (d-1)\right], \text{ and} \\ \tfrac{N}{m} &\text{ large enough.} \end{aligned}$$

As we see, infinite settings with oblivious deletion schemes and the finite setting are tightly related. The reason for this relation is that one can consider the finite setting as a special instance of the infinite setting by putting $\Delta(J) = \infty$ for each job J with $\delta(J) \leq \overline{M} = M$ and $\Delta(J) = \delta(J)$ for the remaining jobs.

The load of an optimal allocation has a trivial lower bound of $\frac{\overline{M}}{N}$. The full strength of our theorem is deployed if we assume that $\frac{\overline{M}}{N}$ is a constant. Then Theorem 5.1.4 shows that the load obtained by the greedy algorithm deviates at most $\log_d \log_{\epsilon_c} N + \mathrm{O}(\sigma_\Xi)$ from the optimum. The number of servers with load exceeding optimal load by more than t, decreases doubly exponentially with t, it is upper bounded by

$$\frac{N}{\epsilon_c^{(d^{t-\mathrm{O}(1)})}}.$$

The impact of the stupidity of the i-th copy distributions is similar to the impact of choosing a larger value for $\frac{\overline{M}}{N}$. The intuitive reason for this is that the stupidity corresponds to the utilization of the servers allowed by the i-th copy distributions. Poor i-th copy distributions have the same effect as a smaller number of servers. In fact a set of i-th copy distributions leaving all but $\frac{N}{a}$ servers untouched has stupidity a.

1.5.2 Example: the c-priority algorithm and a realistic deletion scheme

Considering oblivious deletion schemes is relatively straightforward, it requires few more than a close look to the computations applied to the finite case. This is entirely different for non-oblivious deletion schemes. Non-oblivious deletion schemes possibly depend on the possible servers of a job and the decisions of the allocation algorithm in consideration. As a consequence the random variable $S^{(i)}(J)$ fixing the i-th possible server of job J stochastically depends on whether J is in the system or not. Another problem is that the number of jobs in the system is unbounded. In Section 5.2.1.3 we demonstrate

that it is possible to overcome these problems. We consider the c-priority algorithm as given in Figure 5.1 on Page 79 working on an allocation problem. In the considered infinite parallel setting in each round up to $\vec{M}$ jobs enter the system. The c-priority algorithm has to allocate each job to a server, but in contrast to the sequential setting we allow that the algorithm defers the decision where to allocate a job. The number of rounds a job has to wait before it is allocated is an important quality measure in this setting.

In each round the c-priority algorithm allocates up to c jobs to a server. As a deletion scheme we consider the so-called *server oriented deletion scheme*. In each round this deletion scheme allows each server to delete up to c of the jobs allocated to it. Thus we can assume that a job is deleted from the system right after it is allocated. The c-priority algorithm makes use of a *priority rule* determining which jobs are allocated to a server in each round. Our result requires that this priority rule is *time preserving*, this simply means that a new job is never preferred over an old one for being allocated. This assumption is quite natural as it also prevents starvation.

> **(Theorem 5.2.7:)** Let $\bar{\tau}$ be a round. Let $\epsilon_c = (c!)^{\frac{1}{c}} \cdot 3^{d-1}$, and let $0 < \alpha \leq \frac{1}{12} \cdot (k_{\mathrm{pr}} \cdot c[d-1]+1)$. Further let $\eta > 1$. Consider the c-priority algorithm with time preserving priority rules working on an ordinary allocation problem in the infinite setting using the server oriented deletion scheme. Then at the end of round $\bar{\tau}$ there is no job waiting more than
>
> $$\eta \cdot (\log_{c[d-1]+1} \log_{\epsilon_c} N + k_{\mathrm{pr}})$$
>
> rounds for being allocated to a server, with probability at least $1 - N^{-\alpha + \mathrm{o}(1)}$, if
>
> $$\vec{M} \leq \min\left\{(3 \cdot \sqrt{\alpha} + 1)^{-1},\ 1 - \tfrac{1}{\eta} - \mathrm{o}(1)\right\} \cdot [(d + \mathrm{o}(1)) \cdot \sigma_{\Xi}]^{-1} \cdot c \cdot N.$$

A similar result holds for the c-priority algorithm in the weighted setting. The performance bounds for the ordinary and the weighted setting show the expected behavior: no job has to wait more than $\mathrm{O}(\log \log N)$ rounds for being served. In the ordinary case our upper bound generalizes the preliminary result found in [ABS98]. The new bound applies to arbitrary values of c and arbitrary i-th copy distributions. If uniform i-th copy distributions are used, as in [ABS98], it is possible to compare both results. The new result shows a tradeoff between the number of rounds a job has to wait for being allocated and the allowed throughput. The higher the throughput the higher is the bound on the waiting time. Our result from [ABS98] requires that $\vec{M} < \frac{N}{2de}$, while the new one allows any $\vec{M} \leq (1 - \varepsilon) \cdot \frac{c \cdot N}{d}$ (for any $\varepsilon > 0$). The maximum value of $\vec{M}$ is of particular importance as it determines the utilization of the system. At the first glance, a system with N servers performing c jobs per round should be able to deal with $\vec{M} < (1 - \varepsilon) \cdot c \cdot N$ jobs per round, for any $\varepsilon > 0$. The reason for our results being a factor of d away from this obvious upper bound is that we cannot prevent a job from

being accepted by up to d servers. If a job is accepted by several servers, none of these servers will perform another job in that round, we therefore can assume that a job is performed on each server accepting it. Our upper bound implicitly assumes that this worst case becomes true. If each job is performed on d servers, the natural upper bound on $\vec{M}$ is $(1-\varepsilon)\cdot\frac{N\cdot c}{d}$, which matches our upper bound on $\vec{M}$. Our new technique also allows us to derive a result for the performance of the weighted version of the c-priority algorithm. So far, no bound is known for this scenario.

1.5.3 An overview on the results

As noted, there are three major algorithms for the allocation problem: the greedy algorithm, the c-priority algorithm, and the c-collision algorithm. Each of these algorithms is investigated in each applicable setting. The *greedy algorithm* deals with the allocation problem in sequential settings, we investigate its performance for finite settings and for infinite settings using oblivious deletion schemes. As we will see, infinite settings with oblivious deletion schemes and finite settings are closely related. In either setting we allow any sensible i-th copy distributions and consider the ordinary case as well as the weighted one. To demonstrate the power of our Main Lemma we further apply it to Vöckings Always-go-Left version of the greedy algorithm.

The *c-priority algorithm* may be considered as a kind of parallelized version of the greedy algorithm. We state results on the performance of the c-priority algorithm for finite and for infinite settings and weighted or ordinary allocation problems. For infinite settings we go beyond the possibilities of oblivious deletion schemes and consider the non-oblivious server oriented deletion scheme. Again we allow almost arbitrary i-th copy distributions.

The other algorithm for the parallel setting the *c-collision algorithm* has no sensible application in infinite settings. We give performance bounds for it in the finite setting for weighted and ordinary allocation problems. As every time any sensible i-th copy distributions are allowed.

Beyond the allocation problem we apply the balls-into-bins paradigm to *circuit switching in butterfly type networks*. In particular we consider circuit switching in the *two-fold butterfly* BB_N, compare Figure 6.1 in Chapter 6. The two-fold butterfly is a multistage network formed by two copies of a butterfly network placed one after the other, such that the input nodes of the second butterfly are identified with the corresponding output nodes of the first one. Applying the balls-into-bins paradigm we randomly select two paths for each message. Then an algorithm selects one of the paths for each message. Like in the allocation problem where our aim is to minimize load, our aim is to minimize the maximum number of paths using a link in the network. As we see in Chapter 6 this approach allows to obtain good solutions using fast algorithms.

Some of the results of this thesis have already been published in a similar form. In any case the previous publication is bound to particular i-th copy distributions.

In [BMS97] the performance of the c-collision algorithm is considered for weighted allocation problems with three possible servers for each job in the finite setting. The analysis employed in [BMS97] is similar to a technique employed in [MPR94]. The core of the technique used in Chapter 4 is published in [BMS99], this version of the result is however bound to constant $d \geq 2$. A result on the c-priority algorithm in infinite settings using the server oriented deletion scheme has been published in [ABS98]. The proof given in this thesis differs from the one given there. The proof given here allows a better throughput, but yields a slightly worse bound on the waiting time of the jobs. The contents of Chapter 6 are part of [CMM+98].

1.6 Outline of this Thesis

The remainder of this thesis is organized as follows. In the remaining parts of the Introduction we present the greedy algorithm and the c-collision algorithm. These algorithms will serve as an example to demonstrate the use of the concepts and the results introduced in Chapter 3. Chapter 3 contains the core of our combinatorial argument and the definitions required to set it up. The results of Chapter 3 are subsumed in the Main Lemma (Lemma 3.2.6). In Chapter 4 we generalize the Main Lemma to weighted allocation problems. The "Main Lemma for Weighted Jobs" is presented as Lemma 4.3.3 in that chapter. The possibilities of both lemmas are applied to the various algorithms and settings in Chapter 5. Results which are not discussed here are discussed there. Chapter 6 deals with an application of the balls-into-bins paradigm to a routing problem. This chapter does not refer to the Main Lemma. It is a part of this thesis on its own and contains its own introduction in Section 6.1. Our results on the routing problem are discussed there.

1.7 The Example Algorithms

1.7.1 The c-collision algorithm

The c-collision algorithm is motivated by a conflict resolution rule used to resolve access conflicts in Distributed Memory Machines, called *c-collision rule* or – referring the whole DMM – the *c-collision DMM*. The c-collision rule is very simple: If at most c requests are sent to a memory module, the memory module accepts these requests; if the module receives more than c requests it accepts none of them. The c-collision algorithm mimics the behavior of the DMM modules in accepting jobs. A pseudo-code description of the c-collision algorithm in terms of the allocation problem is given in Figure 1.1. If in step (4) of the c-collision algorithm a job agent receives acknowledgments from more than one server, we do not care to which server the job is actually allocated. In the analysis we assume that job J is allocated to *each* server whose agent sends an acknowledgment to J. The c-collision algorithm defined in Figure 1.1 differs in several points from the

For all $J \in \mathcal{J}$ do in parallel
  $\text{agent}(J)$ becomes *active*.
$\tau := 0$

While there is at least one *active* job agent do (1)
  $\tau := \tau + 1$
  (This run trough the Loop (1) is called *round* τ.)
  For all $J \in \mathcal{J}$ do in parallel
    For $j \in \{1, 2, \ldots, d\}$ do (2)
      If $\text{agent}(J)$ is *active*,
        $\text{agent}(J)$ sends a (J, j, τ)-*request* to server agent $\text{agent}(S^{(j)})$.
  For all $S \in \mathcal{S}$ do in parallel
      If $\text{agent}(S)$ receives at most c $(\cdot, \cdot, \tau)$-*requests*, $\text{agent}(S)$ sends (3)
        a (S)-*acknowledgment* to each job agent which sent a
        $(\cdot, \cdot, \tau)$-request to S.
      (In this case $\text{agent}(S)$ is said to *accept its requests*.)
  For all $J \in \mathcal{J}$ do in parallel
      If for some $S \in \mathcal{S}$ the $\text{agent}(J)$ receives a (S)-*acknowledgment*, (4)
          $\text{agent}(J)$ becomes *inactive*,
          $\text{agent}(J)$ allocates J to server S.

Fig. 1.1: The c-collision algorithm

version given in [DM93, MSS95, Ste96]. A version of the c-collision algorithm which is more akin to the Access Schedule 2 is presented in Figure 5.5.

In prior work the performance of the c-collision algorithm has always been analyzed for particular choices of the i-th copy distributions. Considering the c-collision algorithm as an example application we assume that the i-th copy distributions are uniform on $\mathcal{S}$. Results for different i-th copy distributions are presented in Section 5.2.2.3.

Observation 1.7.1: If the agent of a server S accepts its requests in some round t, it sends an acknowledgment to all job agents sending a request to S. Thus, all these job agents become inactive in round t. As a consequence there is no job agent sending a request to S in any round $t' > t$, and hence, the load of S is at most c.

According to Observation 1.7.1, the load of the allocation produced by the c-collision algorithm is at most c. It is therefore sufficient to determine the running time of the algorithm.

1.7.2 The greedy algorithm in an infinite setting

The *greedy algorithm* which will serve as our second example is taken from [ABKU94]. It is perhaps the most simple and natural allocation algorithm for sequential settings.

The greedy algorithm is described in Figure 1.2. If a tie occurs in step (1), we apply a so-called *tie breaking mechanism* to chose one or several possible servers to allocate the job. Considering the greedy algorithm as an example algorithm we assume that in case of a tie, one arbitrary server is chosen from those with minimum load to allocate the job.

$\tau := 0$
While *true* do
  $\tau := \tau+1$
  agent(J_i) sends a request to the servers $S^{(1)}(J_i), \ldots, S^{(d)}(J_i)$.
  If server S receives a request, agent(S) replies with a message containing $load_\tau(S)$.
  agent(J_j) allocates J_j to a server with minimum load among $S^{(1)}(J_j), \ldots, S^{(d)}(J_j)$. (1)

Fig. 1.2: The greedy algorithm

As we consider the greedy algorithm in an infinite setting, a deletion scheme has to be specified. For some $\overline{M}$, we assume that $\Delta(J) = \delta(J)+\overline{M}$. This ensures that the system contains at most $\overline{M}$ jobs. Different deletion schemes are considered in Section 5.1.1. Treating the greedy algorithm as an example algorithm we restrict ourselves to uniform i-th copy distributions. More general distributions are subject of Section 5.1.1, too.

1.7.3 Comparing the c-collision and the greedy algorithm

The c-collision algorithm and the greedy algorithm have been chosen as examples for two reasons: their simplicity, their importance, and their variety. The c-collision algorithm achieves optimal performance up to a constant factor in the parallel setting; its relation to the parallel setting is quite close as its usage in a sequential setting is absurd. (Compare, however, [Kri99]). The greedy algorithm is optimal in sequential settings; its relation to this setting is very close, too. Moreover the greedy algorithm is able to deal with infinite settings — while the c-collision algorithm depends on the number of jobs being finite. The two algorithms also differ in the performance measure. While the c-collision algorithm always obtains load at most c and the key point of the analysis is to bound the running time, the key point in the analysis of the greedy algorithm is to determine the obtained load, the running time is not important at all.

2 Preliminary Section

This section contains some definitions and some useful Lemmas used in the remainder of this thesis.

2.1 Hypergraphs

Our work will heavily depend on structures described by hypergraphs.

Definition 2.1.1: Let V be a finite set, the set of *nodes*. Further let $E \subseteq \bigcup_{i \geq 2} V^i$ be a finite subset of the set of tuples over V. The set E is called the set of *edges*. An element $e = (v_1, \ldots, v_j) \in E$ is said to be *self-loop-free* if $v_i \neq v_{i'}$ for all $1 \leq i < i' \leq j$. If all elements of E are self-loop-free, the pair $G = (V, E)$ is called a *hypergraph*. A hypergraph $G = (V, E)$ is *d-uniform* if $E \subseteq V^d$. An edge $e \in E$ has *size* d if $e \in V^d$.

Let $e = (v_1, \ldots, v_j) \in E$ be an edge of a hypergraph G and let $v \in V$ be a node of G. The node v is said to be *incident* to e if there is an $1 \leq i \leq j$ with $v_i = v$. An edge e is *incident* to a node v if v is incident to e. If a node v is incident to i edges, the node v has *degree* i ($\deg(v) = i$). If $v \neq v'$ are nodes incident to the same edge $e \in E$, the nodes v and v' are said to be *adjacent*. For $j \geq 1$, a sequence of nodes $v_1, \ldots, v_j$ is called a *path connecting* v_1 *and* v_j if v_i is adjacent to v_{i+1} for each $1 \leq i \leq j-1$. The *length* of a path $v_1, \ldots, v_j$ is $j-1$. Two nodes v and w are said to have *distance* i ($\mathrm{dist}(v, w) = i$) if the shortest path connecting v and w has length i. If $W \subseteq V$ is a subset of the nodes, the *distance* between a node $v \in V$ and W is defined by $\min_{w \in W} \mathrm{dist}(v, w)$. Two paths $v_1, \ldots, v_i$ and $w_1, \ldots, w_j$ are *distinct* if $\{v_1, \ldots, v_i\} \cap \{w_1, \ldots, w_j\} = \emptyset$.

Remark 2.1.2: Sometimes we will treat an edge of a hypergraph like a set writing $v \in e$, if v is incident to e and $e \subseteq V$ if all nodes incident to e are in V. In the sequel we also assume that there is an arbitrary ordering, called *lexicographic ordering*, on the nodes V and edges E of a hypergraph $G = (V, E)$. Such an ordering exists on V and E as both sets are finite.

Definition 2.1.3: A hypergraph $G = (V, E)$ is *circle-free* if for each pair $v, w \in V$ there are no two distinct paths connecting v and w. The *connected component* of a node $v \in V$ is the set of all nodes w connected to v by a path. A graph is *connected* if all nodes are in the same connected component. A connected, circle-free hypergraph $G = (V, E)$ with a *root* ($root(G) \in V$) is called a *tree*.

Consider a tree $G = (V, E)$ and a node $v \in V$. A node $w \in V$, is called a *successor of* v if $w \neq v$ and the path between $\mathrm{root}(G)$ and w does contain v. If w is a successor of

v, then v is a *predecessor* of w. A successor w of v is called a *child of* v if $\text{dist}(v,w) = 1$. If w is a child of v, then v is called the *parent* of w. If an edge e is incident to a node v and its child w, then e is called a *child edge of* v and a *parent edge of* w. If e is the child edge of a node v, then v is the *top node of* e ($top(e)$). If e is the parent edge of a node w, then w is a *bottom node of* e. Note that each edge has a unique top node as a tree is circle-free. Each node has a unique parent edge for the same reason. To simplify our future discussions, we assume that for an edge $e = (e_1, \ldots, e_d)$ of a tree it holds $top(e) = e_1$ and $e_2 \prec e_3 \prec \cdots \prec e_d$.

The *height of a tree* $G = (V, E)$ is defined as the maximum distance of any node to the root ($\max_{v \in V} \text{dist}(v, root(G))$). If G is a tree of height $T \in \mathbb{N}$, the *level* of a node $v \in V$ is defined as $level(v) = T = \text{dist}(root(G), v)$. The *level of an edge* is the maximum of the levels of the incident nodes.

A tree is called *a c-ary tree* if the root of the tree has degree either 0 or c, and each node but the root has either degree 1 or degree $c+1$. A tree is called *a $(c+1, c)$-ary tree* if the root has either degree 0 or degree $c+1$ and each node but the root has either degree 1 or degree $c+1$. A tree is called *a complete tree* if for each two nodes $v, w \in V$ with $\text{dist}(v, root(V) = \text{dist}(w, root(v))$ it holds $\deg(v) > 1 \Leftrightarrow \deg(w) > 1$. In a tree a node $v \neq root(G)$ with degree at least 2 is called *an inner node* of the tree. A node $v \neq root(G)$ with degree at most 1 is called *a leaf* of the tree.

A graph is called a *forest* if each connected component is a tree.

2.2 Deviation Bounds

The following version of the popular Chernoff Bound is found in [HR90], Equation (6) and (7).

Lemma 2.2.1: Let $i \in \mathbb{N}$ and let $0 \leq p_1, \ldots, p_i \leq 1$. Let $X_1, \ldots, X_i$ be independent 0-1 random variables with

$$\text{Prob}[X_j = 1] = p_j \text{ for } j = 1, \ldots, i.$$

Further let $X = \sum_{j=1}^{i} X_j$ be the sum of the defined random variables. Then

$$\text{Prob}\big[X \geq (1+\varepsilon) \cdot \text{E}[X]\big] \leq \text{e}^{-\frac{\varepsilon^2 \text{E}[X]}{3}}$$
$$\text{Prob}\big[X \leq (1-\varepsilon) \cdot \text{E}[X]\big] \leq \text{e}^{-\frac{\varepsilon^2 \cdot \text{E}[X]}{2}}$$

for each $0 \leq \varepsilon \leq 1$.

We also make use of the following generalized version of the Chernoff Bound.

Lemma 2.2.2: Let $i \in \mathbb{N}$, let $0 \leq p_1, \ldots, p_i \leq 1$, and let $0 \leq x_1, \ldots, x_i \leq 1$. Let $X_1, \ldots, X_i$ be independent random variables with

$$\text{Prob}[X_j = x_j] = p_j \text{ and } \text{Prob}[X_j = 0] = 1 - p_j \text{ for } j = 1, \ldots, i.$$

Further let $X = \sum_{j=1}^{i} X_j$ be the sum of the defined random variables. Then

$$\mathrm{Prob}\big(X \geq (1+\varepsilon) \cdot \mathrm{E}(X)\big) \leq \mathrm{e}^{-\frac{\varepsilon^2 \mathrm{E}(X)}{3}}$$

for each $0 \leq \varepsilon \leq 1$.

Proof: Our proof follows the one given in [MR95] for the ordinary Chernoff Bound. But before proving the bound itself, we prove a useful inequality.

For each $0 \leq x \leq 1$ it holds

$$\begin{aligned} \mathrm{e}^{tx} - 1 &= -1 + \sum_{j=0}^{\infty} \frac{1}{j!} \cdot (tx)^j \\ &= \sum_{j=1}^{\infty} \frac{1}{j!} \cdot (tx)^j \\ &= x \cdot \sum_{j=1}^{\infty} \frac{1}{j!} \cdot t^j x^{j-1} \end{aligned}$$

as $0 \leq x \leq 1$ this is

$$\begin{aligned} &\leq x \cdot \sum_{j=1}^{\infty} \frac{1}{j!} \cdot t^j \\ &= x \cdot (-1 + \sum_{j=0}^{\infty} \frac{1}{j!} \cdot t^j) \\ &= x \cdot (\mathrm{e}^t - 1). \end{aligned}$$

For any $t \in (0, 1]$,

$$\mathrm{Prob}[X > (1+\epsilon) \cdot \mathrm{E}[X]] \;=\; \mathrm{Prob}[\mathrm{e}^{tX} > \mathrm{e}^{t(1+\epsilon)\mathrm{E}[X]}].$$

Applying the Markov Inequality to the right side yields

$$\mathrm{Prob}[X > (1+\epsilon) \cdot \mathrm{E}[X]] < \frac{\mathrm{E}[\mathrm{e}^{tX}]}{\mathrm{e}^{t(1+\epsilon)\mathrm{E}[X]}}. \tag{2.1}$$

We bound the right-hand side by observing that

$$\mathrm{E}[\mathrm{e}^{tX}] = \mathrm{E}[\mathrm{e}^{t \cdot \sum_{j=1}^{i} X_j}] = \mathrm{E}[\prod_{j=1}^{i} e^{tX_j}].$$

Since the X_j are independent, the random variables e^{tX_j} are independent, too. It follows that $\mathrm{E}[\prod_{j=1}^{i} e^{tX_j}] = \prod_{j=1}^{i} \mathrm{E}[e^{tX_j}]$. Using these facts in Equation 2.1 gives

$$\mathrm{Prob}[X > (1+\epsilon)\cdot \mathrm{E}[X]] < \frac{\prod_{j=1}^{i} \mathrm{E}[\mathrm{e}^{tX_j}]}{\mathrm{e}^{t(1+\epsilon)\mathrm{E}[X]}}. \tag{2.2}$$

The random variable e^{tX_j} assumes the value e^{tx_j} with probability p_j, and the value 1 with probability $1-p_j$. Computing $\mathrm{E}[e^{tX_j}]$ from these observations, we have that

$$\begin{aligned}\mathrm{Prob}[X > (1+\epsilon)\cdot \mathrm{E}[X]] &< \frac{\prod_{j=1}^{i}(p_j e^{tx_j} + 1 - p_j)}{\mathrm{e}^{t(1+\epsilon)\mathrm{E}[X]}} \\ &= \frac{\prod_{j=1}^{i}(1 + p_j(e^{tx_j} - 1))}{\mathrm{e}^{t(1+\epsilon)\mathrm{E}[X]}}\end{aligned}$$

according to the inequality established at the beginning of the proof

$$\leq \frac{\prod_{j=1}^{i}(1 + p_j x_j(e^{t} - 1))}{\mathrm{e}^{t(1+\epsilon)\mathrm{E}[X]}}.$$

Now we use the inequality $1 + z < \mathrm{e}^z$ with $z = p_j x_j(\mathrm{e}^t - 1) = \mathrm{E}[X_j](\mathrm{e}^t - 1)$ and obtain

$$\begin{aligned}\mathrm{Prob}[X > (1+\epsilon)\cdot \mathrm{E}[X]] &\leq \frac{\prod_{j=1}^{i} \mathrm{e}^{\mathrm{E}[X_j](\mathrm{e}^t-1)}}{\mathrm{e}^{t(1+\epsilon)\mathrm{E}[X]}} \\ &= \frac{\mathrm{e}^{\sum_{j=1}^{i} \mathrm{E}[X_j](\mathrm{e}^t-1)}}{\mathrm{e}^{t(1+\epsilon)\mathrm{E}[X]}} \\ &= \frac{\mathrm{e}^{(e^t-1)\cdot \mathrm{E}[X]}}{\mathrm{e}^{t(1+\epsilon)\mathrm{E}[X]}}.\end{aligned}$$

Choosing $t = \ln(1+\epsilon)$ and observing $\epsilon - (1+\epsilon)\ln(1+\epsilon) \leq -\frac{1}{3}\epsilon^2$ yields the desired result. ■ of Lemma 2.2.2

The following upper bound is not a deviation bound, but it is not worth an own section, so we include it here.

Lemma 2.2.3: For any positive $a, x \in \mathbb{R}$

$$\left(\frac{a\cdot \mathrm{e}}{x}\right)^x \leq \mathrm{e}^a.$$

Proof: The derivative with respect to x of the left side is

$$\left(\frac{a\cdot \mathrm{e}}{x}\right)^x \cdot \left(\ln\left(\frac{a\cdot \mathrm{e}}{x}\right) - 1\right).$$

The derivative is zero if and only if $x = a$ and it is decreasing in x at a, thus the left side is maximal for $x = a$ which yields the claim. ■ of Lemma 2.2.3

3 The Witness Tree Analysis

Our aim is to provide a general proof for different bounds on the performance of several allocation algorithms. The price to be paid for this generality is a quite complex system of notions (compare [Luh84]). The proof concept is called *witness tree analysis* for historical reasons only. We make use of forests rather than trees, as they allow to obtain more general results.

The intrinsic concept of the witness tree analysis is the *indirect proof*. But instead of proving that the converse of the proposition is wrong, the converse of the proposition is shown to be very unlikely in the witness tree analysis. As we are interested in the performance of the algorithm we thus show that bad performance does not occur, with high probability. To prove that, we show that a certain structure — a *witness forest* with a *representation* — occurs whenever an allocation algorithm performs bad. After that we show that there is no witness forest with a representation, with high probability.

Together with Chapter 4 this chapter contains the most technical part of this thesis. We like to emphasize that among all concepts introduced in this chapter only a few are important outside this chapter. The notions "witness forest", "complete explaining representation", and "vivid system" are the only ones being of further importance. Using these concepts we summarize the contribution of the chapter in the Main Lemma (Lemma 3.2.6). Before stating the Main Lemma we show how to use our notions for the example algorithms in an informal way. After that, in Section 3.2, we state our Main Lemma, preceded by the notions needed to state it. Then we apply the Main Lemma to our example algorithms. The remainder of the chapter then contains the proof of the Main Lemma.

3.1 Witness Forests for the Example Algorithms

3.1.1 A witness forest for the c-collision algorithm

As a first step, we define a structure which occurs if a server is active at the end of a round $T \in \mathbb{N}$. To describe the structure we use a complete d-uniform $(c+1, c)$-ary tree $G_{\text{Tree}} = (V_{\text{Tree}}, E_{\text{Tree}})$ of height T. In G_{Tree} each inner node has degree $c+1$. The tree G_{Tree} serves as a framework for the *representation*. The representation associates a server $server(v) \in \mathcal{S}$ with each node v of G_{Tree} and a job $job(e) \in \mathcal{J}$ with each edge of G_{Tree}. A node $v \in V_{\text{Tree}}$ (an edge $e \in E_{\text{Tree}}$) is said to *represent* $server(v)$ ($job(e)$). The key point of the representation is that a server represented by a node v

receives requests from the jobs represented by the edges incident to v. Conversely a job represented by an edge e sends requests to the d servers represented by the d nodes incident to e. To tell apart which request is sent to which server we define a mapping $label_e : \{v_1, \ldots, v_d\} \to \{1, \ldots, d\}$ for each edge $e = (v_1, \ldots, v_d)$, such that for each $v \in e$ we have $server(v) = S^{(label_e(v))}(job(e))$.

Before we actually define the representation, we give some technical definitions using the notions introduced in Section 2.1. Let '$<$' be an ordering $J \times \{1, \ldots, d\}$ with $(J, j) < (J', j')$ if $J < J$ or $J = J'$ and $j < j'$. Further, for an edge $e = (v_1, \ldots, v_d)$ let $label_e^{(j)} : \{v_1, \ldots, v_d\} \to \{1, \ldots, d\}$ be a mapping with

$$label_e^{(j)}(v) = \begin{cases} j & \text{if } v = v_1 = top(e) \\ i - 1 & \text{if } v = v_i \text{ and } 2 \leq i < j \\ i & \text{if } v = v_i \text{ and } i > j \end{cases}, \text{ for each } v \in e. \tag{3.1}$$

To relate the representation to the behavior of the c-collision algorithm, fix the possible servers of the jobs and assume that a server S is active at the end of round T. Assume that the root $root(G_{\text{Tree}})$ represents S. As S is active at the end of round T, it receives $(c+1)$ requests $(J_1, j_1), \ldots, (J_{c+1}, j_{c+1})$ in round T. Thus to obtain a representation as described above, let $e_1 < \cdots < e_{c+1}$ be the child edges of $root(G_{\text{Tree}})$. For each $1 \leq i \leq c+1$ let $job(e_i) = J_i$, and $label_{e_i} = label_{e_i}^{(j_i)}$. Further for each edge $e \in \{e_1, \ldots, e_{c+1}\}$, $e = (v_1, \ldots, v_d)$ let $server(v_i) = S^{(label_e(v))}(job(e))$, for $2 \leq i \leq d$. The job $job(e)$ sends a request to server S in round T, thus it is active at the end of round $T - 1$. Thus the servers represented by the bottom nodes of e are active at the end of round $T - 1$.

Now let v' be a bottom node of e. As the server S' represented by v' is active at the end of round $T - 1$, it receives some requests $(J'_1, j'_1) < \cdots < (J'_c, j'_c)$ besides the request $(job(e), label_e(v'))$. As done above for the root of G_{Tree}, let $e'_1 < \ldots, e'_c$ be the child edges of v' and for each $1 \leq i \leq c$ let $job(e'_i) = J'_i$ and $label_{e'_i} = label_{e'_i}^{(j'_i)}$. For each bottom node v'' of e'_i let $server(v'') = S^{(label_{e'_i}(v''))}(job(e'_i))$. $server(v'')$ is active at the end of round $T - 2$, thus we can continue this construction recursively to the leaves of G_{Tree} which represent servers active at the end of round 0.

If m servers $S_1 < \cdots < S_m$ are active at the end of round T, we can define a representation on a forest $G_{\text{Forest}}^{\text{c-coll}}$ of m trees which are isomorphic to G_{Tree}. To do so we simply start the construction above by choosing S_i to be represented by the root of the i-th tree, for $1 \leq i \leq m$.

3.1.2 A witness forest for the greedy algorithm

The structure occuring when a server has load at least $T' \in \mathbb{N}$ at a time step $\tau \in \mathbb{N}$ of the greedy algorithm is described using a different tree. Let $G_{\text{Tree}} = (V_{\text{Tree}}, E_{\text{Tree}})$

be a complete 2-ary tree of height T where each inner node v has two child edges: one edge of size 2 — called the *irregular child edge of* v — and one edge of size d — called the *regular child edge of* v. For each $v \in V_{\text{Tree}}$ the *irregular component of* v contains all nodes v' connected to v by a path of irregular edges. We define a representation on G_{Tree}. Each node $v \in V_{\text{Tree}}$ represents a server $\mathit{server}(v) \in \mathcal{S}$ and each *regular* edge e represents a job $\mathit{job}(e) \in \mathcal{J}$. The irregular edges do not represent anything. We also define a mapping label_e for each regular edge e, such that if v is incident to e, then $\mathit{server}(v) = S^{(\mathit{label}_e(v))}(\mathit{job}(e))$ as in Section 3.1.1. If a node v' is the bottom node of the irregular child edge of a node v, then $\mathit{server}(v') = \mathit{server}(v)$, thus the mapping *server* is a constant on an irregular component. In defining the representation each node is assigned a *time marker*. The key observation in defining the representation is that each node v of level l has $\mathit{load}_\tau(\mathit{server}(v))$ if its time marker is τ.

To define the representation fix the possible servers of the jobs and assume that there is a server S with $\mathit{load}_\tau(S) \geq T$ at the beginning of time step τ of the greedy algorithm. Let the root v of G_{Tree} represent server S, and let the *time marker* of v be τ. Consider a node v with time marker τ and $\mathit{server}(v) = S$. We show how to choose *job*, *server*, *label*, and a time marker for the child edges and the children of v. Let J be the last job allocated to S before or in step τ. Assume that S received request (J, i). Then let the regular child edge e of v represent J and let $\mathit{label}_e(v) = i$. For the bottom nodes $v_2, \ldots, v_d$ of e let $\mathit{server}(v_j) = S^{(\mathit{label}_e(v_j))}(J)$, and let their time marker be $\delta(J) - 1$. Further let the time marker of the bottom node of v's irregular child be $\delta(J) - 1$, too. Now each level 1 node v' with time marker τ' represents a server S' with $\mathit{load}_{\tau'}(S') \geq T - 1$. Continuing the construction recursively, we define the representation on the whole tree G_{Tree}.

The tree $G_{\text{Tree}} = (V, E)$ in not the tree we are really interested in. To get the correct tree, for some $k_{\text{Gr}} \in \mathbb{N}$, identify the k_{Gr} upmost nodes in the irregular component of the root of G_{Tree}, in the modified tree remove all nodes and edges of level greater than $T - k_{\text{Gr}}$, and call the resulting tree $G^{\text{Gr}}_{\text{Tree}}$. The root of $G^{\text{Gr}}_{\text{Tree}}$ has k_{Gr} regular child edges, each level 1 node of $G^{\text{Gr}}_{\text{Tree}}$ is the root of a complete 2-ary tree of height $T - k_{\text{Gr}} - 1$ whose inner nodes have d children. And as the representation is constant on an irregular component, the representation $\mathcal{R}_{\text{Gr}}$ is well defined on $G^{\text{Gr}}_{\text{Tree}}$.

We consider the greedy algorithm in an infinite setting, thus it may seem that the number of jobs possibly represented by an edge of $G^{\text{Gr}}_{\text{Tree}}$ is unbounded. But it is easy to see that the choices for the jobs are quite restricted. As the server S represented by the root of $G^{\text{Gr}}_{\text{Tree}}$ has $\mathit{load}_\tau(S) \geq T'$ at the end of round τ the jobs represented by the child edges of the root are in the system at the end of time step τ. Similar observations hold for each node v': if its time marker is τ' then the job represented by the regular child edge of v' is in $\mathcal{J}^{\delta,\Delta}_{\tau'}$.

As in the section above we can define a representation on a forest $G_{\text{Forest}} = (V_{\text{Forest}}, E_{\text{Forest}})$ of m trees isomorphic to $G^{\text{Gr}}_{\text{Tree}}$, if m servers have load at least T' at the beginning of time step τ.

3.2 The Main Lemma

In Section 3.1 two kinds of trees are presented on which a structure called 'representation' exists, if the performance of the c-collision algorithm or the greedy algorithm is bad. Both the witness trees and the representation are vital for the understanding of the Main Lemma which states that such a representation does not exist on a witness tree, with high probability. In this section we give a formal definition of the structures considered above, i.e. a formal definition of *witness forests* and a formal definition of a *representation*. We further define the term *"vivid system"* to encapsulate relevant properties of the i-th copy distributions and the allocation problem. The formal definitions allow to upper bound the probability that there is a representation on a witness forest, without referring to a particular algorithm or allocation problem.

Definition 3.2.1: Let $c, d \in \mathbb{N}$. We define a (c, d) *witness tree* recursively.

- A d-uniform tree $G = (V, E)$ where each inner node has c child edges is a (c, d) witness tree. The edges in E are called the *regular edges* of the witness tree G and the nodes in V are called *regular nodes*.

- If $G = (V, E)$ and $G' = (V', E')$ are (c, d) witness trees and $v \in V$ has no *irregular child edge*, then

$$G'' = \left(V \cup V' \,,\, E \cup E' \cup \{(v, root(G'))\} \right)$$

  is also a (c, d) witness tree. The root of G'' is defined to be $root(G)$. The edge $(v, root(G'))$ is called an *irregular edge* or the *irregular child edge of* v. The node $root(G')$ is called the *appendant child* of v and v is called the appendant parent of $root(G')$ and the graph G' is called the *appendant of* v.

A path $v_1, \ldots, v_j$ in G_{Forest} is *irregular* if for each $1 \leq i \leq j - 1$ the edge incident to v_i and v_{i+1} is an irregular edge. The *irregular component of a node* v ($irr(v)$) is the set of all nodes w connected to v by an irregular path. For a level 1 node v of a (c, d) witness tree $G = (V, E)$ let the *level 1 subtree of* v $\text{sub}(v)$ be the subgraph of G induced by the successors of v. Let $e_{\text{sub}}(v)$ denote the number of regular edges of $\text{sub}(v)$. A (c, d) witness tree $G = (V, E)$ is a $(b, e_{\text{sub}}, c, d)$ witness tree, if the root of G has b children and $e_{\text{sub}}(v) \geq e_{\text{sub}}$ for each level 1 node $v \in V$. This ends Definition 3.2.1.

