# Exploiting Storage Redundancy to Speed Up Randomized Shared Memory Simulations

Friedhelm Meyer auf der Heide and Christian Scheideler*
Heinz Nixdorf Institute and Computer Science Department,
University of Paderborn, 33095 Paderborn, Germany

Volker Stemann†
International Computer Science Institute,
University of California at Berkeley, CA 94704-1198, USA

**Abstract**

Assume that a set $U$ of memory locations is distributed among $n$ memory modules, using some number $a$ of hash functions $h_1, \ldots, h_a$, randomly and independently drawn from a high performance universal class of hash functions. Thus each memory location has $a$ copies. Consider the task of accessing $b$ out of the $a$ copies for each of given keys $x_1, \ldots, x_n \in U$, $b < a$. The paper presents and analyses a simple process executing the above task on distributed memory machines (DMMs) with $n$ processors. Efficient implementations are presented, implying

- a simulation of an $n$-processor PRAM on an $n$-processor optical crossbar DMM with delay $O(\log\log n)$,
- a simulation as above on an arbitrary-DMM with delay $O(\frac{\log\log n}{\log\log\log n})$,
- an implementation of a static dictionary on an arbitrary-DMM with parallel access time $O(\log^* n + \frac{\log\log n}{\log a})$, if $a$ hash functions are used. In particular, an access time of $O(\log^* n)$ can be reached if $(\log n)^{1/\log^* n}$ hash functions are used.

We further prove a lower bound for executing the above process by *any* so-called *simple access protocol*, showing that our implementations are optimal.

## 1 Introduction

Parallel machines that communicate via a shared memory, so-called *parallel random access machines* (PRAMs), represent an idealization of a parallel computation model. The user does not have to worry about synchronization, locality of data, communication capacity, delay effects or memory contention.

On the other hand, PRAMs are very unrealistic from a technological point of view; large machines with shared memory can only be built at the cost of very slow shared memory access. A more realistic model is the *distributed memory machine* (DMM), where the memory is partitioned into modules, one per processor. In this case a parallel memory access is restricted in so far as only one access to each module can be performed per parallel step. Thus memory contention occurs if a PRAM algorithm is run on a DMM; parallel accesses to cells stored in one module have to be sequentialized.

Many authors have already investigated methods for simulating PRAMs on DMMs. If one focuses on a complete network between processors and modules, the main problem is the distribution of the shared memory cells over the modules to allow fast accesses. A standard method is to use universal hashing for distributing the shared memory among the memory modules of the DMM. In this paper

*Supported in part by DFG-Sonderforschungsbereich 376 "Massive Parallelität: Algorithmen, Entwurfsmethoden, Anwendungen", and by DFG Leibniz Grant Me 872/6-1.

†Supported by the DFG-Graduiertenkolleg "Parallele Rechnernetzwerke in der Produktionstechnik", ME 872/4-1; work was done at the Heinz Nixdorf Institute, University of Paderborn.

we consider both simulations of PRAMs and implementations of parallel static dictionaries on DMMs, based on distributing the shared memory cells among the modules using not only one but several hash functions, i.e. using a *redundant* storage representation.

## 1.1 Computation models

A *parallel random access machine* (PRAM) consists of processors $P_1, \ldots, P_n$ and a shared memory with cells $U = \{1, \ldots, p\}$, each capable of storing one integer. The processors work synchronously and have random access to the shared memory cells. In this paper we will only consider the exclusive-read exclusive-write PRAM (EREW PRAM) model, that is, no two processors are allowed to access the same shared memory cell at the same time during a read or write step.

A *distributed memory machine* (DMM) consists of $n$ processors $Q_1, \ldots, Q_n$ and $n$ memory modules $M_1, \ldots, M_n$. Each processor has a link to each module. A basic communication step of such a DMM consists of the processors sending read or write requests to the memory modules, at most one request per processor. Each module processes some of the requests directed to it and sends an acknowledgement to each processor whose request was chosen to be processed.

We distinguish between the following rules for choosing requests for processing. ($c \geq 1$ is a fixed integer. For a discussion of the models see [8] or [14].)

- *arbitrary-DMM* : In this case, one arbitrarily chosen request out of all requests arriving at one module is processed per step. The answer given by a module is accessible by all processors accessing the module.

- *c-collision DMM* : In this case, all requests arriving at one module are processed in one step, as long as there are at most $c$ of them; otherwise none is processed. An answer is only accessible by the issuing processor. (Note : For $c = 1$ this model corresponds to a communication mechanism based on optical crossbars (compare [1], [9] and [22]). $c$-collision DMMs can easily be simulated on arbitrary-DMMs with delay $O(c)$.)

Randomized versions of the above models are obtained by adding the capability of choosing a random integer from a finite range.

## 1.2 Dictionaries and Shared Memory Simulations

*Shared memory simulations* on a DMM based on hashing begin with a preprocessing phase. In this phase each processor $P_i$ of the PRAM is mapped to processor $Q_i$ of the DMM and the shared memory cells (we say *keys* for short) of the PRAM are distributed among the modules of the DMM via $a \geq 1$ randomly and independently chosen hash functions from some suitable universal class of hash functions (see below), i.e. each shared memory cell has $a$ *copies*. This *redundant storage representation* needs space $a \cdot |U|$.

In this paper we will only deal with $a \geq 2$. The basic access distribution phase (we say *basic process* for short) will be organized in such a way that each processor $P_i$ that wants to get access to a key $x_i \in U$ tries to send requests to the modules containing copies of $x_i$ until it got access to at least $b$ of the $a$ copies of $x_i$, $b < a$. To resolve conflicts arising from colliding requests the modules will work according to the $c$-collision rule or the arbitrary rule. This process is *direct* in a sense introduced by Goldberg et al. [10]. A process for distributing the requests of the processors to the modules is called *direct* if it runs in rounds and in each round the only messages allowed are requests of an arbitrary number of copies of each key.

If we choose, for example, $b > \frac{a}{2}$ then the basic process yields a simulation of an $n$-processor EREW PRAM on an $n$-processor DMM using the trick introduced in [20], which we will call the *majority trick*:
If each shared memory cell possesses $a > 2$ copies distributed among the memory modules of the DMM then it suffices to access arbitrary $\lfloor \frac{a}{2} \rfloor + 1$ out of these $a$ copies to guarantee a correct simulation of both a read and a write step.

To clarify how this trick works suppose that an update of a copy of a key contains a time stamp indicating the update time. If $b > \frac{a}{2}$ copies of a key $x$ are up-to-date then it suffices to access arbitrary

$b$ copies of $x$. This guarantees that at least one up-to-date copy is accessed. It can be recognized by its time stamp.

The read and write accesses to the shared memory can be looked upon as the lookup and update operations of a *parallel dynamic dictionary*, i.e. a data structure that supports the above operations on the given set $U$ of keys. In case we only want to support lookups, we are allowed to execute some preprocessing such that afterwards parallel lookups can be supported efficiently. We refer to such a data structure as a *parallel static dictionary*. In our framework, the static version is easier to handle as all the copies are up-to-date all the time. Thus choosing $b = 1$ is good enough. Furthermore, we can afford a larger storage overhead because we do not have to execute updates.

Our runtime bounds only hold with a certain probability (w.r.t. the choices of the hash functions). By *'with high probability'* (w.h.p.) we mean a probability of at least $1 - \frac{1}{n^\alpha}$ for a fixed $\alpha > 0$.

## 1.3 Previous Results

Shared memory simulations and static dictionaries that use only one hash function to distribute the shared memory cells over the modules of the DMM have an inherent delay of $\Theta(\log n / \log\log n)$ even if the hash function behaves like a random function (see [7]). Faster static dictionaries are only known for PRAMs, see [16] and [3]. On such machines constant access time can be achieved using linear space.

Karp et al. [13] were the first to consider shared memory simulations using two or more hash functions. They also present a fast implementation of write steps. The simulation runs on an arbitrary-DMM with delay $O(\log\log n)$ and can be made time-processor optimal. Dietzfelbinger and Meyer auf der Heide [8] achieve the same delay with a very simple scheme using the majority trick (see Section 1.2) with three hash functions. The scheme can be executed on the weaker $c$-collision DMM with $c \geq 3$. For a survey of shared memory simulations see [14]. MacKenzie et al. [15] analyze processes for accessing 1 out of 2 copies for $c = 2$ and 1 out of 3 copies for $c = 1$. They use these results to obtain processes for accessing $b < a$ out of $a$ copies that have runtime at most $O(a^2 \log\log n)$. Furthermore, they show that an EREW PRAM can be simulated on a 1-collision DMM with storage overhead of at least five. This result was extended to a time-processor optimal simulation of an $n \log\log n$-processor EREW PRAM on an $n$-processor 1-collision DMM by Goldberg et al. [11]. Their simulation uses only three hash functions.

Recently, Czumaj et al. [4] presented a shared memory simulation with delay $O(\log\log\log n \log^* n)$ using only three hash functions. However, they allow non-oblivious communication in their protocol whereas we only consider communication strategies for the analysis of upper and lower bounds that are, apart from the accesses to the copies, independent of the input keys.

## 1.4 New Results

In this paper we focus on the analysis and the implementation of a simple process for shared memory simulations that generalizes the processes and simulations from [13], [8] and [15] mentioned above. We assume that each key has $a$ copies, for some $a \geq 2$, distributed w.r.t. $a$ hash functions, randomly and independently drawn from an $O(\log^3 n)$-universal* class of hash functions.

We introduce a simple *direct* process for accessing $b$ out of $a$ copies for each requested key on the $c$-collision DMM. Informally, we consider processes for this task as direct if apart from the accesses to the copies the communication is independent of the input keys. This process consists of rounds in which the processors simultaneously try to get access to all copies of the requested keys for which they have not been successful so far, until they have accessed at least $b$ copies for each requested key. The modules answer w.r.t. the $c$-collision rule.

We first analyse the above process and show that it finishes within $\frac{\log\log n}{\log(c(a-b))} + 3$ rounds, w.h.p. This generalizes (and improves constant factors of) the analyses from [13], [8] and [15] from 2 or 3 to an arbitrary number $a$ of hash functions.