In a witness tree each node has at most one appendant child. Typically witness trees are not d-uniform as the edges connecting a node with its appendant child have size 2, only.

Definition 3.2.2: Let $m, b, e_{\text{sub}}, c, d \in \mathbb{N}$, and let $G_1 = (V_1, E_1), \ldots, G_m = (V_m, E_m)$ be $(b, e_{\text{sub}}, c, d)$ witness trees. Let

$$V_{\text{Forest}} = \bigcup_{i=1}^{m} V_i$$
$$E_{\text{Forest}} = \bigcup_{i=1}^{m} E_i$$

and let

$$G_{\text{Forest}} = (V_{\text{Forest}}, E_{\text{Forest}}),$$

then G_{Forest} is called *an* $(m, b, e_{\text{sub}}, c, d)$ *witness forest*. Let

$$root(G_{\text{Forest}}) = \bigcup_{i=1}^{m} root(G_i)$$

be the set of the roots of the witness trees in G_{Forest}. If not stated otherwise, the witness forest in consideration is always denoted as $G_{\text{Forest}} = (V_{\text{Forest}}, E_{\text{Forest}})$ in the sequel. The set of regular nodes of G_{Forest} is denoted by E_{Reg}.

Remark 3.2.3: An example for a witness forest is the forest $G^{\text{c-coll}}_{\text{Forest}}$ considered for the c-collision algorithm in Section 3.1.1. It is a witness forest as its components are d-uniform trees. The forest $G^{\text{Gr}}_{\text{Forest}}$ defined in Section 3.1.2 is a witness forest, too. To obtain a tree $G^{\text{Gr}}_{\text{Tree}}$ of $G^{\text{Gr}}_{\text{Forest}}$ start with a d-uniform tree of height $T - k_{\text{Gr}}$ whose root has k_{Gr} child edges and each node of level l, $1 \leq l \leq T - k_{\text{Gr}} - 1$, has one child edge. After that we recursively add a complete c-ary d-uniform tree of height $T - k_{\text{Gr}} - level(v)$ to each node v with $1 \leq level(v) \leq T - k_{\text{Gr}} - 1$, and connect it using an irregular edge. This construction is applied recursively to all nodes, not only to the ones in the tree we started with.

The witness forest as defined above is just the skeleton of the structure occuring when an algorithm performs bad. To put it to live, we have to relate this structure to the outcomes of the random choices of the possible servers of the jobs. This is done by a *representation*. We define representations not only for witness forests, but also for the weighted witness forests we define in Definition 4.3.1 of Section 4.3. For the sake of this chapter the definition can be considered as being restricted to ordinary witness forests.

Definition 3.2.4: Let G_{Forest} be a (weighted) witness forest. Let $V_{\text{Server}} \subseteq V_{\text{Forest}}$ be a subset of the nodes of G_{Forest}, and $E_{\text{Job}} \subseteq E_{\text{Reg}}$ be a subset of the *regular* edges. For each $e = (v_1, \ldots, v_d) \in E_{\text{Job}}$ let $label_e = label_e^{(j)}$ for some $j \in \{1, \ldots, d\}$ called the *apex*

of e. Let Label $= \{label_e | e \in E_{\text{Job}}\}$ be the set of the mappings $label_e$ of all edges. Further let

$$server : V_{\text{Server}} \to \mathcal{S}$$

be a mapping fulfilling $server(v) = server(v')$ if v is the appendant parent of v'. Let

$$job : E_{\text{Job}} \to \mathcal{J}$$

be an arbitrary mapping. The tuple $\mathcal{R} = (\text{Label}, server, V_{\text{Server}}, job, E_{\text{Job}})$ is called a *representation* on the (weighted) witness forest G_{Forest}. A representation $\mathcal{R}$ is called *explaining on* $V \subseteq V_{\text{Forest}}$ if for each $v, v' \in V$ and each $e, e' \in \overline{E} = \{\bar{e} \in E_{\text{Reg}} | \bar{e} \subseteq V\}$ the following Property 1 is fulfilled.

1. (a) If $v \in V_{\text{Server}}$ is incident to e, and if $e \in E_{\text{Job}}$, then $S^{(label_e(v))}(\text{job}(e)) = server(v)$.
   (b) If $e \in E_{\text{job}}$ is incident to v and $e' \in E_{\text{job}}$ is incident to v', $v' \in irr(v)$, $\text{Job}(e) = \text{Job}(e')$, and $label_e(v) = label_{e'}(v)$, then $e = e'$.
   (c) If $v, v' \in root(G_{\text{Forest}})$ and $v < v'$, then $server(v) < server(v')$.
   (d) If $e, e' \in E_{\text{Job}}$ have the same top node and $e < e'$, then $job(e) < job(e')$ or $job(e) = job(e')$ and $label_e(top(e)) < label_e(top(e'))$.

A representation is called *complete explaining* if it is explaining on $V = V_{\text{Server}} = V_{\text{Forest}}$ and the set E_{Job} contains all regular edges in E_{Forest}. (These conditions are fulfilled by the representations defined in Sections 3.1.1 and 3.1.2, if we assume that $V_{\text{Server}} = V_{\text{Tree}}$ and E_{Job} is the set of regular edges in G_{Forest}.)

The final definition we need to state our Main Lemma, defines what we call a *vivid system* for an allocation problem. Its main task is to cover the properties of the i-th copy distributions of an allocation problem. It is independent of the algorithm in consideration. The proof of our Main Lemma is based upon enumeration of all possible representations. For each root v of a witness forest we therefore have to know how many servers can be represented by v. Throughout this thesis, this value is almost every time N, the number of servers. For each child edge e of v, we need to know how many jobs can be represented by that edge, and we need to know how many ways there are to choose a mapping $label_e$. It is, for instance, possible that only one possible server of a job J can be equal to a server S with non-zero probability. Then if $server(v)$ and $\text{job}(e) = J$, there is only one choice for $label_e$. Once $label_e$ is fixed, this restricts the choices for the servers being represented by the bottom nodes of e, and so on. This is how the vivid system depends on the distributions (not the outcomes) of the possible servers of the jobs.

Another restriction on the possible representation can often be derived from the deletion scheme. As an example consider the representation defined for the greedy

algorithm in the previous section. If an edge e represents job J then each bottom node v of e had time marker $\tau = \delta(J)$. This restricted the choice for the jobs represented by the child edges of v.

Let $J_1, \ldots, J_k \in \mathcal{J}$, $S_1, \ldots, S_k \in \mathcal{S}$ and let $i_1, \ldots, i_k \in \{1, \ldots, d\}$. Then the event

$$\mathcal{E} = \big((S^{(i_1)}(J_1) = S_1) \wedge \cdots \wedge (S^{(i_k)}(J_k) = S_k)\big)$$

is called a *simple event*, if $\text{Prob}[\mathcal{E}] > 0$. The event $\mathcal{E}$ is said to be *about* (J, i), $J \in \mathcal{J}$, $i \in \{1, \ldots, d\}$, if $(J, i) \in \{(J_1, i_1), \ldots, (J_k, i_k)\}$. The most trivial example for a simple event is '$S^{(i)}(J) = S$'.

Definition 3.2.5: Consider an allocation problem with N servers. Let $N', M, d', \in \mathbb{N}$ and $0 \leq p, \mu \leq 1$. Let $G_{\text{Forest}} = (V_{\text{Forest}}, E_{\text{Forest}})$ be a witness forest. Let $\mathcal{R}$ be a representation on G_{Forest}.

- For each root v and each $S \in \mathcal{S}$ let $\mathit{vividrj}(S, v) \subseteq \mathcal{J}$, such that $|\mathit{vividrj}(S, v)| \leq M$.
- For each $J \in \mathcal{J}$, $i \in \{1, \ldots, d\}$ let $\mathit{vivids}(J, i) \subseteq \mathcal{S}$, such that $|\mathit{vivids}(J, i)| \leq N'$, and
- for each $S \in \mathit{vivids}(J, i)$ let $\mathit{vividj}(J, S) \subseteq \mathcal{J}$, such that $|\mathit{vividj}(J, S)| \leq M$.
- Assume further that for each simple event $\mathcal{E}$ ($\mathcal{E}'$) which is not about (J, i) ((J', i))

$$\begin{aligned} &\text{Prob}[S^{(i)}(J) = S \,|\, S \in \mathit{vivids}(J) \wedge \mathcal{E}] \leq p \text{ and} \\ &\text{Prob}[S^{(i)}(J') = S \,|\, S \in \mathit{vivids}(J) \wedge J' \in \mathit{vividj}(J, S) \wedge \mathcal{E}'] \leq p, \end{aligned} \tag{3.2}$$

- Let $d' \leq d$ such that for each $S \in \mathcal{S}$ and for each $J \in \mathcal{J}$ we have

$$\big|\big\{i \in \{1, \ldots, d\} \,\big|\, \text{Prob}[S^{(i)}(J) = S] > 0\big\}\big| \leq d'.$$

- For each node $v \in V_{\text{Server}}$ which is not bottom node of an irregular edge let $\mathit{vivide}_{\mathcal{R}}(v)$ be an event. Let $\mathcal{E}$ denote the event

$$\bigwedge_{\substack{w \in V_{\text{Server}} \\ \mathit{server}(w) \neq \mathit{server}(v)}} \mathit{vivide}_{\mathcal{R}}(w)$$

  Then $\mathit{vivide}_{\mathcal{R}}(v)$ has to fulfill

$$\text{Prob}[\mathit{vivide}_{\mathcal{R}}(v) \,|\, \text{“}\mathcal{R} \text{ is explaining”} \wedge \mathcal{E}] \leq \mu$$

  (Note that $\mathit{vivide}_{\mathcal{R}}$ depends on the representation $\mathcal{R}$.)

$\mathcal{V} = (\mathit{vividrj}, \mathit{vivids}, \mathit{vividj}, \mathit{vivide})$ is called a (N, N', p, M, d', μ)-*vivid system*. The mappings *vividrj* and *vividj* are called *vivid sets of the edges*, *vivids* is called *vivid set of the nodes*, and $\mathit{vivide}_{\mathcal{R}}$ is called *the set of vivid events*.

The representation $\mathcal{R}$ is said to use the vivid system $\mathcal{V}$, if for each root w of G_{Forest} and each regular child edge e_w of w

- $\text{job}(e_w) \in \mathit{vividrj}(\mathit{server}(w), w)$.

and for each $e \in E_{\text{Forest}}$ and each bottom node v of e

- $\mathit{server}(v) \in \mathit{vivids}(\mathit{job}(e), \mathit{label}_e(v))$,
- if v has a child edge e', then $\mathit{job}(e') \in \mathit{vividj}(\mathit{job}(e), \mathit{server}(v))$, and
- $\mathit{vivide}_{\mathcal{R}}(v)$ is true.

If we do not care about the values N, N', p, M, d', μ, then we simply call $\mathcal{V}$ a *vivid system*. Frequently we just do not care about *vivide*, then we say that (*vividrj*, *vivids*, *vividj*) is a $(N, N', p, M, d', 1)$-vivid system. We further allow that the mappings *vividrj* and *vividj* take a third parameter from V_{Forest}. If this parameter is not mentioned, we assume that the mappings are constant with respect to their third parameter. This ends Definition 3.2.5.

An example for a vivid system is the *trivial vivid system* with $\mathit{vividrj}(S, v) = \mathcal{J}$ for each root v of G_{Forest} and each server $S \in \mathcal{S}$, $\mathit{vivids}(J, i) = \mathcal{S}$ for each J, i, and $\mathit{vividj}(J, S) = \mathcal{J}$ for each J and S. If the i-th copy distributions are uniform, the trivial vivid system is a $(N, N, N^{-1}, M, d, 1)$-vivid system. Another example is the *natural vivid system for step* τ. It is defined by $\mathit{vividrj}(S, v) = \mathcal{J}_\tau^{\delta,\Delta}$ for each root of G_{Forest} and each server $S \in \mathcal{S}$, $\mathit{vivids}(J, i) = \mathcal{S}$ for each J, i, and $\mathit{vividj}(J, S) = \mathcal{J}_{\delta(J)}^{\delta,\Delta}$ for each J and S. If the deletion scheme ensures that the system never contains more than $\overline{M}$ jobs and if the i-th copy distributions are uniform, the natural vivid system for step τ, called $\mathcal{V}_\tau$, is a $(N, N, N^{-1}, \overline{M}, d, 1)$-vivid system for any τ.

Similar to the definition for the various kinds of representations, the definition above may seem quite complicated. Its task is however quite simple: the vivid system describes the range of possible decisions in building up a representation. This range depends on the properties of the allocation problem, mainly its i-th copy distributions, and the deletion scheme. In a (N, N', p, M, d', μ)-vivid system N denotes the number of ways to choose a server for the root of a witness tree, in any case we consider in the sequel it is equal to the number of servers. M shows how many servers can be represented by an edge. If J is represented by an edge e and a bottom node $v \in e$ represents S, then $\mathit{vividj}(J, S)$ contains all jobs allowed for the child edges of v. *vividrj* and *vividj* play an important role in the infinite setting, as our Main Lemma is based upon enumeration. Therefore it is vital for us to avoid infinite sets. The parameters N', p, and d' are much less spectacular, they simply help us evading direct consideration of the allocation problems i-th copy distributions. The events $\mathit{vivide}_{\mathcal{R}}$ will contain events which are hard to represented by a witness tree. The simple events $\mathcal{E}$ and $\mathcal{E}'$ used in Equation 3.2 shelter us from possible dependencies.

The following Main Lemma is the main technical result of this thesis.

Lemma 3.2.6 (Main Lemma): Let $m' = \left\lceil \frac{m}{b+1} \right\rceil$ and

$$\bar{e}_{\text{sub}} = \log_{\epsilon_c} \left(4^{\frac{1}{m' \cdot b}} \cdot N^{\frac{\alpha}{m' \cdot b}} \cdot \left(\frac{N}{m'} \cdot \mathrm{e}^3 \right)^{\frac{1}{b}} \right).$$

Let G_{Forest} be a $(m, b, \bar{e}_{\text{sub}}, c, d)$ witness forest. Then the probability of G_{Forest} having a complete explaining representation using a (N, N', p, M, d', μ)-vivid system is at most

$$\left(\frac{N}{m'} \right)^{m'} \cdot e^{3m'} \cdot \epsilon_c^{-\bar{e}_{\text{sub}} \cdot m' \cdot b} \leq N^{-\alpha}$$

for

$$\Lambda = (N' \cdot \mu \cdot p), \quad (\epsilon_c)^c = \frac{c!}{(M \cdot d' \cdot p \cdot \Lambda^{d-1})^c} > 1, \quad \alpha \leq \frac{1}{12} \cdot b \cdot \frac{\ln(\frac{N}{m'} \cdot \mathrm{e}^3) \cdot m'}{\ln N},$$

and N/m' is large enough (cf. Equations 3.20 and 3.21 in the proof of Lemma 3.4.4).

Proof: The proof of this Lemma is subject of Section 3.4. ∎

Remark 3.2.7: Frequently we consider a logarithm of $\bar{e}_{\text{sub}}$ assuming $\alpha \leq \frac{1}{12} \cdot b$. For each $a \geq 2$ we have

$$\begin{aligned}
\log_a \bar{e}_{\text{sub}} &= \log_a \log_{\epsilon_c} \left(4^{\frac{1}{m' \cdot b}} \cdot N^{\frac{\alpha}{m' \cdot b}} \cdot \left(\frac{N}{m'} \cdot \mathrm{e}^3 \right)^{\frac{1}{b}} \right) \\
&\leq \log_a \log_{\epsilon_c} \left(4^{\frac{1}{m' \cdot b}} \cdot \left(\frac{N}{m'} \cdot \mathrm{e}^3 \right)^{\frac{1}{b} \cdot (\alpha+1)} \right) \\
&\leq \log_a \left(\frac{1}{b} \cdot (\alpha+1) \cdot \log_{\epsilon_c} \left(\frac{N}{m} \right) + \log_{\epsilon_c} \left(4 \cdot \mathrm{e}^3 \right) \right) \\
&\leq \log_a \log_{\epsilon_c} \left(\frac{N}{m} \right) + \mathrm{o}(1).
\end{aligned}$$

The parameter 'Λ' of the Main Lemma may seem a little bit awkward. But it has a quite natural interpretation: if an edge e represents some job J, then Λ denotes the expected number of servers which can be represented by one of the bottom nodes of e. Λ^{d-1} thus denotes the expected number of ways to choose servers for being represented by all bottom nodes. If one fixes the server being represented by the top node of e, $M \cdot d' \cdot p \cdot \Lambda^{d-1}$ is the expected number of ways to choose a representation for an edge and its bottom nodes. This also leads to an interpretation for $(\epsilon_c)^c$. $(\epsilon_c)^{-c}$ is the expected number of representations for the child edges and children of an inner node of G_{Forest}. Then e_{sub} has to be large enough to ensure that the expected number of representations

for G_{Forest} gets small enough. The proof for Lemma 3.4.4 would be quite easy, if the expansion nodes and edges won't come in. They are, however, necessary to deal with possible stochastic dependencies of considered events.

The condition $\epsilon_c > 1$ can be satisfied in several ways. In [MSS95], where the witness tree is used for the first time, $M \leq \epsilon \cdot N$ is used for some constant $\epsilon < 1$ small enough. Another natural way is to use a large c. In this thesis we frequently assume that $\Lambda < 1$.

Remark 3.2.8: For the c-collision algorithm and the greedy algorithm representations on some forests are defined in Sections 3.1.1 and 3.1.2. It remains to clarify that in both cases the definitions coincide with the formal definitions made above.

If m servers are active at the end of round T of the *c-collision algorithm* the forest $G^{\text{c-coll}}_{\text{Forest}}$ contains m connected components. Each connected component is isomorphic to a complete d-uniform $(c+1, c)$-ary tree G_{Tree} of height T. The level 1 subtrees of G_{Tree} each contain $e_{\text{sub}} = c \cdot \sum_{i=0}^{T-1} [c(d-1)]^i$ edges, thus $G^{\text{c-coll}}_{\text{Forest}}$ is a $(m, (c+1)(d-1), e_{\text{sub}}, c, d)$ witness forest.

The representation $\mathcal{R}_{\text{c-coll}} = (\text{Label}, \mathit{server}, V_{\text{Forest}}, \mathit{job}, E_{\text{Forest}})$ using the mappings *server* and *job* is a complete explaining representation on $G^{\text{c-coll}}_{\text{Forest}}$. It uses the trivial vivid system.

For the *greedy algorithm* the forest $G^{\text{Gr}}_{\text{Forest}}$ defined in Section 3.1.2 is a witness forest according to Remark 3.2.3. Each root of G_{Tree} has k_{Gr} regular child edges and one irregular child edge. Each level 1 subtree of G_{Tree} contains $e_{\text{sub}} = \sum_{i=0}^{T-1} [(d-1)+1]^i$ edges, as each inner node has one regular child edge with $d-1$ bottom nodes and one irregular child edge with one bottom node. Thus the forest $G^{\text{Gr}}_{\text{Forest}}$ defined in Section 3.1.2 is a $(m, k_{\text{Gr}} \cdot (d-1)+1, e_{\text{sub}}, 1, d)$ witness forest.

Let E_{Reg} be the set of regular edges of $G^{\text{Gr}}_{\text{Forest}}$. Then the representation $\mathcal{R}_{\text{Gr}} = (\text{Label}, \mathit{server}, V_{\text{Forest}}, \mathit{job}, E_{\text{Reg}})$ using the mappings *server* and *job* defined in Section 3.1.2 is a complete explaining representation on $G^{\text{Gr}}_{\text{Forest}}$. If the time marker of the roots of $G^{\text{Gr}}_{\text{Forest}}$ is τ, i.e. if the construction started with the assumption that there are m servers with $\mathit{load}_\tau \geq T'$, then the natural vivid system $\mathcal{V}_{\text{Gr}} = \mathcal{V}_\tau$ is used by $\mathcal{R}_{\text{Gr}}$.

3.3 Consequences for the Example Algorithms

3.3.1 Consequences for the c-collision algorithm

Using Lemma 3.2.6 we bound the probability for bad performance of the c-collision algorithm. Consider the witness forest $G^{\text{c-coll}}_{\text{Forest}}$ defined in Section 3.1.1 for the c-collision algorithm. According to Remark 3.2.8 there is a complete explaining representation on the $(m, (c+1)(d-1), c \cdot \sum_{i=0}^{T-2} [c(d-1)]^i, c, d)$ witness forest $G^{\text{c-coll}}_{\text{Forest}}$ using a $(N, N, N^{-1}, M, d, 1)$-vivid system, if there are m servers active at the end of round T.

Corollary 3.3.1: Consider an allocation problem with $|\mathcal{S}| = N$, $|\mathcal{J}| = M$, number of copies d, and uniform i-th copy distributions. Let $c \in \mathbb{N}$, such that

$$b = (c+1)(d-1) \geq 3, \quad (\epsilon_c)^c = \frac{c!}{(\frac{M}{N} \cdot d)^c} > 1, \quad \text{and } \alpha \leq \frac{1}{12} \cdot b.$$

Let further $m \in \mathbb{N}$ and

$$T \geq \log_{c(d-1)} \log_{\epsilon_c} \left(\frac{N}{m} \right) + 2 + \mathrm{o}(1).$$

Then if N/m is large enough, there are less than m servers whose agents are active at the end of round T of the c-collision algorithm, with probability at least $1 - N^{-\alpha}$. In particular no server agent is active after

$$\log_{c(d-1)} \log_{\epsilon_c} (N) + 2 + \mathrm{o}(1)$$

rounds, with high probability, if N is large enough.

Proof: According to Remark 3.2.8 there is a representation on the witness forest $G^{\text{c-coll}}_{\text{Forest}}$ using a $(N, N, N^{-1}, M, d, 1)$-vivid system if there are m servers active at the end of round T of the c-collision algorithm. We can therefore use the Main Lemma (Lemma 3.2.6) to upper bound the probability for that event.

To provide the preconditions of the Main Lemma it suffices to establish that the forest $G^{\text{c-coll}}_{\text{Forest}}$ is a $(m, (c+1)(d-1), \bar{e}_{\text{sub}}, c, d)$ witness forest, i.e. to show that $c \cdot \sum_{i=0}^{T-2} [c(d-1)]^i \geq c \cdot [c(d-1)]^{T-2} \geq \bar{e}_{\text{sub}}$. According to Remark 3.2.7 this is ensured by our choice of T. And as

$$\frac{\ln(\frac{N}{m'} \cdot \mathrm{e}^3) \cdot m'}{\ln N} \geq 1,$$

we have

$$\alpha \leq \frac{1}{2} \cdot b \tag{3.3}$$

$$\leq \frac{1}{2} \cdot b \cdot \frac{\ln(\frac{N}{m'} \cdot \mathrm{e}^3) \cdot m'}{\ln N}. \tag{3.4}$$

Thus the preconditions of Lemma 3.2.6 are met. ■ of Corollary 3.3.1

We discuss this result in Section 5.2.2.

3.3.2 Consequences for the greedy algorithm

Deriving a result on the performance of the greedy algorithm is also simple. Fix some time step τ. According to Remark 3.2.8 there is a complete explaining representation using a $(N, N, N^{-1}, M, d, 1)$-vivid system on the $(m, k_{\text{Gr}}(d-1)+1, \sum_{i=0}^{T-3}[(d-1)+1]^i, 1, d)$ witness forest $G^{\text{Gr}}_{\text{Forest}}$ defined in Section 3.1.2, if m bins have load at least T at the beginning of round τ. The root of $G^{\text{Gr}}_{\text{Forest}}$ has $k_{\text{Gr}} \cdot (d-1) + 1$ children.

Corollary 3.3.2: Let $\overline{M} \in \mathbb{N}$. And let $d \geq 2$. Consider an infinite allocation problem with $|\mathcal{S}| = N$, number of copies d, and uniform i-th copy distributions. For the deletion scheme assume that job $J_{\tau-\overline{M}}$ is deleted right after job J_τ enters the system. Let

$$b = k_{\mathrm{Gr}} \cdot (d-1) + 1, \ \epsilon_c = \frac{1}{(\frac{\overline{M}}{N} \cdot d')} > 1, \text{ and } \alpha \leq \frac{1}{12} \cdot b.$$

Let further $m \in \mathbb{N}$ and let

$$T \geq \log_d \log_{\epsilon_c} \left(\frac{N}{m}\right) + k_{\mathrm{Gr}} + 2 + \mathrm{o}(1).$$

Then there are less than m servers with load at least T at the beginning of time step τ of the greedy algorithm, with probability at least $1 - N^{-\alpha}$, if N/m is large enough. In particular no server has load greater than

$$\log_d \log_{\epsilon_c} (N) + k_{\mathrm{Gr}} + 2 + \mathrm{o}(1)$$

rounds, with high probability, if N is large enough.

Proof: Similar to the proof above, we use $N' = N$, $p = N^{-1}$, $\mu = 1$, and check the preconditions of Lemma 3.2.6. As each inner node of $G^{\mathrm{Gr}}_{\mathrm{Forest}}$ has d child edges, it is easy to check that the choice of T ensures that each level 1 subtree of $G^{\mathrm{Gr}}_{\mathrm{Forest}}$ has e_{sub} inner nodes, each with one regular child edge (cf. to Remark 3.2.7). As shown above (in Equation 3.3), choosing $\alpha \leq \frac{1}{12} \cdot b$ suffices to fulfill the appropriate precondition of Lemma 3.2.6. ■ of Corollary 3.3.2

The result obtained in Corollary 3.3.2 is not the best possible result for the greedy algorithm. The condition that $\frac{1}{(\frac{\overline{M}}{N} \cdot d)^c} > 1$ does not allow to choose $\overline{M} \geq N$. We overcome this restriction in Section 5.1.1. Applying the witness tree argument to the greedy algorithm is quite simple and straightforward. We include the example above as it shows the necessity of the appendant graphs in the witness forest: without the appendant graphs and for $d = 2$, the level 1 subtrees of the witness forest for the greedy algorithm would degenerate to a linear array of size $T - k_{\mathrm{Gr}}$. This would lead to choosing $T = \Omega(\log(N/m))$.

3.4 The Proof of the Main Lemma

The aim of this Section is to show that there is no complete explaining representation using (N, N', p, M, d', μ)-vivid systems for a $(m, b, e_{\mathrm{sub}}, c, d)$ witness forest, with high probability, — for suitable values of $p, \mu, m, b, e_{\mathrm{sub}}, c, d$, and d'. Refining the formal definition of witness forests and representations given in the previous section, we do not need to refer to any particular algorithm or allocation problem.

The section is organized as follows. We start with refining the definition of a representation given in the previous section. In particular we define a *root-expansion-free valid* representation and show that each complete explaining representation can be converted to a root-expansion-free valid representation. The second part of this section which is formed by Lemma 3.4.4 contains the genuine combinatorial argument.

In the sequel we need an ordering $\preceq$ on the nodes and the edges of G_{Forest}. The ordering '$\preceq$' we define below arranges the nodes and the regular edges of the witness forest level by level from roots to leaves and from left to right. (We write $v \prec w$ if $v \preceq w$ and $v \neq w$.) Let $v \in V_{\text{Forest}}$ ($w \in V_{\text{Forest}}$) and let v' (w') be the parent of v (w) if it has a parent. We write $v \preceq w$, if either

- $\mathit{level}(v) < \mathit{level}(w)$,
- $\mathit{level}(v) = \mathit{level}(w) = 0$ and $v \leq w$,
- $\mathit{level}(v) = \mathit{level}(w) \geq 1$ and $v' \prec w'$,
- $\mathit{level}(v) = \mathit{level}(w) \geq 1$, $v' = w'$, and v is the appendant child of v'.
- $\mathit{level}(v) = \mathit{level}(w) \geq 1$, $v' = w'$, and the parent edge of v is lexicographically smaller than the parent edge of w, or
- $\mathit{level}(v) = \mathit{level}(w) \geq 1$, the parent edge $e = (v_1, \ldots, v_d)$ of v and w is the same, and $v = v_i$, $w = v_j$ for $1 \leq i < j \leq d$.

If $v \preceq w$ and $v \neq w$, then $v \prec w$. If $v \in V_{\text{Forest}}$ has children $v_1 \prec \cdots \prec v_j$, then the node v_1 is called *the leftmost child of* v.
For the regular edges we define a similar ordering. For $e, e' \in E_{\text{Reg}}$ let $e \preceq e'$ if

- $\mathit{top}(e) \prec \mathit{top}(e')$ or
- $\mathit{top}(e) = \mathit{top}(e')$ and $e \leq e'$.

If $e \preceq e'$ and $e \neq e'$ then let $e \prec e'$. The definition of this ordering insures that for two child edges e, e' of a node v it holds $e' \prec e$ if $e' < e$. Using this ordering we refine our terminology on representations.

Definition 3.4.1: A representation $\mathcal{R}$ explaining on some set $V \subseteq V_{\text{Forest}}$ (thus fulfilling Property 1 of Definition 3.2.4) is called *valid on* V, if there are sets $V_{\text{Exp}} \subseteq V_{\text{Server}}$ and $E_{\text{Exp}} \subseteq E_{\text{Job}}$ such that for each $v, v' \in V$ and each $e, e' \in \overline{E} = \{\bar{e} \in E_{\text{Forest}} \mid \bar{e} \subseteq V\}$ the following Properties 2, 3, and 4 are fulfilled. The nodes in V_{Exp} and the edges in E_{Exp} are called *expansion nodes* and *expansion edges*, respectively.

2. (a) If $v \in V_{\text{Server}}$ has a regular parent edge, then its parent edge is in $E_{\text{Job}} \setminus E_{\text{Exp}}$.

   (b) If $v \in V_{\text{Server}}$ has an appendant parent, then its appendant parent is in $V_{\text{Server}} \setminus V_{\text{Exp}}$.

   (c) If $e \in E_{\text{Job}}$, then the top node of e is in $V_{\text{Server}} \setminus V_{\text{Exp}}$.

   (d) If $v \in V_{\text{Server}} \setminus V_{\text{Exp}}$ has regular child edges, then all regular child edges of v are in E_{Job}.

   (e) If $v \in V_{\text{Server}} \setminus V_{\text{Exp}}$ has an appendant child, then the appendant child of v is in V_{Server}.

   (f) If $e \in E_{\text{Job}} \setminus E_{\text{Exp}}$, then all bottom nodes of e are in V_{Server}.

   (Property 2 deals with the properties of the sets V_{Server}, V_{Exp}, E_{Job}, and E_{Exp}. The sets V_{Exp} and E_{Exp} are the border of the sets V_{Server} and E_{Job}: all nodes and edges above the expansion nodes and edges are in V_{Server} and E_{Job}, and the nodes and edges below them are not in V_{Server} and E_{Job}.)

3. (a) If $v \in V_{\text{Server}} \setminus V_{\text{Exp}}$, $\bar{v}$ is the leftmost child of v, $w \in V_{\text{Server}}$ such that $w \prec \bar{v}$, $w \notin \mathit{irr}(v)$, then $\mathit{server}(v) \neq \mathit{server}(w)$.

   (b) If $e \in E_{\text{Job}} \setminus E_{\text{Exp}}$ and $e' \in E_{\text{Job}}$ such that $e' \prec e$, then $\mathit{job}(e) \neq \mathit{job}(e')$.

   (Property 3 clarifies the characteristic of the sets $V_{\text{Server}} \setminus V_{\text{Exp}}$ and $E_{\text{Job}} \setminus E_{\text{Exp}}$. No server is represented by two elements of $V_{\text{Server}} \setminus V_{\text{Exp}}$ and no job by two elements of $E_{\text{Job}} \setminus E_{\text{Exp}}$.)

4. (a) If $v \in V_{\text{Exp}}$, $\bar{v}$ is the leftmost child of v, then there is a $w \in V_{\text{Server}}$, $w \prec \bar{v}$, $w \notin \mathit{irr}(v)$ fulfilling $\mathit{server}(v) = \mathit{server}(w)$.

   (b) If $e \in E_{\text{Exp}}$, then there is an $\bar{e} \in E_{\text{Job}} \setminus E_{\text{Exp}}$, $\bar{e} \prec e$ such that $\mathit{job}(e) = \mathit{job}(\bar{e})$.

   (The elements of the expansion sets V_{Exp} and E_{Exp} have to show up the opposite property than the one required in Property 3: expansion nodes and edges represent something appearing before in the (weighted) witness forest.)

A representation is called *valid* if it is valid on V_{Forest}. It is called *root-expansion-free* if no root of the witness forest is an expansion node. This ends Definition 3.4.1.

Due to Property 2 of Definition 3.4.1 the sets V_{Exp} and E_{Exp} are unique for a given valid representation. In the sequel V_{Exp} and E_{Exp} denote the sets of expansion nodes and edges of the representation in consideration, if not stated otherwise.

Our next step is now to show that existence of a complete explaining representation implies the existence of a root-expansion-free valid representation. First we show how to convert a complete explaining representation to a valid representation. After that valid representations are made root-expansion-free.

Lemma 3.4.2: Let $\mathcal{R}' = (\text{Label}', server', V'_{\text{Server}}, job', E'_{\text{Job}})$ be a complete explaining representation using a (N, N', p, M, d', μ)-vivid system on a witness forest G_{Forest}. Then there is a valid representation using the same (N, N', p, M, d', μ)-vivid system on G_{Forest}.

Proof: To construct a valid representation on G_{Forest}, we define sets V_{Server}, V_{Exp}, E_{Job}, and E_{Exp} using the mappings $server'$ and job'. The new mappings will be a restriction of the old ones to these sets. Let $k = |V_{\text{Forest}}|$, let $v_1 \prec \cdots \prec v_k$ be the nodes of G_{Forest}, and for each $i = 1, \ldots, k$ let $\overline{v_i}$ be the leftmost child of v_i. (Recall further, that $irr(v_i)$ is the irregular component of v_i.)

Let $V_{\text{Server}}^{(v_1)} = V_{\text{Forest}}$ and $V_{\text{Exp}}^{(v_1)} = \emptyset$. For $2 \le i \le k$ let

$$V_{\text{Server}}^{(v_i)} = \begin{cases} V_{\text{Server}}^{(v_{i-1})} & \text{if } v_i \notin V_{\text{Server}}^{(v_{i-1})}, \text{ or} \\ & \text{if } server'(w) \neq server'(\overline{v_i}) \text{ for all } w \prec \overline{v_i},\ w \notin irr(v_i) \\ V_{\text{Server}}^{(v_{i-1})} \setminus \text{succ}(v_i) & \text{otherwise} \end{cases} \tag{3.5}$$

and

$$V_{\text{Exp}}^{(v_i)} = \begin{cases} V_{\text{Exp}}^{(v_{i-1})} & \text{if } v_i \notin V_{\text{Server}}^{(v_{i-1})}, \text{ or} \\ & \text{if } server'(w) \neq server'(\overline{v_i}) \text{ for all } w \prec \overline{v_i},\ w \notin irr(v_i)\ . \\ V_{\text{Exp}}^{(v_{i-1})} \cup \{v_i\} & \text{otherwise} \end{cases} \tag{3.6}$$

Finally, let $V_{\text{Server}} = V_{\text{Server}}^{(v_k)}$, $V_{\text{Exp}} = V_{\text{Exp}}^{(v_k)}$, and

$$\begin{aligned} E_{\text{Exp}} = \Big\{ e \in E_{\text{Reg}} \big|\ & top(e) \in V_{\text{Server}} \setminus V_{\text{Exp}} \wedge \exists\, e' \in E_{\text{Forest}} : \\ & \big(job'(e') = job'(e) \wedge e' \prec e\big) \Big\} \end{aligned} \tag{3.7}$$

$$E_{\text{Job}} = \big\{ e \in E_{\text{Reg}} \big|\ e \subseteq V_{\text{Server}} \big\} \cup E_{\text{Exp}}\ . \tag{3.8}$$

Let $server = server'\big|_{V_{\text{Server}}}$ and $job = job'\big|_{E_{\text{Job}}}$. We claim that $(\{label_e | e \in E_{\text{Job}}\}, server, V_{\text{Server}}, job, E_{\text{Job}})$ is a valid representation on G_{Forest}. To prove this claim, let $v \in V_{\text{Forest}}$ and $e \in E_{\text{Forest}}$.

Property 1

According to the precondition of the lemma.

Property 2 (a)

If $v \in V_{\text{Server}}$, then the parent v' of v is in V_{Server}, too. As one child of $v' \in V_{\text{Server}}$ is in V_{Server}, all children of v' are in V_{Server}, hence all nodes incident to a regular parent edge e of v are in V_{Server}, and $e \in E_{\text{Job}} \setminus E_{\text{Exp}}$.

Property 2 (b)
: If a node $v \in V_{\text{Server}}$ has an appendant parent v', then $v' \in V_{\text{Server}}$, according to Equation 3.5, and $v \notin V_{\text{Exp}}$, according to Equation 3.6.

Property 2 (c)
: If $e \in E_{\text{Job}} \setminus E_{\text{Exp}}$, then all incident nodes are in V_{Server} according to Equation 3.8. If $e \in E_{\text{Exp}}$, then $top(e) \in V_{\text{Server}}$ according to Equation 3.7.

Property 2 (d)
: If $v \in V_{\text{Server}} \setminus V_{\text{Exp}}$, then Equation 3.5 ensures that all children of v are in V_{Server}. Thus for each regular child edge e of v all incident nodes are in V_{Server}, hence $e \in E_{\text{Job}} \setminus E_{\text{Exp}}$, according to Equation 3.8.