A straightforward implementation of this process has a running time of $c \cdot a \cdot (\frac{\log\log n}{\log(c(a-b))} + 3)$. This yields static dictionaries with access time $O(\log\log n)$, w.h.p., for example for $a = 2$ and $b = 1$ on

*In this paper log denotes the logarithm with base 2, $\log^k n$ denotes $(\log n)^k$.

a 2-collision DMM or for $a = 4$ and $b = 1$ on a 1-collision DMM, the optical crossbar DMM. With the help of the majority trick (see section 1.2) the process also yields shared memory simulations, for instance for $a = 3$ on a 2-collision DMM or for $a = 7$ on an optical crossbar DMM. The advantage of this kind of shared memory simulation is that the constants in the running time are small.

If we use the arbitrary-DMM we are able to implement a (still direct) variant of this process in a more efficient way, i.e., we show how to speed up the execution of a round of the process. The time needed for this implementation of the process is $O(\log^* n + b + \frac{\log\log n}{\log a})$, w.h.p., if $b \leq (1-\delta)a$ for some constant $\delta > 0$. It yields static dictionaries with access time $O(\log^* n)$ using $a = (\log n)^{1/\log^* n}$ hash functions. For a shared memory simulation we obtain a delay of $O(\frac{\log\log n}{\log\log\log n})$ with, e.g., $\sqrt{\log\log n}$ hash functions.

Finally, we ask whether we can find faster implementations of direct processes than ours. More precisely, we consider a class of so-called *simple access protocols* for shared memory simulations and allow each processor to access several copies of a key in parallel. Furthermore, we allow the processors to communicate in oblivious mode with other processors, i.e., the communication protocol must be independent of the input keys. The information gathered about the topology of the access graph by communicating with other processors may be used to decide which copies to try to access in the next round. We prove that within this class of direct schemes our implementations are optimal.

### 1.5 Organization of the paper

Section 2 contains information about universal hashing, $\log^*$-algorithms, and useful tail estimates that is necessary for the following sections. In Section 3 we introduce the basic process for our simulations and analyze it. Section 4 shows implementations of the process on $c$-collision and arbitrary-DMMs. Section 5 finally contains the lower bound which holds within the class of simple schemes defined above.

## 2 Preliminaries: Universal Hashing, $\log^*$-Algorithms, and Useful Tail Estimates

This section contains information about the classes of hash functions we use to distribute the shared memory, results about fast parallel algorithms we need for allocating work to the processors in our fast simulations, and tail estimates necessary for the analyses of the stochastic processes underlying our simulations.

### 2.1 Universal Hashing

Our simulations use hash functions $h_1, \ldots, h_a : U \to [n]$ that assign to each memory cell $x \in U$ $a$ modules $M_{h_1(x)}, \ldots, M_{h_a(x)}$ in which a copy of $x$ has to be stored. These hash functions have to fulfill seemingly contradicting properties:

They have to behave like random functions and have to be evaluated in constant time while using only little space.

Carter and Wegman [5] introduced the notion of universality for families of hash functions as a measure of quality concerning the first of these demands.

**Definition 2.1** *Let $\mathcal{H}_{p,n}$ be a family of hash functions mapping $[p]$ to $[n]$. $\mathcal{H}_{p,n}$ is called $(\mu,k)$-*universal*, if for any set $\{x_1,\ldots,x_j\} \subset U$ of keys and locations $l_1,\ldots,l_j \in \{1,\ldots,n\}$, $j \leq k$, it holds that, if the hash function $h$ is drawn with uniform probability from $\mathcal{H}_{p,n}$ then*

$$Pr(h(x_1) = l_1 \wedge \ldots \wedge h(x_j) = l_j) \leq \frac{\mu}{n^j} .$$

*If $\mu = 1$ then we simply call $\mathcal{H}_{p,n}$ $k$-*universal*.*

Carter and Wegman introduced a class of linear hash functions in [5] and showed that it is $(2,2)$-universal. In [6] this is generalized to a class of polynomials of degree $d$. This class is proved there to be $(2, d+1)$-universal, the evaluation time is $O(d)$.

Siegel succeeded in [18] to present $\sqrt{n}$-universal classes of hash functions with constant evaluation time, provided that the universe $U$ is of size polynomial in $n$. Techniques presented in [13] show how to use these functions even in the case of an arbitrarily large finite universe:

First apply a randomly chosen function $h$ from a $(2,2)$-universal class to a set $S \subseteq U$, $|S| \leq n$, to a range of size polynomial in $n$. As shown in [6] this mapping is one-to-one, w.h.p. Now apply a randomly chosen function from Siegels class of hash functions on the image of $S$ under $h$.

For our analyses it is sufficient to assume that the hash functions $h_1, \ldots, h_a$ are randomly and independently drawn from an $O(\log^3 n)$-universal class of hash functions. The above discussion shows that using Siegels function together with the extension from [13] yields such a class with constant evaluation time using little space ($O(\sqrt{n})$ space works). For a detailed description of this class see [13].

## 2.2 $\log^*$-Algorithms

We only consider the *linear approximate compaction* (LAC) problem here. The algorithm solving this problem will be used as a basis in Section 4 to obtain very fast simulations on an arbitrary-DMM.

**Definition 2.2** *Let $m \leq n$ denote the number of keys distributed among $n$ processors such that each processor has at most one key. The* linear approximate compaction *problem is to insert the keys into an array $B$ of size $4m$.*

In [3] and [16] a randomized algorithm for the LAC problem is presented and analyzed. Their algorithms are designed for a CRCW-PRAM that uses $n$ processors and $O(n)$ shared memory cells. As noted in [13], such a PRAM can be simulated on an $n$-processor arbitrary-DMM with constant delay. So we get the following result:

**Theorem 2.3** *The LAC problem can be solved by a randomized arbitrary-DMM with $n$ processors in time $O(\log^* m)$, w.h.p.*

In our algorithms we also need information about intermediate time steps of the above algorithm. Therefore we will shortly describe it.

Initially, each cell of array $B$ is empty. Throughout the algorithm we map more and more items to accomplish a 1-1 embedding into array $B$. An unmapped key is called *active*. Once a key is mapped it becomes *inactive*. Initially, all keys are active.

The algorithm for the LAC-problem consists of $O(\log^* m)$ iterations. Let $q_1 = 1$ and $q_{i+1} = 2^{q_i}$ for $i \geq 1$. Let $d \geq 2$ be some constant (for suitable values for $d$ see [3] or [16]).

**High-level description of iteration $i$:**
The main idea is to enhance the mapping of active keys by reallocating many processors to them.

Input of iteration $i$: the number of active keys is assumed to be at most $m/q_i^d$.
The iteration consists of two basic steps:

- (allocation) Allocate $dq_i$ processors to each active key. Each active key that indeed gets $dq_i$ processors is called *participating*.
- (mapping) Map each participating key to a different empty cell of $B$. Each successful participating key becomes inactive.

Output of the iteration: The number of active keys to be at most $m/(2^{q_i})^d = m/q_{i+1}^d$ with high probability.

Let us call the allocation step of iteration $i$ to be *successful* if the number of non-participating active keys is at most $\frac{1}{2}m2^{-dq_i}$. Analogous, the mapping step of iteration $i$ is said to be *successful* if the number of participating keys that remain active is at most $\frac{1}{2}m2^{-dq_i}$. Then the following lemma holds which is implicitly proved in [3] and [16].

**Lemma 2.4** *Suppose that iteration $i-1$ was successful, that is, at most $m/q_i^d$ keys are still active after iteration $i-1$, $1 \leq i \leq \log^* m$. Then, for any constant $d > 0$ and sufficiently large $m$, there exists a constant $\epsilon > 0$ such that both the allocation step and mapping step are successful with probability at least $1 - 2^{-m^\epsilon}$, using only a constant number of time steps.*

## 2.3 Useful Tail Estimates

There are two tail estimates we will use in this paper. First, we state a result from [17] which generalizes the Chernoff-Hoeffding bounds for the sums of certain types of dependent random variables. This will be used in the lower bound proof in Section 5.

**Definition 2.5** *Given $n$ 0-1 random variables $X_1, \ldots X_n$, we call them self-weakening if for all $i, 1 \leq i \leq n$ and all subsets $\{j_1, \ldots, j_k\} \subseteq \{1, 2, \ldots, i-1\}$ it holds:*

$$P(X_i = 1 | X_{j_1} = X_{j_2} = \cdots = X_{j_k} = 1) \leq P(X_i = 1) \ .$$

**Lemma 2.6** *Consider $n$ 0-1 random variables $X_1, X_2, \ldots, X_n$ which are self-weakening. Let $X = \sum_{i=1}^n X_i$ and let $E(X) \leq \mu^*$, for some $\mu^*$. Then*

$$P(X \geq (1+\epsilon)\mu^*) \leq \left( \frac{e^\epsilon}{(1+\epsilon)^{(1+\epsilon)}} \right)^{\mu^*} \leq e^{\epsilon^2 \mu^*/3} \ .$$

**Proof**. For a proof see [17] and [12]. ∎

Second, we use the General Markov Inequality (see, e.g., [19]) to obtain bounds for sums of $s$-wise independent binary random variables.

**Lemma 2.7** *(General Markov Inequality)*
*Let $X$ be an arbitrary random variable then, for every $\epsilon > 0$ and $k \geq 0$, it holds:*

$$Prob\left( |X - E(X)| \geq \epsilon \sqrt[k]{E(|X - E(X)|^k)} \right) \leq \left( \frac{1}{\epsilon} \right)^k \ .$$

The following more technical tail estimate will be used in the analysis of our protocols in Section 3.