Property 2 (e)
: If $v \in V_{\text{Server}} \setminus V_{\text{Exp}}$, then all children of v are in V_{Server}.

Property 2 (f)
: Confer to Equation 3.8.

Property 3 (a)
: If for a node $v \in V_{\text{Server}}$ with leftmost child $\bar{v}$ there were a node $w \prec \bar{v}$, $w \notin irr(v)$ fulfilling $server(v) = server(w)$, then $v \in V_{\text{Exp}}$, according to Equation 3.6.

Property 3 (b)
: If for an $e \in E_{\text{Job}}$ there were an $e' \prec e$ with $job(e') = job(e)$, then Equation 3.7 would ensure $e \in E_{\text{Exp}}$.

Property 4
: According to Equation 3.5 and Equation 3.6.

■ of Lemma 3.4.2

Our next step is to show that there is a root-expansion-free valid representation if there is a valid representation on a witness forest. We first observe that no two roots of a witness forest can represent the same server, according to Property 1 (d). Thus a root v of G_{Forest} can be an expansion node only if there is a node w preceding the leftmost child v' of v with $server(v) = server(w)$.

We call a node $v \in V_{\text{Sever}}$ of a valid representation on a witness forest a *blameless node* if for the leftmost child $\bar{v}$ of v there is a node $w \in V_{\text{Server}} \setminus irr(v)$, $v \prec w \prec \bar{v}$ with $server(w) = server(v)$ but no $w' \in V_{\text{Server}} \setminus irr(v)$ with $w' \prec v$ and $server(w') = server(v)$. The set of blameless nodes is denoted as $V_{\text{B-less}}$. If a node v is a blameless node and $\bar{v}$ is its leftmost child, then there is a node w with $v \prec w \prec \bar{v}$, $w \notin irr(v)$ and $server(w) = server(v)$. In this case node w is said to *blame* w. As the considered representation is valid, a blameless node is always an expansion node.

Lemma 3.4.3: Consider a $(m, b, e_{\text{sub}}, c, d)$ witness forest $G_{\text{Forest}} = (V_{\text{Forest}}, E_{\text{Forest}})$. Assume further that there is a complete explaining representation using a (N, N', p, M, d', μ)-vivid system on G_{Forest}. Then there is a $(\lceil \frac{m}{b+1} \rceil, e_{\text{sub}}, c, d)$ witness forest which is a subforest of G_{Forest} and has a root-expansion-free complete explaining representation using a (N, N', p, M, d', μ)-vivid system.

Proof: If a root of G_{Forest} is a blameless node, there are nodes blaming it. Let $G = (V, E)$ be a directed graph with m vertices associated to the m connected components of G_{Forest}. For each $x \in V$ let $\mathit{tree}(x)$ be the connected component associated to x. For any x the component $\mathit{tree}(x)$ is a witness tree. Consider two nodes $x, y \in V$, and let v be the root of $\mathit{tree}(x)$ and $\bar{v}$ its leftmost child. The nodes x, y of G are connected by an edge (x, y) if and only if $\mathit{tree}(y)$ contains a node w fulfilling

$$\begin{aligned} \mathit{level}(w) &\leq 1\,, \\ w &\prec \bar{v}\,, \text{ and} \\ \mathit{server}(w) &= \mathit{server}(v)\,. \end{aligned} \tag{3.9}$$

Thus a 2-tuple (x, y) is in E, if and only if the root v of $\mathit{tree}(x)$ is a blameless node and $\mathit{tree}(y)$ contains a node w blaming v. In this case x is said to *be connected to y due to w*. As a consequence of the construction the root of $\mathit{tree}(x)$ is a blameless node if and only if the out-degree of x is non-zero. Further G fulfills the following properties.

The in-degree of the nodes of G is bounded by b.

Consider a node $y \in V$. As the representation on G_{Forest} fulfills Property 1(c) of Definition 3.2.4, the roots of the witness trees represent different servers. Hence no $x \in V$ is connected to y due to $\mathit{root}(\mathit{tree}(y))$. If $x, x' \in V$, $x \neq x'$, x is connected to y due to w, and x' is connected to y due to w', then $w' \neq w$ as $\mathit{server}(\mathit{root}(\mathit{tree}(x))) \neq \mathit{server}(\mathit{root}(\mathit{tree}(y)))$. Since $\mathit{tree}(y)$ contains b nodes of level 1, the in-degree of y is at most b.

G is circle-free.

Let $(x, y) \in E$ and let x be connected to y due to w. As w is not the root of $\mathit{tree}(y)$, w is a node of level 1. If $\bar{v}$ is the leftmost child of the root of $\mathit{tree}(x)$, then $w \prec \bar{v}$, thus $\mathit{root}(\mathit{tree}(y)) \prec \mathit{root}(\mathit{tree}(x))$ and G has a topological order.

G is $b+1$ colorable.

Let $x_1, \ldots, x_m$ be the nodes of G in a topological order, thus if $(x_i, x_j) \in E$, then $i < j$. Let $\mathit{color}(x_m) = 0$. For each $1 \leq i \leq m-1$ let

$$\mathit{color}(x_i) = \min\Big(\{0, \ldots, b\} \setminus \{j \,|\, \exists y \in V : \mathit{color}(y) = j \wedge (y, x_i) \in E\}\Big)\,.$$

Note that $\big|\{j \in \{0, \ldots, b\} \,|\, \exists y \in V : \mathit{color}(y) = j \wedge (y, x_i) \in E\}\big| \leq b$ as the in-degree of G is bounded, thus the mapping $\mathit{color} : V \to \{1, \ldots, b+1\}$ is well-defined.

Consider a subset V' of V containing nodes of only one color, let $x, y \in V'$ and let v be the root of $tree(x)$. Then $tree(y)$ does not contain a node w fulfilling the properties of Equation 3.9. Thus, the representation on the $(|V'|, T, c, d)$ witness forest containing the witness trees associated to the nodes of V' does not contain a root which is an expansion node. At least one of the sets $\{x \in V \mid color(x) = 1\}, \ldots, \{x \in V \mid color(x) = b\}$ contains at least $\lceil \frac{m}{b+1} \rceil$ nodes of G. ■ of Lemma 3.4.3

As a consequence of Lemma 3.4.2 and Lemma 3.4.3 there is a root-expansion-free valid representation on a $(\lceil \frac{m}{b+1} \rceil, b, e_{\text{sub}}, c, d)$ witness forest, provided that there is a complete explaining representation on a $(m, b, e_{\text{sub}}, c, d)$ witness forest. To upper bound the probability for existence of a complete explaining representation it therefore suffices to upper bound the probability that there is a root-expansion-free valid representation. This is subject of Lemma 3.4.4.

The following statements and definitions are used in the proof of Lemma 3.4.4. An edge $e \in E_{\text{Job}} \setminus E_{\text{Exp}}$ is called a *conflict edge to an edge* e' (or simply a *conflict edge*) if $e \prec e'$ and $job(e) = job(e')$. The set of conflict edges is denoted by E_{Confl}. Let $e \in E_{\text{Confl}}$ be a conflict edge to $e' \in E_{\text{Exp}}$, and consider the top node $top(e')$ of e'. According to Property 1(a), there is a node v incident to e such that $server(v) \in server(top(e'))$. If the top nodes of e and e' were in different irregular components, $top(e')$ would be an expansion node as v precedes the leftmost child of $top(e')$. Thus the top nodes of e and e' are in the same irregular component. Thus Property 1(b) of Definition 3.2.4 ensures $label_e(v) \neq label_{e'}(v')$. Let w be the node incident to e' with $label_{e'}(w) = label_e(v)$. According to Property 1(a), $server(w) = server(top(e)) = server(top(e'))$, thus w is an expansion node. The node w is called the *apex-mirror node of* e or simply an *apex-mirror node*. The set of apex-mirror nodes is denoted by V_{Mirr}. Note, that each apex-mirror node is apex-mirror node of at most one expansion edge, thus $|V_{\text{Mirr}}| = |E_{\text{Exp}}|$.

Let $V_{\text{Reg}} \subseteq V_{\text{Forest}}$ denote the set nodes whose parent is not an appendant parent, i.e. the set of nodes being either a root of G_{Forest} or bottom node of a regular edge. The elements of V_{Reg} are called the *regular nodes* of G_{Forest}. A node is called *irregular* if it is not regular. Note that an irregular node v is an expansion node only if it is a blameless node, too. If there were a node $w \prec v$ with $server(w) = server(v)$, then the appendant parent v' of v would be an expansion node, as w precedes its leftmost child.

Lemma 3.4.4: Let G_{Forest} be a $(m', b, \bar{e}_{\text{sub}}, c, d)$ witness forest with $b \geq 3$. The probability of G_{Forest} having a root-expansion-free valid representation on a (N, N', p, M, d', μ)-vivid system is at most

$$\left(\frac{N}{m'}\right)^{m'} \cdot e^{3m} \cdot \epsilon_c^{-\bar{e}_{\text{sub}} \cdot m' \cdot b} \leq N^{-\alpha},$$

for

$$\Lambda = (N' \cdot \mu \cdot p), \quad (\epsilon_c)^c = \frac{c!}{(M \cdot d' \cdot p \cdot \Lambda^{d-1})^c} > 1, \quad \alpha \leq \frac{1}{12} \cdot b \cdot \frac{\ln(\frac{N}{m'} \cdot \mathrm{e}^3) \cdot m'}{\ln N},$$

and N/m' large enough (cf. Equations 3.20 and 3.21).

Proof: Assume that the size of $|E_{\text{Exp}}|$ and $|V_{\text{Exp}} \setminus (V_{\text{B-less}} \cup V_{\text{Mirr}})|$ is fixed. Then the expected number E of root-expansion-free valid representations for the $(m', b, \bar{e}_{\text{sub}}, c, d)$ witness forest is upper bounded by (using $R = root(G_{\text{Forest}})$)

$$
\begin{aligned}
\mathrm{E} \leq\ & \underbrace{N^{|R|} \cdot N'^{|[V_{\text{Server}} \setminus (V_{\text{Exp}} \cup R) \cup V_{\text{B-less}}] \cap V_{\text{Reg}}|}}_{(1)} \\
& \cdot \underbrace{\frac{1}{m'!}}_{(2)} \cdot \underbrace{(M \cdot d')^{|E_{\text{Job}} \setminus E_{\text{Exp}}|}}_{(3)} \cdot \underbrace{\left(\frac{1}{c!}\right)^{\frac{|E_{\text{Job}}|}{c}}}_{(4)} \\
& \cdot \underbrace{\binom{|E_{\text{Forest}}|}{|E_{\text{Exp}}|}}_{(5)} \cdot \underbrace{\left(d' \cdot |E_{\text{Job}}|\right)^{|E_{\text{Exp}}|}}_{(6)} \\
& \cdot \underbrace{\binom{|V_{\text{Forest}} \setminus V_{\text{Mirr}}|}{|V_{\text{Exp}} \setminus (V_{\text{B-less}} \cup V_{\text{Mirr}})|}}_{(7)} \cdot \underbrace{|V_{\text{Server}}|^{|V_{\text{Exp}} \setminus (V_{\text{B-less}} \cup V_{\text{Mirr}})|}}_{(8)} \\
& \cdot \underbrace{p^{-d \cdot |E_{\text{Job}} \setminus E_{\text{Exp}}|} \cdot \mu^{|[(V_{\text{Server}} \setminus V_{\text{Exp}}) \cup V_{\text{B-less}}] \cap V_{\text{Reg}}|}}_{(9)} \ .
\end{aligned}
\tag{3.10}
$$

(1) and (3)
: The representation for G_{Forest} is chosen level by level starting at the roots. We use the fact that the representation has to use a (N, N', p, M, d', μ)-vivid system. Further we assume that the expansion nodes and edges are fixed, see (5) and (7).

  There are N possibilities to choose a server for a root, and M possible jobs for each level 1 edge. If for an edge e, $job(e)$ and $server(top(e))$ are fixed, there are at most d' possibilities to choose $label_e$. For each node v with $level(v) \geq 1$, assume that $job(e)$ and $label_e$ are fixed for the top edge e of v. Then $server(v) \in vivids(J, label_e(v))$, thus there are N' possibilities to choose a server for v. The server for a child edge e' is chosen from $vividj(job(e), server(v))$ which allows M possibilities. We do not fix the choices for expansion edges, here. Edges in E_{Exp} are subject of (6). Similar, expansion nodes which are not blameless are considered in (8) if they are not apex-mirror nodes. Apex-mirror nodes are implicitly treated in (6), see there. Any non-regular node represents the same node as its parent node, thus there is no need to consider them.

(2)
: As servers represented by roots of G_{Forest} are distinct and appear sorted in a valid representation (Property 1(c) of Definition 3.2.4), superfluous permutations created by Term (1) are eliminated.

(4)
: According to Property 1(d) of Definition 3.2.4, the regular child edges of each node appear sorted. There are $|E_{\text{Job}}|/c$ nodes having child edges, for each of them Term (3) creates $c!$ superfluous permutations.

(5) Denotes the number of possibilities to choose expansion edges from E_{Forest}.

(6) Each expansion edge e represents a job appearing in an edge $e' \prec e$ (Property 4 (c) of Definition 3.4.1). There are at most $|E_{\text{Job}}|$ possibilities to choose e'. Further there are at most d possibilities to choose $label_e$. Choosing $label_e$ fixes also the apex-mirror node of e. The apex-mirror node represents the same server as the top node of its parent edge.

(7) Denotes the number of possibilities to choose expansion nodes which are neither blameless nodes nor apex-mirror nodes. Blameless nodes are considered in (8), apex-mirror nodes are considered in (6).

(8) Each expansion node represents a server appearing in a node of $(V_{\text{Server}} \setminus V_{\text{Exp}}) \cup V_{\text{B-less}} \subseteq V_{\text{Server}}$. (cf. Property 4 (a) and definition of $V_{\text{B-less}}$.) Choosing the servers represented by nodes in $V_{\text{Exp}} \setminus (V_{\text{B-less}} \cup V_{\text{Mirr}})$ fixes the positions of the blameless nodes, too. Apex-mirror nodes are considered in (6).

(9) The product of the terms (1) to (8) upper bounds the number of representations which use a particular (N, N', p, M, d', μ)-vivid system and fulfill all conditions of a root-expansion-free valid representation, but not necessarily fulfill Property 1(a) and the simple events defined by $vivide_{\mathcal{R}}$.

Consider an edge $e \in E_{\text{Job}} \setminus E_{\text{Exp}}$ and a node v incident to e. Let $\mathcal{E}$ be the event that Property 1(a) is fulfilled for each edge $e' \in E_{\text{Job}}$, $e' \prec e$ and each node $v' \in V_{\text{Server}}$, $v' \prec v$. Then $\mathcal{E}$ is a simple event, and as $e \notin E_{\text{Exp}}$ the event $\mathcal{E}$ is not about $(job(e), i)$, for any i. Thus according to Equation 3.2 from the definition of vivid systems

$$\text{Prob}\left[S^{(label_e(v))}(J) = server(v) \;\middle|\; J \in \mathcal{J}^{(e)}_{\text{Vivid}} \wedge S \in \mathcal{S}^{(v)}_{\text{Vivid}} \wedge \mathcal{E}\right] \leq p^{-1},$$

This holds for each of $d \cdot |E_{\text{Job}} \setminus E_{\text{Exp}}|$ pairs of edges in $E_{\text{Job}} \setminus E_{\text{Exp}}$ on one side and nodes incident to them on the other.

Using the representation $\mathcal{R}$ defined by Terms (1)-(8) we evaluate the probability that the vivid events defined by $vivide_{\mathcal{R}}$ are true. For nodes v, w in $[(V_{\text{Server}} \setminus V_{\text{Exp}}) \cup V_{\text{B-less}}]$ we have $server(v) \neq server(w)$. Thus the probability that $vivide_{\mathcal{R}}(v)$ is true for each $v \in [(V_{\text{Server}} \setminus V_{\text{Exp}}) \cup V_{\text{B-less}}] \cap V_{\text{Reg}}$ is at most

$$\mu^{|[(V_{\text{Server}} \setminus V_{\text{Exp}}) \cup V_{\text{B-less}}]|}.$$

The rough idea of the remaining part of the proof is to treat the terms of Equation 3.10 in three groups: one group dealing with the nodes in $V_{\text{Server}} \setminus V_{\text{Exp}}$, one group for the nodes in $V_{\text{Exp}} \setminus V_{\text{Mirr}}$ and one group for the edges in E_{Exp} (which are closely related to the nodes in V_{Mirr}). This rough idea cannot be used directly as some terms of Equation 3.10 are associated with several groups and a part of the second and the third

group has to be treated in the first one. We thus separate the proof into three parts: In the first part Terms (1), (3), (4), and (9) of Equation 3.10 are bounded. This part is finished by Equation 3.12 on Page 50 where the three groups are separated. After that the terms of Equation 3.12 are bounded separately, which leads to Equation 3.16 on Page 52. Here a term of the second and the third group is moved to the first one. After that the three groups are considered separately again until the desired proposition is set up on Page 55.

As each edge in $E_{\text{Job}} \setminus E_{\text{Exp}}$ is a regular edge containing $d-1$ regular bottom nodes and each node in $(V_{\text{Server}} \cap V_{\text{Reg}}) \setminus R$ is a bottom node of one edge in $E_{\text{Job}} \setminus E_{\text{Exp}}$,

$$|V_{\text{Server}} \setminus (V_{\text{Reg}} \cap R)| = (d-1) \cdot |E_{\text{Job}} \setminus E_{\text{Exp}}| \tag{3.11}$$

The set V_{Server} contains the set of expansion nodes V_{Exp} as a subset. As stated in the definitions of blameless and apex-mirror nodes, blameless and apex-mirror nodes are always expansion nodes. Further, no apex-mirror node is a blameless node, as the server represented by an apex-mirror node is also represented by the parent of the apex-mirror node. Thus

$$V_{\text{Server}} \supseteq V_{\text{Exp}}\,, \qquad V_{\text{B-less}} \cap V_{\text{Mirr}} = \emptyset\,,$$
$$V_{\text{Exp}} \supseteq V_{\text{B-less}}\,, \text{ and} \qquad V_{\text{Exp}} \supseteq V_{\text{Mirr}}\,.$$

Further

$$\begin{aligned}
&[(V_{\text{Server}} \setminus (V_{\text{Exp}} \cup R)) \cup V_{\text{B-less}}] \cap V_{\text{Reg}} \\
&\quad = \left[(V_{\text{Server}} \setminus R \cap V_{\text{Reg}}) \setminus (V_{\text{Exp}} \cap V_{\text{Reg}})\right] \cup (V_{\text{B-less}} \cap V_{\text{Reg}}) \\
&\quad = (V_{\text{Server}} \setminus R \cap V_{\text{Reg}}) \setminus \left[(V_{\text{Exp}} \setminus V_{\text{B-less}}) \cap V_{\text{Reg}}\right]
\end{aligned}$$

as each irregular expansion node is a blameless node this is

$$= (V_{\text{Server}} \setminus R \cap V_{\text{Reg}}) \setminus (V_{\text{Exp}} \setminus V_{\text{B-less}}).$$

As a consequence of this the product of Terms (1) and (9) of Equation 3.10 is

$$\begin{aligned}
&(N \cdot \mu)^{|R|} \cdot (N' \cdot \mu)^{|[(V_{\text{Server}} \setminus (V_{\text{Exp}} \cup R)) \cup V_{\text{B-less}}] \cap V_{\text{Reg}}|} \cdot p^{d \cdot |E_{\text{Job}} \setminus E_{\text{Exp}}|} \\
&\quad = (N \cdot \mu)^{|R|} \cdot (N' \cdot \mu)^{|V_{\text{Server}} \setminus R \cap V_{\text{Reg}}|} \cdot (N' \cdot \mu)^{-|V_{\text{Exp}} \setminus V_{\text{B-less}}|} \cdot p^{d \cdot |E_{\text{Job}} \setminus E_{\text{Exp}}|},
\end{aligned}$$

using Equation 3.11 and $\Lambda = (N' \cdot \mu \cdot p)$ this is

$$\begin{aligned}
&= (N \cdot \mu)^{m'} \cdot (p \cdot \Lambda^{d-1})^{|E_{\text{Job}} \setminus E_{\text{Exp}}|} \cdot (N' \cdot \mu)^{|V_{\text{Exp}} \setminus V_{\text{B-less}}|} \\
&= (N \cdot \mu)^{m'} \cdot (p \cdot \Lambda^{d-1})^{|E_{\text{Job}} \setminus E_{\text{Exp}}|} \cdot (N' \cdot \mu)^{|V_{\text{Exp}} \setminus (V_{\text{B-less}} \setminus V_{\text{Mirr}})|} \cdot (N' \cdot \mu)^{|V_{\text{Mirr}}|}.
\end{aligned}$$

To bound the product of Terms (3) and (4), substitute $c! = (\epsilon_c)^c \cdot (M \cdot d' \cdot \Lambda^{d-1})^c$

$$(M \cdot d')^{|E_{\text{Job}} \setminus E_{\text{Exp}}|} \cdot \left(\frac{1}{c!}\right)^{\frac{E_{\text{Job}}}{c}} \leq \left(\frac{1}{\epsilon_c \cdot p \cdot \Lambda^{d-1}}\right)^{|E_{\text{Job}} \setminus E_{\text{Exp}}|} \cdot \left(\frac{1}{c!}\right)^{\frac{E_{\text{Exp}}}{c}}$$

Thus the product of Terms (1), (9), (3), and (4) is upper bounded by

$$(N \cdot \mu)^{m'} \cdot \left(\frac{1}{\epsilon_c}\right)^{|E_{\text{Job}} \setminus E_{\text{Exp}}|} \cdot \left(\frac{1}{c!}\right)^{\frac{E_{\text{Exp}}}{c}} \cdot (N' \cdot \mu)^{-|V_{\text{Exp}} \setminus (V_{\text{B-less}} \cup V_{\text{Mirr}})|} \cdot (N' \cdot \mu)^{-|V_{\text{Mirr}}|}.$$

Using Equation 3.10 and $|V_{\text{Mirr}}| = |E_{\text{Exp}}|$, E is upper bounded by

$$\begin{aligned} \mathrm{E} \leq \underbrace{(N \cdot \mu)^{m'} \cdot \frac{1}{m'!}}_{(1)} \cdot \underbrace{\epsilon_c^{-|E_{\text{Job}} \setminus E_{\text{Exp}}|}}_{(2)} \\ \underbrace{\cdot \binom{|V_{\text{Forest}} \setminus V_{\text{Mirr}}|}{|V_{\text{Exp}} \setminus (V_{\text{B-less}} \cup V_{\text{Mirr}})|} \cdot |V_{\text{Server}}|^{|V_{\text{Exp}} \setminus (V_{\text{B-less}} \cup V_{\text{Mirr}})|} \cdot (N' \cdot \mu)^{-|V_{\text{Exp}} \setminus (V_{\text{B-less}} \cup V_{\text{Mirr}})|}}_{(3)} \\ \underbrace{\cdot \left(\frac{1}{c!}\right)^{\frac{|E_{\text{Exp}}|}{c}} \cdot \binom{|E_{\text{Forest}}|}{|E_{\text{Exp}}|} \cdot \left(d' \cdot |E_{\text{Job}}|\right)^{|E_{\text{Exp}}|} \cdot (N' \cdot \mu)^{-|E_{\text{Exp}}|}}_{(4)}. \end{aligned} \quad (3.12)$$

We bound the terms of Equation 3.12.

Term (1) It holds $1/m'! \leq (\mathrm{e}/m')^{m'}$, thus Term (1) is upper bounded by

$$(N \cdot \mu)^{m'} \cdot \frac{1}{m'!} \leq \left(\frac{(N \cdot \mu) \cdot \mathrm{e}}{m'}\right)^{m'}.$$

Term (2) If v is a level 1 node of G_{Forest}, its subtree $\text{sub}(v)$ is called *defect* if either

- $\text{sub}(v)$ contains an expansion node,
- $\text{sub}(v)$ contains an expansion edge, or
- the parent edge of v is an expansion edge.

An expansion edge affects at most one level 1 subtree by being contained in, and it affects at most $d-1$ level 1 subtrees by being the parent edge of their root. An expansion node affects at most one level 1 subtree. Hence the number of defect subtrees in G_{Forest} is upper bounded by

$$(d-1) \cdot |E_{\text{Exp}}| + |V_{\text{Exp}}| = d \cdot |E_{\text{Exp}}| + |V_{\text{Exp}} \setminus V_{\text{Mirr}}|.$$

The witness forest G_{Forest} contains at least $m' \cdot b$ level 1 nodes, thus at least

$$m' \cdot b - d \cdot |E_{\text{Exp}}| - |V_{\text{Exp}} \setminus V_{\text{Mirr}}|$$

level 1 subtrees are not defect. If a level 1 subtree is not defect, all its edges are in $E_{\text{Job}} \setminus E_{\text{Exp}}$, thus

$$|E_{\text{Job}} \setminus E_{\text{Exp}}| \geq \left[m' \cdot b - d \cdot |E_{\text{Exp}}| - |V_{\text{Exp}} \setminus V_{\text{Mirr}}| \right] \cdot \bar{e}_{\text{sub}} \,.$$

Hence

$$\begin{aligned} \epsilon_c^{-|E_{\text{Job}} \setminus E_{\text{Exp}}|} &\leq \big(\underbrace{\epsilon_c^{-\bar{e}_{\text{sub}}}}_{p_{\text{Tree}}} \big)^{m' \cdot b - d \cdot |E_{\text{Exp}}| - |V_{\text{Exp}} \setminus V_{\text{Mirr}}|} \\ &= p_{\text{Tree}}^{m' \cdot b - d \cdot |E_{\text{Exp}}| - |V_{\text{Exp}} \setminus V_{\text{Mirr}}|} . \end{aligned} \tag{3.13}$$

Term (3) Bounding the binomial coefficient in Term (3) we obtain

$$\left(\frac{|V_{\text{Forest}} \setminus V_{\text{Mirr}}| \cdot |V_{\text{Server}}| \cdot \mathrm{e}}{N' \cdot \mu \cdot |V_{\text{Exp}} \setminus (V_{\text{B-less}} \cup V_{\text{Mirr}})|} \right)^{|V_{\text{Exp}} \setminus (V_{\text{B-less}} \cup V_{\text{Mirr}})|} ,$$

as $|V_{\text{Server}}| \leq |V_{\text{Forest}}|$ and $|V_{\text{Forest}} \setminus V_{\text{Mirr}}| \leq |V_{\text{Forest}}|$ this is

$$\begin{aligned} &\leq \left(\frac{|V_{\text{Forest}}|^2 \cdot \mathrm{e}}{N' \cdot \mu \cdot |V_{\text{Exp}} \setminus (V_{\text{B-less}} \cup V_{\text{Mirr}})|} \right)^{|V_{\text{Exp}} \setminus (V_{\text{B-less}} \cup V_{\text{Mirr}})|} \\ &= \left(\frac{\mathrm{e} \cdot [m' \cdot |V_{\text{Tree}}|]^2}{N' \cdot \mu \cdot |V_{\text{Exp}} \setminus (V_{\text{B-less}} \cup V_{\text{Mirr}})|} \right)^{|V_{\text{Exp}} \setminus (V_{\text{B-less}} \cup V_{\text{Mirr}})|} \\ &\leq \left(\frac{\mathrm{e} \cdot m'}{|V_{\text{Exp}} \setminus (V_{\text{B-less}} \cup V_{\text{Mirr}})|} \right)^{|V_{\text{Exp}} \setminus (V_{\text{B-less}} \cup V_{\text{Mirr}})|} \\ &\qquad \cdot \left(\underbrace{\frac{m' \cdot |V_{\text{Tree}}|^2}{N' \cdot \mu \cdot |V_{\text{Exp}} \setminus (V_{\text{B-less}} \cup V_{\text{Mirr}})|}}_{p_{V_{\text{Exp}}}} \right)^{|V_{\text{Exp}} \setminus (V_{\text{B-less}} \cup V_{\text{Mirr}})|} , \end{aligned} \tag{3.14}$$

using Lemma 2.2.3

$$\leq \mathrm{e}^{m'} \cdot p_{V_{\text{Exp}}}^{|V_{\text{Exp}} \setminus (V_{\text{B-less}} \cup V_{\text{Mirr}})|} .$$

Term (4) We use Lemma 2.2.3 to get Term (4) upper bounded by

$$\left(\frac{\mathrm{e}}{c} \right)^{|E_{\text{Exp}}|} \cdot \left(\frac{|E_{\text{Forest}}| \cdot \mathrm{e}}{|E_{\text{Exp}}|} \right)^{|E_{\text{Exp}}|} \cdot [d' \cdot |E_{\text{Job}}|]^{|E_{\text{Exp}}|} \cdot (N' \cdot \mu)^{-|E_{\text{Exp}}|}$$

as $|E_{\text{Job}}| \leq |E_{\text{Forest}}|$

$$
\begin{aligned}
&= \left(\frac{\mathrm{e}^2 \cdot d' \cdot |E_{\text{Forest}}|^2}{c \cdot |E_{\text{Exp}}| \cdot N' \cdot \mu}\right)^{|E_{\text{Exp}}|} \\
&= \left(\frac{\mathrm{e}^2 \cdot d' \cdot m'^2 \cdot |E_{\text{Tree}}|^2}{c \cdot |E_{\text{Exp}}| \cdot N' \cdot \mu}\right)^{|E_{\text{Exp}}|} \\
&\leq \left(\frac{m' \cdot \mathrm{e}}{|E_{\text{Exp}}|}\right)^{|E_{\text{Exp}}|} \cdot \left(\underbrace{\frac{\mathrm{e} \cdot d' \cdot m' \cdot |E_{\text{Tree}}|^2}{c \cdot N' \cdot \mu}}_{p_{E_{\text{Exp}}}}\right)^{|E_{\text{Exp}}|},
\end{aligned}
\tag{3.15}
$$

using Lemma 2.2.3,

$$\leq \mathrm{e}^{m'} \cdot p_{E_{\text{Exp}}}^{|E_{\text{Exp}}|}.$$

Rewriting Equation 3.12 yields

$$
\begin{aligned}
\mathrm{E} \leq{}& \left(\frac{N \cdot \mu}{m'}\right)^{m'} \cdot \mathrm{e}^{3m} \cdot p_{\text{Tree}}^{m' \cdot b - d \cdot |E_{\text{Exp}}| - |V_{\text{Exp}} \setminus V_{\text{Mirr}}|} \\
&\cdot p_{V_{\text{Exp}}}^{|V_{\text{Exp}} \setminus (V_{\text{B-less}} \cup V_{\text{Mirr}})|} \\
&\cdot p_{E_{\text{Exp}}}^{|E_{\text{Exp}}|} \\
\leq{}& \underbrace{\left(\frac{N \cdot \mu}{m'}\right)^{m'} \cdot \mathrm{e}^{3m} \cdot p_{\text{Tree}}^{m' \cdot b}}_{(1)} \\
&\cdot \underbrace{p_{V_{\text{Exp}}}^{|V_{\text{Exp}} \setminus (V_{\text{B-less}} \cup V_{\text{Mirr}})|} \cdot p_{\text{Tree}}^{-|V_{\text{Exp}} \setminus V_{\text{Mirr}}|}}_{(2)} \\
&\cdot \underbrace{p_{E_{\text{Exp}}}^{|E_{\text{Exp}}|} \cdot p_{\text{Tree}}^{-d \cdot |E_{\text{Exp}}|}}_{(3)}.
\end{aligned}
\tag{3.16}
$$

Using Equation 3.13, $\mu \leq 1$, and the definition of $\bar{e}_{\text{sub}}$, Term (1) is obviously bounded by

$$\frac{1}{4} \cdot N^{-\alpha}.$$

In our next step we show that Term (2) and (3) are bounded by $(\frac{1}{2})^{|V_{\text{Exp}} \setminus (V_{\text{B-less}} \cup V_{\text{Mirr}})|}$ and $(\frac{1}{2})^{|E_{\text{Exp}}|}$, respectively. In both cases we need an upper bound on p_{Tree}^{-1} computed

next. According to Equation 3.13, the inverse of the probability p_{Tree} is

$$p_{\text{Tree}}^{-1} = (4 \cdot N^{\alpha})^{\frac{1}{m' \cdot b}} \cdot \left(\frac{N \cdot \mathrm{e}^3}{m'}\right)^{\frac{1}{b}},$$

as $b \geq 3$

$$\leq 4^{\frac{1}{b}} \cdot \left(\frac{N}{m'} \cdot \mathrm{e}^3\right)^{\frac{\ln N}{m' \cdot \ln(\frac{N}{m'} \cdot \mathrm{e}^3)} \cdot \frac{\alpha}{b}} \cdot \left(\frac{N}{m'} \cdot \mathrm{e}^3\right)^{\frac{1}{b}}.$$

As $\alpha \leq \frac{1}{12} \cdot b \cdot \frac{m' \cdot \ln(\frac{N}{m'} \cdot \mathrm{e}^3)}{\ln N}$, and $b \geq 3$

$$p_{\text{Tree}}^{-1} \leq 2 \cdot \mathrm{e}^2 \cdot \left(\frac{N}{m'}\right)^{\frac{5}{12}}. \tag{3.17}$$

Term (2) of Equation 3.16

We claim that

$$p_{V_{\text{Exp}}}^{|V_{\text{Exp}} \setminus (V_{\text{B-less}} \cup V_{\text{Mirr}})|} \cdot p_{\text{Tree}}^{-|V_{\text{Exp}} \setminus V_{\text{Mirr}}|} \leq \frac{1}{2}^{|V_{\text{Exp}} \setminus (V_{\text{B-less}} \cup V_{\text{Mirr}})|}.$$

For each blameless node v there has to be at least one node v' blaming v. As the node v' represents the same server as v and $v \prec v'$, the node v' is not a blameless node itself. Moreover v' is not blaming any other node $w \neq v$, and v' is not an apex-mirror node. Thus at most half the expansion nodes not being an apex-mirror node are blameless nodes, hence

$$|V_{\text{Exp}} \setminus (V_{\text{B-less}} \cup V_{\text{Mirr}})| \geq \frac{1}{2} \cdot |V_{\text{Exp}} \setminus V_{\text{Mirr}}|,$$

hence

$$2 \cdot |V_{\text{Exp}} \setminus (V_{\text{B-less}} \cup V_{\text{Mirr}})| \geq |V_{\text{Exp}} \setminus V_{\text{Mirr}}|.$$

As $p_{\text{Tree}}^{-1} \geq 1$

$$p_{V_{\text{Exp}}}^{|V_{\text{Exp}} \setminus (V_{\text{B-less}} \cup V_{\text{Mirr}})|} \cdot p_{\text{Tree}}^{-|V_{\text{Exp}} \setminus V_{\text{Mirr}}|} \leq p_{V_{\text{Exp}}}^{|V_{\text{Exp}} \setminus (V_{\text{B-less}} \cup V_{\text{Mirr}})|} \cdot p_{\text{Tree}}^{-2 \cdot |V_{\text{Exp}} \setminus (V_{\text{B-less}} \cup V_{\text{Mirr}})|}.$$

Thus to prove our claim it suffices to establish

$$\frac{m' \cdot |V_{\text{Tree}}|^2}{\frac{1}{2} \cdot N' \cdot \mu \cdot |V_{\text{Exp}} \setminus V_{\text{Mirr}}|} \cdot p_{\text{Tree}}^{-2} \leq \frac{1}{2}. \tag{3.18}$$

Putting

$$\beta = \frac{|V_{\text{Tree}}|}{\bar{e}_{\text{sub}}} \text{ and } \rho = \frac{N'}{N} \tag{3.19}$$

we get

$$\frac{m' \cdot |V_{\text{Tree}}|^2}{\frac{1}{2} \cdot N' \cdot \mu \cdot |V_{\text{Exp}} \setminus V_{\text{Mirr}}|} \cdot p_{\text{Tree}}^{-2} \leq \frac{\beta^2 \cdot \log_{\epsilon_c}^2(p_{\text{Tree}}^{-1})}{\frac{1}{2} \cdot \mu \cdot \rho \cdot |V_{\text{Exp}} \setminus V_{\text{Mirr}}|} \cdot p_{\text{Tree}}^{-2} \cdot \frac{m'}{N},$$

using the bound on p_{Tree}^{-1} from Equation 3.17 this is upper bounded by

$$\left(4 \cdot \mu^{-1} \cdot \mathrm{e}^4 \cdot \beta^2 \cdot \rho^{-1} \cdot \log_{\epsilon_c}^2(p_{\text{Tree}}^{-1})\right) \cdot \left(\frac{N}{m'}\right)^{\frac{10}{12}-1} \leq \frac{1}{2}, \tag{3.20}$$

if $\frac{N}{m'}$ large enough.

Term (3) of Equation 3.16

We bound $|E_{\text{Tree}}|$ using the definition for β from Equation 3.19.

$$|V_{\text{Tree}}| \leq \beta \cdot \log_{\epsilon_c}(p_{\text{Tree}}^{-1}).$$

As each edge contains d nodes, we get

$$|E_{\text{Tree}}| \leq d \cdot \beta \cdot \log_{\epsilon_c}(p_{\text{Tree}}^{-1}).$$

Thus using the definition of $p_{E_{\text{Exp}}}$ from Equation 3.15,

$$p_{E_{\text{Exp}}} \cdot p_{\text{Tree}}^{-1} \leq \left(\frac{\mathrm{e} \cdot d' \cdot m' \cdot d^2 \cdot \beta^2 \cdot \log_{\epsilon_c}^2(p_{\text{Tree}}^{-1})}{c \cdot N' \cdot \mu}\right) \cdot p_{\text{Tree}}^{-1},$$

and the bound on p_{Tree} from Equation 3.17,

$$\begin{aligned} &\leq 2 \cdot \mu^{-1} \cdot \mathrm{e}^3 \cdot d' \cdot \frac{d^2}{c} \cdot \beta^2 \cdot \rho^{-1} \log_{\epsilon_c}^2(p_{\text{Tree}}^{-1}) \cdot \frac{m'}{N} \cdot \left(\frac{N}{m'}\right)^{\frac{5}{12}} \\ &\leq 2 \cdot \mu^{-1} \cdot \mathrm{e}^3 \cdot d' \cdot \frac{d^3}{c} \cdot \beta^2 \cdot \rho^{-1} \cdot \log_{\epsilon_c}^2(p_{\text{Tree}}^{-1}) \cdot \left(\frac{m'}{N}\right)^{\frac{7}{12}} \\ &\leq \frac{1}{2}, \end{aligned} \tag{3.21}$$

for $\frac{N}{m'}$ large enough (using β and ρ from Equation 3.19).

We may now write Equation 3.16 as

$$\mathrm{E} \leq \left(\frac{N}{m'}\right)^{m'} \cdot e^{3m} \cdot \epsilon_c^{-\bar{e}_{\mathrm{sub}} \cdot m' \cdot b} \cdot \left(\frac{1}{2}\right)^{|V_{\mathrm{Exp}} \setminus (V_{\text{B-less}} \cup V_{\mathrm{Mirr}})|} \cdot \left(\frac{1}{2}\right)^{|E_{\mathrm{Exp}}|}.$$

Recall that the E denotes the expected number of root-expansion-free valid representations with fixed size of $|V_{\mathrm{Exp}} \setminus (V_{\text{B-less}} \cup V_{\mathrm{Mirr}})|$, and $|E_{\mathrm{Exp}}|$. To bound the expected number of root-expansion-free valid representations $\widetilde{\mathrm{E}}$, we take the sum over all possible values of $|V_{\mathrm{Exp}} \setminus (V_{\text{B-less}} \cup V_{\mathrm{Mirr}})|$ and E_{Exp}.

$$\begin{aligned} \widetilde{\mathrm{E}} &\leq \sum_{i=0}^{\infty} \sum_{j=0}^{\infty} \left(\frac{N}{m'}\right)^{m'} \cdot e^{3m} \cdot \epsilon_c^{-\bar{e}_{\mathrm{sub}} \cdot m' \cdot b} \cdot \left(\frac{1}{2}\right)^{i} \cdot \left(\frac{1}{2}\right)^{j} \qquad (3.22) \\ &= 4 \cdot \left(\frac{N}{m'}\right)^{m'} \cdot e^{3m} \cdot \epsilon_c^{-\bar{e}_{\mathrm{sub}} \cdot m' \cdot b} \end{aligned}$$

using Equation 3.13 and the definition of $\bar{e}_{\mathrm{sub}}$

$$\leq N^{-\alpha}.$$

■ of Lemma 3.4.4

This allows to prove the Main Lemma (Lemma 3.2.6).

Proof: If there is a complete explaining representation using a (N, N', p, M, d', μ)-vivid system on G_{Forest}, Lemma 3.4.2 ensures that there is a valid representation on G_{Forest}. Then Lemma 3.4.3 provides a root-expansion-free representation using a (N, N', p, M, d', μ)-vivid system on a $(m', b, \bar{e}_{\mathrm{sub}}, c, d)$ witness forest. Applying Lemma 3.4.4 yields the desired result. ■ of Lemma 3.2.6

4 The Argument for Weighted Jobs

The aim of this chapter is to obtain results for weighted allocation problems. Our results are similar to the results for ordinary allocation problems in the previous chapter. Again we seek for a general proof which is able to deal with various allocation problems. As we will see, weighted allocation problems are closely related to ordinary ones. The close relation is the basis of the analysis given in the sequel. The rough concept is to relate representations for weighted allocation problems to representations for ordinary allocation problems. After that we make use of the Main Lemma (Lemma 3.2.6).

As done in Chapter 3 we use an example algorithm to illustrate our technique. As an example algorithm we introduce a weighted version of the c-collision algorithm the so-called *load collision algorithm*. For the load collision algorithm we show how to define a representation on a witness forest if the performance of the algorithm is bad. After that we state the Main Lemma for Weighted Jobs and show how to apply it to the example algorithm. The proof of the Main Lemma for Weighted Jobs is contained in the last section of the chapter.

4.1 The load collision algorithm

The load collision algorithm or, more precisely, the c_w-*load collision algorithm* is shown in Figure 4.1. The main difference between the c-collision algorithm and the load collision algorithm occurs in step (2), where the load collision algorithm deals with the sum of the jobs weights while the c-collision algorithm deals with the number of jobs. In fact the c_{w}-load collision algorithm is just a generalization of the c-collision algorithm: if $c_{\mathrm{w}} = c$ then the behavior of the c-collision algorithm and the c_{w}-load collision algorithm is the same, if they both deal with the same ordinary allocation problem and the same random choices for the servers.

As we do for the c-collision algorithm, we call each run through the while loop (1) a *round* of the algorithm. If in Step (3) a job agent receives acknowledgments from more than one server, we do not care which server is actually chosen.

4.2 A Representation for the Load Collision Algorithm

As stated above, the main difference between the c-collision algorithm and the c_{w}-load collision algorithm is that the latter deals with a sum of weights in step (2) while the c-collision algorithm uses the number of requests. This difference is also the main

For all $J \in \mathcal{J}$ do in parallel
  agent(J) becomes *active*.

While there is at least one *active* job agent do (1)
  For all $J \in \mathcal{J}$ do in parallel
    For $j \in \{1, 2, \ldots, d\}$ do
      If agent(J) is *active*,
        agent(J) sends a $(J, j, W(J))$-*request* to agent($S^{(j)}(J)$).

  For all $S \in \mathcal{S}$ do in parallel
    If the sum of the weights of all jobs sending a request to S (2)
      does not exceed c_w,
        agent(S) sends a (S)-*acknowledgment* to each job agent which
          sent a request to S.
        (In this case agent(S) is said to *accept its requests*.)

  For all $J \in \mathcal{J}$ do in parallel
    If for some $S \in \mathcal{S}$ the agent(J) receives a (S)-*acknowledgment*, (3)
      agent(J) becomes *inactive*,
      agent(J) allocates J to server S

Fig. 4.1: The c_w-load collision algorithm

difference in the construction of the representation. In the representation for the c-collision algorithm, each inner node v represents a server getting $c+1$ requests from the $c+1$ jobs represented by the $c+1$ edges incident to v. In the representation for the c_w-load collision algorithm each inner node v represents a server receiving requests with weight larger than c_w from the jobs represented by the edges incident to v. As the weights of the jobs may differ, the number of jobs sending a request to *server*(v) differs, too. For this reason the degree of the inner nodes will differ in the witness forest for the c_w-load collision algorithm. It is, however, possible to give a lower bound on the degree of the inner nodes. As the weight of a job is at most 1, at least $c_w + 1$ requests are required to obtain weight more than c_w.

Let $G_{\text{Tree}} = (V_{\text{Tree}}, E_{\text{Tree}})$ be a tree specified in the sequel. Fix the random possible servers of the jobs and assume that there is a server S whose agent is active at the end of some round T of the c_w-load collision algorithm. We define mappings *server*, *job*, and $label_e$ for each edge $e \in E_{\text{Tree}}$, as in Section 3.1.1.

Assume that at the end of round T of the c_w-load collision algorithm there is a server S, whose server agent is still active. Let the root of G_{Tree} represent server S. As S is active at the end of round T, it receives some requests $(J_1, j_1) < \cdots < (J_k, j_k)$ in round T. Let the root of G_{Tree} have k child edges $e_1 < \cdots < e_k$ and let edge $job(e_i) = J_i$ and $label_{e_i} = label_{e_i}^{(j_i)}$ for $1 \le i \le k$.

Consider a child edge $e = (v_1, \ldots, v_d)$ of $root(G_{\text{Tree}})$ and let $server(v_i) = S^{(label_e(v_i))}(job(e))$, for $2 \le i \le d$. The job agent of $job(e)$ sends a request to S in

round T. Thus $job(e)$ is active at the end of round $T-1$ and sends a request to the servers represented by the bottom nodes of e in round $T-1$ and this request is not accepted. Hence $server(v_2), \ldots, server(v_d)$ are active at the end of round $T-1$.

Now let v' be a bottom node of e. As $job(v')$ is active at the end of round $T-1$ it receives some requests $(J'_1, j'_1) < \cdots < (J'_{k'}, j'_{k'})$ besides the request $(job(e), label_e(v'))$. Let v' have k' child edges $e'_1 < \cdots < e'_{k'}$, and for $1 \leq i \leq k'$ let $job(e'_i) = J'_i$ and $label_{e'_i} = label_{e'_i}^{(j'_i)}$. For each bottom node v'' of e'_i let $server(v'') = S^{(label_{e'_i}(v''))}(job(e'_i))$. This server is active at the end of round $T-2$. It is possible to continue the construction recursively to the level T nodes of G_{Tree} which represent servers active at the end of round 0. The tree G_{Tree} defined in the recursion is a d-uniform tree whose inner nodes have at least c_{w} child edges. All leaves of G_{Tree} are level T nodes.

If m servers are active at the end of round T it is possible to define a representation $\mathcal{R}_{\text{Load}}$ on a forest $G^{\text{Load}}_{\text{Forest}}$ of m d-uniform trees whose inner nodes have at least c_{w} child edges and whose leaves are level T nodes. The forest $G^{\text{Load}}_{\text{Forest}}$ is a witness forest. It also fulfills a stronger condition defined next.

4.3 The Main Lemma for Weighted Jobs

Definition 4.3.1: Let $m, e_{\text{sub}}, c, d \in \mathbb{N}$. Let $b = c(d-1)$. Let $G_{\text{wForest}} = (V_{\text{wForest}}, E_{\text{wForest}})$ be a d-uniform forest with m connected components. Then G_{wForest} is called a *weighted* (m, b, e_{sub}, c, d) *witness forest* if for each set $E \subseteq E_{\text{wForest}}$ it holds

> Let $G_{\text{Forest}} = (V_{\text{Forest}}, E_{\text{Forest}})$ be the subgraph of G_{wForest} induced by E. If each node $v \in G_{\text{wForest}}$ has exactly c child edges which are in E, then G_{Forest} is a $(m, b, e_{\text{sub}}, c, d)$ witness forest, called the *witness forest induced by E in the weighted witness forest $G_{wForest}$*.

A weighted $(1, b, e_{\text{sub}}, c, d)$ witness forest is called a *weighted* $(b, e_{\text{sub}}, c, d)$ *witness tree.*

The definition for weighted witness forests may seem a little bit strange, but it fills exactly our needs. In the sequel we choose a witness forest as a subforest of a weighted witness forest, and the above definition allows this to be done in a greedy fashion. An example for a weighted $(c(d-1), e_{\text{sub}}, c, d)$ witness tree is a complete c-ary, d-uniform tree of height T (using $e_{\text{sub}} = c \cdot \sum_{i=0}^{T-2}[c(d-1)]^i$). For each $c' \geq c$ a complete c'-ary, d-uniform tree of height T is a weighted $(c, e_{\text{sub}}, c, d)$ witness tree. An example for a tree which is not a weighted $(c, e_{\text{sub}}, c, d)$ witness tree is a complete c-ary, d-uniform tree of height T where the root has one extra child edge whose bottom nodes are leaves. (If the extra edge is in E but another child edge of the root is not, the induced graph is not connected.)

An explaining representation for a witness forest assures that $c+1$ jobs are represented by the edges incident to an inner node. In an ordinary allocation problem these jobs have weight $c+1$. We need a similar property for weighted allocation problems and weighted witness forests.

Definition 4.3.2: Let $c_{\mathrm{w}} \in \mathbb{N}$. Consider a weighted witness forest G_{wForest} and a representation $\mathcal{R}_{\mathrm{w}}$. Then $\mathcal{R}_{\mathrm{w}}$ is called a *c_{w}-fat representation* or *fat representation* for short, if for each inner node v of G_{Forest}

> If v has i incident edges and $J_1, \ldots, J_i$ are the jobs represented by these edges, then
>
> $$\sum_{j=1}^{i} W(J_i) \geq c_w.$$

A set $\mathcal{J}' \subseteq \mathcal{J}$ of jobs has *weight* M, if

$$\sum_{J \in \mathcal{J}'} W(J) = M.$$

A vivid system has weight M if each vivid set of an edge has weight M.

The representation $\mathcal{R}_{\mathrm{Load}}$ defined in Section 4.2 is a c_{w}-fat representation. The corresponding vivid system has weight M as $\mathcal{J}$ has weight M.

Lemma 4.3.3 (Main Lemma for Weighted Jobs): Let $m', c, c_{\mathrm{w}}, d, e_{\mathrm{sub}} \leq N$, $\Lambda = (N \cdot \mu \cdot p)$, $m' = \left\lceil \frac{m}{b+1} \right\rceil$. Let

$$b \geq 3,\ c_{\mathrm{w}} \geq 5 \cdot c + 1, \quad \text{and } c_{\mathrm{w}} \geq -\frac{25}{8} \cdot \ln(1 - \epsilon_c^{-\frac{1}{11}}) + 1.$$

Further let

$$(\epsilon_c)^c = \frac{c!}{((1+\sqrt{3}) \cdot M \cdot d' \cdot p \cdot \Lambda^{d-1})^c} > 1,$$

$$e_{\mathrm{sub}} = \log_{\epsilon_c} \left(4^{\frac{1}{m' \cdot b}} \cdot N^{\frac{\alpha}{m' \cdot b}} \cdot \left(\frac{N}{m'} \cdot \mathrm{e}^3 \right)^{\frac{1}{b}} \right), \text{ such that } e_{\mathrm{sub}} \geq b + 1,$$

and let $G_{\mathrm{wForest}} = (V_{\mathrm{wForest}}, E_{\mathrm{wForest}})$ be a weighted $(m, b, e_{\mathrm{sub}}, c, d)$ witness forest. Then the probability that there is a c_{w}-fat complete explaining representation using a $(N, N', p, \infty, d', \mu)$-vivid system of weight M on G_{wForest} is at most

$$N^{-\frac{1}{132} \cdot b},$$

if N/m' (cf. to Lemma 3.2.6) and M (cf. to Equation 4.1) are large enough.

4.4 Consequences for the Load Collision Algorithm

Suppose that there are m server agents active at the end of round T of the c_{w}-load collision algorithm. Then there is a complete explaining representation on the weighted $(m, b, e_{\mathrm{sub}}, c, d)$ witness forest $G^{\mathrm{Load}}_{\mathrm{Forest}}$. The representation is also c_{w}-fat. Let each vivid set of an edge be $\mathcal{J}$ and each vivid set of a node be $\mathcal{S}$. Then for $N = |\mathcal{S}|$ and $M = \sum_{J\in\mathcal{J}} W(J)$ the representation uses a $(N, N', N^{-1}, \infty, d, \mu)$-vivid system of weight M.

Corollary 4.4.1: Consider an allocation problem with $|\mathcal{S}| = N$, $\sum_{J\in\mathcal{J}} W(J) = M$, number of copies d, and uniform i-th copy distributions. Let $c \in \mathbb{N}$, such that

$$b = c(d-1) \geq 3$$

and

$$(\epsilon_c)^c = \frac{c!}{((1+\sqrt{3})\cdot\frac{M}{N}\cdot d)^c} > 1.$$

Then for each $m \leq N$, $c \leq N$,

$$c_{\mathrm{w}} \geq 5\cdot c + 1, \text{ such that } N \geq c_{\mathrm{w}} \geq -\frac{25}{8}\cdot\ln(1-\epsilon_c^{-\frac{1}{11}}) + 1,$$

and

$$T \geq \log_{c(d-1)}\log_{\epsilon_c}\left(\frac{N}{m}\right) + \log_{c(d-1)}\left(\frac{1}{b}\cdot(\alpha+1)\right) + 2 + \omega(1)$$

there are less than m servers whose agents are active at the end of round T of the c_{w}-load collision algorithm, with high probability, if N/m is large enough. In particular no server agent is active after

$$\log_{c(d-1)}\log_{\epsilon_c}(N) + 2 + \omega(1)$$

rounds, with high probability, if N is large enough.

Proof: It is easily checked that the preconditions of Lemma 4.3.3 are met. ■

We discuss this result in Section 5.2.2.

4.5 The Proof for the Weighted Case

In order to prove the Main Lemma for Weighted Jobs (Lemma 4.3.3) it may seem tempting, to apply Lemma 3.2.6 in order to get a bound on the performance of weighted allocation problems. But the crucial thing about weighted allocation problems is that

the proximate vivid system leads to a poor result in the Main Lemma. For a weighted allocation problem the vivid set of the edges can be very large compared to the weights of the jobs. For instance consider an allocation problem with set of servers $\mathcal{S}$, $|\mathcal{S}| = N$, set of jobs $\mathcal{J}$, $|\mathcal{J}| = 2^N + 1$, number of copies d, and uniform i-th copy distributions. Assume further that $W(J_{\max}) = 1$ and $W(J) = \frac{N-1}{2^N}$ for each $J \in \mathcal{J} \setminus \{J_{\max}\}$, thus $\sum_{J\in\mathcal{J}} W(J) = N$. Such an allocation problem would give raise to a representation using a $(N, N, N^{-1}, 2^N+1, d, 1)$-vivid system. For such a vivid system the Main Lemma cannot guarantee more than a $O(2^N/N)$ upper bound on the load, while in fact the load can be shown to be constant. To overcome this problem we relate a weighted allocation problem with $\sum_{J\in\mathcal{J}} W(J) = M$ to an allocation problem with $O(M)$ jobs. The relation is established using the vivid system.

To relate a weighted allocation problem to an ordinary one, we deploy a process called *the flipping*. Consider a weighted witness forest G_{wForest}, a representation $\mathcal{R}_{\text{w}}$ on it, and a vivid system $\mathcal{V}_{\text{w}} = (\mathit{vividrj}_{\text{w}}, \mathit{vivids}, \mathit{vividj}_{\text{w}}, \mathit{vivide}_{\mathcal{R}})$ used by $\mathcal{R}_{\text{w}}$. We assume that the mapping *vividj* takes a third parameter from V_{Forest}. Let $v \in V_{\text{wForest}}$, $J \in \mathcal{J}$, $i \in \{1, \ldots, d\}$, and $S \in \mathit{vivids}(J, i)$. Then for each $J' \in \mathit{vividj}_{\text{w}}(J, S, v)$ we flip a coin: with probability $W(J')$ the job J' *survives* the flipping and with probability $1 - W(J)$ it *vanishes*. Let the set $\mathit{vividj}(J, S, v)$ contain all jobs from $\mathit{vividj}_{\text{w}}(J, S, v)$ which survive the flipping. Similar flip a coin for each $S \in \mathcal{S}$, $v \in \mathit{root}(G_{\text{wForest}})$, $J' \in \mathit{vividrj}_{\text{w}}(S, v)$ and let $\mathit{vividrj}(S, v)$ contain the surviving jobs from $\mathit{vividrj}_{\text{w}}(S, v)$. We call $\mathcal{V} = (\mathit{vividrj}, \mathit{vivids}, \mathit{vividj}, \mathit{vivide}_{\mathcal{R}})$ the *new vivid system*. $\mathcal{V}_{\text{w}}$ is called the *old vivid system*.

Our next step is to establish two facts about the new vivid system. At first we would like to show that the new vivid system is a $(N, N', p, O(M), d', \mu)$-vivid system given that the old one is a $(N, N', p, \infty, d', \mu)$-vivid system of weight M. Next we would like to show that the representation $\mathcal{R}_{\text{w}}$ uses the new vivid system. Neither of these statements is really true, but in each case we are able to show a sufficiently strong similar statement.

Let $m' = \lceil \frac{m}{b+1} \rceil$. Assume that there is a c_{w}-fat complete explaining representation $\mathcal{R}_{\text{w}} = (\mathit{vividrj}, \mathit{vivids}, \mathit{vividj}, \mathit{vivide}_{\mathcal{R}})$ on a weighted $(m, b, e_{\text{sub}}, c, d)$ witness forest $G_{\text{wForest}} = (V_{\text{wForest}}, E_{\text{wForest}})$ using the old vivid system. Assume that the vivid sets of the edges have weight at most M, each. Consider an edge e representing job $J = \mathit{job}(e)$. With probability $W(J)$ the job J survives the flipping, in this case we say that *edge e survives the flipping*. A *node v does c-survive the flipping* if at least c child edges of v survive the flipping.

Constructing a complete explaining representation Assume that each inner node of G_{wForest} does c-survive the flipping. Then there is a set $E \subseteq E_{\text{wForest}}$ containing *exactly* c surviving child edges for each inner node. Let G'_{Forest} be the $(m, b, e_{\text{sub}}, c, d)$ witness forest induced by E from G'_{wForest}. Then the restriction $\mathcal{R}'$ of $\mathcal{R}_{\text{w}}$ to the nodes and edges of G'_{Forest} is a complete explaining representation on G'_{Forest}. According to Lemma 3.4.3

there is a $(m, b, e_{\text{sub}}, c, d)$ witness forest G_{Forest} being a subforest of G'_{Forest} such that the restriction $\mathcal{R}$ of $\mathcal{R}'$ to G_{Forest} is root-expansion-free. As the jobs represented by the edges of G_{wForest} survive the flipping, $\mathcal{R}$ uses the new vivid system.

The assumption that each inner node of G_{wForest} does c-survive the flipping is actually stronger than necessary. Considering G_{Forest} as a subgraph of G_{wForest}, it suffices if only the inner nodes of G_{Forest} and the roots of G_{wForest} do c-survive the flipping. In the next paragraph we bound the probability that this condition is fulfilled.

Bounding the probability Suppose that $c \leq \frac{c_{\text{w}}-1}{5}$. Consider a node $v \in V_{\text{Forest}}$. After the flipping, some of the child edges of v in G_{wForest} are vanished, others survive. As the representation is c_{w}-fat, the expected number of surviving child edges is at least $c_{\text{w}} - 1$. Using a Chernoff Bound from Lemma 2.2.1, the probability that v does *not* c-survive the flipping is upper bounded by

$$\mathrm{e}^{-\frac{(1-\frac{c}{c_{\text{w}}-1})^2 \cdot (c_{\text{w}}-1)}{2}}.$$

Consequently, v does c-survive the flipping with probability at least

$$1 - \mathrm{e}^{-\frac{(1-\frac{c}{c_{\text{w}}-1})^2 \cdot (c_{\text{w}}-1)}{2}} \geq 1 - \mathrm{e}^{-\frac{8}{25} \cdot (c_{\text{w}}-1)}$$

For the roots of G'_{Forest} and the other inner node v of G_{Forest} let X_v be a $0,1$ random variable which is 1 if v does c-survive and 0 otherwise. Then the random variables X_v are independent random variables as the flipping is performed independently for each inner node v of G_{wForest}. The forest G_{Forest} has $m' \cdot b \cdot \frac{e_{\text{sub}}}{c}$ inner nodes which are not roots of G_{wForest}. G_{wForest} has $m \leq (b+1) \cdot m'$ roots. Thus the probability that there is a complete explaining representation on G_{Forest} using the new vivid system is at least

$$\left(1 - \mathrm{e}^{-\frac{8}{25} \cdot (c_{\text{w}}-1)}\right)^{m' \cdot (b \cdot \frac{e_{\text{sub}}}{c} + b + 1)}$$

if there is a c_{w}-fat complete explaining representation on a weighted $(m, b, e_{\text{sub}}, c, d)$ witness forest using the old vivid system.

The size of the new vivid sets If the vivid sets of the edges of the old vivid system have weight at most M, each, the expected size of the corresponding new vivid sets is at most M. Thus using a Chernoff Bound from Lemma 2.2.1 the probability that a new vivid set for an edge contains more than $(1 + \sqrt{3}) \cdot M$ jobs is at most e^{-M}. But even this small probability does not allow to show that *all* vivid sets of the edges have size at most $(1 + \sqrt{3}) \cdot M$. The reason for this is that the number of jobs in the weighted allocation problem is unbounded and so is the number of old vivid sets of the edges in the old vivid system. On the other hand it suffices to bound the size of those sets which are important for us. For instance consider a level 1 edge e. If a job J is not contained

in $\mathit{vividrj}(S, \mathrm{top}(e))$ for any $S \in \mathcal{S}$, then we can put $\mathit{vivids}(J, S, v) = \emptyset$ for each S and each bottom node v of e. To apply this modification to sets for edges of level larger than 2, let

$$\mathcal{J}_{\mathrm{Vivid}}(v) = \bigcup_{S \in \mathcal{S}} \mathit{vividrj}(S, v)$$

for each root v of G_{Forest}. For each node v with $\mathit{level}(v) \geq 2$ with parent v' let

$$\mathcal{J}^{(v)}_{\mathrm{Vivid}} = \bigcup_{\substack{J \in \mathcal{J}_{\mathrm{Vivid}}(v') \\ i \in \{1,\ldots,d\} \\ S \in \mathit{vivids}(J,i)}} \mathit{vividj}(J, S, v).$$

Thus $\mathcal{J}_{\mathrm{Vivid}}(v)$ contains all jobs possibly represented by the regular child edges of v. We therefore put $\mathit{vividj}'(J, S, v) = \emptyset$ for each node v with parent v' and each $S \in \mathcal{S}$, if $J \notin \mathcal{J}_{\mathrm{Vivid}}(v')$. The vivid system $\mathcal{V}' = (\mathit{vividrj}, \mathit{vivids}, \mathit{vividj}', \mathit{vivide}_{\mathcal{R}})$ is called the *cleaned vivid system*. Any representation using the new vivid system uses the cleaned vivid system, too.

With probability at least $p_1 = e^{-M}$ the set $\mathcal{J}_{\mathrm{Vivid}}(v)$ has size at most $r_1 = N \cdot (1 + \sqrt{3}) \cdot M$. Consider a level l edge e with top node v whose parent is v'. Assume that with probability p_{l-1} it holds

$$|\mathcal{J}_{\mathrm{Vivid}}(v')| \leq r_{l-1}.$$

The probability that $|\mathit{vividj}(J, S, v)| \leq (1 + \sqrt{3}) \cdot M$ for each $J \in \mathcal{J}_{\mathrm{Vivid}}(v')$, $S \in \mathit{vivids}(J, i)$ for some i is at least

$$1 - \left[r_{l-1} \cdot N' \cdot d' \cdot \mathrm{e}^{-M} + (1 - p_{l-1})\right] = p_l.$$

Thus with probability at least p_l

$$\left|\mathcal{J}_{\mathrm{Vivid}}(v)\right| \leq r_l = r_{l-1} \cdot N' \cdot d' \cdot (1 + \sqrt{3}) \cdot M.$$

Thus

$$r_l = \mathrm{O}(1)^l \cdot N \cdot (N' \cdot d')^{l-1} \cdot M^l$$

and

$$\begin{aligned} p_l &= [\mathrm{O}(1)^l \cdot N \cdot (N' \cdot d' \cdot M)^l \cdot \mathrm{e}^{-M} + (1 - p_{l-1})] \\ &\geq 1 - [\mathrm{O}(1)^l \cdot N \cdot (N' \cdot d' \cdot M)^l \cdot \mathrm{e}^{-M}] \end{aligned}$$

As the witness forest G_{Forest} has height at most $\log^{\mathrm{O}(1)} N$ and at most N inner nodes, each vivid set of an edge in G_{Forest} contains at most $(1 + \sqrt{3}) \cdot M$ jobs, with probability at least

$$1 - \mathrm{O}(1) \cdot (M \cdot d' \cdot N')^{\log^{\mathrm{O}(1)} N} \cdot \mathrm{e}^{-M} \leq \frac{1}{2} \tag{4.1}$$

if $M = \omega(\log^{\varpi} N)$, for some $\varpi = \mathrm{O}(1)$. Thus with probability at least one half the cleaned vivid system is a $(N, N', p, (1+\sqrt{3}) \cdot M, d', \mu)$-vivid system. This allows to prove the Main Lemma for Weighted Jobs.

Proof: Let p_{w} denote the probability that there is a c_{w}-fat, complete explaining representation using a $(N, N', p, \infty, d', \mu)$-vivid system of weight M on G_{wForest}. Given such a representation, the probability that there is a root-expansion-free complete explaining representation using a $(N, N', p, (1+\sqrt{3}) \cdot M, d', \mu)$-vivid system on a $(m, b, e_{\mathrm{sub}}, c, d)$ witness forest is at least

$$\frac{1}{2} \cdot \left(1 - \mathrm{e}^{-\frac{8}{25} \cdot (c_{\mathrm{w}} - 1)}\right)^{m' \cdot (b \cdot \frac{e_{\mathrm{sub}}}{c} + b + 1)} \geq \frac{1}{2} \cdot \epsilon_c^{-\frac{1}{11} \cdot m' \cdot (b \cdot \frac{e_{\mathrm{sub}}}{c} + b + 1)}$$

as $e_{\mathrm{sub}} \geq b + 1$

$$\geq \frac{1}{2} \cdot \epsilon_c^{-\frac{2}{11} \cdot e_{\mathrm{sub}} \cdot m' \frac{b}{c}}.$$

The Main Lemma (Lemma 3.2.6) upper bounds the probability that there is such a representation by

$$\left(\frac{N}{m'}\right)^{m'} \cdot e^{3m'} \cdot \epsilon_c^{-e_{\mathrm{sub}} \cdot m' \cdot b}.$$

On account of the above remarks

$$\begin{aligned} p_{\mathrm{w}} &\leq 2 \cdot \left(\frac{N}{m'}\right)^{m'} \cdot e^{3m'} \cdot \epsilon_c^{-e_{\mathrm{sub}} \cdot m' \cdot b \cdot (1 - \frac{2}{11})} \\ &\leq 2 \cdot \left(\frac{N}{m'} \cdot e^3\right)^{m'} \cdot 4^{-\frac{9}{11} \cdot m'} \cdot N^{-\frac{9}{11} \cdot \alpha} \cdot \left(\frac{N}{m'} \cdot \mathrm{e}^3\right)^{-\frac{9}{11} \cdot m'} \end{aligned}$$

using the definition of α and $b \geq 3$

$$\begin{aligned} &\leq N^{\frac{\ln(\frac{N}{m'} \cdot \mathrm{e}^3) \cdot m'}{\ln N} \cdot (-\frac{9}{11} \cdot \frac{3}{12} \cdot \frac{b}{3} + \frac{2}{11} \cdot \frac{b}{3})} \\ &\leq N^{-\frac{1}{132} \cdot b} \end{aligned}$$

■ of Lemma 4.3.3

5 Applications of the Main Lemma

5.1 The Sequential Setting

5.1.1 The greedy algorithm

The greedy algorithm is the most important algorithm in the sequential setting. It is presented in Figure 1.2 on Page 23. So far (compare Section 3.1.2, Corollary 3.3.2) we have considered the greedy algorithm as an example to demonstrate our technique, only. We have investigated the performance of the greedy algorithm assuming $\overline{M} < N/d$ and considering a particular tie breaking mechanism for step (1). In this Section we generalize the result on the greedy algorithm to higher values of $\overline{M}$ and various i-th copy distributions.

Other deletion schemes and the finite setting The deletion scheme defined in Section 1.7.2 assumes that job J is deleted in round $\Delta(J) = \delta(J) + \overline{M}$. In this section we generalize this result to a large class of deletion schemes including all schemes where the sequence of insertions and deletions is independent of the possible servers and the allocation. These deletion schemes are called *oblivious deletion schemes*. Important oblivious deletion schemes are the *random deletion scheme* — where in each round a job is chosen independently and uniformly at random to be deleted from the system — and the *finite deletion scheme*. In the finite deletion scheme we have $\Delta(J) = \infty$, if $\delta(J) \leq \overline{M}$ and $\Delta(J) = \delta(J)$, otherwise. This way the finite deletion scheme mimics the behavior of the finite setting with $\overline{M}$ jobs.

Our results are not restricted to oblivious deletion schemes, they hold for any deletion scheme ensuring

$$\mathrm{Prob}\big[S^{(i)}(J) = S \,\big|\, J \in \mathcal{J}_\tau^{\delta,\Delta}\big] = \mathrm{Prob}\big[S^{(i)}(J) = S\big].$$

for each $J \in \mathcal{J}$, $S \in \mathcal{S}$, $i \in \{1, \ldots, d\}$, and $\delta(J) \leq \tau < \Delta(J)$; and

$$\max \left|\mathcal{J}_\tau^{\delta,\Delta}\right| \leq \overline{M}.$$

Actually these two assumptions suffice to ensure that the vivid system $\mathcal{V}_{\mathrm{Gr}}$ as defined in Remark 3.2.8 uses vivid sets for the edges with size at most $\overline{M}$. Thus Corollary 3.3.2 holds for any deletion scheme fulfilling the conditions above.

Other i-th copy distributions Considering the greedy algorithm as an example for our analysis, we have restricted ourselves to uniform i-th copy distributions. In this paragraph we generalize our argument to arbitrary i-th copy distributions. The construction used in this paragraph is not restricted to the analysis for the greedy algorithm. It applies to any use of non-uniform i-th copy distributions.

Consider a representation $\mathcal{R}$ on a witness forest G_{Forest} using a natural vivid system $\mathcal{V} = (\mathit{vividrj}, \mathit{vivids}, \mathit{vividj})$. We show how to modify $\mathcal{V}$ such that the modified vivid system has better parameters. For sake of that we remove all elements from the vivid sets of the edges and the vivid sets of the nodes, which cannot appear in an explaining representation. For instance, if node v represents server S, then a job J with $\text{Prob}[S^{(i)}(J)] = 0$ for any i cannot be represented by a child edge of v.

For round τ let $\mathcal{V}_\tau = (\mathit{vividrj}, \mathit{vivids}, \mathit{vividj})$ be the natural vivid system for round τ. For each $S \in \mathcal{S}$ and each root v of G_{Forest} let

$$\mathit{vividrj}'(S, v) = \{J \in \mathit{vividrj}(S, v) \,|\, \exists j \in \{1, \ldots, d\} : \text{Prob}[S^{(j)}(J) = S] > 0\}.$$

For each $J \in \mathcal{J}$, $i \in \{1, \ldots, d\}$ let

$$\mathit{vivids}'(J, i) = \{S \in \mathit{vivids}(J, i) \,|\, \text{Prob}[S^{(i)}(J) = S] > 0\}.$$

Similar let for each $J \in \mathcal{J}$, $i \in \{1, \ldots, d\}$, $S \in \mathit{vividj}(J, i)$

$$\mathit{vividj}'(J, S) = \{J \in \mathit{vividj}(J, S) \,|\, \exists j \in \{1, \ldots, d\} : \text{Prob}[S^{(j)}(J) = S] > 0\}.$$

Then the vivid system $\mathcal{V}'_\tau = (\mathit{vividrj}', \mathit{vivids}', \mathit{vividj}', \mathit{vivide}_{\mathcal{R}})$ is called the Ξ-*vivid system of round* τ. If the meaning is clear, we drop the "τ" from our notation. As each representation uses the natural vivid system, it also uses the Ξ-vivid system.
For

$$\begin{aligned}
N'_\Xi &= \max_{\substack{J \in \mathcal{J} \\ i \in \{1, \ldots, d\}}} \left|\mathit{vivids}'(J, i)\right|, \\
M_\Xi &= \max \left\{ \max_{\substack{J \in \mathcal{J} \\ i \in \{1, \ldots, d\} \\ S \in \mathit{vivids}'(J, i)}} \left|\mathit{vividj}'(J, S)\right|, \max_{\substack{S \in \mathcal{S} \\ v \in \mathit{root}(G_{\text{Forest}})}} \left|\mathit{vividrj}'(S, v)\right| \right\} \\
p_\Xi &= \max_{\substack{J \in \mathcal{J} \\ i \in \{1, \ldots, d\} \\ S \in \mathcal{S}}} \text{Prob}[S^{(i)}(J) = S], \\
d'_\Xi &= \max_{\substack{J \in \mathcal{J} \\ S \in \mathcal{S}}} \left| \{j \in \{1, \ldots, d\} \,|\, \text{Prob}[S^{(j)}(J) = S] > 0\} \right|
\end{aligned}$$

the Ξ-vivid system $\mathcal{V}'$ is a $(N, N'_\Xi, p_\Xi, M_\Xi, d'_\Xi, 1)$-vivid system. The parameters N, N'_Ξ, p_Ξ, M_Ξ, and d'_Ξ are called *the parameters of the* Ξ-*vivid system.* p_Ξ is called the *maximum probability of* Ξ, and d'_Ξ is called the *mixing of* Ξ. In any case M_Ξ is the maximum size

of a set $\mathcal{J}_\tau^{\Delta,\delta}$, as the i-th copy distributions are the same for each job. This implies $\overline{M} = M_\Xi$. It would be possible to define something like a Ξ-vivid system for *dependent* i-th copy distributions, where the i-th possible server of a job depends stochastically on other possible servers of that job. This would only affect the definition of p_Ξ. As the vivid system completely encapsulates the properties of the i-th copy distributions, a generalized definition would allow us to deal with dependent i-th copy distributions. We did not use this possibility, as we wanted to avoid an even more complicated definition of the vivid system.