**Lemma 2.8** *Let $X_1, \ldots, X_m$ be (not necessarily independent) binary random variables, let $X = \sum_{i=1}^m X_i$. Furthermore, let $\text{Prob}(X_{i_1} = \ldots = X_{i_s} = 1) \leq p^s$ for all $\{i_1, \ldots, i_s\} \subseteq \{1, \ldots, m\}$, $1 \leq s \leq \lceil \alpha \log m + 1 \rceil$, let $\frac{\lceil \alpha \log m + 1 \rceil}{m} \leq p < 1$ for a constant $\alpha > 0$. Then it holds for every $\epsilon > 0$:*

$$Prob(|X - E(X)| \geq \epsilon \cdot m \cdot p) \leq \left( \frac{1}{m} \right)^{\alpha(\log \epsilon - 1)} \ .$$

**Proof**. We use here the General Markov Inequality described above.
Let $k \in \{\lceil \alpha \log m \rceil, \lceil \alpha \log m + 1 \rceil\}$ be even for a constant $\alpha > 0$. Then it holds

$$\begin{aligned} E(|X - E(X)|^k) &= E((X - E(X))^k) = E\left( \sum_{i=0}^k \binom{k}{i} X^i (-E(X))^{k-i} \right) \\ &= \sum_{i=0}^k \binom{k}{i} E(X^i)(-E(X))^{k-i} \ . \qquad (*) \end{aligned}$$

Furthermore we get

$$E(X^i) = E\left( \left( \sum_{j=1}^m X_j \right)^i \right) = \sum_{\substack{s_1, \ldots, s_m \in \{0, \ldots, i\}, \\ s_1 + \ldots + s_m = i}} \frac{i!}{s_1! \cdot \ldots \cdot s_m!} E(X_1^{s_1} \cdot \ldots \cdot X_m^{s_m}) \ .$$

Let us replace the $X_1, \ldots, X_m$ by random variables $Y_1, \ldots, Y_m$ satisfying

$$\begin{aligned} &\text{Prob}(Y_{i_1} = Y_{i_2} = \ldots = Y_{i_s} = 1) \\ &= \begin{cases} p^s & : \quad s \leq \lceil \alpha \log m + 1 \rceil \\ \text{Prob}(X_{i_1} = X_{i_2} = \ldots = X_{i_s} = 1) & : \quad s > \lceil \alpha \log m + 1 \rceil \end{cases} \end{aligned}$$

for all subsets $\{i_1, i_2, \ldots, i_s\} \subseteq \{1, \ldots, m\}$. According to the definition of the $Y_1, \ldots, Y_m$ we get for all $i_1, \ldots, i_k \in \{1, \ldots, m\}$ and $s_1, \ldots, s_k \geq 1$ with $k \leq \lceil \alpha \log m + 1 \rceil$:

$$E(Y_{i_1}^{s_1} Y_{i_2}^{s_2} \ldots Y_{i_k}^{s_k}) = E(Y_1 Y_2 \ldots Y_k) \ .$$

Therefore we can simplify $E(X^i)$ to

$$E(X^i) \leq \sum_{j=1}^{i} \binom{m}{j} \cdot s(i,j) \cdot E(Y_1 \cdot \ldots \cdot Y_j) \ ,$$

where $s(i,j) = \sum_{l=0}^{j} (-1)^l \binom{j}{l} (j-l)^i$ is the number of surjective mappings from $\{1, \ldots, i\}$ to $\{1, \ldots, j\}$. Since $s(i,j) \leq j^i$ for all $i, j$ and $p \geq \frac{k}{m}$ it follows:

$$\begin{aligned} E(X^i) &\leq \sum_{j=1}^{i} \binom{m}{j} j^i \cdot p^j \\ &\leq \sum_{j=1}^{i} \left( \frac{e \cdot m}{j} \right)^j j^i \cdot p^j \ = \ m^i \sum_{j=1}^{i} e^j \left( \frac{j}{m} \right)^{i-j} p^j \\ &\leq m^i \sum_{j=1}^{i} e^j \cdot p^i \ \leq \ 2 \left( e \cdot m \cdot p \right)^i \ . \end{aligned}$$

Using this inequality in equation $(*)$ yields

$$\begin{aligned} E(|X - E(X)|^k) &\leq (m \cdot p)^k + \sum_{i=1}^{k} \binom{k}{i} 2 \left( e \cdot m \cdot p \right)^i \cdot (-m \cdot p)^{k-i} \\ &\leq (m \cdot p)^k + 2((e-1)m \cdot p)^k - 2(m \cdot p)^k \\ &\leq 2 \left( (e-1)m \cdot p \right)^k \ . \end{aligned}$$

Thus it holds

$$\sqrt[k]{E(|X - E(X)|^k)} \leq 2m \cdot p$$

for sufficiently large $m$ (note that $k \in \{\lceil \alpha \log m \rceil, \lceil \alpha \log m + 1 \rceil\}$). With the help of the General Markov Inequality we get for every $\epsilon > 0$:

$$\text{Prob}(|X - E(X)| \geq \epsilon \cdot 2m \cdot p) \leq \left( \frac{1}{\epsilon} \right)^{\alpha \log m}$$

and with $\epsilon' := 2\epsilon$:

$$\text{Prob}(|X - E(X)| \geq \epsilon' \cdot m \cdot p) \leq \left( \frac{2}{\epsilon'} \right)^{\alpha \log m} \leq \left( \frac{1}{m} \right)^{\alpha (\log \epsilon' - 1)} \ .$$

■

# 3 The Basic Process

In order to get a clean description of our process we assume a DMM with $n$ processors and $a \cdot n$ memory modules $M_{j,k}$, $j \in \{1,\ldots,a\}$, $k \in \{1,\ldots,n\}$. Suppose that $a$ hash functions $h_1,\ldots,h_a : U \to \{1,\ldots,n\}$ distribute the keys from $U$ among the modules, so each key $x$ has $a$ copies stored in $M_{1,h_1(x)},\ldots,M_{a,h_a(x)}$. Let $x_1,\ldots,x_{\epsilon n} \in U$, $\epsilon \leq 1$, be some keys for which we want to get access to at least $b$ out of $a$ copies. Let $I_i \subseteq \{1,\ldots,a\}$, $i \in \{1,\ldots,\epsilon n\}$, be the set of all copies of $x_i$ for which we have already been successful. The following process forms the basis of our algorithms to implement static dictionaries and shared memory simulations.

$(n,\epsilon,a,b,c)$**-process:**

initially: active keys $x_1,\ldots,x_{\epsilon n}$, $I_1 = \ldots = I_{\epsilon n} = \emptyset$

execute the following round until all keys are inactive:
  **for each** $j \in \{1,\ldots,a\}$:
    **for each** active key $x_i$ with $j \notin I_i$:
      $x_i$ tries to access $M_{j,h_j(x_i)}$
      { each module $M_{j,l}$ works according to the $c$-collision rule }
      **if** $x_i$'s access is accepted **then** $I_i := I_i \cup \{j\}$
      **if** $|I_i| \geq b$ **then** $x_i$ becomes inactive

The following holds for the number of rounds needed by our process.

**Main Theorem:** *Let $h_1,\ldots,h_a : U \to \{1,\ldots,n\}$ be randomly and independently chosen from a $O(log^3 n)$-universal class of hash functions. Let $0 < \epsilon \leq 1$, $2 \leq a \leq \sqrt{\log n}$, $b < a$ and $c = O((\frac{\sqrt{\log n}}{a-b})^{1/3})$ be chosen such that*

$$\frac{c^2(a-b)}{c+1} > 1+\delta \quad \textit{and} \quad \epsilon \cdot \binom{a-1}{a-b}\left(\frac{1}{c!}\right)^{a-b} \leq \frac{1}{2}$$

*for some constant $\delta > 0$.*

*(a) Then, for each $t$, $4 \leq t \leq \frac{\log\log n}{\log(c(a-b))} + 1$, and $c \geq \frac{\log(4a)}{a-b}$ at most $n/2^{(c(a-b))^{t-2}}$ keys are still active after $t$ rounds of the $(n,\epsilon,a,b,c)$-process, w.h.p.*

*(b) In particular, the $(n,\epsilon,a,b,c)$-process finishes within $\frac{\log\log n}{\log(c(a-b))} + 3$ rounds, w.h.p.*

**Proof**. Let $x_1,..,x_{\epsilon n} \in U$, $\epsilon \in (0,1]$. A module is called *blocked* at round $t$ of the $(n,\epsilon,a,b,c)$-process if it gets more than $c$ requests at round $t$. Note that a module will be blocked consecutively until it gets at most $c$ requests for the first time. In this round it answers all of them. Afterwards, it gets no request any more.

We view the distribution of the requests among the modules as a graph, the *access graph* $G = (V_K, V_M, E)$. $V_K = \{1,\ldots,\epsilon n\}$ represents the keys $x_1,\ldots,x_{\epsilon n}$ and $V_M = \{(j,k) \mid j \in \{1,\ldots,a\}, k \in \{1,\ldots,n\}\}$ the modules $M_{1,1},\ldots,M_{a,n}$. Node $i \in V_K$ is connected with $(j,k) \in V_M$ in $G$ if $h_j(x_i) = k$, that is, $M_{j,k}$ possesses the $j$-th copy of the key $x_i$. Thus $E = \{\{i,(j,k)\} \mid i \in V_K, (j,k) \in V_M, h_j(x_i) = k\}$.

Assume now that a module $M_{j,k}$ is still blocked at round $t$ of the $(n,\epsilon,a,b,c)$-process. Then $M_{j,k}$ must have received more than $c$ requests at round $t$. So there must have been at least $c+1$ keys $x_{i_1},\ldots,x_{i_{c+1}}$ with $h_j(x_{i_l}) = k$ for all $l \in \{1,\ldots,c+1\}$ that were still active at round $t$. Let $x_i$ be one of those keys. As $x_i$ was still active at round $t$ there must have been at least $a-b$ hash functions $h_{j_1},\ldots,h_{j_{a-b}}$ in addition to $h_j$ for which $x_i$ was not successful at round $t-1$. In other words, modules $M_{j_1,h_{j_1}(x_i)},\ldots,M_{j_{a-b},h_{j_{a-b}}(x_i)}$ must have been blocked at round $t-1$. Let $M_{j',k'}$ be one of those modules. Continuing with the argumentation for $M_{j',k'}$ as we did for $M_{j,k}$ we find:
If module $M_{j,k}$ is still blocked at round $t$ of the $(n,\epsilon,a,b,c)$-process, the following tree can be embedded

into $G$ such that its root is embedded in the $V_M$-node $(j,k)$, $V_{TM}$-nodes are mapped to $V_M$-nodes and $V_{TK}$-nodes are mapped to $V_K$-nodes.