For the greedy algorithm there is one set of non-uniform i-th copy distributions which is of particular importance (compare [Vöc99a]). To define these distributions let $\Xi'(i)$ be a distribution on $\mathcal{S} = \{S_1, \ldots, S_N\}$ which distributes the i-th possible servers uniformly on $\{S_{(i-1)\cdot\frac{N}{d}+1}, \ldots, S_{i\cdot\frac{N}{d}}\}$ (for convenience we assume that N is a multiple of d). We call the distribution functions $\Xi'(i)$, $1 \leq i \leq d$, the *separated i-th copy distributions* on $\mathcal{S}$. The separated i-th copy distributions have mixing 1.

Lemma 5.1.1: Let $\tau \in \mathbb{N}$ be a round, and assume that there are m servers with load $T'+1$ at the beginning of round τ of the greedy algorithm working on an allocation problem with separated i-th copy distributions. Then there is a complete explaining representation on the witness forest $G^{\mathrm{Gr}}_{\mathrm{Forest}}$ using a $(N, \frac{N}{d}, (\frac{N}{d})^{-1}, \overline{M}, 1, 1)$-vivid system.

Proof: The vivid system is just the Ξ-vivid system of the natural vivid system. ■

A drawback of our construction is revealed, if the considered distributions are non-uniform but $\mathrm{Prob}[S^{(i)}(J) = S] > 0$ for each i and each S. In this case our analysis is not sharp. The reason for this is that we do not distinguish between two different tasks of 'p_Ξ'. On one hand $p_\Xi \cdot N$ denotes the expected number of servers suitable to be represented by a node, if the representation for its parent edge is fixed. This requires p_Ξ to be an upper bound for

$$p_\Lambda = \max_{J\in\mathcal{J}} \max_{i\in\{1,\ldots,d\}} \frac{1}{N'_\Xi} \cdot \sum_{s\in\mathcal{S}} \left(\mathrm{Prob}[S^{(i)}(J) = S]\right),$$

as the worst case for the representation on the edge has to be covered. On the other hand $p_\Xi \cdot d'_\Xi \cdot M_\Xi$ denotes the expected number of ways to choose a representation for an edge, given a server represented by its top node. This requires p_Ξ being at least

$$p_{\mathrm{edge}} = \max_{s\in\mathcal{S}} \frac{1}{d'_\Xi \cdot M_\Xi} \cdot \sum_{J\in\mathcal{J}} \sum_{i\in\{1,\ldots,d\}} \left(\mathrm{Prob}[S^{(i)}(J) = S]\right).$$

In the non uniform case the second value is typically larger than N^{-1}, while the first one typically equals N^{-1}. Choosing $p_\Xi > p_\Lambda$ yields an unnecessarily large value for ϵ_c (compare the discussion after Lemma 3.2.6). But even if we would use p_Λ where its appropriate, the performance for non-uniform distributions would suffer from $p_{\mathrm{edge}} >$

N^{-1}, which indicates that non-uniform distributions may lead to poorer performance. On the other hand the (non-uniform) separated i-th copy distributions yield the same performance as uniform i-th copy distributions. Thus only particular, stupid i-th copy distributions impair performance. Our analysis overestimates this effect a little bit, but a sharp result would require more subtle analysis of the representation of the witness forest. Anyhow only the additive constants of our performance bounds are affected, as we see in the next paragraph.

Larger values of $\overline{M}$ The result on the greedy algorithm presented in Corollary 3.3.2 on Page 40 is restricted to $\overline{M} < N$. To obtain results for larger values $\overline{M}$, we use a property of the representation $\mathcal{R}_{\mathrm{Gr}}$ noted in Section 3.1.2: if there are m servers with load $\overline{T}'$ at some round $\bar{\tau}$ and the witness forest $G^{\mathrm{Gr}}_{\mathrm{Forest}}$ has height $\overline{T}$, then each node v of $G^{\mathrm{Gr}}_{\mathrm{Forest}}$ with time marker τ' has

$$load_{\tau'}(server(v)) \geq \overline{T}' - \overline{T} - k_{\mathrm{Gr}} + 1.$$

If τ'' is the time marker of a child edge of v, then

$$load_{\tau''-1}(server(v)) \geq \overline{T}' - \overline{T} - k_{\mathrm{Gr}} - 1 = g.$$

This observation is useful for $g > 0$. It allows us to define vivid events for the representation $\mathcal{R}_{\mathrm{Gr}}$.

Lemma 5.1.2: Consider the greedy algorithm working on an ordinary or weighted allocation problem. For the tie breaking mechanism, allow any mechanism which never allocates a job to more than one server. Let $\sigma_\Xi = N \cdot p_\Xi \cdot \frac{d'_\Xi}{d}$ be the *stupidity of the i-th copy distributions*. Let

$$g = \left(\min\{5.275 + \tfrac{1}{d}, d\} + \mathrm{o}(1)\right) \cdot \frac{\overline{M}}{N} \cdot \sigma_\Xi, \text{ and}$$
$$\mu_\Xi = (3 \cdot (N \cdot p_\Xi) \cdot (\overline{M} \cdot p_\Xi \cdot d'_\Xi))^{-1}.$$

Assume that there is a representation $\mathcal{R}$ on a witness forest G_{Forest} using the Ξ-vivid system $\mathcal{V}$, such that each node $v \in V_{Server}$ has

$$load_\tau(server(v)) \geq g, \quad \text{for } \tau = \min_{e \ni v}\{\delta(job(e))\} - 1.$$

Then there is a mapping $vivide_{\mathcal{R}}$ such that

- The vivid system $\mathcal{V}' = (vividrj, vivids, vividj, vivide_{\mathcal{R}})$ is a $(N, N'_\Xi, p_\Xi, \overline{M}, d'_\Xi, \mu_\Xi)$-vivid system (a $(N, N'_\Xi, p_\Xi, \infty, d'_\Xi, \mu_\Xi)$-vivid system of weight $\overline{M}$), and
- $\mathcal{R}$ uses $\mathcal{V}'$.

Proof: Fix a representation $\mathcal{R}$. We define a vivid event for each node $v \in V_{Server} \setminus V_{\mathrm{Exp}}$. Among all edges incident to nodes representing *server*(v), let e be the one representing the oldest job. Let this job be J_τ. Then let *vivide*$_\mathcal{R}(v)$ be the event "*load*$_{\tau-1}$(*server*$(v)) \geq g$". It remains to show that for

$$\mathcal{E} = \bigwedge_{\substack{w \in V_{\mathrm{Server}} \\ server(w) \neq server(v)}} vivide_\mathcal{R}(w)$$

it holds

$$\mathrm{Prob}[vivide_\mathcal{R}(v) \,\big|\, \text{“}\mathcal{R} \text{ is explaining”} \wedge \mathcal{E}] \leq \mu_\Xi.$$

For each job J represented by an edge of G_{Forest}, *vivide*$_\mathcal{R}(v)$ is not about J, as either $\delta(J) < \tau$ or $S^{(i)}(J) \neq server(v)$ for all $i \in \{1, \ldots, d\}$, thus

$$\mathrm{Prob}[vivide_\mathcal{R}(v) \,\big|\, \text{“}\mathcal{R} \text{ is explaining”} \wedge \mathcal{E}] \leq \mathrm{Prob}[vivide_\mathcal{R}(v) \,\big|\, \mathcal{E}\,].$$

The event *vivide*$_\mathcal{R}(v)$ is not independent from $\mathcal{E}$ as requests sent to a server S cannot be sent to another server $S' \neq S$. But for the same reason this dependency only decreases the probability that *vivide*$_\mathcal{R}(v)$ is fulfilled, thus $\mathrm{Prob}[\mathrm{vivide}_\mathcal{R} \,|\, \mathcal{E}] \leq \mathrm{Prob}[\mathrm{vivide}_\mathcal{R}]$.

Case 1: $d \geq 5$

Let τ be a round, let S be a server. Consider the set $\mathcal{J}_\tau^{\delta,\Delta}$ of jobs being in the system at the beginning of round τ. If S reaches *load*$_\tau(S) = g$, there has to be a round τ' such that the weight of jobs from $\mathcal{J}_\tau^{\delta,\Delta}$ being allocated to S before τ' is

$$k = \lceil \mathrm{e} \cdot (\frac{9}{4} \cdot e)^{\frac{1}{4}} \cdot \frac{\overline{M}}{N} \cdot \sigma_\Xi \rceil.$$

If a job J with $\delta(J) = \hat{\tau}$ is allocated to S after round τ', each possible server S' of J has to have load at least k at the beginning of round $\hat{\tau}$. As the system contains jobs with weight at most $\overline{M}$, there are at most $(\mathrm{e} \cdot (\frac{9}{4} \cdot \mathrm{e})^{\frac{1}{4}} \cdot \sigma_\Xi)^{-1} \cdot N$ such servers. Hence the probability that a particular job is allocated to S after round τ' is at most

$$\begin{aligned}
& d'_\Xi \cdot p_\Xi \cdot \left((\mathrm{e} \cdot (\frac{9}{4} \cdot \mathrm{e})^{\frac{1}{4}} \cdot \sigma_\Xi)^{-1} \cdot N \right)^{d-1} \cdot p_\Xi^{d-1} \\
& \quad \leq d'_\Xi \cdot p_\Xi \cdot \left(\left(\frac{1}{\frac{9}{4} \cdot \mathrm{e}} \right)^{\frac{1}{4}} \cdot \frac{1}{p_\Xi} \cdot \frac{d}{d'_\Xi} \cdot \frac{1}{d-1} \right)^{d-1} \cdot p_\Xi^{d-1} \\
& \quad \leq p_\Xi \cdot \left(\frac{4}{9} \right)^{d-1} \cdot \mathrm{e}^{-1} \cdot \left(\frac{d}{d-1} \right)^{d-1} \\
& \quad \leq p_\Xi \cdot \frac{1}{d^2},
\end{aligned}$$

as $d \geq 5$. The expected weight of the jobs $J \in \mathcal{J}_\tau^{\delta,\Delta}$ allocated to S after $J_{\tau'}$ is at most $\overline{M} \cdot p_\Xi \cdot \frac{d'_\Xi}{d^2} = \frac{\overline{M}}{N} \cdot \sigma_\Xi \cdot \frac{1}{d}$. Using the generalized Chernoff Bound from Lemma 2.2.2, the probability that this weight exceeds

$$\frac{\overline{M}}{N} \cdot \sigma_\Xi \cdot \frac{1}{d} + 3 \cdot \sqrt{\frac{\overline{M}}{N} \cdot \sigma_\Xi \cdot \frac{1}{d}} \cdot \ln(3\frac{\overline{M}}{N} \cdot d^2 \cdot \sigma_\Xi^2) = \frac{\overline{M}}{N} \cdot \sigma_\Xi \cdot \frac{1}{d} + \mathrm{o}(\frac{\overline{M}}{N} \cdot \sigma_\Xi)$$

is at most

$$\left(3 \cdot \frac{\overline{M}}{N} \cdot d^2 \cdot \sigma_\Xi^2\right)^{-1} \leq \left(3 \cdot N'_\Xi \cdot p_\Xi \cdot \overline{M} \cdot p_\Xi \cdot d'_\Xi\right)^{-1}. \tag{5.1}$$

Case 2: The probability that S receives requests with load exceeding $\overline{M} \cdot p_\Xi \cdot d'_\Xi + 3 \cdot \sqrt{\overline{M} \cdot p_\Xi \cdot d'_\Xi} \cdot \ln(2 \cdot N \cdot p_\Xi \cdot \overline{M} \cdot p_\Xi \cdot d'_\Xi) = \frac{\overline{M}}{N} \cdot \sigma_\Xi \cdot d + \mathrm{o}(\frac{\overline{M}}{N} \cdot \sigma_\Xi \cdot d)$ from jobs in $\mathcal{J}_\tau^{\delta,\Delta}$ is at most

$$\mathrm{e}^{-\ln(3 \cdot N \cdot p_\Xi \cdot \overline{M} \cdot p_\Xi \cdot d'_\Xi)} \leq \left(3 \cdot N'_\Xi \cdot p_\Xi \cdot \overline{M} \cdot p_\Xi \cdot d'_\Xi\right)^{-1},$$

according to the generalized Chernoff Bound from Lemma 2.2.2.

■ of Lemma 5.1.2

Remark 5.1.3: The restriction that the tie breaking mechanism never allocates a job to more than one server, is only required for Case 1 of the proof. Thus if $g = (d + \mathrm{o}(1)) \cdot \frac{\overline{M}}{N} \cdot \sigma_\Xi$, the proposition of Lemma 5.1.2 holds for arbitrary tie breaking mechanisms.

Theorem 5.1.4: Consider an ordinary allocation problem in the finite setting with $\overline{M}$ jobs or an allocation problem in the sequential setting using an oblivious deletion scheme which ensures that there are at most $\overline{M}$ jobs in the system. Let $\sigma_\Xi = N \cdot p_\Xi \cdot \frac{d'_\Xi}{d}$ be the *stupidity of the i-th copy distributions*. Then any version of the greedy algorithm which never allocates a job to more than one server assures that less than m servers obtain load larger than

$$\overline{T} = \log_d \log_{\epsilon_c} \frac{N}{m} + \left(\min\{5.275 + \tfrac{1}{d}, d\} + \mathrm{o}(1)\right) \cdot \frac{\overline{M}}{N} \cdot \sigma_\Xi + k_{\mathrm{Gr}} + 2 + \mathrm{o}(1),$$

with probability at least $1 - N^{-\alpha}$, for

$$\begin{aligned} \epsilon_c &= 3^{d-1} \\ k_{\mathrm{Gr}} \cdot (d-1) &\geq 2, \\ \alpha &\leq \frac{1}{12}\left[1 + k_{\mathrm{Gr}} \cdot (d-1)\right], \text{ and} \\ \tfrac{N}{m} &\text{ large enough.} \end{aligned}$$

Proof: The Theorem is an immediate consequence of the discussion above, the Main Lemma (Lemma 3.2.6), and Remark 3.2.7. ■ of Theorem 5.1.4

As a consequence of Remark 5.1.3 the proposition of Theorem 5.1.4 is also true for arbitrary tie breaking mechanisms if

$$T_m = \log_d \log_{\epsilon_c} \frac{N}{m} + \big(d + \mathrm{o}(1)\big) \cdot \frac{\overline{M}}{N} \cdot \sigma_\Xi + k_{\mathrm{Gr}} + 2 + \mathrm{o}(1).$$

Remark 5.1.5: Theorem 5.1.4 does not contain an explicit restriction on the i-th copy distributions. Implicitly such a restriction applies as the requirement that $\frac{N}{m}$ has to be large enough (cf. Equations 3.20 and 3.21 in the proof of Lemma 3.4.4) depends on the parameters of the vivid system which depends on the i-th copy distributions. There are i-th copy distributions that do not allow to fulfill Equations 3.20 and 3.21. As long as $N' = \omega(N^{1/6})$ and $p_\Xi = \Theta(\mathrm{o}(\log N) \cdot \frac{1}{N'})$, the mentioned equations are easily fulfilled. The same restrictions hold for any theorem presented in this chapter. Violating these conditions would yield poor performance, anyway.

Our bound shows that the greedy algorithm obtains load at most $\log_d \log_{\epsilon_c} N + \mathrm{O}\big(\frac{\overline{M}}{N}\big) \cdot \sigma_\Xi$, with high probability. This matches the best previously known results for uniform distributions (compare [ABKU94, ACMR95, Vöc99a]). It is of particular interest to compare the load obtained here to the load obtained if only one possible server per job is used ($d = 1$). If in the latter case a uniform i-th copy distribution is used, the load of the allocation is $\frac{M}{N} + \Theta(\sqrt{\frac{M}{N} \cdot \ln N})$ ([Vöc99b]). Compared to this bound, the greedy algorithm — being the most proximate algorithm at all in the sequential setting — shows that using $d \geq 2$ allows to obtain an exponential improvement in the obtained load. In contrast to previous results our analysis applies to arbitrary i-th copy distributions, showing that stupid choice of the i-th copy distributions affects only the additive constant of the performance bound. The only result getting ahead of ours is the result on the Always-go-Left version of the greedy algorithm from [Vöc99a]. We treat this version of the greedy algorithm in the next section.

The result of Azar et al. considers a slightly different situation than ours. At first their analysis is restricted to the random deletion scheme. Despite of that they assume that the first $\overline{M}$ jobs are allocated to arbitrary servers and only subsequent jobs are placed to randomly chosen possible servers according to the greedy algorithm. In this situation no sensible bound on the load can be given for round $\tau = \overline{M}$ and the same holds for the next rounds. It is therefore a natural question to ask when the load bound applies for the first time. Azar et al. show that after N^3 rounds the load in the system is $\log_d \ln N$, with high probability. This result is improved by Czumaj in [Czu97] who shows that $(1 + \mathrm{o}(1)) \cdot N \cdot \ln N$ rounds are sufficient to obtain this bound, with high

probability. He uses Markov Chains and coupling (see e.g. [Lin92]) to obtain this result. A slightly weaker result is easily shown using our approach.

To obtain the result we just have to ensure that the vivid system used by the representation does not involve any of the first $\overline{M}$ jobs, as only those jobs do not use random possible servers. In each round the probability for a job to be deleted is $\frac{1}{N}$. Thus the probability that a job J is not deleted during $\alpha \cdot N \cdot \ln N$ rounds is at most

$$\begin{aligned}(1-\tfrac{1}{N})^{\alpha \cdot N \cdot \ln N} &\leq \mathrm{e}^{-\alpha \cdot \ln N} \\ &\leq N^{-\alpha}.\end{aligned}$$

Thus after round $\alpha \cdot N \cdot \ln N + N - 1 + \overline{M}$ no arbitrarily placed job is in the system. Let the jobs J with $\delta(J) \leq \overline{M}$ be called the 0-*dirty jobs*. Further we call a job J *i-dirty*, if the system contains an $(i-1)$-dirty job in round $\delta(J)$. As shown above, the system contains no 0-dirty job after round $\alpha \cdot N \cdot \ln N + N - 1 + \overline{M}$, with probability at least $1 - N^{-\alpha}$. After round $2 \cdot (\alpha \cdot N \cdot \ln N + N - 1) + \overline{M}$, there is no 1-dirty job in the system with probability at least $1 - 2 \cdot N^{-\alpha}$. After $T \cdot (\alpha \cdot N \cdot \ln N + N)$ rounds no job in the system is a $(T-1)$-dirty job, with probability at least $1 - T \cdot N^{-\alpha}$. Thus if $\tau \geq \overline{M} + \overline{T}' \cdot (\alpha \cdot N \cdot \ln N + N)$ the representation used in Theorem 5.1.4 does not contain any 0-dirty job. We therefore can remove the 0-dirty jobs from the vivid sets of the edges and the analysis succeeds. (Note that the random events considered here are events of the deletion scheme, not of the possible servers of the jobs.) Compared to the result of Czumaj we just loose a factor of $\overline{T}' = \mathrm{O}(\log_d \log N)$ and some minor constant factors. Thus in combination with the witness tree technique and the vivid systems notation our quite coarse argument suffices to almost match the best known bound.

5.1.2 The greedy algorithm on weighted allocation problems

The greedy algorithm can be applied to weighted allocation problems without any change. To analyze the performance of the greedy algorithm on weighted allocation problems we use the Main Lemma for Weighted Jobs (Lemma 4.3.3 from Chapter 4). We consider the greedy algorithm working on a weighted allocation problem in the infinite setting, using some oblivious deletion scheme Δ.

We first show how to obtain a witness forest and a representation. Fix a round $\bar{\tau}$, fix the possible servers of the jobs and assume that there is a server $S \in \mathcal{S}$ with $load_{\bar{\tau}}(S) \geq \overline{T}' \cdot (c_{\mathrm{w}} + 1)$ for some $c_{\mathrm{w}} \in \mathbb{R}$. Let G_{Tree} be a d-uniform tree of height $\overline{T} \leq \overline{T}'$ specified in the sequel. We show how to obtain a c_w-fat complete explaining representation on G_{Tree}. Let the root of G_{Tree} represent server S and let its time marker be $\bar{\tau}$. Consider a node v of G_{Tree} of level l and assume that the server S represented by v has load at least $(\overline{T}' - l) \cdot (c_{\mathrm{w}} + 1)$ at the end of the round indicated by the time marker τ' of v. Let $J_1 \in \mathcal{J}_{\tau'}^{\delta,\Delta}$ be the last job allocated to S before round τ', let $J_2 \in \mathcal{J}_{\tau'}^{\delta,\Delta}$, $J_2 < J_1$ be the last job allocated to S before $\delta(J_1)$, and so on, such that

$J_1, J_2, \ldots, J_i \in \mathcal{J}_{\tau'}^{\delta,\Delta}$ are the last i jobs allocated to S before round τ'. We choose an $i_{c_\mathrm{w}} \in \mathbb{N}$, such that

$$c_\mathrm{w} \leq \sum_{i=1}^{i_{c_\mathrm{w}}} W(J_i) \leq c_\mathrm{w} + 1.$$

Then let v have i_{c_w} child edges, and for each $i \in \{1, \ldots, i_{c_\mathrm{w}}\}$ let the i-th child edge of v represent job J_i. For each child edge e let $label_e$ ensure that $S^{(label_e(v))}(\mathrm{job}(e)) = S$. Further let each bottom node v' of e represent server $S^{(label_e(v'))}(\mathrm{job}(e))$ and let its time marker be $\delta(J) - 1$. Each child of v now represents a server S with load at least $(\overline{T}' - (l+1)) \cdot (c_\mathrm{w}+1)$ at the round indicated by its time marker. Note further that v has at least c_w children. Applying this construction recursively to each inner node of G_Tree yields a c_w-fat complete explaining representation $\mathcal{R}_\mathrm{wGr}$ on G_Tree. The tree G_Tree is a weighted $(c(d-1), e_\mathrm{sub}, c, d)$ witness tree for any $c \leq c_\mathrm{w}$ and $e_\mathrm{sub} \leq c \cdot \sum_{i=0}^{T-2} [c(d-1)]^i$.

If there are m servers with load at least $\overline{T}' \cdot (c_\mathrm{w} + 1)$ there is a representation on a weighted $(m, c(d-1), e_\mathrm{sub}, c, d)$ witness forest $G_\mathrm{Forest}^\mathrm{wGr}$. If there are m servers with load at least $\overline{T}' \geq \overline{T} \cdot (c_\mathrm{w} + 1) + g$, then each server appearing in the representation has load at least g at the round indicated by its time marker. This allows to apply Lemma 5.1.2.

If $\mathcal{V}_\mathrm{wGr}$ is the Ξ-vivid system $\mathcal{V}_\tau$, then $\mathcal{R}_\mathrm{wGr}$ uses $\mathcal{V}_\mathrm{wGr}$. If the deletion scheme ensures that $\sum_{J \in \mathcal{J}_{\tau'}^{\delta,\Delta}} W(J) \leq \overline{M}$ for any τ', then $\mathcal{V}_\mathrm{wGr}$ has weight $\overline{M}$.

Theorem 5.1.6: Consider a weighted allocation problem in the finite setting with jobs of weight $\overline{M}$ or an allocation problem in the sequential setting using an oblivious deletion scheme ensuring that the weight of the jobs in the system does not exceed $\overline{M}$. Then any version of the greedy algorithm which never allocates a job to more than one server assures that less than m servers obtain load larger than

$$\overline{T}' = \mathrm{O}\Big(\log_d \log_{\epsilon_c} \frac{N}{m}\Big) + \big(\min\{5.275 + \tfrac{1}{d}, d\} + \mathrm{o}(1)\big) \cdot \frac{\overline{M}}{N} \cdot \sigma_\Xi,$$

with high probability, for

$$\epsilon_c = \frac{3^{d-1}}{1 + \sqrt{3}} > 1$$

and $\frac{N}{m}$ large enough.

Proof: Let $c \geq 3$, $b = c(d-1)$, $m' = \lceil \frac{m}{b} \rceil$, $c_\mathrm{w} \geq 5 \cdot c + 1$, such that $c_\mathrm{w} \geq -\frac{25}{8} \cdot \ln(1 - \epsilon_c^{-\frac{1}{11}}) + 1$. For

$$\begin{aligned} \overline{T} &= (c_\mathrm{w} + 1) \cdot \log_{c(d-1)} \bar{e}_\mathrm{sub} \\ &= \mathrm{O}\Big(\log_d \log_{\epsilon_c} \frac{N}{m}\Big) \end{aligned}$$

the witness forest $G^{\text{wGr}}_{\text{Forest}}$ is a weighted $(m, c(d-1), e_{\text{sub}}, c, d)$ witness forest. If there are m servers with load higher than $\overline{T}'$ there is a representation $\mathcal{R}_{\text{wGr}}$ being a c_{w}-fat complete explaining representation on $G^{\text{wGr}}_{\text{Forest}}$. The representation uses the Ξ-vivid system. As each server appearing in the representation has load at least $\overline{T}' - \overline{T} = \big(\min\{5.275 + \frac{1}{d}, d\} + \text{o}(1)\big) \cdot \frac{\overline{M}}{N} \cdot \sigma_\Xi$, Lemma 5.1.2 allows to conclude that $\mathcal{R}_{\text{wGr}}$ uses a $(N, N', p_\Xi, \infty, d'_\Xi, \mu_\Xi)$-vivid system of weight $\overline{M}$. The Main Lemma for Weighted Jobs (Lemma 4.3.3) ensures that such a representation does not exist with probability at least $1 - N^{-\frac{b}{132}}$. ■ of Theorem 5.1.6

Remark 5.1.7: As a consequence of Remark 5.1.3 the proposition of Theorem 5.1.6 is also true for arbitrary tie breaking mechanisms if

$$\overline{T}' = \text{O}\Big(\log_d \log_{\epsilon_c} \frac{N}{m}\Big) + \big(d + \text{o}(1)\big) \cdot \frac{\overline{M}}{N} \cdot \sigma_\Xi.$$

The performance in the weighted case is almost the same as the performance in the ordinary case. Only in the $\text{O}(\log_d \log_{\epsilon_c} N)$ term we lost some constant factor compared to the result for ordinary allocation problems. Our analysis allows to state that this is typical when weighted results are compared to ordinary results.

5.1.3 The Always-go-Left version of the greedy algorithm

As noted by Azar et al. no allocation algorithm for a sequential allocation problem using uniform i-th copy distributions can do better (up to an additive constant) than the greedy algorithm, no matter which tie breaking mechanism is used. Surprisingly this is not true for the separated i-th copy distributions defined in Section 5.1.1. In this case the so-called *Always-go-left* tie breaking mechanism achieves better performance than other tie breaking mechanisms for the greedy algorithm. The Always-go-left tie breaking mechanism is quite simple: in case of a tie, the smallest server (with respect to '$<$') is chosen. This allows to obtain an upper bound of

$$\text{O}\Big(\frac{\ln \log_{\epsilon_c} N}{d}\Big)$$

instead of

$$\text{O}\Big(\frac{\ln \log_{\epsilon_c} N}{\ln d}\Big) = \text{O}\Big(\log_d \log \epsilon_c N\Big)$$

as achieved by other versions. For large d the difference between the Always-go-left version and other tie-breaking mechanisms is substantial. This fact was noticed by Vöcking, [Vöc99a]. He proves it using a witness tree argument. We state a proof for his result using our Main Lemma.

To give the witness forest and the representation we recursively define a (T,i,d)-*Fibonacci tree*. For $T=0$ and any $i \in \{1,\ldots,d\}$, the (T,i,d)-Fibonacci tree consists out of a single node. For $T>0$ the root of the (T,i,d)-Fibonacci tree has two child edges, one edge of size d — called *regular child edge* — and one edge of size 2 — called *irregular child edge*. The bottom node of the roots irregular child edge is root of a $(T-1,i,d)$-Fibonacci tree. The j-th bottom node of the regular child edge of v is the root of a (T,j,d)-Fibonacci tree if $j<i$, and it is the root of a $(T-1,j+1,d)$-Fibonacci tree otherwise.

A server $S \in \mathcal{S}$ is said to *be in the i-th group*, if $S \in \{S_{(i-1)\cdot\frac{N}{d}+1},\ldots,S_{i\cdot\frac{N}{d}}\}$. If a server is in the i-th group, then only an i-th copy of a job can be S. We claim that there is a complete explaining representation on a $(\overline{T}+k_{\mathrm{Gr}},d,d)$-Fibonacci tree, if there is a server S in the d-th group having load $\overline{T}' \geq \overline{T}+k_{\mathrm{Gr}}$ at the end of some round $\bar{\tau}$.

Let v be the root of a (T,i,d)-Fibonacci tree, let the time marker of v be τ, and assume that $S=\mathit{server}(v)$ is in the i-th group and has $\mathit{load}_\tau(\mathit{server}(v)) \geq T$. Let the bottom node v' of the irregular child edge of v have time marker τ and let it represent server S. The node v' is the root of a $(T-1,i,d)$-Fibonacci tree, and it represents a server of the i-th group with load at least $T-1$ at the end of the round indicated by its time marker. Let $J \in \mathcal{J}_\tau^{\delta,\Delta}$ be the last job allocated to $S=\mathit{server}(v)$ before or in round τ. Then let the regular child edge e of v represent job J, and let $\mathit{label}_e(v)=i$. Let the j-th bottom node v_j of e represent server $S^{(\mathit{label}_e(v_j))}(J)$. If $j<i$ then v_j is the root of a (T,j,d)-Fibonacci tree and the load of $\mathit{server}(v_j)$ at the end of round $\delta(J)-1$ is at least T. If $j \geq i$, the load of $\mathit{server}(v_j)$ is at least $T-1$ and v_j is the root of a $(T-1,j+1,d)$-Fibonacci tree. Applying this construction recursively yields a complete explaining representation on G_{Tree}.

We identify the upmost k_{Gr} nodes in the irregular component of the root of G_{Tree} and call the resulting tree $G_{\mathrm{Tree}}^{\mathrm{Left}}$. The representation defined above is called $\mathcal{R}_{\mathrm{Left}}$. $\mathcal{R}_{\mathrm{Left}}$ is a complete explaining representation on $G_{\mathrm{Tree}}^{\mathrm{Left}}$. It uses a $(N,\frac{N}{d},(\frac{N}{d})^{-1},\overline{M},1,1)$-vivid system. Let g be defined as in Theorem 5.1.4. If there is a server S in the d-th group with load $\overline{T}' \geq \overline{T}+k_{\mathrm{Gr}}+1+g$, each node of $G_{\mathrm{Tree}}^{\mathrm{Left}}$ fulfills

$$\mathit{load}_\tau(\mathit{server}(v)) \geq g, \quad \text{for } \tau = \min_{e \ni v}\{\delta(\mathit{job}(e))\}-1.$$

Thus according to Lemma 5.1.2 there is a $(N,\frac{N}{d},(\frac{N}{d})^{-1},\overline{M},1,\mu_\Xi)$-vivid system used by $\mathcal{R}_{\mathrm{Left}}$. To apply the Main Lemma, it remains to determine the size of the level 1 subtrees of $G_{\mathrm{Tree}}^{\mathrm{Left}}$.

If $s_d(T\cdot d+i)$ denotes the number of nodes in a (T,i,d)-Fibonacci tree, we have $s(j)=1$, for any $j \in \{1,\ldots,d\}$. For $j \geq d$, a $(\lfloor \frac{j}{d} \rfloor,(j \mod d),d)$-Fibonacci tree contains

$$s_d(j)=\sum_{l=j-d}^{j-1} s_d(l)$$

nodes. The sequence $(s_2)_{\mathbb{N}}$ is the well-known sequence of Fibonacci numbers. As in [Vöc99a] let $\phi_d = \lim_{k=\infty} \sqrt[k]{s_d(k)}$. Then ϕ_2 is the inverse value of the golden ratio, see e.g. [Knu98]. According to [Vöc99a] we further have $\phi_d > 2^{(d-1)/d}$, hence $d \cdot \ln \phi_d > (d-1) \cdot \ln 2$.

The size of a (T, i, d)-Fibonacci tree is at least $(\epsilon_\phi \cdot \phi_d)^{T \cdot d + i}$ for any $\epsilon_\phi < 1$ and $T \cdot d + i$ large enough. The smallest level 1 subtree of $G^{\text{Left}}_{\text{Tree}}$ a $(T - k_{\text{Gr}} - 1, 1, d)$-Fibonacci tree, has

$$(\epsilon_\phi \cdot \phi_d)^{(T - k_{\text{Gr}} - 1) \cdot d + 1}$$

nodes. Thus $G^{\text{Left}}_{\text{Tree}}$ is a $(1, k_{\text{Gr}} \cdot (d-1) + 1, (\epsilon_\phi \cdot \phi_d)^{(T - k_{\text{Gr}} - 1) \cdot d + 1}, c, d)$ witness forest. Summarizing we have

Theorem 5.1.8: Consider an allocation problem using separated i-th copy distributions. Assume that the preconditions of Theorem 5.1.4 are fulfilled, let $\epsilon_\phi < 1$, let N be large enough and let

$$\overline{T}' \geq \frac{1}{d \cdot \ln(\epsilon_\phi \cdot \phi_d)} \cdot \ln \log_{\epsilon_c} N + \left(\min\{5.275 + \tfrac{1}{d}, d\} + \mathrm{o}(1)\right) \cdot \frac{\overline{M}}{N} + k_{\text{Gr}} + 2 + \mathrm{o}(1)$$

Then for any round $\bar{\tau}$ the Always-go-left version of the greedy algorithm ensures that the system has load at most $\overline{T}'$ at the end of round τ, with probability at least $1 - N^{-\alpha}$.

Theorem 5.1.8 almost coincides with Theorem 3 in [Vöc99a].

5.2 The Parallel Setting

The parallel setting is more complicated than the sequential setting. Not only that we have to consider two different allocation algorithms, comparing performance also involves a new parameter, the number of rounds used to allocate the jobs. In the parallel setting a job agent can benefit from using multiple communication rounds, as they may help to determine the possible servers of other jobs. Using multiple communication rounds allows jobs to coordinate their decisions, which is impossible in the sequential setting. Besides the load of an allocation, the number of rounds used to determine the allocation is therefore an important measure in comparing the performance of parallel allocation algorithms. Our results show up a tradeoff between load and time: allowing more communication rounds allows better load.

5.2.1 The c-priority algorithm

The c-priority algorithm is introduced in [KLM92], it is the first algorithm presented in the literature for the parallel setting. We present it in Figure 5.1. The c-priority algorithm is outperformed by the c-collision algorithm, which achieves better load than

For all $J \in \mathcal{J}$ do in parallel
  $\text{agent}(J)$ becomes *active*.
$\tau := 0$

While there is at least one *active* job agent do (1)
  $\tau := \tau + 1$
  (This run trough the Loop (1) is called *round* τ.)
  For all $J \in \mathcal{J}$ do in parallel
    For $j \in \{1, 2, \ldots, d\}$ do (2)
      If $\text{agent}(J)$ is *active*,
        $\text{agent}(J)$ sends a (J, j, τ)-*request* to server agent $\text{agent}(S^{(j)}(J))$.
  For all $S \in \mathcal{S}$ do in parallel
    The agent(S) chooses the c $(\cdot, \cdot, \tau)$-*requests* with highest (3)
      priority and sends an (S)-*acknowledgment* to each of them
    (Theses requests are said to be *chosen for acceptance*.
      The other requests are said to be *rejected*.)
  For all $J \in \mathcal{J}$ do in parallel
    If for some $S \in \mathcal{S}$ the $\text{agent}(J)$ receives an (S)-*acknowledgment*, (4)
      $\text{agent}(J)$ becomes *inactive*,
      $\text{agent}(J)$ allocates J to server S

Fig. 5.1: The c-priority Algorithm

the c-priority algorithm using approximately the same running time. But in contrast to the c-collision algorithm the c-priority algorithm also suits for infinite settings.

We distinguish different versions of the c-priority algorithm by different *priority rules* used in step (3). Priority rules are given by orderings $\vdash_{S,\tau}$ on $\mathcal{J} \times \{1, \ldots, d\}$. In round τ server S prefers Request (J, j) to request (J', j') if $(J, j) \vdash_{S,\tau} (J', j')$. A special instance of the c-priority algorithm is the *c-arbitrary algorithm* which uses no specific priority rule at all. In the c-arbitrary algorithm the c requests chosen for acceptance in Round (3) are chosen arbitrarily by an adversary. Thus no priority rule for the c-priority algorithm performs worse than the c-arbitrary algorithm. If in step (4) of the c-collision algorithm a job agent receives acknowledgments from more than one server, a tie breaking mechanism is employed — just like for the greedy algorithm. As for the c-collision algorithm there are versions of the c-priority algorithm and the c-arbitrary algorithm which assume that the Loop (2) is the outmost loop (cf. to Figures 5.5 and 1.1). We call these versions *separated versions of the c-priority algorithm or c-arbitrary algorithm.*

The weighted counterpart of the c-priority algorithm is defined by choosing a slightly different description for step (3). In the weighted version step (3) is replaced by

> The agent(S) selects requests of the highest priority until their total weight is c_{w} at least, and sends a (S)-acknowledgment to each selected job.

Note that the weighted version of the c-priority algorithm is just a generalization of the ordinary one. If the weighted version acts on an allocation problem with unit weight jobs, it acts exactly like its ordinary counterpart.

In the case $c = 1$ the ordinary c-priority algorithm is akin to the greedy algorithm. For particular priority rules both algorithms even yield the same allocation, if a job is allocated to both servers in case of tie. For that assume that the priority rule is given by a single ordering $\vdash$ for each server S and each round τ. Then the allocation obtained by the 1-priority algorithm is the same as the one obtained by the greedy algorithm if $\delta(J) < \delta(J') \Leftrightarrow J \vdash J'$. For general priority rules and for larger values of c, however, there is no such tight relation between the greedy algorithm and the c-priority algorithm. We analyze the c-priority algorithm using the witness tree technique. When we establish the existence of the representation on the witness tree, we consider the weighted version of the greedy algorithm. The ordinary version is then just a special case of the weighted one.

5.2.1.1 The representation for the c-priority algorithm

Consider the weighted version of the c-priority algorithm working on an allocation problem. Assume that each priority rule $\vdash_{S,\tau}$ ensures that $(J, j) \vdash_{S,\tau} (J', j')$ whenever $\delta(J) < \delta(J')$, thus new jobs are never preferred to old ones. We call such a priority rule *time preserving*. Fix a round $\bar{\tau}$ and fix the possible servers of the jobs. A server is said to be τ'*-active in round* τ, if in round τ it rejects requests issued by jobs with entry time τ' or earlier. Let $\overline{T}' \in \mathbb{N}$. Assume that in round $\bar{\tau}$ there is a $(\bar{\tau} - \overline{T}')$-active server $\overline{S}$. Let $\overline{G}^{\mathrm{pr}}_{\mathrm{Tree}}$ be a tree of height $T \leq \overline{T}'$. Let each inner node of $\overline{G}^{\mathrm{pr}}_{\mathrm{Tree}}$ have one child edge of size 2 — called *irregular child edge* — and at least c_{w} child edges of size d — called *regular child edges*. We show how to obtain a c_{w}-fat complete explaining representation on $\overline{G}^{\mathrm{pr}}_{\mathrm{Tree}}$. Let the root v of $\overline{G}^{\mathrm{pr}}_{\mathrm{Tree}}$ represent server S and let the time marker of v be $\bar{\tau}$. As S is $(\bar{\tau} - \overline{T}')$-active, in round $\bar{\tau}$ server S rejects a job $\overline{J}$ with $\delta(J) \leq \bar{\tau} - \overline{T}'$.

Next consider an inner node v of $\overline{G}^{\mathrm{pr}}_{\mathrm{Tree}}$ and assume that v has time marker τ and represents some server S. Assume further that server S is $(\bar{\tau} - \overline{T}')$-active in round τ. We show how to define a representation for the regular child edges and the children of v. Server S accepts some requests $(J_1, j_1) < \cdots < (J_k, j_k)$ in round τ. The total weight of jobs $J_1, \ldots, J_k \in \mathcal{J}^{\delta,\Delta}_{\tau}$ is at least c_{w}, thus $k \geq c_{\mathrm{w}}$. Let v have k regular child edges $e_1, \ldots, e_k$, let edge e_i represent job J_i, and let $\mathit{label}_{e_i}(v) = j_i$, for each i. For each bottom node v' of each regular child edge e of v let $\mathit{server}(v') = S^{(\mathit{label}_e(v'))}(\mathrm{job}(e))$ and let the time marker of v' be $\tau - 1$. For the bottom node v'' of the irregular child of v let $\mathit{server}(v'') = S$ and let its time marker be $\tau - 1$. Now each child of v represents a server being $(\bar{\tau} - \overline{T}')$-active in round τ. Applying the construction recursively yields a c_{w}-fat complete explaining representation on $\overline{G}^{\mathrm{pr}}_{\mathrm{Tree}}$. The leaves of $\overline{G}^{\mathrm{pr}}_{\mathrm{Tree}}$ represent servers being $(\bar{\tau} - \overline{T}')$-active in round $\bar{\tau} - T$. If the considered allocation problem is ordinary, the inner nodes of $\overline{G}^{\mathrm{pr}}_{\mathrm{Tree}}$ have exactly c_{w} regular child edges, each. The constructed representation uses the natural vivid system for round $\bar{\tau}$.

Lemma 5.2.1: Let $c \leq c_{\mathrm{w}}$. Let $\bar{\tau}$ be a round, let $\overline{T} \leq \overline{T}'$, and let $e_{\mathrm{sub}}^{\mathrm{wpr}} = c \cdot \sum_{i=0}^{\overline{T}-2} [c(d-1)]^i$. Consider the c-priority algorithm using time preserving priority rules working on a weighted allocation problem. If at the end of round $\bar{\tau}$ the system contains m servers being $(\bar{\tau} - \overline{T}')$-active, then there is a c_{w}-fat complete explaining representation using the natural vivid system for round $\bar{\tau}$ on a weighted $(m, c(d-1), e_{\mathrm{sub}}^{\mathrm{wpr}}, c, d)$ witness forest. In round $\bar{\tau} - \overline{T}$ each server in the representation accepts a job generated in round $\bar{\tau} - \overline{T}'$ or earlier.

Proof: Let $\overline{G}_{\mathrm{Tree}}^{\mathrm{pr}}$ have height $\overline{T}$. According to the discussion above, there is a c_{w}-fat complete explaining representation on the tree $\overline{G}_{\mathrm{Tree}}^{\mathrm{pr}}$ if there is a $(\bar{\tau} - \overline{T})$-active server in round $\bar{\tau}$. However $\overline{G}_{\mathrm{Tree}}^{\mathrm{pr}}$ is not a weighted witness tree as it is not d-uniform. Let $G_{\mathrm{Tree}}^{\mathrm{wpr}}$ be the subtree of $\overline{G}_{\mathrm{Tree}}^{\mathrm{pr}}$ induced by the nodes of $\overline{G}_{\mathrm{Tree}}^{\mathrm{pr}}$ which are neither a bottom node of an irregular edge nor are successors of such a bottom node. As the irregular edges do not represent jobs, the representation is still c_{w}-fat and complete explaining on $G_{\mathrm{Tree}}^{\mathrm{wpr}}$. ■ of Lemma 5.2.1

Lemma 5.2.2: Let $k_{\mathrm{pr}} \in \mathbb{N}$ and $c = c_{\mathrm{w}}$. Let $\bar{\tau}$ be a round, $\overline{T} \leq \overline{T}' - k_{\mathrm{pr}}$, and let $e_{\mathrm{sub}}^{\mathrm{pr}} = c \cdot \sum_{i=0}^{\overline{T}-2} [c(d-1)+1]^i$. Consider the c-priority algorithm using time preserving priority rules working on an ordinary allocation problem. If at the end of round $\bar{\tau}$ there are m $(\bar{\tau} - \overline{T}')$-active servers, then there is a complete explaining representation using the natural vivid system for round $\bar{\tau}$ on a $(m, k_{\mathrm{pr}} \cdot c(d-1)+1, e_{\mathrm{sub}}^{\mathrm{pr}}, c, d)$ witness forest. In round $\bar{\tau} - (\overline{T} + k_{\mathrm{pr}})$ each server in the representation accepts a job generated in round $\bar{\tau} - \overline{T}'$ or earlier.

Proof: Let $\overline{G}_{\mathrm{Tree}}^{\mathrm{pr}}$ have height $\overline{T} + k_{\mathrm{pr}}$. We identify the upmost k_{pr} nodes in the irregular component of the root of $\overline{G}_{\mathrm{Tree}}^{\mathrm{pr}}$ and call the resulting tree G_{Tree}. The root of G_{Tree} has k_{pr} regular child edges, thus G_{Tree} is a $(k_{\mathrm{pr}} \cdot c(d-1)+1, e_{\mathrm{sub}}^{\mathrm{pr}}, c, d)$-witness tree. If the considered allocation problem is ordinary each inner node of $\overline{G}_{\mathrm{Tree}}^{\mathrm{pr}}$ has exactly $c = c_{\mathrm{w}}$ regular child edges, thus $\overline{G}_{\mathrm{Tree}}^{\mathrm{pr}}$ is uniquely determined by its height. ■ of Lemma 5.2.2

5.2.1.2 The finite setting

In the finite setting we have $\delta(J) = 1$ for all jobs $J \in \mathcal{J}$. This implies that any priority rule is time preserving.

Theorem 5.2.3: Let $\epsilon_c = (c!)^{\frac{1}{c}} \cdot 3^{d-1}$. Consider the c-priority algorithm working on an ordinary allocation problem in the finite setting with $M \in \mathbb{N}$ jobs. Assume that the i-th copy distributions of the allocation problem have stupidity σ_Ξ. Then after

$$\overline{T}' = \log_{[c(d-1)+1]} \log_{\epsilon_c} \frac{N}{m} + \frac{1}{c} \cdot (d + \mathrm{o}(1)) \cdot \frac{M}{N} \cdot \sigma_\Xi + k_{\mathrm{pr}} + 3 + \mathrm{o}(1)$$

rounds there are less than m 1-active servers, with probability at least $1 - N^{-\alpha}$, for $\alpha \leq \frac{1}{12} \cdot (k_{\mathrm{pr}} \cdot c(d-1)+1)$, $k_{\mathrm{pr}} \cdot c(d-1) \geq 2$, and $\frac{N}{m}$ large enough.

Proof: The theorem is an immediate consequence of a more general lemma.

Lemma 5.2.4: Let τ be a round. Consider the c-priority algorithm with time preserving priority rules working on an ordinary allocation problem in the infinite setting. Then for $\overline{T} = \log_{c\lfloor d-1\rfloor+1} \log_{\epsilon_c} \frac{N}{m} + 2 + \mathrm{o}(1)$, any $T_{\mathrm{L}} \geq T_{\mathrm{S}} \geq \overline{T} + k_{\mathrm{pr}} + (d + \mathrm{o}(1)) \cdot \frac{\overline{M}'}{c \cdot N} \cdot \sigma_{\Xi} + \mathrm{O}(1)$ and $\overline{M}' = (T_{\mathrm{L}} - T_{\mathrm{S}}) \cdot \vec{M}$ it holds

> Assume that at the end of round $\tau - T_{\mathrm{S}}$ the system does not contain any job J with $\delta(J) \leq \tau - T_{\mathrm{L}}$. Then at the end of round τ there are less than m servers being $(\tau - T_{\mathrm{S}})$-active.

with probability at least $1 - N^{-\alpha}$, for $\alpha \leq \frac{1}{12} \cdot (k_{\mathrm{pr}} \cdot c(d-1) + 1)$

Proof: According to Lemma 5.2.2 there is a complete explaining representation $\mathcal{R}_{\mathrm{pr}}$ on $G^{\mathrm{pr}}_{\mathrm{Forest}}$, if there are m servers being $(\tau - T_{\mathrm{S}})$-active at the end of some round τ. The representation uses the natural vivid system for round τ.

The assumption that the system contains no job J with $\delta(J) \leq \tau - T_{\mathrm{L}}$ in round $\tau - T_{\mathrm{S}}$, ensures that all jobs appearing in the representation $\mathcal{R}_{\mathrm{pr}}$ are generated between round $\tau - T_{\mathrm{L}}$ and round $\tau - T_{\mathrm{S}}$. We call such jobs *well-timed*. There are at most $(T_{\mathrm{L}} - T_{\mathrm{S}}) \cdot \vec{M}$ well-timed jobs, thus the vivid sets of the edges can be reduced to size at most $\overline{M}' = (T_{\mathrm{L}} - T_{\mathrm{S}}) \cdot \vec{M}$. Using this bound on the size of the vivid sets of the edges, we apply a similar argument as the one applied in Lemma 5.1.2.

For $\mu = (3 \cdot (N \cdot p_{\Xi}) \cdot (\overline{M}' \cdot p_{\Xi} \cdot d'_{\Xi}))$ we show how to obtain a $(N, N'_{\Xi}, p_{\Xi}, \overline{M}', d'_{\Xi}, \mu)$-vivid system used by $\mathcal{R}_{\mathrm{pr}}$. According to Lemma 5.2.2 in round $\tau - (\overline{T} + k_{\mathrm{pr}})$ each server S occuring in the representation accepts a job J. Job J waits for $T_{\mathrm{S}} - (\overline{T} + k_{\mathrm{pr}})$ rounds to be accepted by S, thus S accepts $c \cdot (T_{\mathrm{S}} - (\overline{T} + k_{\mathrm{pr}}))$ jobs before J is accepted. All these requests are accepted before round $T_{\mathrm{S}} - (\overline{T} + k_{\mathrm{pr}})$, thus S gets at least $c \cdot (T_{\mathrm{S}} - (\overline{T} + k_{\mathrm{pr}}))$ requests besides the ones occuring in the representation. For each node v let $\mathrm{vivide}_{\mathrm{pr}}(v)$ be the event "*server*(v) receives at least $c \cdot (T_{\mathrm{S}} - (\overline{T} + k_{\mathrm{pr}}))$ requests issued by well-timed jobs not appearing in $\mathcal{R}_{\mathrm{pr}}$". This definition assures that the vivid event for a node does not depend stochastically on the fact that $\mathcal{R}_{\mathrm{pr}}$ is explaining.

We have $T_{\mathrm{S}} - (\overline{T} + k_{\mathrm{pr}}) = (d + \mathrm{o}(1)) \cdot \frac{\overline{M}'}{c \cdot N} \cdot \sigma_{\Xi}$. Consider a node v. The probability that server $S = \mathit{server}(v)$ receives more than

$$\overline{M}' \cdot p_{\Xi} \cdot d'_{\Xi} + 3 \cdot \sqrt{\overline{M}' \cdot p_{\Xi} \cdot d'_{\Xi}} \cdot \ln(3 \cdot N \cdot p_{\Xi} \cdot \overline{M}' \cdot p_{\Xi} \cdot d'_{\Xi})$$

requests from well-timed jobs not occuring in the representation is at most

$$\mathrm{e}^{-\ln(3 \cdot N \cdot p_{\Xi} \cdot \overline{M}' \cdot p_{\Xi} \cdot d'_{\Xi})} = 3 \cdot N \cdot p_{\Xi} \cdot \overline{M}' \cdot p_{\Xi} \cdot d'_{\Xi} = \mu,$$

according to a Chernoff Bound from Lemma 2.2.1. Thus $\mathrm{Prob}[\mathrm{vivide}_{\mathrm{pr}}(v)] \leq \mu$. As a request sent to a server $S' \neq S$ cannot be sent to S, the event $\mathrm{vivide}(v)$ is not stochastically independent from

$$\mathcal{E} = \bigwedge_{\substack{w \in V_{\mathrm{Server}} \\ \mathit{server}(w) \neq \mathit{server}(v)}} \mathit{vivide}_{\mathrm{pr}}(w).$$

But for the same reason, this dependency only decreases the probability for vivide(v) being fulfilled, thus

$$\begin{aligned}\text{Prob}[\text{vivide}_{\text{pr}}(v)] &\geq \text{Prob}[\text{vivide}_{\text{pr}}(v) \,|\, \mathcal{E}] \\ &= \text{Prob}[\text{vivide}_{\text{pr}}(v) \,|\, \text{“}\mathcal{R}_{\text{pr}} \text{ is explaining ”} \wedge \mathcal{E}].\end{aligned}$$

The trees in the witness forest have height $\overline{T} = \log_{[c(d-1)+1]} \log_{\epsilon_c} \frac{N}{m} + 2 + \text{o}(1)$. Thus the preconditions on the Main Lemma (Lemma 3.2.6) are met, which yields the desired result.

The event “at the end of round $\tau - T_{\text{S}}$ the system does not contain any job J with $\delta(J) \leq \tau - T_{\text{L}}$” is stochastically independent of the possible servers of any job with entry time later than $\tau - T'$, and vice versa, as the priority rule is time preserving. ■ of Lemma 5.2.4

Theorem 5.2.3 is an immediate consequence of the lemma for $\tau = \overline{T}'$, $T_{\text{S}} = \tau - 1$, and $T_{\text{L}} = \tau$. ■ of Theorem 5.2.3

Theorem 5.2.3 only considers the number of rounds used to compute an allocation. The load of the allocation is not considered. The c-priority algorithm shows however a tight relation between the number of rounds used to compute an allocation and its load. As in each round at most c jobs are allocated to a particular server, the load of an allocation is at most c times the number of rounds used to compute it. Thus under the preconditions of Theorem 5.2.3, the load is at most

$$\overline{T}' = \tfrac{c}{\ln(c(d-1)+1)} \cdot \ln \log_{\epsilon_c} N + (d + \text{o}(1)) \cdot \frac{M}{N} \cdot \sigma_{\Xi} + c \cdot (k_{\text{pr}} + 3 + \text{o}(1)).$$

It is easy to see, that while the number of rounds used to compute an allocation decreases with increasing c, the load increases. Very large values of c allow to obtain an allocation very fast. If $M = N$ and $\sigma_{\Xi} = 1$, choosing $c = \sqrt[T]{\frac{\log N}{\log \log N}}$ yields an allocation in $T + \text{O}(1)$ rounds, with high probability. The allocation has load $(T + \text{O}(1)) \cdot \sqrt[T]{\frac{\log N}{\log \log N}}$. For constant values of T, this behavior matches the lower bounds for the parallel setting of the allocation problem. It would be easy to convert the result for the finite setting to infinite settings using oblivious deletion schemes as in Section 5.1.

Compared with the previously known results found in [KLM92] and [ACMR95], our result is much broader. The result on the c-priority algorithm from [ACMR95] is restricted to uniform i-th copy distributions and $M = N$, but does not gain more accuracy from this restriction. In this special case covered by the result from [ACMR95], our result almost differs only in the additive part. The c-priority algorithm considered by Karp et al. is the separated c-priority algorithm mentioned on Page 5.2.1. It assumes that in each round τ each server agent accepts up to c $(\cdot, j, \tau)$-requests, for *each* $j \in \{1, \ldots, d\}$. Hence up to $d \cdot c$ requests are accepted by a particular server in each round.

We call this version of the c-priority algorithm the *separated version of the c-priority algorithm*. To compare our result with theirs, consider the c-arbitrary algorithm working on an allocation problem with $d \cdot N$ servers using the separated i-th copy distributions. If after the allocation algorithm terminated the servers S_i with $i \mod N = k$ are identified for each k, the resulting allocation is at most worse than the one obtained by applying the separated version of the c-priority algorithm to the corresponding allocation problem on N servers.

Theorem 5.2.5: Let $\epsilon_c = (c!)^{\frac{1}{c}} \cdot 3^{d-1} \cdot (1+\sqrt{3})^{-1}$, assume that c_{w} fulfills $c_{\mathrm{w}} \geq 5 \cdot c + 1$, and $c_{\mathrm{w}} \geq -\frac{25}{8} \cdot \ln(1 - \epsilon_c^{-\frac{1}{11}}) + 1$. Consider the c_{w}-priority algorithm with time preserving priority rules working on a weighted allocation problem in the finite setting with jobs of weight M. Assume that the i-th copy distributions of the allocation problem have stupidity σ_Ξ. Then after

$$\overline{T}' = \log_{[c(d-1)]} \log_{\epsilon_c} \frac{N}{m} + \frac{1}{c} \cdot (d + \mathrm{o}(1)) \cdot \frac{M}{N} \cdot \sigma_\Xi + 3 + \mathrm{o}(1)$$

rounds all but less than m jobs are allocated to a server, with probability at least $1 - N^{-\alpha}$, for

$$\alpha \leq \frac{1}{132} \cdot c(d-1) \quad \text{and}$$
$$c(d-1) \geq 3.$$

Proof: The proof for the weighted case is almost the same as the one for the ordinary case. We use a weighted version of Lemma 5.2.4 presented in Lemma 5.2.6. The lemma is proven using the generalized Chernoff Bound from Lemma 2.2.2 and the Main Lemma for weighted jobs (Lemma 4.3.3). ■ of Theorem 5.2.5

Lemma 5.2.6: Let τ be a round. Consider the c-priority algorithm with time preserving priority rules working on a weighted allocation problem in the infinite setting. Then using, ϵ_c from the theorem above, for $\overline{T} = \log_{c[d-1]} \log_{\epsilon_c} N + 2 + \mathrm{o}(1)$, any $T_{\mathrm{L}} \geq T_{\mathrm{S}} \geq \overline{T} + k_{\mathrm{pr}} + (d + \mathrm{o}(1)) \cdot \frac{\overline{M}'}{N} \cdot \sigma_\Xi$ and $\overline{M}' = (T_{\mathrm{L}} - T_{\mathrm{S}}) \cdot \vec{M}$ it holds

> Assume that at the end of round $\tau - T_{\mathrm{S}}$ the system does not contain any job J with $\delta(J) \leq \tau - T_{\mathrm{L}}$. Then at the end of round τ there are less than m servers being $(\tau - T_{\mathrm{S}})$-active.

with probability at least $1 - N^{-\alpha}$, for $\alpha \leq \frac{1}{132} \cdot c(d-1)$.

Theorem 5.2.5 is the first result for the performance of the weighted version of the c-priority algorithm in the finite setting. Most things mentioned in appreciation of the result for the ordinary case also hold for the weighted setting. In particular the correlation between the amount of communication and the obtained load is — up to a constant factor — the same. Compared with its ordinary counterpart Theorem 5.2.5 looses up to a factor of $\frac{\ln(c(d-1)+1)}{\ln(c_{\mathrm{w}}(d-1)+1)}$, corresponding to reduction of the degree of the witness trees inner nodes, which drops from at least $c_{\mathrm{w}} + 1$ to $c + 1$ when the weighted witness forest is converted to an ordinary one.

5.2.1.3 The infinite setting

The c-priority algorithm in the finite setting gained much attraction from PRAM-Simulations. Other applications, such as the VoD example are better modeled using an infinite setting. The oblivious deletion schemes as considered in Section 5.1 do, however, not cover real world situations very well. Oblivious deletion schemes would correspond to situations where server speed *increases* with the load of a server — a fairly unrealistic assumption. A better model should assume that server speed is independent from the number of jobs allocated to a server. This requires to consider a non-oblivious deletion scheme.

We assume that in each round $\vec{M}$ jobs enter the system, and each server can delete up to c jobs allocated to it. We call this deletion scheme the *server oriented deletion scheme*. Using this deletion scheme the number of jobs in the system is not bounded, the same holds for the number of rounds a job stays in the system. On the other hand this deletion scheme is more realistic than the oblivious deletion schemes considered in the sequential setting. The server oriented deletion scheme assumes that each server is able to perform c tasks per round — a far more realistic scenario than any oblivious deletion scheme. Moreover oblivious deletion schemes do not make any sense at all if the decision where to allocate a job can be deferred. In this case a job may be deleted from the system before it is allocated to a server.

In [ABS98] we present a modified witness tree argument to deal with the possibly unbounded number of jobs in the system. In this thesis we present a new approach which allows to employ our Main Lemma (Lemma 3.2.6) and further allows a generalization to the weighted setting. Our aim is to prove the following result.

Theorem 5.2.7: Let $\bar{\tau}$ be a round. Let $\epsilon_c = (c!)^{\frac{1}{c}} \cdot 3^{d-1}$, and let $0 < \alpha \leq \frac{1}{12} \cdot (k_{\text{pr}} \cdot c[d-1]+1)$. Further let $\eta > 1$. Consider the c-priority algorithm with time preserving priority rules working on an ordinary allocation problem in the infinite setting using the server oriented deletion scheme. Then at the end of round $\bar{\tau}$ there is no job waiting more than

$$\eta \cdot (\log_{c[d-1]+1} \log_{\epsilon_c} N + k_{\text{pr}})$$

rounds for being allocated to a server, with probability at least $1 - N^{-\alpha+o(1)}$, if

$$\vec{M} \leq \min\left\{(3 \cdot \sqrt{\alpha}+1)^{-1},\ 1 - \tfrac{1}{\eta} - o(1)\right\} \cdot [(d+o(1)) \cdot \sigma_{\Xi}]^{-1} \cdot c \cdot N.$$

Proof: We prove Theorem 5.2.7 by induction using Lemma 5.2.4. To avoid repetition we state the weighted version of the theorem without proof afterwards. The proof for the weighted case is almost the same as the one for the ordinary case.

Let $\overline{T}$ and $\overline{M}'$ be defined as in Lemma 5.2.4. Further let

$$\begin{aligned} T_0 &= 0 \\ T_1 &= \eta \cdot (\overline{T} + k_{\text{pr}}) \\ T_{i+1} &= T_i + 1 \quad \text{for } i \geq 1. \end{aligned}$$

Now assume that for some $i \geq 0$ at the end of round $\bar{\tau} - \sum_{j=0}^{i+1} T_j$ there are no jobs J with $\delta(J) \leq \bar{\tau} - \sum_{j=0}^{i+2} T_j$. Then for

$$\vec{M} \leq c \cdot N \cdot \left[(d + \mathrm{o}(1)) \cdot \sigma_{\Xi}\right]^{-1}$$

Lemma 5.2.4 assures that there are no $(\bar{\tau} - \sum_{j=0}^{i+1} T_j)$-active jobs at the end of round $\bar{\tau} - \sum_{j=0}^{i} T_j$.

To check the preconditions of Lemma 5.2.4, let

$$\begin{aligned} \tau &= \bar{\tau} - \sum_{j=0}^{i} T_j \\ T_{\mathrm{S}} &= T_{i+1} \\ T_{\mathrm{L}} &= T_{i+2} + T_{i+1} \end{aligned}$$

It remains to check that

$$T_{\mathrm{S}} \geq (\overline{T} + k_{\mathrm{pr}}) + (d + \mathrm{o}(1)) \cdot \frac{\overline{M}'}{c \cdot N} \cdot \sigma_{\Xi} + \mathrm{O}(1).$$

The right side is equal to

$$= (\overline{T} + k_{\mathrm{pr}}) + (d + \mathrm{o}(1)) \cdot \frac{\vec{M}}{c \cdot N} \cdot (T_{\mathrm{L}} - T_{\mathrm{S}}) \cdot \sigma_{\Xi} + \mathrm{O}(1),$$

as $T_{\mathrm{L}} - T_{\mathrm{S}} = T_{\mathrm{S}} + 1$

$$\begin{aligned} &= (\overline{T} + k_{\mathrm{pr}}) + (d + \mathrm{o}(1)) \cdot \frac{\vec{M}}{c \cdot N} \cdot \sigma_{\Xi} \cdot (T_{\mathrm{S}} + 1) + \mathrm{O}(1) \\ &= (T_{\mathrm{S}} + 1) \cdot \Big[\frac{(\overline{T} + k_{\mathrm{pr}}) + \mathrm{O}(1)}{T_{\mathrm{S}} + 1} + (d + \mathrm{o}(1)) \cdot \frac{\vec{M}}{c \cdot N} \cdot \sigma_{\Xi}\Big]. \end{aligned}$$

Thus the condition is fulfilled, if

$$\frac{T_{\mathrm{S}}}{T_{\mathrm{S}} + 1} = 1 - \mathrm{o}(1) \geq (\eta + \mathrm{o}(1)) + (d + \mathrm{o}(1)) \cdot \sigma_{\Xi} \cdot \frac{\vec{M}}{c \cdot N}$$

The latter condition is fulfilled according to the condition on $\vec{M}$. Thus the preconditions of Lemma 5.2.4 are fulfilled.

To obtain the theorem it remains to show that there is an i such that the preconditions of Lemma 5.2.4 are fulfilled, i.e. to show that there is an i such that at the end of round $\bar{\tau} - \sum_{j=0}^{i+1} T_j$ there is no job with entry time $\sum_{j=0}^{i+2} T_j$ or earlier.

Lemma 5.2.8: Assume that $\alpha \geq 1$. Let $\vec{M} \leq \frac{c \cdot N}{\sigma_\Xi \cdot d} \cdot (3 \cdot \sqrt{\alpha} + 1 + o(1))^{-1}$. Consider the c-priority algorithm working on a weighted allocation problem in the infinite setting. Let τ be a round. At the end of round τ the system contains no job J with $\delta(J) \leq \tau - \ln N$, with probability at least $1 - N^{-\alpha}$.

Proof: We say that a round τ' is *S-good*, if in round τ' server S accepts all requests it receives that round. To prove the proposition of the lemma, we show that for each server S, there is an S-good round after round $\tau - \ln N$ and before round τ, with probability at least $1 - N^{-\alpha}$.

Let S be a server and assume that our assertion is wrong, i.e. assume that $\tau_0 < \tau - \ln N$ is the last S-good round before round τ. Between rounds τ_0 and round τ server S receives requests with weight at least $c \cdot (\tau - \tau_0)$. The expected weight of requests received by S is at most $(\tau - \tau_0) \cdot \vec{M} \cdot p_\Xi \cdot d'_\Xi = (\tau - \tau_0) \cdot (\sqrt{\alpha} + 1 + o(1))^{-1}$. Thus using the generalized Chernoff Bound from Lemma 2.2.2 the probability that S receives requests with weight at least $c \cdot (\tau - \tau_0)$ is at most

$$\begin{aligned} e^{-(\alpha + o(1)) \cdot (\tau - \tau_0)} &\leq e^{-(\alpha + o(1)) \cdot \ln N} \cdot e^{-(\tau - \tau_0 - \ln N)} \\ &\leq \tfrac{e-1}{e} \cdot N^{-\alpha - o(1)} \cdot e^{-(\tau - \tau_0 - \ln N)}. \end{aligned}$$

Summing up over all possible values of τ_0 shows that the probability that S has no S-good round between $\tau - \ln N$ and τ is at most

$$\tfrac{e-1}{e} \cdot N^{-\alpha} \cdot \sum_{j=0}^{\infty} e^{-j} \leq N^{-\alpha}.$$

■ of Lemma 5.2.8

The proof of Lemma 5.2.8 makes use of the possible servers of jobs with entry time after $\tau - \ln N$. The real event "at the end of round τ the system contains no job with entry time before $\tau - \ln N$" does not depend on the possible servers of jobs with entry time later than $\tau - \ln N$. This is due to the fact that the priority rules are time preserving. The assertion of Lemma 5.2.8 is therefore stochastically independent of the possible servers of jobs entering the system after round $\tau - \ln N$. ■ of Theorem 5.2.7

Theorem 5.2.9: Let $c(d-1) \geq 3$. Let $\epsilon_c = (c!)^{\frac{1}{c}} \cdot 3^{d-1} \cdot (1 + \sqrt{3})^{-1}$, $c_w \geq 5 \cdot c + 1$, $c_w \geq -\frac{25}{8} \cdot \ln(1 - \epsilon_c^{-\frac{1}{11}})$, $\eta > 1$, and let $0 < \alpha \leq \frac{1}{132} \cdot (c[d-1])$. Consider the weighted version of the c-priority algorithm with time preserving priority rules working on a weighted allocation problem in the infinite setting using the server oriented deletion scheme and time preserving priority rules. Then at the end of round $\bar{\tau}$ there is no job waiting more than

$$\eta \cdot (\log_{c[d-1]} \log_{\epsilon_c} N)$$

rounds for being allocated to a server, with probability at least $1 - N^{-\alpha + o(1)}$, if

$$\vec{M} \leq \min\left\{(3 \cdot \sqrt{\alpha} + 1)^{-1}, 1 - \tfrac{1}{\eta} - o(1)\right\} \cdot [(d + o(1)) \cdot \sigma_\Xi]^{-1} \cdot c_w \cdot N.$$

For all $J \in \mathcal{J}$ do in parallel
  $\mathrm{agent}(J)$ becomes *active*.
  For $j \in \{1, 2, \ldots, d\}$ do
    $\mathrm{agent}(J)$ sends a $(J, j, W(J))$-*request* to server agent $\mathrm{agent}(S^{(j)}(J))$.

— Ensure that each request has been received —

For all $S \in \mathcal{S}$ do in parallel (1)
  Let $\#(S)$ be the weight of the requests received by S
  While $\#(S) > 0$ do
    If $\#(S) \leq c$,
      then $\mathrm{agent}(S)$ sends a (S)-acknowledgment to each
        active job agent which sent a request to S.
    For each $W(J)$-*delete* received by S let $\#(S) := \#(S) - W(J)$.

For all $J \in \mathcal{J}$ do in parallel (2)
  While $\mathrm{agent}(J)$ is *active* do
    If for some $S \in \mathcal{S}$ the $\mathrm{agent}(J)$ receives a (S)-*acknowledgment*,
      For $j \in \{1, 2, \ldots, d\}$ do
        Send a $W(J)$-*delete* to $\mathrm{agent}(S^{(j)}(J))$

Fig. 5.2: The asynchronous version of the c-collision algorithm

5.2.2 The c-collision algorithm revisited

The c-collision algorithm, introduced in [DM93], offers the best performance of all known allocation algorithms in the finite parallel setting. It allows to obtain almost optimal load using $\mathrm{O}(\log \log N)$ rounds. The c-collision algorithm is presented in Figure 1.1 on Page 22. The major drawback of the c-collision algorithm is that it does not suit for infinite settings. Actually an infinite version of the c-collision algorithm would not gain any further improvement over the c-priority algorithm. The key point in the infinite parallel setting is the number of rounds used to compute an allocation. According to this measure the c-priority algorithm even outperforms the c-collision algorithm a little bit.

The performance of the c-collision algorithm in the ordinary and the weighted setting is bounded in Section 3.3.1 and in Section 4.4. These bounds are, however, restricted to uniform i-th copy distributions. We give a generalization to non-uniform i-th copy distributions in the present section. Moreover we consider two alternative formulations of the c-collision algorithm overcoming some drawbacks of the original.

5.2.2.1 The asynchronous version of the c-collision algorithm

It is easy to see that the respective runs through the "while" loop (1) of the c-collision algorithm as depicted in Figure 1.1 on Page 22 have to proceed in a strictly synchronous

manner. The analysis of the c-collision algorithm assumes that all requests of all jobs are received before step (3) is performed. Otherwise a server agent may wrongly accept the received requests.

This very strong assumption on the execution of the algorithm is avoided by the asynchronous version of the c-collision algorithm as presented in Figure 5.2. A first asynchronous version of the c-collision algorithm was presented by Stemann in [Ste96]. The version presented here is a generalization of his algorithm to the weighted case. If one assumes that Loops (1) and (2) are performed synchronous, the allocation obtained by the asynchronous version of the c-collision algorithm is the same as the one obtained by the basic c-collision algorithm. A special discussion of the asynchronous versions performance is therefore superfluous.

5.2.2.2 The adaptive collision algorithm

To motivate the adaptive collision algorithm consider the c-collision algorithm allocating M jobs with unit weights to N servers. Let $c = (1+\varepsilon) \cdot d\frac{M}{N}$ for some $\varepsilon > 0$. In the c-collision algorithm in each round each job sends a request to its possible servers, and a server S accepts the request of a job if and only if S gets at most c requests. A job which is not accepted keeps sending requests until it is accepted. Obviously, c has to be chosen properly in order to make the algorithm work, and proper choice of c requires knowledge about the number M of jobs. In many applications this number is unknown. To avoid this problem, assume that the algorithm is executed for all possible values of c simultaneously, i.e. in each round every job sends a request including the values of c for which it is already accepted and each server answers a request with the values of c for which it accepts its requests. This allows to obtain a good allocation by simply choosing the allocation corresponding to the smallest value of c for which the algorithm succeeds.

But as there are infinitely many possible values of c, this approach would require an infinite amount of communication. To avoid this, our algorithm operates slightly different. Each job J keeps track of the smallest value of c for which it has been accepted in the last round. We call this value $c(J)$. At the beginning of each round each job J sends a request including $c(J)$ and its own weight $W(J)$ to each of its possible servers. Next every server S computes the value $c(S)$ as the minimum possible value such that every job sending a request to S is accepted at S or by another server. For sake of that S assumes that every other server S' chooses the same value $c(S')$ (while other servers may choose and assume different values). Afterwards S answers each request with $c(S)$ if S assumes the job to be accepted at S. The other requests do not get a reply. A formal definition of the algorithm is given in Figure 5.2. It remains to state how a server S computes the value $c(S)$ in Step (2) of some round τ. Let $J_1, \ldots, J_k$ be the jobs sending a request to S, in an order such that the sequence $c(J_1), \ldots, c(J_2)$ is decreasing. In case of a tie assume that jobs with $S_\tau(J) = S$ are preferred. All remaining ties are broken with respect to the ordering $<$ on $\mathcal{J}$. We say that job J has *rank i at S in step τ* if $J = J_i$. Now let $c(S) = \sum_{i=0}^{l} W(J_i)$ where l is the smallest number such that

$W(J)$	$c(J)$	rank	accepted	reply $= c(S)$
2	10	1	yes	7
2	9	2	yes	7
1	7	3	yes	7
2	6	4	no	no reply
1	2	5	no	no reply

Fig. 5.3: The server chooses $c(S) = 6$. The last column contains the values replied to the jobs. (For simplicity we use weights from $\mathbb{N}$ rather than real values from $[0, 1]$ as assumed in the definition of the allocation problem).

$\sum_{i=0}^{l} W(J_i) \geq c(J_{i+1})$. The jobs $J_1, \ldots, J_l$ are *accepted by* S. Figure 5.3 contains an example.

We call the adaptive collision algorithm a version of the c-collision algorithm as both algorithms are tightly related. If the c-collision algorithm and the adaptive collision algorithm work on the same allocation problem with the same possible servers for the jobs and the c-collision algorithm is terminated after $\tilde{\tau}$ rounds, then the adaptive collision algorithm obtains load at most c. The two different algorithms address the tradeoff between the number of rounds used and the load of the obtained allocation from two different sides: the c-collision algorithm targets a fixed load and uses as many rounds as it takes to obtain that load, while the adaptive collision algorithm targets a fixed number of rounds accepting whatever load it achieves.

Lemma 5.2.10: Assume that the adaptive collision algorithm and the c-collision algorithm work on the same allocation problem with the same possible servers for the jobs. Assume further that all jobs are inactive after $\bar{\tau}$ rounds of the c-collision algorithm. Then the load obtained by the adaptive collision algorithm with $\tilde{\tau} = \bar{\tau}$ rounds is at most c.

Proof: We first prove the following claim.

In each round each job gets at least one reply.

We show by induction that in round τ each job J receives a reply from server $S_{\tau-1}(J)$ and that the value $c(J)$ does not increase during round τ. In the first round each server S accepts all jobs sending a request to S. For the induction step assume that the proposition is true for all rounds $1 \leq \tau' < \tau$ and all jobs. Let $S = S_{\tau-1}(J)$. Then J receives a reply from S in round $\tau - 1$. The rank of J at S in step τ is not smaller than the rank of J at S in step $\tau - 1$ as the value $c(J')$ does not increase for any job J' during round $\tau - 1$. Thus if S accepts J in round $\tau - 1$, it also accepts J in round τ. This also assures that the value $c(J)$ does not increase during round τ.