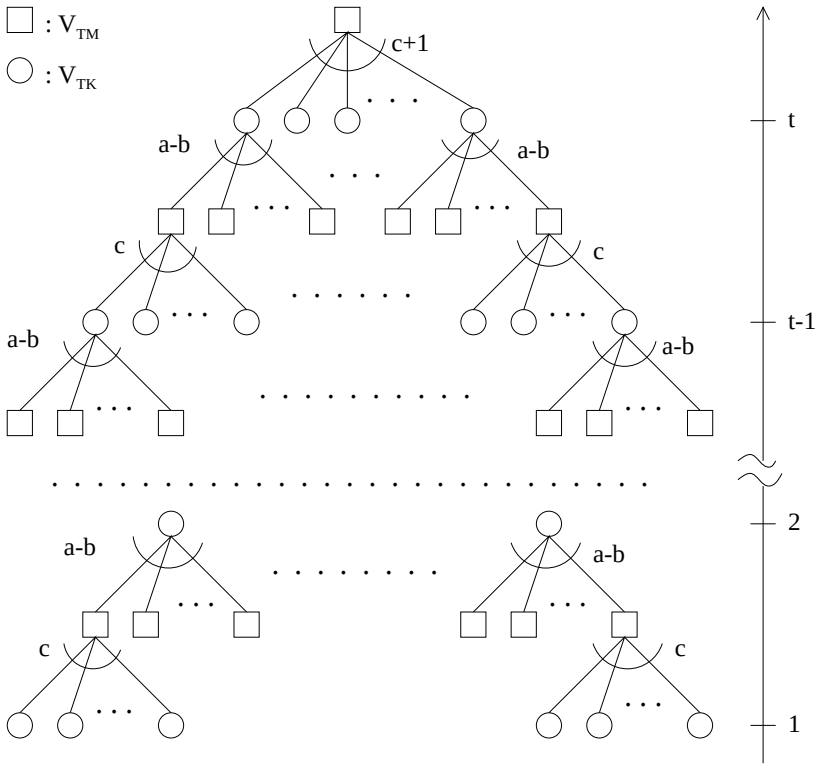


**Figure 1: A witness tree**

Let us call this tree $T_t = (V_{TK}, V_{TM}, E_T)$ a *witness tree* of depth $t$. Note that, if two $V_{TK}$-nodes $v$ and $w$ are embedded into the same $V_K$-node, then we can *identically* embed modules and keys into nodes of $T_t$ below $v$ and $w$ in a way we just described above. The following definition formalizes what kind of embeddings of $T_t$ into $G$ we only have to consider.

**Definition 3.1** *An embedding $\phi$ of $T_t$ into $G$ is called* valid *if*

*(1) for each $v \in V_{TM}$, all neighbours $v'$ of $v$ are embedded into different $V_K$-nodes,*

*(2) for each $v \in V_{TK}$, all neighbours $v'$ of $v$ are embedded into $V_M$-nodes $(j,k)$ with different $j$,*

*(3) for each $\{u,v\} \in E_T$ we have $\{\phi(u), \phi(v)\} \in E$.*

*(4) For each $v, w \in V_{TK}$ let $T_v$, $T_w$ be the subtrees of $T_t$ rooted in $v$ and $w$, resp., $d = depth(T_v) \leq depth(T_w)$. If $v$ and $w$ are embedded identically then $T_v$ is embedded identical to the top part of $T_w$ of depth $d$.*

The following lemma is a direct result of our discussion above.

**Lemma 3.2** *If module $M_{j,k}$ is still blocked at round $t$ of the $(n, \epsilon, a, b, c)$-process then there is a valid embedding $\phi$ of $T_t$ into $G$ that maps the root of $T_t$ to $M_{j,k}$.*

The above lemma implies that it suffices to find a suitable upper bound for the probability (w.r.t. random choices of $h_i$'s as indicated in the Main Theorem) that $T_t$ has a valid embedding into $G$ in order to prove part (b) of the Main Theorem.

From the above definition of valid embeddings one can conclude that a valid embedding of $T_t$ into $G$ is already completely described by a small (at most size $\epsilon|V_{TK}|$) subtree $\tilde{T}$ of $T_t$, called *base tree*, and a set $\mathcal{E}$ of $V_{TK}$-nodes below $\tilde{T}$, called *set of expansion nodes*, with the following properties:

(1) $\tilde{T}$ and $T_t$ have the same root,

(2) no two $V_{TK}$-nodes in $\tilde{T}$ are embedded into the same $V_K$-node,

(3) if a $V_{TK}$-node is a leaf of $\tilde{T}$ then it is a leaf of $T_t$ (that is, in the inner part of $T_t$, $\tilde{T}$ can only have $V_{TM}$-nodes as leaves),

(4) all $V_{TK}$-nodes in

$$\mathcal{E} := \{w \in V_{TK} \mid w \text{ is a son of a leaf in } \tilde{T}\}$$

are embedded in $V_K$-nodes that are already used for the embedding of $V_{TK}(\tilde{T})$,

(5) there are no nodes $v \in V_{TK}(\tilde{T})$ and $w \in \mathcal{E}$ with $\phi(v) = \phi(w)$, but the distance from $v$ to the root of $T_t$ is greater than from $w$ to the root of $T_t$.

It is easy to write a breadth first search algorithm which finds a base tree $\tilde{T}$ and a node set $\mathcal{E}$ in $T_t$ for every valid node embedding $\phi$. The following picture will give an example for such a subtree for $a = 7$, $b = 6$ and $c = 2$.

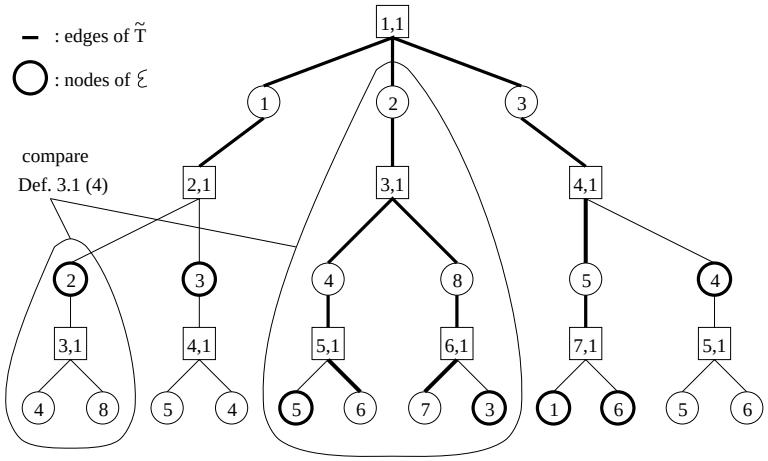


**Figure 2: A valid embedding of a witness tree**

Note that $\mathcal{E}$ can be empty. Suppose now that we create a valid node embedding for a subtree $\tilde{T}$ of $T_t$ and a node set $\mathcal{E}$ such that the requirements above are fulfilled. Then we can find an embedding for the rest of $T_t$ by first copying the embedding of subtrees of $\tilde{T}$ below the expansion nodes such that the embedding of the roots of the subtrees coincides with the embedding of the expansion nodes and continue copying until every node of $T_t$ is embedded. This process has the effect that the rest of $T_t$ is folded back into the subgraph over nodes of $G$ induced by the embedding of $\tilde{T}$ and $\mathcal{E}$.

These observations allow us to use an (involved) counting argument to show the following probability bounds. The proof can be found in Section 3.1.

**Lemma 3.3** *Let $h_1, \ldots, h_a$, $\epsilon$, $a$, $b$ and $c$ be chosen as in the Main Theorem. Consider a module $M_{j,k}$. Then it holds :*

$$\textit{Prob( There is a valid embedding of } T_t \textit{ in } G \textit{ with root embedding } M_{j,k} \textit{ )}$$

$$\leq \begin{cases} \left(\frac{1}{2}\right)^{\sum_{j=1}^{t-2}(c(a-b))^j} & : \quad 2 \leq t \leq \frac{\log\log n}{\log(c(a-b))} + 2 \\ (\log n)^2 \cdot \left(\frac{1}{n}\right)^{\frac{c^2(a-b)}{c+1}} & : \quad t \geq \frac{\log\log n}{\log(c(a-b))} + 3 \,. \end{cases}$$

The reason why this probability does not approach 0 as $t$ goes to infinity is that structures may occur in the access graph that prevent the $(n, \epsilon, a, b, c)$-process from terminating, e.g. a subset $U \subseteq V_K$ such that all $V_M$-nodes in $G|_{U \cup \Gamma(U)}$ have degree at least $c + 1$, where $\Gamma(U) = \{(j,k) \mid i \in U, j \in$

$\{1,\ldots,a\}, h_j(x_i) = k\}$. Thus $(\log n)^2 \cdot (\frac{1}{n})^{c^2(a-b)/(c+1)}$ is an upper bound for the probability that such structures occur in the access graph.