For all $J \in \mathcal{J}$ do in parallel
  $S_0(J) := S^{(1)}(J)$
  $c(J) = \infty$
  agent(J) becomes *active*.

For τ from 1 to $\tilde{\tau}$ do (1)
  For all $J \in \mathcal{J}$ do in parallel
    For all $j \in \{1, \ldots, d\}$ do
      send a $(J, j, W(J), c(J), S_\tau(J))$-request to $S^{(j)}(J)$
  For all $S \in \mathcal{S}$ do in parallel
    compute $c(S)$ (2)
    send a $(c(S), \tau)$-reply to each accepted job
  For all $J \in \mathcal{J}$ do
    Let $c(J)$ be the minimum value, such that J received a $(c(J), \tau)$-reply
    Let $S_\tau(J)$ be a server sending a $(c(J), \tau)$-reply

For all $J \in \mathcal{J}$ do
  Allocate job J to server $S_{\tilde{\tau}}(J)$

Fig. 5.4: The adaptive collision algorithm

The weight of the jobs receiving a reply from S is at most $c(S)$. If $c'(S)$ is the value of $c(S)$ at the end of round $\tilde{\tau}$, then $load(S) \leq c'(S)$.

It now remains to prove that at the end of round $\tilde{\tau}$ each job has $c(J) \leq c$, if the preconditions of the Lemma are fulfilled. The proof is done by induction on the number of rounds. We show that a job J being inactive at the end of round τ of the c-collision algorithm has $c(J) \leq c$ after round τ of the adaptive collision algorithm.

$\tau = 1$: No job is inactive at the beginning of round 1 of the c-collision algorithm.

$\tau - 1 \to \tau$:
Consider a job J being inactive at the beginning of round 1 of the c-collision algorithm. If the job becomes inactive in round $\tau' < \tau - 1$, then $c(J) \leq c$ at the end of round τ' according to the induction hypothesis and $c(J) \leq c$ at the beginning of round τ as the $c(J)$ do not increase.

Thus, assume that J becomes inactive in round $t - 1$. W.l.o.g. we assume that J gets an accept message from $S = S^{(1)}(J)$ in the c-collision algorithm. Then S accepts all its requests in round $\tau - 1$. Thus, the weight of the requests sent to S during round $\tau - 1$ of the c-collision algorithm is at most c. Due to the induction hypothesis the weight of jobs J sending a request with $c(J) > c$ is at most c. Thus $c(J) \leq c(S) \leq c$ at the end of round τ.

■ of Lemma 5.2.10

5.2.2.3 The c-collision algorithm and its variants on arbitrary i-th copy distributions

For all previous algorithms, we have considered their performance on allocation problems with arbitrary i-th copy distributions. In each case we have seen that choice of the i-th copy distributions only affects the additive constant of the performance bound. This is different for the c-collision algorithm. The choice of the i-th copy distributions can directly affect the possibilities for distribution of the jobs. For instance consider i-th copy distributions which distribute all copies uniformly on $\frac{N}{a}$ servers. These i-th copy distributions have stupidity a and a lower bound for the load of any allocation is $a \cdot \frac{M}{N}$. Thus the choice of the i-th copy distributions affects the possibilities to choose the parameter c, which strongly affects the performance of the c-collision algorithm. Our upper bound suffers a little bit more than necessary from poor choice of the i-th copy distributions, as it involves an extra factor $(N'_\Xi \cdot p_\Xi)^{d-1}$. This factor is only due to simplifications made in Lemma 3.4.4.

Theorem 5.2.11: Consider an ordinary allocation problem using i-th copy distributions with stupidity σ_Ξ. Let $c \in \mathbb{N}$, such that

$$b = (c+1)(d-1) \geq 3, \quad (\epsilon_c)^c = \frac{c!}{(\frac{M}{N} \cdot d \cdot \sigma_\Xi \cdot (N' \cdot p_\Xi)^{d-1})^c} > 1, \quad \text{and } \alpha \leq \frac{1}{12} \cdot b.$$

Let further $m \in \mathbb{N}$ and

$$T \geq \log_{c(d-1)} \log_{\epsilon_c} \left(\frac{N}{m}\right) + 2 + \mathrm{o}(1).$$

Then if N/m is large enough, there are less than m servers whose agent is active at the end of round T of the c-collision algorithm, with probability at least $1 - N^{-\alpha}$. In particular no server agent is active after

$$\log_{c(d-1)} \log_{\epsilon_c} (N) + 2 + \mathrm{o}(1)$$

rounds, with high probability, if N is large enough.

Proof: As the representation constructed in Section 3.1.1 uses the natural vivid system, it also uses the Ξ-vivid system. The theorem is then a simple consequence of the Main Lemma (Lemma 3.2.6). ■ of Theorem 5.2.11

To demonstrate the accuracy of our results it is worth to compare the minimum number of copies d and the minimum value of c required for the analysis to work. The strongest result with respect to this measure is the result from [MSS95, MSS96]. Unfortunately this result deals with a slightly different version of the c-collision algorithm. They deal with the separated version of the c-collision algorithm given in Figure 5.5. The main difference between the c-collision algorithm and its separated version is that

For all $J \in \mathcal{J}$ do in parallel
  $\text{agent}(J)$ becomes *active*.
$\tau := 0$

While there is at least one *active* job agent do (1)
  $\tau := \tau + 1$
  For $j \in \{1, 2, \ldots, d\}$ do (2)
    For all $J \in \mathcal{J}$ do in parallel
      If $\text{agent}(J)$ is *active*,
        $\text{agent}(J)$ sends a (J, j, τ)-*request* to server agent $\text{agent}(S^{(j)}(J))$.
    For all $S \in \mathcal{S}$ do in parallel
      If $\text{agent}(S)$ receives at most c $(\cdot, j, \tau)$-*requests*, (3)
        $\text{agent}(S)$ sends a (S)-*acknowledgment* to each job agent
          which sent a $(\cdot, j, \tau)$-request to S.
      (In this case $\text{agent}(S)$ is said to *accept its* j-*requests*.)
    For all $J \in \mathcal{J}$ do in parallel
      If for some $S \in \mathcal{S}$ the $\text{agent}(J)$ receives a (S)-*acknowledgment*, (4)
        $\text{agent}(J)$ becomes *inactive*,
        $\text{agent}(J)$ allocates J to server S.

Fig. 5.5: The separated version of the c-collision Algorithm

the Loop (2) becomes the outmost loop of a round. As a consequence, a $(\cdot, j, \cdot)$-request cannot get in conflict with a $(\cdot, j', \cdot)$-request, if $j \neq j'$. For each $j \in \{1, \ldots, d\}$ a server can j-accept its requests, thus the load of the separated version of the c-collision algorithm is at most $c \cdot d$. The result from [MSS95, MSS96] considers the separated c-collision algorithm on an allocation problem with $N \cdot d$ servers and separated i-th copy distributions.

A performance bound for the separated version of the c-collision algorithm is easily obtained using our techniques. The only thing to do is to set up a representation on the witness forest $G^{\text{c-coll}}_{\text{Forest}}$. For sake of this discussion we simply note that the performance bound from Corollary 3.3.1 also holds for the separated c-collision algorithm. The result from [MSS95, MSS96] deals with an allocation problem using $N \cdot d$ servers and uniform i-th copy distributions.

The first difference between the result from [MSS96] and ours is that our conditions on c and d are weaker than theirs. While for $c = 1$ it suffices to choose $d = 3$ in our case, their result requires $d = 4$. For larger values of c choosing $d \geq 2$ suffices in both cases. To compare the bounds on the minimum possible value of c, consider an allocation problem with $N \cdot d$ servers. Our bound allows to choose $c = 1$ as long as $\frac{M}{N} \leq (1 - \varepsilon)$ for an arbitrary $\varepsilon > 0$. The bound from [MSS95, MSS96] requires $\frac{M}{N} < \frac{1}{2}$ in the same setting. Moreover their result is restricted to $\frac{M}{N} \leq 1$.

6 An Application of the Balls-Into-Bins Paradigm to Routing

This chapter goes beyond the capabilities of the Main Lemma. It deals with an application of the balls-into-bins paradigm to circuit switching on multistage networks related to the popular butterfly network. We devise algorithms that route messages by constructing circuits (or paths) for the messages with small congestion, dilation, and setup time.

6.1 Introduction

Underlying every parallel computer is a network that delivers messages between processors or between processors and memory modules. Similar networks are found in the switches that route telephone calls and internet traffic. Typically, a message is sent from its input node (source) to its output node (destination) via a path in the network. Methods for routing messages include circuit-switching, store-and-forward routing, and wormhole routing. With circuit switching, each message must first lock down (i.e., reserve) a path (i.e., circuit) in the network from its input node to its output node. The path is then used to transmit the message through the network. This differs from store-and-forward routing and worm-hole routing where paths are not reserved before transmission.

Circuit-switching has enjoyed widespread popularity since its early use in telephony and subsequently in the design of parallel computers. Recent trends in network design emphasize the need for providing quality of service (QoS) guarantees for communication. To provide guarantees as opposed to just best-effort service, network resources must be reserved before communication begins. Consequently, several modern high-speed multimedia switches and ATMs reserve a (virtual) circuit through the network for each communication request [RCM94, TY97].

6.1.1 Circuit routing algorithms and their performance

In a circuit-switched network, a message arrives requesting a path from its source to its destination. A *routing algorithm* determines which of many possible paths is locked down for each message. We measure the performance of a routing algorithm in terms of three parameters: congestion, dilation, and setup time.

Congestion and dilation are properties of the paths locked down for the messages by the routing algorithm. The *congestion* of a set of paths is defined to be the maximum number of paths that pass through any link in the network. Congestion is a measure of the maximum number of paths that must be simultaneously supported by a link of the network, and hence determines the bandwidth that a link should possess. The *dilation* of a set of paths is defined to be the maximum length of a path in the set. Dilation is a measure of maximum distance (in links) that a message must travel to reach its destination. Finally, the *setup time* is the time taken by the routing algorithm to allocate paths through the network. This is the time overhead involved in path selection before the actual message transmissions begin. The goal of this section is to devise routing algorithms with small congestion, dilation, and setup time.

6.1.2 Network and problem definitions

The results in this section apply to variants of a popular type of multi-stage interconnection network called the *butterfly network*. Butterfly networks and its variants have been widely used for packet routing in a number of commercial and experimental networks [Got87, Nak91, PBG+87]. More recently, several proposed designs for the switching fabric of scalable high-speed ATM networks use the butterfly and its variants for routing virtual circuits [RCM94, TY97].

We define an N-input *butterfly network* B_N as follows. An N-input butterfly has $N(\log N + 1)$ nodes arranged in $\log N + 1$ *levels* of N nodes each.[1] An example of an N-input butterfly ($N = 8$) with depth $\log N$ ($\log N = 3$) is shown in Figure 6.1. Each node has a distinct label $\langle w, i\rangle$ where i is the level of the node ($0 \leq i \leq \log N$) and $w = w_1 w_2 \ldots w_{\log N}$ is a $\log N$-bit binary number that denotes the *row* of the node. All nodes of the form $\langle w, i\rangle$, $0 \leq i \leq \log N$, are said to belong to row w. Two nodes $\langle w, i\rangle$ and $\langle w', i'\rangle$ are linked by an edge if $i' = i + 1$ and either w and w' are identical or w and w' differ only in the bit in position i'. (The bit positions are numbered 1 through $\log N$.) We call the first type of edge a *straight edge* and the second a *cross edge*. The nodes on level 0 are called the *inputs* of the network, and the nodes on level $\log N$ are called the *outputs*. Sometimes the level 0 node in each row is identified with the level $\log N$ node in the same row. In this case, the butterfly B_N is said to *wrap around*.

We define a *randomly-wired butterfly* RB_N as follows. Network RB_N has the same set of nodes and edges as B_N, except that the cross edges incident on the input nodes of RB_N are permuted randomly according to the following rule. Let $D = \log N$ Each node $\langle w_1 \ldots w_D, 0\rangle$ of RB_N is connected by a cross edge to node $\langle w'_1 \ldots w'_D, 1\rangle$ if and only if $w_1 \neq w'_1$ and $\sigma_{w_1}(w_2 \ldots w_D) = w'_2 \ldots w'_D$, where σ_0 and σ_1 are random permutations of the set of $(\log N - 1)$-bit numbers.

We define a *two-fold butterfly* BB_N as follows. Network BB_N consists of two copies of B_N placed one after the other such that each output node in the first copy is identified with the corresponding input node of the second copy with the same row number. Note

[1] Throughout this section we use $\log N$ to denote $\log_2 N$.

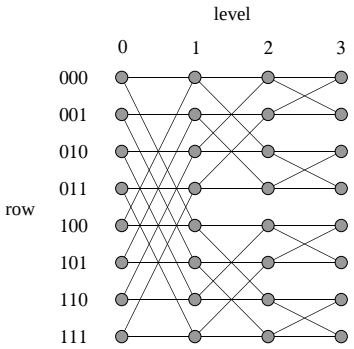

Fig. 6.1: An 8-input butterfly network.

that BB_N is a multistage network with N rows and $2\log N + 1$ levels. The nodes in level 0 are called the inputs of BB_N and the nodes in level $2\log N$ are called the outputs of BB_N. Also, observe that a routing algorithm on BB_N can be simulated by making two passes through a butterfly B_N that wraps around.

It is important to contrast the BB_N network with another common variant of the butterfly, the *Beneš network*. An N-node Beneš network consists of two copies of B_N placed "back-to-back" such that each output node of the first copy is identified with the corresponding output node of the second copy.

In this section, we study a canonical circuit routing problem that is known as the permutation routing problem. In a *permutation routing problem* at most one message originates at each input of the network and at most one message is destined for each output of the network.

We distinguish two kinds of permutation routing problems: static and dynamic. In a *static problem*, all the messages that constitute a permutation routing problem are present at time 0, before the routing begins. The routing algorithm constructs paths for all the messages in a "batch" mode. All the messages are delivered to their respective destinations before the routing of the next batch of messages begins. In contrast, in a *dynamic problem*, messages are injected or deleted one by one. The routing algorithm routes a path for each injected message in an on-line fashion with no knowledge of future message arrivals. We assume that at any time, the messages being routed form a partial permutation; that is, each input and output node correspond to at most one routed message.

6.1.3 Previous work

There are several different sub-areas of research that relate to our work. We provide a summary of the most relevant.

Routing in Butterfly Networks. There is a vast literature on routing in butterfly networks [Lei92, Lei92b]. Much of the early work focuses on store-and-forward routing [LMRR94, Pip84, Ran91, Val82, Val81].

More recently, there has been progress in analyzing wormhole routing algorithms [CMS96, CMSV96, FRU92, RSW94]. Since we present no new results in these two routing methods, we focus only on the butterfly circuit-switching literature.

In two early papers, Beizer [Bei62] and Beneš [Ben64] showed that any static permutation routing problem can be routed with congestion 1 and dilation $2\log N$ on an N-input Beneš network. Subsequently, Waksman [Wak68] provided an elegant algorithm that takes $O(N \log N)$ time to determine all the paths, but requires global knowledge of the source and destination of all the messages. Later, Nassimi and Sahni [NS82] showed how to implement Waksman's algorithm in parallel on the Beneš and related networks in time $O(\log^4 N)$. However, their algorithm is complex and requires the Beneš network to emulate a complete network by executing a series of sorting routines.

Although the Beneš network and the BB_N are closely related in structure, it is a long-standing open problem whether or not it is possible to route an arbitrary permutation routing problem in an offline fashion with congestion 1 on the BB_N.

In this section, we devise routing algorithms that *minimize congestion*. A complementary approach aims to maximize throughput. Previous work has studied the model where each link can support at most q paths, and the goal is to maximize the number of messages that lock down paths. Kruskal and Snir [KS83] showed that if each input in a butterfly network B_N sends a message to a randomly chosen output, and at most one message can use any edge of the network (i.e., $q = 1$), then the expected number of messages that succeed in locking down paths to their destinations is $\Theta(N/\log N)$. Koch [Koc88] generalized the result of Kruskal and Snir by showing that if each edge can support q messages, $q \geq 1$, then the expected fraction of messages that succeed in locking down paths is $\Theta(N/\log^{1/q} N)$. Maggs and Sitaraman [MS92] generalized the previous two results by showing that, by making two passes through a butterfly, it is possible to route an $\Omega(N/\log^{1/q} N)$ fraction of any permutation (rather than only a random permutation), with high probability.

Use of Randomness. An early example of the use of randomization for circuit-switching in butterfly networks is the work of Valiant [Val82, Val81]. Valiant showed that any permutation routing problem can be transformed into two random problems by first routing a path for each message to a random intermediate destination, and then on to its true destination. This implies that we can route paths for a (static or dynamic) permutation routing problem on a two-fold butterfly BB_N with congestion $\Theta(\log N/\log\log N)$, and dilation $2\log N$. Note that the paths for each message can be set up independently without complete knowledge of the permutation in $O(\log N)$ time. We show how to use randomization to route permutations with substantially smaller congestion and the same dilation.

Ranade [Ran87] observed that a smaller amount of randomness is sufficient to implement Valiant's algorithm. Note that each switch has two input links and two output links. Ranade noted that it is sufficient that each switch in the first $\log N$ levels of BB_N shunts a message from each input link to a random (and distinct) outgoing link. Thus, messages are sent to random but not independent destinations using one random bit per switch. The first $\log N$ levels of such a BB_N constitute a *flip network*. A flip network was subsequently used in [MS92] in the context of circuit routing. We use flip networks in our routing algorithms in Section 6.2.

Randomness can be used in constructing the network itself. The use of randomness to design multistage networks dates back to Bassalygo and Pinsker [Bas74]. Networks such as the randomly-wired multibutterfly are known to have good routing and fault tolerance properties [Upf92, LM92]. In [MSS98] a balls-into-bins approach is used to devise a multistage network of optical crossbars. The construction allows networks of constant depth on which the time to route a permutation is $O(\log\log N)$. Recent results provide algorithms for routing circuits for any permutation routing problem with congestion 1 in multibutterfly and multi-Beneš networks with set-up time $O(\log N)$ [ALM96, Pip96]. Unlike these networks, our results in Section 6.2 apply to commonly-used networks like B_N and BB_N that require neither random wiring nor expanders.

Circuit routing in general topology networks. Dynamic circuit-switching has been extensively studied in an on-line competitive framework for arbitrary network topologies. (See [Plo95] for a survey). Results are known for minimizing congestion [AAF+94] and for the maximizing throughput [GG92]. This framework can incorporate more general parameters such as the circuit bandwidth and circuit holding time. However, these results do not yield routing algorithms with congestion smaller than $\Theta(\log N)$ for the regularly-structured muti-stage networks that are the focus of this chapter.

6.1.4 Our results

We introduce two new algorithms for circuit-routing: the *network c-collision algorithm* and the *network greedy algorithm*. Both algorithms are simple adaptions of their counterparts for allocation problems as presented in Figure 1.1 and Figure 1.2. While the network c-collision algorithm allocates a job to a server S if the number of unallocated jobs sending requests to S is at most c, the network c-collision algorithm chooses a path if the congestion along this path is at most c. The network greedy algorithm allocates a request to the path with the smaller congestion just like the greedy algorithm allocates a job to the server with smaller load.

Static Permutation Routing. In Section 6.2.1, we show the network c-collision algorithm routes any permutation on the two-fold butterfly BB_N with congestion $O(\log\log N/\log\log\log N)$, with high probability, and dilation $2\log N$. The setup time is $O(\log N\log\log N/\log\log\log N)$. Our routing algorithm achieves a substantially smaller

congestion bound than Valiant's algorithm. Comparing our result with Waksman's algorithm, which achieves congestion 1 on a Beneš network, we require substantially smaller setup time. Further, we do not require complete knowledge about the permutation being routed and our routing algorithm can be implemented on the network itself. Comparing our result to the algorithm of Nassimi and Sahni [NS82] for the Beneš network, our algorithm is much simpler and faster, although their algorithm achieves smaller congestion.

Dynamic Permutation Routing. In Section 6.2.2, we analyze the minimum algorithm for routing any dynamic permutation routing problem on network BB_N. The congestion is $O(\log\log N)$ with high probability, the dilation is $2\log N$, and the setup time for each new message is $O(\log N)$. Prior to this work, every known algorithm for the dynamic permutation routing problem on the butterfly and related networks required $\Omega(\log N/\log\log N)$ congestion. Our algorithm is optimal in that any routing algorithm on BB_N that considers only a constant number of alternate paths per message must incur $\Omega(\log\log N)$ congestion [ABKU94].

6.2 Routing in the Two-Fold Butterfly

6.2.1 Routing in BB_N in a finite parallel setting

We describe a simple, efficient off-line algorithm for routing permutations on the two-fold butterfly BB_N. Recall that the two-fold butterfly BB_N has N inputs at level 0 and N outputs at level $2D$, where $D = \log N$. Given a permutation π, our routing algorithm connects each input node i to the corresponding output node $\pi(i)$; each pair $(i, \pi(i))$ of input and output nodes is called a *request*. Our randomized algorithm routes paths such that the maximum congestion on an edge is $\Theta(\log\log N/\log\log\log N)$, with high probability. Further, the time required by the algorithm to set up all the paths is at most $\Theta(\log N\log\log N/\log\log\log N)$, with high probability.

The network c-collision algorithm. We utilize the network c-collision algorithm described below to perform the routing. The network c-collision algorithm initially chooses at random two possible paths for each request. These possible paths correspond to the possible servers of a job in the network c-collision algorithm on an allocation problem with number of copies $d = 2$.

The possible paths for each request are chosen as follows. The switches on the node levels $0, \ldots, D/2-1$ and $D+D/2+1, \ldots, 2D$ are flipped randomly. In particular, each input and output node maps the *first possible path* P of a request to its straight edge and its *second possible path* P' to its cross edge with probability $\frac{1}{2}$,and with probability $\frac{1}{2}$ the order is reversed. Similarly, each switch on the node levels $1, \ldots, D/2-1$ and $D+D/2+1, \ldots, 2D-1$ with probability $\frac{1}{2}$ connects its input straight edge with its output straight edge and its input cross edge with its output cross edge, and with probability $\frac{1}{2}$

the connections are reversed. Note that the random choices for the switches completely determine the possible paths P and P' of each request, because there is exactly one path connecting a node on level $D/2$ with a node on level $D + D/2$ in a BB_N network. For a path P, the other path P' connecting the same input and output nodes is called the *buddy* of P. The random switching ensures that any edge on the levels $1, \ldots, D/2$ and $D + D/2 + 1, \ldots, 2D$ is traversed by at most one of the randomly generated paths. However, each edge on the interior levels, i.e., on the levels $D/2 + 1, \ldots, D + D/2$, is potentially traversed by several of these paths. We call these edges *collision edges*, and we say that two paths that cross the same collision edge *collide*.

The network c-collision algorithm proceeds in rounds to select a path for each request as follows. Initially all paths are *active* and not *selected*. A path P is selected if for each edge $e \in P$ the number of active paths traversing e is at most c. If P and its buddy P' are both eligible to be selected, one is chosen arbitrarily. A path P ceases to be active in a round if P is selected or the buddy of P is selected in that round. The algorithm terminates when there are no more active paths.

Each round of the network c-collision algorithm can be implemented using a store-and-forward algorithm as a subroutine: in a first pass, for each active path, a packet is sent along the path from level 0 to level $2D$. During this pass, for each edge, the number of packets traversing the edge is counted. Then, in a second pass, all packets are routed backward along their respective paths from level $2D$ to level 0. During this pass the congestion for each active path is computed. Note that, in this model, when computing the setup time the packets and edges of the network can act in parallel, and hence a round may complete in $o(N)$ time.

The network c-collision algorithm selects a path P in a round only if P collides with no more than $c - 1$ other active paths on any of the edges in P. This implies that any edge that is included in at least one selected path is included in at most $c - 1$ other selected or active paths. As a consequence, the congestion of all selected paths is at most c. Note that the algorithm as described is not guaranteed to terminate. However, in Theorem 6.2.1, we show that if c is sufficiently large, the algorithm will terminate on its own with maximum congestion at most c after a small number of rounds with high probability. In practice, we may terminate the algorithm after some fixed number of rounds; all requests that still have two active paths at the termination point may choose one randomly.

Theorem 6.2.1: For any constant $\epsilon > 0$ and c such that $c! = (1 + \epsilon) \cdot \log N$, the probability that the network c-collision algorithm on BB_N takes more than $t = \Theta(\log \log N / \log \log \log N)$ rounds to select a path for every request is at most $N^{-c/4+1+o(1)}$. Further, each round can be computed in time $O(\log N)$, with high probability.

Proof: First, we show that if the algorithm does not terminate after t rounds, there exists a "delay tree". This delay tree corresponds to what we called "a witness tree with an explaining representation" when considering allocation problems. Next,

we show how the delay tree can be pruned to avoid stochastic dependencies. This pruning corresponds to the conversion of a complete explaining representation into a valid representation as conducted in Lemma 3.4.2. Finally, we show by enumeration that the probability of occurrence of a pruned delay tree is at most $N^{-c/4+1+o(1)}$. This step corresponds to Lemma 3.4.4.

Constructing a delay tree. Fix a permutation π to be routed, and fix the settings of the randomly flipped switches on the levels $0, \ldots, D/2-1$ and $D+D/2+1, \ldots, 2D$. This determines the two paths chosen for each request. Assume that there is a request with possible paths P and P', and neither path has been selected by round t, where the proper value of t is to be determined later. Then P collides with at least c paths of other requests in round t at some edge e. Let $P_1, \ldots, P_c$ denote the c paths that collide with P in round t at e. The root of the delay tree is the request corresponding to P and the requests corresponding to $P_1, \ldots, P_c$ are its children. Now $P_1, \ldots, P_c$ and their buddies $P'_1, \ldots, P'_c$ were not selected by round $t-1$. Applying the argument recursively to $P'_1, \ldots, P'_c$ we can construct a complete c-ary tree of height t. This tree is called the *delay tree*.

Each node v in the delay tree corresponds to a request corresponding to two paths, one of which collides with the siblings and the parent of v (unless v is the root), and the other of which collides with the children of v (unless v is a leaf). We call the first path the *up path* of v and the other path the *down path* of v. The up path of the root and the down paths of the leaves are defined to be empty paths. Note that by the term "collision represented by node v" we mean the collision of the down path of v with the up paths of the children of v in the delay tree. Finally, to give each tree a unique representation, we assume that the IDs of the input nodes of the requests associated to the children of a node are increasing from left to right.

The requests corresponding to the nodes of a delay tree are not necessarily pairwise distinct. Further, the up and down paths of distinct requests may overlap in the randomly flipped levels, so that a randomly flipped switch can be included in more than one of these paths. Hence, the collision events represented by a delay tree are not necessarily stochastically independent. Note that, if they were stochastically independent, it would be relatively straightforward to argue the theorem.

Pruning the delay tree. The intuitive reason why the dependencies do not affect the final conclusion is that there are only $O(\log N)$ nodes in the delay tree, hence the dependencies are "rare". In order to handle dependencies, we prune nodes from the delay tree as necessary. This pruning is done by a traversal through the tree visiting the internal nodes in breath-first-search order starting at the root. When a node v is visited during this traversal, the dependencies between the collision represented by v and the collisions represented by nodes visited before v are checked. If the dependencies significantly affect our calculations, the subtree rooted at v is pruned, and these pruned nodes are excluded from the subsequent traversal.

The detailed pruning rules follow. For a node v visited during the traversal, let $B(v)$ denote the set of nodes visited before v. Further, let $\Sigma(v)$ denote the set of nodes that are children of the nodes in $B(v)$, that are not pruned before v is visited, and that are not in $B(v)$ themselves. For the root *root* of the delay tree, $B(root)$ and $\Sigma(root)$ are empty since our traversal starts at *root*. The following pruning rule ensures that, for every node v visited after the root *root*, the subgraph induced by $B(v) \cup \Sigma(v)$ is connected; that is, $B(v) \cup \Sigma(v)$ induces a subtree of the full delay tree with root r. When a node v is visited, up to $2c$ subtrees of maximum height $t-2$ could be pruned from the tree. These subtrees do not include any node from $B(v) \cup \Sigma(v)$. Hence, the subtree induced by this set is non-decreasing during the traversal. We distinguish two pruning rules:

1. If a path associated with one of v's non-pruned children traverses a randomly flipped switch that is also traversed by a path associated with a node u from $\Sigma(v)$ then the c subtrees rooted at the children of v are removed from the tree, and the c subtrees rooted at the children of u are also removed from the tree. The node v is called a *pruning node*. The node u that caused the pruning is called the *conflicting node* of v.

2. If a path associated with one of v's non-pruned children traverses a randomly flipped switch that is also traversed by a path associated with a node u from $B(v)$ then the c subtrees rooted at the children of v are removed from the tree. The nodes v and u are again called pruning and conflicting nodes respectively.

When there is more than one choice for a conflicting node for a certain pruning node we make the choice arbitrarily, so that each pruning node can be associated with exactly one conflicting node. Further, the second pruning rule is considered only if the conditions for the first pruning rule are not met. We continue the pruning process till either there are no more nodes to visit or there are $\kappa = \lceil c/2 \rceil$ pruning nodes. In the latter case, we apply a final pruning. If v is the κth pruning node, we remove from the tree all nodes not included in $B(v) \cup \Sigma(v)$. This effectively stops the pruning process at the κth pruning node.

The delay tree pruned in this fashion is called the *pruned delay tree*. Let m denote the number of internal nodes in this tree, and $m' \leq \kappa$ denote the number of pruning nodes. Let $v_1, \ldots, v_m$ denote the internal nodes and $w_1, \ldots, w_{m'}$ the pruning nodes in order of visitation, respectively. Further, let u_i denote the conflicting node of w_i, for $1 \leq i \leq m'$. The pruned tree possesses the following properties.

1) Any internal node v represents a collision of the down path of v and the c up paths of the children of v.

2) For any internal node v, the pruning ensures that the up paths of the children of v do not share a randomly flipped switch with a path associated to a node in $B(v) \cup \Sigma(v)$ except for the down path of v. (As a consequence, all nodes of the tree correspond to distinct requests.)

3) The down path of a pruning node v either collides with a path P that is associated to the conflicting node u, or it collides with a path P such that P or its buddy shares a random switch with a path associated to u. This path P is denoted the *conflicting path* of v.

4) The down path of a pruning node w_i is not the conflicting path of a pruning node w_k with $k < i$. (This can be proved as follows. For contradiction, assume the opposite. Then $w_i = u_k$ and $w_i \in \Sigma(w_k)$. Hence, the subtree below w_i is removed when w_k is visited. This means that w_i has no non-pruned children when w_i is visited and consequently, w_i is not a pruning node.)

5) For each pruning node w_i, the down path P of w_i shares at most $5c$ randomly flipped switches with up and down paths associated with any other node and conflicting paths associated with the pruning nodes $w_1, \ldots, w_i$. (This is because, according to Properties 2 and 4, the down path of w_i is not equivalent to any such up, down, or conflicting path. Further, according to Property 2, the down path of w_i does not share a random switch with any other up or down path, except for the up and down paths of the siblings of w_i, and the up path of w_i. With each of these $2c-1$ paths, the down paths overlaps at most twice in the randomization levels, once in each of the butterflies in BB_N. The same holds for the i conflicting paths associated with $w_1, \ldots, w_i$. Thus, there are at most $4c-2+2\kappa \leq 5c$ overlappings with these paths in the randomization levels.)

Bounding the probability of occurrence of a pruned delay tree. We bound the probability of occurrence of a pruned delay tree via enumeration. Define the *tree shape* to be a description of the topology of a delay tree including the pruning and the conflicting nodes. Define an *admissible delay tree configuration* to be a tree shape with associated requests, up and down paths, and conflicting paths which eventually, i.e., for some setting of the random switching, matches to a pruned delay tree. In particular, any admissible delay tree configuration has to fulfill the 5 properties above.

Let $\mathcal{Q}$ denote the set of tree shapes corresponding to at least one admissible delay tree configuration, and let $\mathcal{K}_Q$ denote the set of all admissible delay tree configurations with tree shape $Q \in \mathcal{Q}$. An admissible configuration K is said to be *active* if the outcome of the random switching corresponds to all paths of the configuration. Hence, each admissible configuration K has a probability to become active, which is just $2^{-\rho(K)}$ with $\rho(K)$ denoting the total number of randomly flipped switches covered by all paths of K. As a consequence, the probability that the network c-collision algorithm takes more than t rounds can be bounded by

$$\sum_{Q \in \mathcal{Q}} \underbrace{\sum_{K \in \mathcal{K}_Q} 2^{-\rho(K)}}_{=: \mathrm{E}(Q).}$$

We aim to give an upper bound on $\mathrm{E}(Q)$, for a fixed tree shape $Q \in \mathcal{Q}$. $\mathrm{E}(Q)$ is equal to the *expected number of active delay tree configurations* with tree shape Q. Note that the tree shape Q only restricts the number of admissible configurations, that is, it defines the set $\mathcal{K}_Q$, but does not influence the probability for a given configuration $K \in \mathcal{K}_Q$ to become active. This probability depends only on $\rho(K)$ and hence on the overlapping of the paths in the randomization levels.

In the following, we utilize Properties 2 and 5 that govern how paths may overlap to compute $\mathrm{E}(Q)$. Instead of summing over all admissible configurations in $\mathcal{K}_Q$ and multiplying each individual configuration with its probability, we consider the nodes of the delay tree one by one and calculate an upper bound on the expected number of configurations for each individual node. In particular, we consider first all the internal tree nodes and then all the collision nodes; both sets of nodes are considered in the order of visitation.

Define the *configuration of an internal node* v_i to consist of the down path of v_i and the up paths of the children of v_i, for $1 \leq i \leq m$. Further, define the *configuration of a pruning node* w_i to be the down path of w_i and the two paths belonging to the colliding request of w_i, for $1 \leq i \leq m'$. A collection of node configurations is said to be admissible, if they are a subset of an admissible tree configuration. Note that a collection of admissible configurations for all internal and all pruning nodes (in conjunction with the tree shape) completely defines the configuration of the delay tree.

For an internal node v_i and a collection K of configurations for the nodes $v_1, \ldots, v_{i-1}$, let $\mathrm{E}_{\mathrm{coll}}(v_i, K)$ denote the expected number of active configurations for v_i under the assumption that the configurations in K are active. Note that K already specifies the request associated to v_i. (For the root v_1 we assume that K specifies only this request.) Let $\mathrm{E}_{\mathrm{coll}}(v_i)$ be the maximum over all configurations K of $\mathrm{E}_{\mathrm{coll}}(v_i, K)$.

Lemma 6.2.2: $\mathrm{E}_{\mathrm{coll}}(v_i) \leq \log N/c!$.

Proof: We bound the expected number of active configurations for v_i by choosing the down path P of v_i arbitrarily and then deriving an upper bound on the expected number of choices of active up paths $P_1, \ldots, P_c$ of the children of v_i that fulfill Properties 1 and 2.

The expected number of active down paths P is at most one. This is because, there are several different paths in BB_N that connect the two input and output nodes which are given by the configuration K. However, at most two of them are active, and the configuration K determines which of them is the up path and which is the down path of v_i.

Given path P, there are $D = \log N$ possible choices for the collision edge at which the down path collides with $P_1, \ldots, P_c$. Let e denote this edge and ℓ the level of this edge. W.l.o.g., we assume that $D/2 + 1 \leq \ell \leq D$.

We calculate an upper bound on the expected number of active up paths $P_1, \ldots, P_c$ traversing e and fulfilling Property 2. This Property ensures that $P_1, \ldots, P_c$ use only unrevealed random switches. Therefore, we assume for the following that all switches are

unrevealed. Note that this does not decrease the number of admissible configurations, and, hence, not decrease the expected number of active configurations for $P_1, \ldots, P_c$. The main problem in calculating the number of active configurations for $P_1, \ldots, P_c$ is to handle overlappings among these paths and overlapping between these paths and the down path P in the randomization levels.

The number of nodes on level 0 from which e can be reached is $2^{\ell-1}$. We select an input node for each of the P_i's from these nodes. The number of possible ways to choose these c nodes is $\binom{2^{\ell-1}}{c}$ because the requests associated to the children of a node are ordered according to the ID's of the input nodes. Let $s_1, \ldots, s_c$ denote the source nodes of the paths $P_1, \ldots, P_c$ on level 0 and $D_1 = \pi(s_1), \ldots, D_c = \pi(s_c)$ the destination nodes of these paths on level $2D$.

Next we choose an intermediate destination D_i' for each path P_i on node level $D+\ell$. For every P_i, there are (eventually) several possibilities to choose these intermediate destination. However, independent from the other paths of the configuration of v_i, the number of active destinations is at most one. Hence, the expected number of active intermediate destinations is at most one.

Now assume the intermediate destinations are fixed. Note that this also fixes the path from level $D + \ell$ to level $2D$. It remains to consider the number of active configurations of c paths $P_1', \ldots, P_k'$ such that P_i' connects s_i and D_i' and traverses e. Paths $P_1', \ldots, P_k'$ and P do not overlap in the randomization levels. This can be shown as follows. If two paths share a random switch s then these paths arrive and leave s on different edges. Further, these paths do not overlap at any other switch with distance less than $D + 1$ from s. Hence, two paths that traverse edge e cannot have used a random switch with distance less than $D + 1$ from the two switches incident to e, and consequently, they cannot meet on a random switch on the levels $0, \ldots, D/2 - 1$ or the levels $D + D/2 + 1, \ldots, D + \ell$.