Part (b) of the Main Theorem follows directly from the above two lemmas. For the proof of part (a) of the Main Theorem we first observe that, by Lemma 3.3, the expected number of modules still blocked at round $t$, $2 \leq t \leq \frac{\log\log n}{\log(c(a-b))} + 2$, is at most $a \cdot n/2^{\sum_{j=1}^{t-2}(c(a-b))^j}$. Thus part (a) of the Main Theorem would easily follow by the well-known Chernoff bound if the events "There is a valid embedding of $T_t$ in $G$ with root embedding $M_{j,k}$" were independent for different $(j,k)$. This is not true because the embeddings may overlap. On the other hand, we can show that an overlap of more than one node is very unlikely, as long as we only consider up to $\alpha \log n$ many $(j,k)$ for some constant $\alpha > 0$. Let

$$p_{a,t} := \frac{(\log n)^2}{2} \cdot \left(\frac{1}{n}\right)^{\frac{c^2(a-b)}{c+1}} + \frac{2}{3} \cdot \left(\frac{1}{2}\right)^{c-1+\sum_{j=1}^{t-2}(c(a-b))^j}.$$

Formally, we prove in Section 3.2:

**Lemma 3.4** *For any $\{(j_1,k_1),\ldots,(j_s,k_s)\} \subseteq \{1,\ldots,a\}\times\{1,\ldots,n\}$ with $s \leq \alpha \log n$, $\alpha > 0$ constant, it holds for all $t$, $2 \leq t \leq \frac{\log\log n}{\log(c(a-b))} + 1$:*

$$\mathit{Prob}(M_{j_1,k_1},\ldots,M_{j_s,k_s} \text{ are blocked at round } t) \leq \left(p_{a,t} + \frac{1}{n}\right)^s.$$

Now we can complete the proof of part (a) of the Main Theorem. Let $X_{j,k}^t$ be defined as

$$X_{j,k}^t = \begin{cases} 1 & : \text{ module } M_{j,k} \text{ is blocked at round } t \\ 0 & : \text{ otherwise} \end{cases}$$

for all $j \in \{1,\ldots,a\}$, $k \in \{1,\ldots,n\}$, and let $X_t = \sum_{j=1}^{a}\sum_{k=1}^{n} X_{j,k}^t$. From the proof of Lemma 3.3 (see Section 3.1) we know that $\mathrm{Prob}(X_{j,k}^t = 1) \leq p_{a,t}$. Let $t \leq \frac{\log\log n}{\log(c(a-b))} + 1$. Then it holds with $c$ chosen as in part (a) of the Main Theorem:

$$\begin{aligned} p_{a,t} &\geq \left(\frac{1}{2}\right)^{c-1+\frac{(c(a-b))^{t-1}-1}{c(a-b)-1}-1} \geq \left(\frac{1}{2}\right)^{c-1+\frac{2^{\log\log n}-1}{c(a-b)-1}-1} \\ &\geq \left(\frac{1}{2}\right)^{c-2+\frac{\log n}{2}} = \frac{2^{2-c}\sqrt{n}}{n}. \end{aligned}$$

Thus Lemma 3.4 and Lemma 2.8 can be used to prove that, for $c \geq \frac{\log(4a)}{a-b}$ and for all $4 \leq t \leq \frac{\log\log n}{\log(c(a-b))} + 1$ it holds:

$$\begin{aligned} \mathrm{Prob}(X_t \geq \frac{n}{c \cdot 2^{(c(a-b))^{t-2}}}) &\leq \mathrm{Prob}(X_t \geq \frac{4a \cdot n}{2^{c-1} \cdot 2^{c(a-b)} \cdot 2^{(c(a-b))^{t-2}}}) \\ &\leq \mathrm{Prob}(X_t \geq 4a \cdot n \cdot p_{a,t}) \leq \left(\frac{1}{a \cdot n}\right)^{\alpha}. \end{aligned}$$

for $\alpha$ chosen as in Lemma 3.4. Hence, for $t$, $a$, $b$ and $c$ chosen as in the Main Theorem, at most $n/(c \cdot 2^{(c(a-b))^{t-2}})$ modules are still blocked at round $t$ of the $(n,\epsilon,a,b,c)$-process, w.h.p.

If the $(n,\epsilon,a,b,c)$-process terminates then each module has to answer at most $c$ requests. (Note that it can happen that the process does not terminate. In this case the access graph contains a highly connected subgraph, e.g. a graph $G|_{U\cup\Gamma(U)}$ for some subset $U \subseteq V_K$ such that all $V_M$-nodes in it have degree at least $c+1$. The choice of $\epsilon$ and $c$ relative to $a$ and $b$ makes this event sufficiently unlikely.) From this we can conclude that, if at round $t$ of the $(n,\epsilon,a,b,c)$-process $m$ modules are still blocked and the $(n,\epsilon,a,b,c)$-process terminates, then at most $c \cdot m$ keys are still active after round $t$. Thus we get with the help of the estimate above:

$$\mathrm{Prob}\left(\geq \frac{n}{2^{(c(a-b))^{t-2}}} \text{ keys are still active after round } t\right) \leq \left(\frac{1}{a \cdot n}\right)^{\alpha}.$$

Note that we only consider events with $\alpha \log n \cdot |E_T| = O(\log^3 n)$ values of hash functions. Thus for the analysis $O(\log^3 n)$-universal hash functions are sufficient. ∎

It remains to prove Lemma 3.3 and Lemma 3.4.

## 3.1 Proof of Lemma 3.3

Let us first introduce the following notations :

- By $B_{k,i}$ we denote the set of all possible base trees with $k$ $V_{TK}$-nodes such that $i$ of these $k$ nodes are *not* leaves of $T_t$.
- An embedding $\phi$ of $T_t$ into $G$ is called *valid node embedding* if all items in Definition 3.1 are fulfilled except item (3). If all items are fulfilled we simply call $\phi$ a *valid embedding*.
- For every $\tilde{T} \in B_{k,i}$ let $\Phi_{\tilde{T}}$ be the set of all valid node embeddings of $\tilde{T}$ and the set of expansion nodes $\mathcal{E}$ induced by $\tilde{T}$. Furthermore let
$$\phi_{k,i} := \max\{|\Phi_{\tilde{T}}| \mid \tilde{T} \in B_{k,i}\} .$$
- Let $h_1, \ldots, h_a$ be randomly and independently chosen hash functions. Then for every tuple $(\tilde{T}, \varphi)$ of base trees $\tilde{T} \in B_{k,i}$ and valid node embeddings $\varphi \in \Phi_{\tilde{T}}$ the upper bound for the probability that $\varphi$ is a valid embedding will be denoted by $p_{k,i}$.
- For each node $v \in T_t$ let $\Gamma(v)$ denote the set of all direct neighbours of $v$ in $T_t$ and $\gamma(v)$ denote the set of all sons of $v$ in $T_t$.

Let $k$ and $i$ be fixed. Then the probability that there is a base tree $\tilde{T} \in B_{k,i}$ and an embedding $\varphi \in \Phi_{\tilde{T}}$ such that $T_t$ can be embedded into $G$ is at most
$$|B_{k,i}| \cdot \phi_{k,i} \cdot p_{k,i} .$$
In the next three propositions bounds for $|B_{k,i}|$, $\phi_{k,i}$ and $p_{k,i}$ will be presented.

**Proposition 3.5** *Let r(k,i) be the number of expansion nodes induced by any base tree in $B_{k,i}$. Then we have*
$$r(k,i) = i \cdot c(a-b) - k + c + 1 ,$$
$$|B_{k,i}| \leq \binom{i \cdot c(a-b)}{r(k,i)} .$$

**Proof**. Suppose a base tree $\tilde{T}$ comprises $i$ inner $V_{TK}$-nodes of $T_t$. For each of these $i$ nodes $v$, $|\gamma(\gamma(v))| = c(a-b)$, for any other node $v \in V_{TK}(\tilde{T})$ we have $|\gamma(\gamma(v))| = 0$, because the others must be leaves of $T_t$. So $|\gamma(\gamma(V_{TK}(\tilde{T})))| = i \cdot c(a-b)$. Because $\tilde{T}$ is connected, $k-(c+1)$ of these $i \cdot c(a-b)$ nodes belong to $\tilde{T}$. (Note that the top $c+1$ $V_{TK}$-nodes always belong to $\tilde{T}$.) The others by definition must be the expansion nodes.

It remains to show that it suffices to count all possibilities to label $r(k,i)$ out of all $i \cdot c(a-b)$ grandsons of $i$ $V_{TK}$-nodes as expansion nodes in order to count all possible base trees $\tilde{T}$ in $B_{k,i}$. For this purpose we need the following definition.

**Definition 3.6** *Let $\tilde{T}$ be a base tree in $B_{k,i}$. For each $V_{TK}$-node $u$ in $T_t$ let $p_v(u)$ be the distance of $u$ from the root of $T_t$ and $p_h(u)$ be the position of $u$ in its row. (If $p_v(u) = r$ than there are $(c+1)(c(a-b))^{(r-1)/2}$ $V_{TK}$-nodes in the same row as $u$. So the leftmost node $w$ in this row has $p_h(w) = 1$, the rightmost node $w$ has $p_h(w) = (c+1)(c(a-b))^{(r-1)/2}$.) Let '$<$' be a relation over the set of $V_{TK}$-nodes of $\tilde{T}$ with*
$$v_1 < v_2 \quad :\Leftrightarrow \quad p_v(v_1) < p_v(v_2) \text{ or } [p_v(v_1) = p_v(v_2) \text{ and } p_h(v_1) < p_h(v_2)].$$
*Then we will call the node sequence $(v_1, v_2, \ldots, v_k)$ consisting of all $V_{TK}$-nodes in $\tilde{T}$* sorted, *if for every $i \in \{1, \ldots, k-1\}$ we have $v_i < v_{i+1}$.*

Suppose we have two different base trees $\tilde{T}_1, \tilde{T}_2 \in B_{k,i}$. Let $(v_1, \ldots, v_k)$ be the sorted sequence of $V_{TK}$-nodes of $\tilde{T}_1$ and $(w_1, \ldots, w_k)$ be the sorted sequence of $V_{TK}$-nodes of $\tilde{T}_2$. Because $\tilde{T}_1$ and $\tilde{T}_2$ are different there must be an $l \in \{1, \ldots, k\}$ with $v_l \neq w_l$.

Let $m = \min\{l \in \{1, \ldots, k\} \mid v_l \neq w_l\}$, let without loss of generality be $v_m < w_m$. According to the definition of base trees the highest row of $V_{TK}$-nodes belongs to every base tree, so $m > c+1$. Then $v_m$ and $w_m$ must have a common $V_{TK}$-node $v_j = w_j$ as grandfather, $j < m$, otherwise $m$ is not chosen correctly. Let $v_m$ occupy position $p_1$ and $w_m$ occupy position $p_2$ out of $c(a-b)$ possible below $v_j$. Then because of $v_m < w_m$ it must hold $p_1 < p_2$ which means that at the position of $v_m$ in $\tilde{T}_2$ there must be an expansion node. Because the expansion nodes are distributed different below $v_j$ from below $w_j$ we have that any two different base trees must have different distributions of expansion nodes below their $V_{TK}$-nodes.