The number of different paths connecting s_i with D_i' and traversing e is one. Thus, the number of admissible configuration for the P_i''s is at most one. All paths in the admissible configuration do not share a randomly flipped switch with another path from the configuration of v_i. Hence, the number of unrevealed random switches traversed by each of these paths is $D/2 + (D + \ell) - (D + D/2) = \ell$. Except for the switch on level 0, all of these switches must correspond to the course of the respective path. The probability for this event is $2^{-(\ell-1)}$. As a consequence, the probability that all k paths are active is at most $2^{-c\cdot(\ell-1)}$.

Putting it all together, the expected number of active configurations for v_i is

$$D \cdot \binom{2^{\ell-1}}{c} \cdot 2^{-c\cdot(\ell-1)} \leq \frac{D}{c!}.$$

■ of Lemma 6.2.2

Now we give an upper bound on the expected number of the active configurations for the pruning nodes. For a pruning node w_i and a collection K of configurations

for all internal nodes and the pruning nodes $w_1, \ldots, w_{i-1}$, let $\mathrm{E}_{\mathrm{prune}}(w_i, K)$ denote the expected number of active configurations for w_i under the assumption that all configurations in K are active. Let $\mathrm{E}_{\mathrm{prune}}(w_i)$ be the maximum over all configurations K of $\mathrm{E}_{\mathrm{prune}}(w_i, K)$.

Lemma 6.2.3: $\mathrm{E}_{\mathrm{prune}}(w_i) \leq 2^{5c+3} \cdot (\log N + 1)/\sqrt{N}$.

Proof: The conflicting path P of pruning node w_i is either associated with the conflicting node u_i or P or its buddy shares a randomly flipped switch with a path associated to u_i. The tree shape specifies u_i, and the configuration K fixes the request associated to u_i. For any consistent setting of the random switches, the number of paths sharing a randomly flipped switch with the two paths belonging to this request is at most $2 \cdot (\log N + 1)$ (inclusive the two paths themselves). Consequently, for any setting of the switches, the number of candidates for the collision request is at most $2 \cdot (\log N + 1)$, and hence the number of candidates for the collision path is at most $4 \cdot (\log N + 1)$.

Now suppose the collision path is fixed. The down path of w_i collides with this path. First, we assume that the collision is in level ℓ, with $D/2 + 1 \leq \ell \leq D$. Let e denote the respective collision edge. There is at most one admissible course for the down path of w_i from its source node on level 0, which is determined by K, to the collision edge e.

The course of the down path from level 0 to level ℓ is determined by the randomly flipped switches. Property 5 ensures that at most $5c$ of the switches traversed by the down path are shared with other paths in K. Hence, at least $\ell - 5c$ of the randomly flipped switches determining the course of the path from level 0 to level ℓ are independent of K, and consequently, the probability that the down path of w_i is equivalent to the only admissible path in these levels is $2^{-\ell+5c}$. Summing over all collision levels ℓ, with $D/2+1 \leq \ell \leq D$, yields an upper bound on the probability that the switches along the collision path are set appropriately of $2^{-D/2+5c}$. Since the same bound holds also for collisions when $D+1 \leq \ell \leq 3D/2$, the probability that the down path is equivalent to the only admissible path is at most $2^{-D/2+5c+1}$. As a consequence, the expected number of active configurations for w_i is at most $2^{-D/2+5c+1} \cdot 4 \cdot (\log N+1) = 2^{5c+3} \cdot (\log N+1)/\sqrt{N}$.

■ of Lemma 6.2.3

The bound for $\mathrm{E}_{\mathrm{coll}}(v_i)$ on the expected number of active configurations for an internal node v_i is independent of the configurations of the internal nodes $v_1, \ldots, v_{i-1}$. Further, the bound for $\mathrm{E}_{\mathrm{prune}}(w_i)$ on the expected number of active configurations for a pruning node w_i is independent of the configurations on all internal nodes and the pruning nodes $w_1, \ldots w_{i-1}$. Consequently, these bounds are independent estimations of expected values and can be multiplied in order to get an upper bound on the expected number of all configurations. Since the number of choices for the initial configuration K in $\mathrm{E}(v_1, K)$ specifying the request associated with the root is N, we get the following upper bound on the expected number of active delay tree configurations.

$$\begin{aligned}
\sum_{Q\in\mathcal{Q}} \mathrm{E}(Q) &\leq \sum_{Q\in\mathcal{Q}} N \cdot \prod_{i=1}^{m} \mathrm{E}_{\mathrm{coll}}(v_i) \prod_{j=1}^{m'} \mathrm{E}_{\mathrm{prune}}(w_j) \\
&\overset{(1)}{\leq} N \cdot \sum_{Q\in\mathcal{Q}} \left(\frac{\log N}{c!}\right)^{m} \left(\frac{2^{5c+3}\cdot(\log N+1)}{\sqrt{N}}\right)^{m'} \\
&\overset{(2)}{\leq} N \cdot \sum_{Q\in\mathcal{Q}} \left(\frac{2^{5c+3}\cdot(\log N+1)}{\sqrt{N}}\right)^{\kappa} \\
&\overset{(3)}{\leq} N \cdot c^{2\kappa t+\kappa} \cdot \left(\frac{2^{5c+3}\cdot(\log N+1)}{\sqrt{N}}\right)^{\kappa} \\
&\leq N^{-c/4+1+o(1)} ,
\end{aligned}$$

for $\kappa = \lceil c/2 \rceil = \Theta\left(\log\log N / \log\log\log N\right)$ and a suitably large

$$t = \Theta\left(\log\log N / \log\log\log N\right).$$

Equation 1
: is an immediate consequence of Lemma 6.2.2 and Lemma 6.2.3.

Equation 2
: is based on the relationship between m and m': The full delay tree includes c disjoint subtrees of height $t-1$. For each of the m' pruning nodes, some nodes from at most two of these subtrees are removed. Consequently, at least $c-2m'$ of the subtrees remain untouched. Since each of them include at least c^{t-2} internal nodes, we get

$$m \geq (c-2m')\cdot c^{t-2} \geq (\kappa - m')\cdot c^{t-2} \ .$$

Applying this equation and substituting $c! = (1+\epsilon)\cdot\log N$ yields

$$\begin{aligned}
\left(\frac{\log N}{c!}\right)^{m} &\leq (1+\epsilon)^{-c^{t-2}\cdot(\kappa-m')} \\
&\leq \left(\frac{2^{5c+3}\cdot(\log N+1)}{\sqrt{N}}\right)^{\kappa-m'} ,
\end{aligned}$$

for $t \geq \log_c \log_{1+\epsilon} N + 2 = \Theta(\log\log N / \log\log\log N)$.

Equation 3
: results from a bound on the number of different tree shapes. In particular, there are at most

$$\sum_{j=0}^{\kappa} \binom{(c^t-1)/(c-1)}{j} \leq c^{\kappa t}$$

possible choices for the at most κ pruning nodes among the $(c^t-1)/(c-1)$ internal nodes of the delay tree, and at most

$$\left(\frac{c^{t+1}-1}{c-1}\right)^{m'} \leq c^{\kappa(t+1)}$$

possibilities to choose the m' conflicting nodes among the $(c^{t+1}-1)/(c+1) \leq c^{t+1}$ nodes of the full delay tree. Since specifying these nodes completely determines the shape of the tree, the total number of different tree shapes is at most $c^{\kappa t} + c^{\kappa(t+1)} \leq c^{2\kappa t+\kappa}$.

We have already shown that $\sum_{Q\in\mathcal{Q}} \mathrm{E}(Q)$ is an upper bound on the probability that the network c-collision algorithm takes more than t rounds. Hence, this probability is at most $N^{-c/4+1+o(1)}$. It remains to show that determining which paths become inactive each round can be done in time $\mathrm{O}(\log N)$, with high probability. Recall that, in our model, this computation is accomplished by sending a packet back and forth along each active path through the network using a store-and-forward algorithm. According to [LMRR94], such a computation can be done in time $\mathrm{O}(\text{congestion}+\text{dilation})$, with high probability, using only constant size buffers at each edge. Note here that the congestion we wish to bound is the congestion caused using this store-and-forward scheme, not the congestion under the network c-collision algorithm. However, this congestion is easily bounded. Let C denote the congestion of all $2n$ paths.

Lemma 6.2.4: $C \leq \alpha \cdot \log N/\log\log N$, with probability $N^{-\alpha+\mathrm{O}(1)}$.

Proof: The congestion in the randomization levels is 1. Therfore, we only have to consider the collision levels. The probability that a fixed collision edge is traversed by at least C paths is at most $1/C!$. This bound follows analogously to the proof of Lemma 6.2.2. Hence, the probability that one of the $2 \cdot N \cdot \log N$ collision edges has congestion C is at most

$$2 \cdot N \cdot \log N \cdot 1/C! \leq N^{-\alpha+\mathrm{O}(1)} \ ,$$

for $C > \alpha \cdot \log N/\log\log N$. ■ of Lemma 6.2.4

Applying Lemma 6.2.4 yields that each round can be computed in time $\mathrm{O}(\log N)$, with high probability. This completes the proof of Theorem 6.2.1. ■ of Theorem 6.2.1

6.2.2 Routing in BB_N in an infinite sequential setting

We now describe a simple algorithm that routes paths dynamically in network BB_N. As before, a *request* is an input-output pair. These requests are inserted and deleted similar as specified by an oblivious deletion scheme (cf. to the beginning of Section 5.1.1). An

oblivious adversary specifies an infinite sequence $\sigma_1, \sigma_2, \ldots$ of requests. The request σ_i must be handled at time i, where time flows in the natural manner. If at time i neither the input nor the output of σ_i is already locked, then the algorithm must establish and lock a path in the network between the input and output of σ_i. This is an *arrival*. If there is already a locked path between the input-output pair, then the path is released. This is a *departure*. In all other cases the request may be ignored. That is, the algorithm only connects an input-output pair if neither is already involved in a connection. Without loss of generality we may assume that the sequence of requests includes only valid arrival and departure events. An input-output pair is said to *exist* at each time k between its arrival and departure.

The network greedy algorithm To solve the dynamic routing problem on the two-fold butterfly BB_N, we initialize BB_N as in Section 6.2.1. Let s_i denote an arrival event. A path for the corresponding request r_i is chosen as follows. For an edge e in the collision levels, define $c(e)$ to be the number of paths that traverse e at time i. The algorithm examines the two paths P and P' that connect the input to the output of r_i. The congestion $c(P)$ of a path P is defined to be $\max_{e \in P}(c(e))$. If $c(P) \leq c(P')$, path P is chosen for request r_i; Otherwise, path P' is chosen.

Theorem 6.2.5: At any time t, the probability that the congestion is greater than $\Theta(\log \log N)$ is at most $N^{-\Theta(\log \log N)}$.

Proof: The proof is similar to that of Theorem 6.2.1.

Constructing a delay tree. First, we fix the settings of the randomly flipped switches. This determines two allowable paths for each request. Assume that there is an edge e with congestion larger than $4c$ at some time t, where $c = \lceil \log \log N \rceil$. Let P denote the last path mapped to edge e on or before time t. When P was mapped to e there were already $4c$ other paths present at this edge. Let $P_1, \ldots, P_{4c}$ denote these paths such that P_i was mapped to e at time step t_i with $t_i < t_{i+1}$. The root of the tree is the request corresponding to P and the requests corresponding to $P_1, \ldots, P_{4c}$ are its children. Now we consider the buddies $P'_1, \ldots, P'_{4c}$ of these paths. Path P'_i traverses an edge with congestion at least $i - 1$ at time step t_i, because the congestion of P_i is not larger than the congestion of P'_i at time i, and when P_i was mapped to e there were already $i - 1$ other paths present at this edge. As a consequence, we can construct a tree by applying the argument above recursively to $P'_2, \ldots, P'_{4c}$.

The tree constructed above is irregular in that nodes have varying degrees. However, it contains a c-ary tree of height c, which we call the delay tree, with the following properties.

- The node on level 0, i.e., the root, has c children that are internal nodes.
- Each internal node on levels $1, \ldots, c-2$ has 2 children that are internal nodes and $c - 2$ children that are leaves, and each internal node on level $c - 1$ has c children that are leaves.

Pruning the delay tree. The pruning is done by a breadth-first traversal of the tree. We use the same definitions for $B(v)$ and $\Sigma(v)$ as in Section 6.2.1. However, the pruning rules are slightly different. When a node v is visited, the following rules are applied.

1. If a path associated with one of v's non-pruned children traverses a randomly flipped switch that is also traversed by a path associated with a node u from $B(v) \cup \Sigma(v)$ then all nodes below v are pruned. Node u is denoted the conflicting node of v. Note that the down path of v either shares a collision edge with a path P that is associated to u, or it shares a collision edge with a path P such that P or its buddy shares a random switch with a path associated with u. This path P is denoted the *conflicting path* of v.

2. Depending on the conflicting path P we apply a further pruning. For each node $u \in \Sigma(v)$ such that either the input or output node of u coincides with the input or output node of path P, we prune all the nodes below u. The first pruning rule ensures that there is at most one request in $B(v) \cup \Sigma(v)$ incident on each input and output of the network, even though the requests in $B(v) \cup \Sigma(v)$ exist at possibly non-overlapping times. Thus, at most two nodes, call them u and u', get pruned due to an application of this rule. Nodes u and u' are defined to be the conflicting nodes of v. (For simplicity, we pretend that each pruning node v has two conflicting nodes u and u'; if this is not the case we simply set u and u' to be the same node.) The second pruning rule ensures that Properties 4 and 5 as stated in Section 6.2.1 hold for the pruned delay tree – specifically, the down path of a pruning node cannot share more than two randomly-flipped switches with a given conflicting path.

We continue the pruning process till either there are no more nodes to visit or there are $\kappa = \lceil c/3 \rceil$ pruning nodes. In the latter case, we apply a final pruning. If v is the κth pruning node, we remove from the tree all nodes not included in $B(v) \cup \Sigma(v)$. The remaining tree is called the *pruned delay tree*.

Bounding the probability of occurrence of a pruned delay tree. The terms *tree shape*, *admissible configuration*, and *active configuration* are defined as in Section 6.2.1. Let $\mathcal{Q}$ denote the set of all tree shapes, and, for $Q \in \mathcal{Q}$, let $\mathrm{E}(Q)$ denote the expected number of active delay tree configurations with tree shape Q. Let $v_1, \ldots, v_m$ be the m internal nodes of Q. Further, for a collection K of configurations for the nodes $v_1, \ldots, v_{i-1}$, let $\mathrm{E}_{\mathrm{coll}}(v_i, K)$ denote the expected number of active configurations for v_i under the assumption that K is active, and let $\mathrm{E}_{\mathrm{coll}}(v_i)$ denote the maximum over all configurations K of $\mathrm{E}_{\mathrm{coll}}(v_i, K)$.

Lemma 6.2.6: $\mathrm{E}_{\mathrm{coll}}(v_i) \leq \log N/c!$.

Proof: The proof is identical to that of Lemma 6.2.2, since the pruned delay constructed here fulfills Properties 1 and 2 as stated in Section 6.2.1. ■ of Lemma 6.2.6

Let $w_1, \ldots, w_{m'}$ denote the m' pruning nodes of Q, and let u_i and u_i' denote the conflicting nodes associated with w_i. For a collection K of configurations for the nodes $v_1, \ldots, v_m$ and $w_1, \ldots, w_{i-1}$, let $\mathrm{E}_{\mathrm{prune}}(w_i, K)$ denote the expected number of active configurations for w_i under the assumption that K is active. Further, let $\mathrm{E}_{\mathrm{coll}}(w_i)$ denote the maximum over all configurations K of $\mathrm{E}_{\mathrm{coll}}(v_i, K)$.

Lemma 6.2.7: $\mathrm{E}_{\mathrm{prune}}(w_i) \leq 2^{5c+3} \cdot (\log N + 1)/\sqrt{N}$.

Proof: The pruned delay tree described above fulfills Properties 3, 4 and 5 stated in Section 6.2.1. Hence, the proof of Lemma 6.2.3, which is based only on these three properties, holds also for this lemma. ■ of Lemma 6.2.7

The probability that the congestion exceeds $4c$ is at most the probability that a pruned delay tree exists. The latter probability is at most

$$\begin{aligned}
\sum_{Q \in \mathcal{Q}} \mathrm{E}(Q) &\leq \sum_{Q \in \mathcal{Q}} N \cdot \prod_{i=1}^{m} \mathrm{E}_{\mathrm{coll}}(v_i) \prod_{j=1}^{m'} \mathrm{E}_{\mathrm{prune}}(w_j) \\
&\leq N \cdot \sum_{Q \in \mathcal{Q}} \left(\frac{\log N}{c!} \right)^m \cdot \left(\frac{2^{5c+3} \cdot (\log N + 1)}{\sqrt{N}} \right)^{m'} \\
&\overset{(1)}{\leq} N \cdot \sum_{Q \in \mathcal{Q}} \left(\frac{2^{5c+3} \cdot (\log N + 1)}{\sqrt{N}} \right)^{\kappa} \\
&\overset{(2)}{\leq} N \cdot c^{5\kappa} \cdot 2^{3c\kappa} \cdot \left(\frac{2^{5c+3} \cdot (\log N + 1)}{\sqrt{N}} \right)^{\kappa} \\
&\leq N^{-c/6+1+o(1)},
\end{aligned}$$

where $\kappa = \lceil \frac{c}{3} \rceil = \Theta(\log \log N)$.

Equation 1

follows from the relationship between m and m': Each of the c children of the root of the full delay tree is a root of a subtree with $2^{c-1} - 1$ internal nodes. For each of the m' pruning nodes, nodes from at most 3 of these subtrees are removed. Thus, at least $c - 3m'$ of the subtrees remain untouched. As a consequence,

$$m \geq (c - 3m') \cdot (2^{c-1} - 1) \geq (\kappa - m') \cdot (2^{c-1} - 1).$$

Applying this equation and substituting $c = \lceil \log \log N \rceil$ yields

$$\left(\frac{\log N}{c!} \right)^m \leq 2^{-(2^{c-1}-1)\cdot(\kappa - m')} \leq \left(\frac{2^{5c+3} \cdot (\log N + 1)}{\sqrt{N}} \right)^{\kappa - m'},$$

for sufficiently large N.

Equation 2

results from a bound on the number of different tree shapes. In particular, there are at most

$$\sum_{j=0}^{\kappa} \binom{c \cdot 2^{c-1}}{j} \leq 2 \cdot c^{\kappa} \cdot 2^{(c-1)\cdot\kappa}$$

possible ways of chooosing the at most κ pruning nodes from the at most $c \cdot 2^{c-1}$ internal nodes of the delay tree. Further, there are at most

$$\left(c^2 \cdot 2^{c-1}\right)^{2m'} \leq c^{4\kappa} \cdot 2^{(c-1)\cdot 2\kappa}$$

possibilities to choose the $2m'$ conflicting nodes from the at most $c^2 \cdot 2^{c-1}$ nodes of the full delay tree. Multiplying the bounds yields that the total number of different tree shapes is at most $c^{5\kappa} \cdot 2^{3c\kappa}$.

■ of Theorem 6.2.5

7 Conclusions

In this thesis we examined the allocation problem as a very pure problem that has many applications in various fields and therefore gained quite much attention. Our interest in the problem was threefold.

At first our interest was to give new results on the performance of allocation algorithms, specifically we were interested in settings of the allocation problem motivated by properties of applications. In particular we considered a new deletion scheme: the *server oriented* deletion scheme. Oblivious deletion schemes which are closely related to finite settings have the disadvantage to assume that server speed *increases* with load. This is avoided by the server oriented deletion scheme we presented and analyzed here. Moreover we considered weighted allocation problems overcoming the assumption that the amount of consumed resources is the same for each job.

Our second interest was to systematize the allocation problem and the allocation algorithms and to fill holes between known results. The variety of different settings and the different algorithms with their variants showed the need for a general approach. We found this approach in the witness tree technique. Extending the possibilities of the technique we devised a general lemma that allows to obtain results for any known algorithm in any setting. Albeit of its generality our technique allows to devise quite exact results. In most cases our performance bounds match previous results. Frequently our results are applicable to a wider scope of parameters, and in any case we are able to deal with any sensible i-th copy distributions. Despite of that we considered several settings of the allocation problem that have not been considered before. In particular this holds for weighted allocation problems.

At third we showed how to apply the balls-into-bins paradigm beyond the scope of the allocation problem. Presenting an algorithm for circuit-switching in a butterfly-type network, we showed that picking-one-out-of-several-random-possibilities applies also for much more complicated problems of allocation. Our application again demonstrated the power of the witness tree technique, which allows to deal with stochastic dependencies in the random variables used.

In setting up our Main Lemma and the notions needed for it, we had to compromise. In several cases this implied that possible generalizations have not been included. Extending the Main Lemma to possibly dependent i-th copy distributions as considered by Vöcking in [Vöc99a] would just require to change the definition of the vivid system and its parameters a little bit. We excluded this case as we believe that the definition is already quite complicated.

Another interesting generalization would be to consider allocation problems where each job has to be allocated to $2 \leq f < d$ servers. This generalized allocation problem has been considered in the PRAM Simulations mentioned in the introduction. Another important application of this generalized allocation problem addresses the storage overhead in data-server (VoD) applications. In such applications the jobs model requests for reading a block of data containing F bits. Using the allocation problem each block has to be stored on d disks, which requires $d \cdot F$ bits of memory. If the allocation allows to access $f = d - 1$ servers for each job, it is possible to reduce this overhead significantly using a simple error correcting code. Divide the data block into f sub-blocks of size F/f. Let the first f servers store one sub-block each, and let the d-th possible server store a sub-block containing the bit-wise XOR over the i-th bit of each of the first $d-1$ sub-blocks in its i-th bit. Then access to an arbitrary subset of size $f = d - 1$ of the d possible servers allows to reconstruct the entire block. This technique allows to reduce storage overhead from d to $d/(d-1)$. A generalized Main Lemma would just have to allow generalized mappings $label_e$ in the representation, and some minor changes in the definition of the vivid system.

As noted several time this version of the witness tree argument is not the only one. Our main interest was generality not simplicity. Witness tree arguments set up for single problems can be much simpler than the one considered here.

The allocation problem in the sequential setting is well understood. The upper and lower bounds in [Vöc99a] match up to an additive constant, and the lower bound applies to arbitrary allocation algorithms and arbitrary i-th copy distributions. The situation is worse for the parallel setting. Although the c-collision algorithm matches the existing lower bounds up to a constant factor. As described in the introduction of this thesis, the lower bounds for the parallel setting are each restricted to some class of allocation algorithms. We believe that devising general lower bounds that match the existing (or improved) upper bounds up to an additive constant is a very challenging task. Devising a general lower bound that matches up to a constant factor should be much easier — as long as the considered allocation algorithms are direct. As also mentioned in [Mit96b] there are simple arguments that indicate that allowing labels for jobs and servers is quite simple.

Bibliography

[AAF+94] J. Aspnes, Y. Azar, A. Fiat, S. Plotkin, and O. Waarts. On-line machine scheduling with applications to load balancing and virtual circuit routing. In *Proceedings of the 25th ACM Symposium on Theory of Computing*, pages 240–249, November 1994.

[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.

[ABS98] M. Adler, P. Berenbrink, and K. Schöder. Analyzing an infinite parallel job allocation process. In G. Bilardi, G. Italiano, A. Piertracaprina, and G. Pucci, editors, *Proceedings of the Annual European Symposium on Algorithms*, pages 417–428, 1998.

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

[ALM96] S. Arora, T. Leighton, and B. Maggs. On-line algorithms for path selection in a non-blocking network. *SIAM Journal on Computing*, 25(3), pages 600–625, June 1996.

[ANR92] Y. Azar, J. (S.) Naor, and R. Rom. The competitiveness of on-line assignments. In *Proceedings of the 3rd ACM-SIAM Symposium on Discrete Algorithms*, pages 203–210, 1992.

[ANR95] Y. Azar, J. (S.) Naor, and R. Rom. The competitiveness of on-line assignments. *Journal of Algorithms*, 18:221–237, 1995.

[Aza98] Y. Azar. On-line load balancing. In A. Fiat and G. Woeginger, editors, *Online Algorithms: The State of the Art*, volume 1442 of *LNCS*, pages 178–195. Springer, 1998.

[Bas74] L. A. Bassalygo and M. S. Pinsker. Complexity of an optimum nonblocking switching network without reconnections. *Problems of Information Transmission*, 9:64–66, 1974.

[Bat68] K. Batcher. Sorting networks and their applications. In *Proceedings of the AFIPS Spring Joint Computing Conference*, volume 32, pages 307–314, 1968.

[BE98] A. Borodin and R. El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, Cambridge, U.K.; New York, 1998.

[Bei62] B. Beizer. The analysis and synthesis of signal switching networks. In *Proceedings of the Symposium on Mathematical Theory of Automata*, pages 563–576, Brooklyn, NY, 1962. Brooklyn Polytechnic Institute.

[Ben64] V. E. Beneš. Optimal rearrangeable multistage connecting networks. *Bell System Technical Journal*, 43:1641–1656, July 1964.

[BFL+95] A. Z. Broder, A. Frieze, C. Lund, S. Phillips, and N. Reingold. Balanced allocations for tree-like inputs. *Information Processing Letters*, 55(6):329–332, September 1995.

[BFM98] P. Berenbrink, T. Fridetzky, and E. Mayr. Parallel continuous randomized load balancing. In *Proceedings of the 10th Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 192–201, Puerto Vallarta, Mexico, 1998. ACM.

[BFS99] P. Berenbrink, T. Fridetzky, and A. Steger. Randomized and adversarial load balancing. In *Proceedings of the 11th Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 175–184, 1999.

[BMS97] P. Berenbrink, F. Meyer auf der Heide, and K. Schröder. Allocating weighted jobs in parallel. In *Proceedings of the 9th Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 302–310, Newport, Rhode Island, USA, 1997.

[BMS99] P. Berenbrink, F. Meyer auf der Heide, and K. Schröder. Allocating weigted jobs in parallel. *Theory of Computing Systems*, 32:281–300, 1999.

[CMM+98] R. Cole, B. Maggs, F. Meyer auf der Heide, M. Mitzenmacher, A. Richa, K. Schr"oder, R. Sitaraman, and B. V"ocking. Randomized protocols for low-congestion circuit routing in multistage interconnection networks. In *Proceedings of the 30th ACM Symposium on Theory of Computing*, pages 378–388, Dallas, Texas, May 1998.

[CMS95a] A. Czumaj, F. Meyer auf der Heide, and V. Stemann. Improved optimal shared memory simulations, and the power of reconfiguration. In *Proceedings of the 3rd Israel Symposium on Theory of Computing and Systems (ISTCS)*, pages 11–19. IEEE Computer Society Press, January 4-6 1995.

[CMS95b] A. Czumaj, F. Meyer auf der Heide, and V. Stemann. Shared memory simulations with triple-logarithmic delay. In P. Spirakis, editor, *Proceedings of the 3rd Annual European Symposium on Algorithms*, volume 979 of *Lecture Notes in Computer Science*, pages 46–59. Springer-Verlag, September 1995.

[CMS95c] A. Czumaj, F. Meyer auf der Heide, and V. Stemann. Simulating shared memory in real time: On the computation power of reconfigurable meshes. In *Proceedings of the 2nd IEEE Workshop on Reconfigurable Architectures (RAW)*. IEEE, April 1995.

[CMS96] R. Cole, B. Maggs, and R. Sitaraman. On the benefit of supporting virtual channels in wormhole routers. In *Proceedings of the 8th Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 131–141, June 1996.

[CMS97] A. Czumaj, F. Meyer auf der Heide, and V. Stemann. Simulating shared memory in real time: On the computation power of reconfigurable architectures. *Information and Computation*, 137(2):103–120, September 1997.

[CMSV96] R. Cypher, F. Meyer auf der Heide, C. Scheideler, and B. Vöcking. Universal algorithms for store-and-forward and wormhole routing. In *Proceedings of the 28th Annual ACM Symposium on the Theory of Computing*, May 1996.

[CS97] A. Czumaj and V. 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.

[Czu97] A. Czumaj. Recovery time of dynamic allocation processes. In *Proceedings of the 10th Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 202–211, Puerto Vallarta, Mexico, June 1997. ACM, ACM Press, New York, NY.

[DM93] M. Dietzfelbinger and F. Meyer auf der Heide. Simple efficient shared memory simulations. In *Proceedings of the 5th Annual ACM Symposium on Parallel Algorithms and Architectures*, 1993.

[ELZ86] D. L. Eager, E. D. Lazowska, and J. Zahorjan. Adaptive load sharing in homogenous distributed systems. *IEEE Transaktions on Software Engeneering*, SE-12(5):662–675, May 1986.

[FRU92] S. Felperin, P. Raghavan, E. Upfal. A theory of wormhole routing in parallel computers. In *Proceedings 33rd IEEE Symposium on Foundations of Computer Science*, pages 563-572, 1992.

[GG92] J.A. Garay, and I. Gopal. Call preemption in communication networks. In *Proceedings of INFOCOM '92*, Vol 44, pages 1043–1050, Florence, Italy, 1992.

[Gon81] G. H. Gonnet. Expected length of the longest probe sequence in hash code searching. *Journal of the ACM*, 28(2):289–304, April 1981.

[Got87] A. Gottlieb. An overview of the NYU Ultracomputer Project. In J. J. Dongarra, editor, *Experimental Parallel Computing Architectures*, pages 25–95. Elsevier Science Publishers, B. V., Amsterdam, The Netherlands, 1987.

[HR90] T. Hagerup and C. Rüb. A guided tour of chernoff bounds. *Information Processing Letters*, pages 305–308, 1990.

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

[Knu98] D. E. Knuth. *The Art of Computer Programming, Volume 3.* Addison-Wesley, 1998.

[Kri99] C. Krick. Resource allocation in batches. Diplomarbeit, Paderborn University, 1999.

[KS83] C. P. Kruskal and M. Snir. The performance of multistage interconnection networks for multiprocessors. *IEEE Transactions on Computers*, C–32(12):1091–1098, December 1983.

[Koc88] R. R. Koch. Increasing the size of a network by a constant factor can increase performance by more than a constant factor. In *Proceedings of the 29th Annual Symposium on Foundations of Computer Science*, pages 221–230. IEEE Computer Society Press, October 1988.

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

[KVV90] R. M. Karp, U. V. Vazirani, and V. V. Vazirani. An optimal algorithm for on-line bipartite matching. In *Proceedings of the 22nd ACM Symposium on Theory of Computing*, pages 352–358, Baltimore, MD, May 1990.

[Lei92] F. T. Leighton. *Introduction to Parallel Algorithms and Architectures: Arrays • Trees • Hypercubes.* Morgan Kaufmann, San Mateo, CA, 1992.

[Lei92b] T. Leighton. Methods for message routing in parallel machines. In *Proceedings of the 24th Annual ACM Symposium on the Theory of Computing*, pages 77–96, May 1992.

[Lin92] T. Lindvall. *Lectures on the Coupling Method*. Wiley Series in Probability and Mathematical Statistics. John Wiley & Sons, New York, NY, 1992.

[LM92] F. T. Leighton and B. M. Maggs. Fast algorithms for routing around faults in multibutterflies and randomly-wired splitter networks. *IEEE Transactions on Computers*, C–41(5):578–587, May, 1992.

[LMRR94] F. T. Leighton, B. M. Maggs, A. G. Ranade, and S. B. Rao. Randomized routing and sorting on fixed-connection networks. *Journal of Algorithms*, 17(1):157–205, July 1994.

[LS91] H. Li and Q. F. Stout, editors. *Reconfigurable Massively Parallel Computers*. Prentice Hall, Englewood Cliffs, New Jersey, 1991.

[Luh84] N. Luhmann. *Soziale Systeme. Grundriß einer allgemeinen Theorie*, page 49. Suhrkamp Verlag, Frankfurt am Main, Germany, 1984. "Nur Komplexität kann Komplexität reduzieren." The english translation is "Only complexity can reduce complexity".

[Mit96a] M. 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, Washington - Brussels - Tokyo, 1996. IEEE.

[Mit96b] M. Mitzenmacher. *The Power of Two Random Choices in Randomized Load Balancing*. PhD thesis, Graduate Division of the University of California at Berkley, 1996.

[Mit97] M. 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. SIGACT/SIGARCH and EATCS.

[MPR94] P. D. MacKenzie, C. G. Plaxton, and R. Rajaraman. On contention resolution protocols and associated probabilistic phenomena. In *Proceedings of the 26th ACM Symposium on Theory of Computing*, pages 153–162, Montréal, Québec, Canada, 1994.

[MR95] R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, Cambridge, England, 1995.

[MS92] B. M. Maggs and R. K. Sitaraman. Simple algorithms for routing on butterfly networks with bounded queues. In *Proceedings of the 24th Annual ACM Symposium on Theory of Computing*, pages 150–161, May 1992.

[MSS95] F. Meyer auf der Heide, C. Scheideler, and V. 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.

[MSS96] F. Meyer auf der Heide, C. Scheideler, and V. Stemann. Exploiting storage redundancy to speed up randomized shared memory simulations. Technical Report tr-rsfb-96-10, Sonderforschungsbereich 376, Massive Parallelität, University Paderborn, 1996.

[MSS98] F. Meyer auf der Heide, K. Schröder, and F. Schwarze. Routing on networks of optical crossbars. *Theoretical Computer Science*, 196:181–200, 1998.

[MV95] F. Meyer auf der Heide and B. Vöcking. A packet routing protocol for arbitrary networks. In *Proceedings of the 12th Symposium on Theoretical Aspects of Computer Science*, pages 291–302, March 1995.

[Nak91] T. Nakata, S. Matsushita, N. Tanabe, N. Kajihara, H. Onozuka, Y. Asano, and N. Koike. Parallel programming on Cenju: A multiprocessor system for modular circuit simulation. *NEC Research & Development*, 32(3):421–429, July 1991.

[NS82] D. Nassimi and S. Sahni. Parallel algorithms to set up the Beneš permutation network. *IEEE Transactions on Computers*, C–31(2):148–154, Feb, 1982.

[PBG+87] G. F. Pfister, W. C. Brantley, D. A. George, S. L. Harvey, W. J. Kleinfelder, K. P. McAuliffe, E. A. Melton, V. A. Norton, and J. Weiss. An introduction to the IBM Research Parallel Processor Prototype (RP3). In J. J. Dongarra, editor, *Experimental Parallel Computing Architectures*, pages 123–140. Elsevier Science Publishers, B. V., Amsterdam, The Netherlands, 1987.

[Pip84] N. Pippenger. Parallel communication with limited buffers. In *Proceedings of the 25th Annual Symposium on Foundations of Computer Science*, pages 127–136. IEEE Computer Society Press, October 1984.

[Pip96] N. Pippenger. Self-Routing Superconcentrators. *Journal of Computer and System Sciences*, 52(1):53–60, Feb, 1996.

[Plo95] S. Plotkin. Competitive routing in ATM networks. *IEEE Journal on Selected Areas in Communication*, pages 1128–1136, August 1995.

[Ran87] A. G. Ranade. Constrained randomization for parallel communication. Technical Report YALEU/DCS/TR-511, Department of Computer Science, Yale University, New Haven, CT, 1987.

[Ran91] A. G. Ranade. How to emulate shared memory. *J. Comp. Syst. Scis. 42*, pages 307-326, 1991.

[RCM94] R. Rooholamini, V. Cherkassky, and M. Garver. Finding the right ATM switch for the market. *IEEE Computer*, pages 16–28, April 1994.

[Reh99] U. Rehberg. Worst-Case-Laufzeitanalyse von verteilten Speicherzugriffsprotokollen. Diplomarbeit, Paderborn University, 1999.

[RSW94] A. Ranade, S. Schleimer, and D. S. Wilkerson. Nearly tight bounds for wormhole routing. In *Proceedings of the 35th Annual Symposium on Foundations of Computer Science*, 1994.

[ST85] D.D. Sleator and R.E. Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28(2):202–208, 1985.

[Ste95] V. Stemann. *Contention Resolution in Hashing Based Shared Memory Simulations*. PhD thesis, University Paderborn, 1995. appears in: HNI-Verlagsschriftenreihe, Bd. 3.

[Ste96] V. 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.

[TY97] J. Turner and N. Yamanaka. Architectural Choices in Large Scale ATM Switches. Technical Report WUCS–97–21. Department of Computer Science, Washington University St. Louis, MO. 1997.

[Upf92] E. Upfal. An $O(\log N)$ deterministic packet routing scheme. *Journal of the ACM*, 39(1), pages 55–70, Jan 1992.

[Val81] L. G. Valiant and G. J. Brebner. Universal schemes for parallel communication. In *Proceedings of the 13th Annual ACM Symposium on Theory of Computing*, pages 263–277, May 1981.

[Val82] L. G. Valiant. A scheme for fast parallel communication. *SIAM Journal on Computing*, 11(2):350–361, May 1982.

[VDK96] N. D. Vvedenskaya, R. L. Dobrushin, and F. I. Karpelevich. Queueing system with selection of the shortest of two queues: an asymptotic approach. Problems of Information Transmission, 32(1), pages 15–27, 1996.

[Vöc99a] B. Vöcking. How asymmetry helps load balancing. In *Proceedings of the 40th IEEE Symposium on Foundations of Computer Science*, 1999.

[Vöc99b] B. Vöcking. *Static and Dynamic Data Management in Networks*. PhD thesis, University Paderborn, Paderborn, 1999. appears in: HNI-Verlagsschriftenreihe Bd. 46.

[Wak68] A. Waksman. A permutation network. *Journal of the ACM*, 15(1):159–163, January 1968.