So we proved that if two base trees $\tilde{T}_1, \tilde{T}_2 \in B_{k,i}$ are different then the distribution of the $r(k,i)$ expansion nodes over the $ic(a-b)$ grandsons of their $i$ inner $V_{TK}$-nodes must be also different. Thus $\binom{ic(a-b)}{r(k,i)}$ is an upper bound for the number of base trees in $B_{k,i}$.

∎

**Proposition 3.7**

$$\phi_{k,i} \leq \underbrace{\binom{\epsilon n}{k}}_{(1)} \underbrace{\binom{k}{c+1}(k-(c+1))!}_{(2)} \cdot \underbrace{\left[\binom{a-1}{a-b} n^{a-b}\right]^i}_{(3)} \underbrace{k^{r(k,i)}}_{(4)} \underbrace{\left(\frac{1}{c!}\right)^{i(a-b)}}_{(5)}.$$

**Proof**.

(1) There are $\binom{\epsilon n}{k}$ possibilities to choose $k$ $V_K$-nodes of $G$ out of $\epsilon n$ for an injective embedding of $V_{TK}(\tilde{T})$ into $V_K$.

(2) There are $\binom{k}{c+1}$ possibilities to embed $c+1$ out of $k$ nodes chosen in (1) into the highest row of $V_{TK}$-nodes in $\tilde{T}$ without getting redundancies. Moreover there are $(k-(c+1))!$ possibilities to distribute the remaining $V_K$-nodes over the remaining $V_{TK}$-nodes of $\tilde{T}$.

(3) For every $V_{TK}$-node $v$ of $\tilde{T}$ which belongs to the inner nodes of $T_t$ there are $\binom{a-1}{a-b} n^{a-b}$ possibilities to embed the $a-b$ $V_{TM}$-nodes under $v$ into $V_M$-nodes of $G$ such that we have a valid node embedding.

(4) There are at most $k$ possibilities for each of the $r(k,i)$ expansion nodes to embed it into one of the $k$ nodes chosen in (1).

(5) Let $\tilde{T}$ be a base tree with $i$ inner $V_{TK}$-nodes of $T_t$. Each of these nodes $v$ is connected with a set $\gamma(v)$ of $(a-b)$ $V_{TM}$-nodes. Every node $w$ of these $i(a-b)$ $V_{TM}$-nodes has a set $\gamma(w)$ of $c$ $V_{TK}$-nodes which were embedded either as a node of $\tilde{T}$ by the term $(k-(c+1))!$ or as an expansion node by the term $k^{r(k,i)}$. This kind of embedding creates superfluous permutations which we want to eliminate by $(1/c!)^{i(a-b)}$.

∎

**Proposition 3.8**

$$p_{k,i} \leq \left(\frac{1}{n}\right)^{i(a-b+1)+(k-i)} \cdot \left(\frac{1}{n}\right)^{\left\lceil \frac{r(k,i)}{c+1} \right\rceil}.$$

**Proof**. Let $\tilde{T} \in B_{k,i}$ and $\varphi \in \Phi_{\tilde{T}}$. The $i$ inner nodes $v$ of $T_t$ in $\tilde{T}$ have degree $(a-b+1)$ and are embedded into $i$ different $V_K$-nodes. Therefore the probability that each one of these $V_K$-nodes has the same links to $V_M$-nodes in $G$ as induced by the embedding of $\Gamma(v)$ in $\tilde{T}$ is $n^{-i(a-b+1)}$. The remaining $(k-i)$ nodes in $\tilde{T}$ are leaves of $T_t$ and thus have degree 1. Hence the probability that the links induced by an embedding $\varphi$ of $\tilde{T}$ are the same as in $G$ is $n^{-(k-i)}$.

It remains to show that we have an additional probability of $n^{-\lceil r(k,i)/(c+1)\rceil}$ for the expansion nodes. If one considers only the embedding of $\tilde{T}$ without its set of expansion nodes then for all $V_M$-nodes $(j,k)$ embedded into $V_{TM}$-nodes of $\tilde{T}$ the values for $k$ are independent from each other. Otherwise there must exist a $V_K$-node embedded in $\tilde{T}$ which is adjacent to two $V_{TM}$-nodes with embeddings in $(j,k)$ and $(j',k')$ such that $j = j'$. This obviously violates the conditions a valid node embedding has to fulfill.

Let $\mathcal{I}$ be the set of all $x \in \mathcal{E}$ with father $w$ that have

- no node $y \in V_{TK}(\tilde{T})$, $\varphi(x) = \varphi(y)$, with a $V_M$-node embedded in a node in $\Gamma(y)$ which lies in the same hash function as $\varphi(w)$ and
- no node $y \in \mathcal{E}$ with father $v$ such that $\varphi(x) = \varphi(y)$, $\varphi(v)$ and $\varphi(w)$ lie in the same hash function and $y < x$ (for $'<'$ defined as in Definition 3.6)

In other words, $\mathcal{I}$ consists of all expansion nodes of lowest order according to '$<$' that do not demand an equality between the embeddings of any two $V_{TM}$-nodes in $\tilde{T}$, but all other nodes in $\mathcal{E} \setminus \mathcal{I}$ do to maintain the status of a valid embedding for $\varphi$. Instead, the embedding of each node in $\mathcal{I}$ induces a new condition for the edges in $G$ to keep $\varphi$ a valid embedding. Each condition is fulfilled with probability $\frac{1}{n}$. So the probability that $\varphi$ remains a valid embedding for $\tilde{T}$ expanded by $\mathcal{I}$ is $(\frac{1}{n})^{|\mathcal{I}|}$. To analyze the probability that $\varphi$ also remains a valid embedding for all nodes in $\mathcal{E} \setminus \mathcal{I}$ we have to consider the following dependency graph $G_D$.

**Definition 3.9** *Let $\tilde{T} \in B_{k,i}$ be a base tree, $\mathcal{E}$ be the set of expansion nodes induced by $\tilde{T}$ and $\varphi$ be a valid node embedding of $\tilde{T}$ and $\mathcal{E}$. The* dependency graph $G_D = (V_D, E_D)$ *of $\tilde{T}$ is a directed multigraph with*

- $V_D = V_{TM}(\tilde{T})$
- $(v,w) \in E_D$ *iff $\varphi(v)$ and $\varphi(w)$ lie in the same hash function and there exists an $x \in \Gamma(v)$ and a $y \in \Gamma(w)$ such that $\varphi(x) = \varphi(y)$, $x \in \mathcal{E} \setminus \mathcal{I}$ and $y \in V_{TK}(\tilde{T}) \cup \mathcal{I}$*

Clearly, $|E_D| = r(k,i) - |\mathcal{I}|$. Each edge $(v,w)$ in $G_D$ induces that the $V_M$-nodes embedded in $v$ and $w$ have to be the same, otherwise $\varphi$ is not a valid embedding. Thus each edge represents a probability of $\frac{1}{n}$. Because $G_D$ can have circles we can not add a probability of $(\frac{1}{n})^{|E_D|}$ but only $(\frac{1}{n})^{|E_{SF}|}$ where $E_{SF}$ is the set of edges of a spanning forest in $G_D$. The next graph theoretic claim will give a lower bound for $|E_{SF}|$.

**Claim 3.10** *Let $G = (V,E)$ be a directed multigraph with $|V| = n$, $|E| = m$. Let $e(v,w)$ be the number of edges from $v$ to $w$ in $G$ and $e(v) = \sum_{w \in V} e(v,w)$. If $e(v,v) = 0$ for all $v \in V$ and there is a $d \geq 1$ such that*

$$e(u,v) + e(v) \leq d \quad \text{for all } u,v \in V \qquad (*)$$

*then it holds for the number $|E_{SF}|$ of edges in any spanning forest of $G$ :*

$$|E_{SF}| \geq \left\lceil \frac{m}{d} \right\rceil .$$

**Proof:** Let $G$ and $d$ be defined as above, let $G' = (V', E')$ be a connected component of $G$ with $|V'| = r$ and $|E'| = s$. Furthermore let $A = (a_{i,j})_{1 \leq i,j \leq r}$ be the adjacence matrix of $G'$ and $s_i = \sum_{j=1}^{r} a_{i,j}$ for all $i \in \{1, \ldots, r\}$. Then it holds $a_{i,i} = 0$ and $a_{i,j} + s_j \leq d$ for all $i,j \in \{1, \ldots, r\}$ according to $(*)$. So we get

$$\begin{aligned} \sum_{1 \leq i,j \leq r} a_{i,j} + \sum_{1 \leq j \leq r} (r-1)s_j &= \sum_{1 \leq i,j \leq r, i \neq j} (a_{i,j} + s_j) \leq (r^2 - r)d \\ \Rightarrow r \cdot s &\leq r(r-1)d \\ \Rightarrow r - 1 &\geq \frac{s}{d} . \end{aligned}$$

Let $e_{G'}$ be the number of edges of a spanning tree in $G'$. Then $e_{G'} = r - 1$, that is, $e_{G'} \geq \frac{s}{d}$. Combining this result with the results for all other connected components in $G$ yields the claim. ∎

Obviously, $G_D$ meets condition $(*)$ of the claim above with $d := c+1$. So the expansion nodes induce a total probability of at least

$$\left(\frac{1}{n}\right)^{r(k,i)-|E_D|+\left\lceil\frac{|E_D|}{c+1}\right\rceil} \leq \left(\frac{1}{n}\right)^{\left\lceil\frac{r(k,i)}{c+1}\right\rceil}$$

that $\varphi$ is a valid embedding. ∎

Let $u(k)$ be the lower bound of the number $i$ of inner $V_{TK}$-nodes of $T_t$ a base tree with $k$ $V_{TK}$-nodes can possess. Then we get :

$$\begin{aligned} &\text{Prob}(T_t \text{ can be embedded into } G \text{ with root embedding } M_{x,y}) \\ \leq &\sum_{k=c+1}^{|V_{TK}|} \sum_{i=u(k)}^{k} |B_{k,i}| \cdot \phi_{k,i} \cdot p_{k,i} \end{aligned}$$

To be able to simplify the formula above we need the following two propositions.

**Proposition 3.11** *Let $M := c + \sum_{i=1}^{t-1}(c(a-b))^i$. Then for $t \geq 2$ we have*

*(1)*

$$u(k) \geq \begin{cases} c+1 & : \quad k \in \{c+1, \ldots, M-1\} \\ c + \sum_{j=1}^{t-2}(c(a-b))^j & : \quad otherwise \end{cases}$$

*(2)*

$$r(k,i) \geq \begin{cases} c^2(a-b)+1 & : \quad k \in \{c+1, \ldots, M\} \\ 1 & : \quad k \in \{M+1, \ldots, |V_{TK}|-1\} \\ 0 & : \quad k = |V_{TK}| \end{cases}$$

**Proof**.
(1): For $k < M$ it is easy to see that the inequality is true.
Let $k \geq M+1$. Suppose, a base tree $\tilde{T} \in B_{k,i}$ has $s$ $V_{TK}$-nodes which are leaves in $T_t$. Then at least $\frac{s}{c(a-b)}$ $V_{TK}$-nodes of the next highest row of $\tilde{T}$ are necessary so that $\tilde{T}$ can possess so many leaves of $T_t$. In addition to this at least $\frac{s}{(c(a-b))^2}$ $V_{TK}$-nodes of the second row above the leaves of $T_t$ are necessary so that $\tilde{T}$ can have $\frac{s}{c(a-b)}$ $V_{TK}$-nodes in the first row above the leaves of $T_t$. Continuing with this argumentation we find that if $\tilde{T}$ possesses $s$ leaves of $T_t$ then for the number $i$ of $V_{TK}$-nodes in $\tilde{T}$ which are not leaves in $T_t$ it holds:

$$i \geq c+1+\sum_{j=1}^{t-2} \frac{s}{(c(a-b))^j} \ .$$

Because $k = i+s$ it follows:

$$k \geq c+1+\sum_{j=0}^{t-2} \frac{s}{(c(a-b))^j} \ . \quad (*)$$

Let $s = (c(a-b))^{t-1} + r$, $r \geq 0$. Then we have:

$$\begin{aligned} k &\geq c+1+\sum_{j=0}^{t-2} \frac{(c(a-b))^{t-1}+r}{(c(a-b))^j} \\ &= c+1+\sum_{j=0}^{t-1}(c(a-b))^j + \sum_{j=0}^{t-2}\frac{r}{(c(a-b))^j} \geq M+1+r \ . \end{aligned}$$

So for $k = M+1+r$ $\tilde{T}$ can have at most $(c(a-b))^{t-1}+r$ leaves of $T_t$. Hence for all $k = M+1+r$, $r \geq 0$, the number $i$ of $V_{TK}$-nodes in $\tilde{T}$ which are not leaves in $T_t$ is:

$$i \geq k - [(c(a-b))^{t-1}+r] = c+1+\sum_{j=1}^{t-2}(c(a-b))^j =: U\,.$$

Clearly, then for all $k = M+1-r \geq c+1$, $r \geq 1$, we have $i \geq U-r$. So inequality (2) is also true for $k = M$.

(2): For $k > M$ it is easy to see that the inequality is true.
Assume $r(k,i) \leq c^2(a-b)$. Then because of $r(k,i) = i \cdot c(a-b) - k + c + 1$ we get

$$i \leq c + \frac{k-(c+1)}{c(a-b)}\,.$$

So for the number $s = k-i$ of $V_{TK}$-nodes in $\tilde{T}$ which are leaves of $T_t$ it holds:

$$s \geq k - c - \frac{k-(c+1)}{c(a-b)}\,. \quad (**)$$

Using the inequality $(**)$ repeatedly for the $(j=0)$-term in $(*)$ yields:

$$k \geq c+1+\sum_{j=1}^{t-1}(c(a-b))^j = M+1\,.$$

So if $k < M+1$ it has to hold $r(k,i) > c^2(a-b)$.

∎

**Proposition 3.12** *Let $a$, $b$ and $c$ be chosen as in the Main Theorem, $2 \leq t \leq \frac{\log\log n}{\log(c(a-b))}+3$ and $n$ be big enough. Then it holds for all $i \leq k$:*

$$\binom{i\cdot c(a-b)}{r(k,i)} k^{r(k,i)} n^{-\lceil \frac{r(k,i)}{c+1} \rceil} \leq \begin{cases} n^{-\frac{c^2(a-b)}{c+1}} & : \quad k \in \{c+1,\ldots,M\} \\ n^{-\frac{r(k,i)}{2(c+1)}} & : \quad k \in \{M+1,\ldots,|V_{TK}|\} \end{cases}$$

**Proof**. Let $a$, $b$ and $c$ be chosen as in the Main Theorem and $t$ be chosen as above. Then it follows with $c = O((\frac{\sqrt{\log n}}{a-b})^{1/3})$:

$$\begin{aligned} |V_{TK}| &= (c+1)\sum_{i=1}^{t-1}(c(a-b))^i \\ &= O((c+1)(c(a-b))^{t-1}) = O((c+1)\cdot\log n\cdot(c(a-b))^2) \\ &= O\left(\left(\frac{\sqrt{\log n}}{a-b}\right)^{1/3}\cdot\log n\cdot\sqrt[6]{\log n}\cdot(a-b)^{4/3}\right) = O((\log n)^2)\,. \end{aligned}$$

Let $k \leq M$. At first we show that for $n$ big enough $\binom{i\cdot c(a-b)}{r(k,i)} k^{r(k,i)} n^{-r(k,i)/(c+1)}$ gets the smaller the bigger $r(k,i)$ gets:

$$\begin{aligned} \binom{i\cdot c(a-b)}{r(k,i)+1} k^{r(k,i)+1} n^{-\frac{r(k,i)+1}{c+1}} &\leq \binom{i\cdot c(a-b)}{r(k,i)} k^{r(k,i)} n^{-\frac{r(k,i)}{c+1}} \\ \Leftrightarrow \frac{r(k,i)!(ic(a-b)-r(k,i))!}{(r(k,i)+1)!(ic(a-b)-r(k,i)-1)!}\cdot k &\leq n^{\frac{1}{c+1}} \\ \Leftarrow \left(\frac{(ic(a-b)-r(k,i))k}{r(k,i)+1}\right)^{c+1} &\leq n \end{aligned}$$

which is true for $n$ big enough since $i,k,r(k,i) \leq |V_{TK}|$, $c = O((\frac{\sqrt{\log n}}{a-b})^{1/3})$. So with the help of Proposition 3.11 we can conclude:

$$\begin{aligned} & & \binom{i\cdot c(a-b)}{r(k,i)} k^{r(k,i)} n^{-\frac{r(k,i)}{c+1}} &\leq n^{-\frac{c^2(a-b)}{c+1}} \\ \Leftarrow \quad & \left(\frac{e\cdot i\cdot c(a-b)\cdot k}{c^2(a-b)+1}\right)^{c^2(a-b)+1} \left(\frac{1}{n}\right)^{\frac{c^2(a-b)+1}{c+1}} &\leq \left(\frac{1}{n}\right)^{\frac{c^2(a-b)}{c+1}} \\ & \Leftarrow \quad \left(\frac{e\cdot i\cdot k}{c}\right)^{c^2(a-b)+1} &\leq n^{\frac{1}{c+1}} \\ & \Leftarrow \quad \left(\frac{e\cdot i\cdot k}{c}\right)^{(c+1)(c^2(a-b)+1)} &\leq n \end{aligned}$$

which is true for $n$ big enough because with $k,i \leq |V_{TK}| = O((\log n)^2)$ and $c = O((\frac{\sqrt{\log n}}{a-b})^{1/3})$ it holds:

$$\log\left(\frac{e\cdot i\cdot k}{c}\right)^{(c+1)(c^2(a-b)+c)} = O(\log\log n\sqrt{\log n}) = o(\log n)\,.$$

Let $k$ be chosen such that $M < k < |V_{TK}|$. Then we can conclude because of $1 \leq r(k,i) \leq |V_{TK}| = O((\log n)^2)$:

$$\begin{aligned} n^{-\frac{r(k,i)}{2(c+1)}} &\geq \binom{i\cdot c(a-b)}{r(k,i)} k^{r(k,i)} n^{-\frac{r(k,i)}{c+1}} \quad (*) \\ \Leftarrow \quad n &\geq \left(\frac{e\cdot i\cdot c(a-b)\cdot k}{r(k,i)}\right)^{2(c+1)} = 2^{O(\log\log n\cdot\sqrt[6]{\log n})}. \end{aligned}$$

For $k = |V_{TK}|$ we have $r(k,i) = 0$. Thus inequality $(*)$ is also fulfilled for $k = |V_{TK}|$.

∎

Now we can simplify the formula of the probability that an embedding of $T_t$ into $G$ with root embedding $M_{x,y}$ is possible. Let $\epsilon \in (0,1]$, $1 < a \leq \sqrt{\log n}$, $b < a$ and $c$ be chosen such that

$$\epsilon\binom{a-1}{a-b}\left(\frac{1}{c!}\right)^{a-b} \leq \frac{1}{2}\,.$$

Then it holds for all $t$, $2 \leq t \leq \frac{\log\log n}{\log(c(a-b))} + 3$ and $n$ big enough:

$$\begin{aligned} &\text{Prob}(T_t \text{ can be embedded into } G \text{ with root embedding } M_{x,y}) \\ \leq\ & \sum_{k=c+1}^{|V_{TK}|}\sum_{i=u(k)}^{k} |B_{k,i}|\cdot\phi_{k,i}\cdot p_{k,i} \\ =\ & \sum_{k=c+1}^{|V_{TK}|}\binom{\epsilon n}{k}\binom{k}{c+1}(k-(c+1))!\sum_{i=u(k)}^{k}\left[\binom{a-1}{a-b}n^{a-b}\right]^i\left(\frac{1}{c!}\right)^{i(a-b)}\cdot \\ & \binom{i\cdot c(a-b)}{r(k,i)}k^{r(k,i)}\left(\frac{1}{n}\right)^{i(a-b+1)+(k-i)}\left(\frac{1}{n}\right)^{\lceil\frac{r(k,i)}{c+1}\rceil} \\ \leq\ & \frac{1}{(c+1)!}\sum_{k=c+1}^{|V_{TK}|}\sum_{i=u(k)}^{k}\left[\epsilon^k\left(\tfrac{a-1}{a-b}\right)^i\left(\tfrac{1}{c!}\right)^{i(a-b)}\right]\cdot\left[\binom{i\cdot c(a-b)}{r(k,i)}k^{r(k,i)}\left(\tfrac{1}{n}\right)^{\lceil\frac{r(k,i)}{c+1}\rceil}\right] \\ \leq\ & \frac{1}{(c+1)!}\sum_{k=c+1}^{|V_{TK}|}\sum_{i=u(k)}^{k}\left(\tfrac{1}{2}\right)^i\cdot\left[\binom{i\cdot c(a-b)}{r(k,i)}k^{r(k,i)}\left(\tfrac{1}{n}\right)^{\lceil\frac{r(k,i)}{c+1}\rceil}\right]. \qquad (*) \end{aligned}$$

**Bounding $(*)$ for $k = c+1$ to $M$:**
Let $U = c + \sum_{j=1}^{t-2}(c(a-b))^j$. With the help of Proposition 3.11 (1) and Proposition 3.12 we get

$$\begin{aligned}
&\frac{1}{(c+1)!}\sum_{k=c+1}^{M}\sum_{i=u(k)}^{k}\left(\frac{1}{2}\right)^i \cdot \left[\binom{i\cdot c(a-b)}{r(k,i)} k^{r(k,i)}\left(\frac{1}{n}\right)^{\left\lceil\frac{r(k,i)}{c+1}\right\rceil}\right] \\
&\leq \sum_{k=c+1}^{M}\sum_{i=u(k)}^{k}\left(\frac{1}{2}\right)^i\left(\frac{1}{n}\right)^{\frac{c^2(a-b)}{c+1}} \\
&\leq \sum_{k=c+1}^{M}\left(\frac{1}{n}\right)^{\frac{c^2(a-b)}{c+1}}\left(\frac{1}{2}\right)^{u(k)-1} \leq \frac{(\log n)^2}{2}\left(\frac{1}{n}\right)^{\frac{c^2(a-b)}{c+1}}.
\end{aligned}$$

**Bounding $(*)$ for $k = M+1$ to $|V_{TK}|$:**
Let $U = c + \sum_{j=1}^{t-2}(c(a-b))^j$. With the help of Propostion 3.11 and Propostion 3.12 we get

$$\begin{aligned}
&\frac{1}{(c+1)!}\sum_{k=M+1}^{|V_{TK}|}\sum_{i=u(k)}^{k}\left(\frac{1}{2}\right)^i \cdot \left[\binom{i\cdot c(a-b)}{r(k,i)} k^{r(k,i)}\left(\frac{1}{n}\right)^{\left\lceil\frac{r(k,i)}{c+1}\right\rceil}\right] \\
&\leq \frac{1}{(c+1)!}\sum_{k=M+1}^{|V_{TK}|}\sum_{i=u(k)}^{k}\left(\frac{1}{2}\right)^i\left(\frac{1}{n}\right)^{\frac{r(k,i)}{2(c+1)}} \\
&\leq \frac{1}{(c+1)!}\left[\sum_{k=M+1}^{|V_{TK}|-1} n^{-\frac{1}{2(c+1)}}\sum_{i=u(k)}^{k}\left(\frac{1}{2}\right)^i + \left(\frac{1}{2}\right)^{U-1}\right] \\
&\leq \frac{1}{(c+1)!}\left[\left(\frac{|V_{TK}|}{n^{\frac{1}{2(c+1)}}}+1\right)\left(\frac{1}{2}\right)^{U-1}\right] \\
&\leq \frac{2}{3}\cdot\left(\frac{1}{2}\right)^{U-1} \leq \frac{2}{3}\cdot\left(\frac{1}{2}\right)^{c-1+\sum_{j=1}^{t-2}(c(a-b))^j}.
\end{aligned}$$

So the probability that $T_t$ with root embedding $M_{x,y}$ can be embedded into $G$ is bounded above by

$$\frac{(\log n)^2}{2}\cdot\left(\frac{1}{n}\right)^{\frac{c^2(a-b)}{c+1}} + \frac{2}{3}\cdot\left(\frac{1}{2}\right)^{c-1+\sum_{j=1}^{t-2}(c(a-b))^j}.$$

This proves Lemma 3.3. Note that for $t = \frac{\log\log n}{\log(c(a-b))} + 3$ and $a$, $b$ and $c$ chosen as in the Main Theorem we get $|E_T| = |V_{TK}| + |V_{TM}| - 1 = O(\log^2 n)$. So $O(\log^2 n)$-universal hash functions suffice to prove the results of Lemma 3.3. ∎

## 3.2 Proof of Lemma 3.4

We first want to analyze the probability that two modules $M_{j,k}$ and $M_{j',k'}$ are still blocked at round $t$. Then we have to analyze the probabilities for all combinations of base trees for these two modules. Let $\tilde{T}_1$ be a base tree for $M_{j,k}$ and $\tilde{T}_2$ be a base tree for $M_{j',k'}$. We want to consider the following three cases :

(1) The set of $V_K$-nodes embedded in $\tilde{T}_1$ and the set of $V_K$-nodes embedded in $\tilde{T}_2$ are disjoint. Then we can analyze the two base trees independently as it is done in the proof of Lemma 3.3.

(2) The embeddings of $\tilde{T}_1$ and $\tilde{T}_2$ only have one common $V_K$-node. Then we can still analyze the two base trees independently, as we will see. Let $\varphi_1$ be a valid node embedding of $\tilde{T}_1$ and $\varphi_2$ be a valid node embedding for $\tilde{T}_2$. Let $G_D^1$ be the dependency graph of $\tilde{T}_1$ and $G_D^2$ be the

dependency graph of $\tilde{T}_2$ (recall that the edges of such graphs symbolize dependencies between embeddings in $V_{TM}$-nodes, see Definition 3.9). For all $i = 1, 2$, $x \in V_K$ and $j \in \{1, \ldots, a\}$ let

$$C^i_{x,j} = \{y \in V_K \mid \exists v, w \in V_{TM}(\tilde{T}_i) : v, w \text{ lie in the same connected component in } G^i_D \text{ built by modules in hash function } h_j, \text{ and } x \in \varphi_i(\Gamma(v)),\ y \in \varphi_i(\Gamma(w))\}.$$

The sense behind the definition of $C^i_{x,j}$ is that the embedding of $\tilde{T}_i$ can only be valid if all $V_K$-nodes in every $C^i_{x,j}$ have an edge to the same $V_M$-node as $x$ in the access graph with regard to hash function $h_j$. So all dependencies of $V_K$-nodes embedded in $\tilde{T}_i$ can be expressed by $\bigcup_{x,j} G^i_{x,j}$, where $G^i_{x,j}$ is the complete graph over nodes in $C^i_{x,j}$. Note that for all $x, y \in V_K$ and $j, j' \in \{1, \ldots, a\}$ it holds: if $y \in G^i_{x,j}$ and $j = j'$ then $G^i_{x,j} = G^i_{y,j'}$, otherwise $G^i_{x,j} \cap G^i_{y,j'} = \emptyset$. If we remove so many edges that all $G^i_{x,j}$ are circle-free, but still connected then we get for each remaining edge independently a probability of $\frac{1}{n}$ that the embedding of $\tilde{T}_i$ is valid. Because the embeddings of $\tilde{T}_1$ and $\tilde{T}_2$ only have one $V_K$-node in common, we get: if all $G^i_{x,j}$ are reduced in a way that they have no circles then also $\bigcup_{x,j}(G^1_{x,j} \cup G^2_{x,j})$ can have no circles any more, that is, we can analyze the dependencies independently in $\tilde{T}_1$ and $\tilde{T}_2$.

(3) $\tilde{T}_1$ and $\tilde{T}_2$ are embedded into at least two common $V_K$-nodes. Then we can not assume independency any more.

Let us use these conclusions for the general case that $s \geq 2$ modules $M_{j_1,k_1}, \ldots, M_{j_s,k_s}$ are still blocked at round $t$. Let $\tilde{T}_1, \ldots, \tilde{T}_s$ be the base trees of these modules, let $Y_i$ be the number of $V_{TK}$-nodes in base tree $\tilde{T}_i$ that are also used in base trees $\tilde{T}_j$, $j < i$, $Y = \sum_{i=1}^{s} Y_i$. Furthermore, let $X^t_{j,k}$ be a random variable with

$$X^t_{j,k} = \begin{cases} 1 & : \text{ module } M_{j,k} \text{ still blocked at round } t \\ 0 & : \text{ otherwise} \end{cases}$$

for all $j \in \{1, \ldots, a\}$, $k \in \{1, \ldots, n\}$. Then it holds:

$$\begin{aligned} &\text{Prob}(X^t_{j_1,k_1} = \ldots = X^t_{j_s,k_s} = 1) \\ &= \sum_{g_1,\ldots,g_s \geq 0} \text{Prob}(Y_1 = g_1, \ldots, Y_s = g_s) \cdot \\ &\quad \text{Prob}(X^t_{j_1,k_1} = \ldots = X^t_{j_s,k_s} = 1 \mid Y_1 = g_1, \ldots, Y_s = g_s). \end{aligned}$$

We want to consider three different cases.

(1) Independent case ($Y \leq 1$): Let $p$ be chosen as $p_{a,t}$ in chapter 2. Then it holds:

$$\text{Prob}(Y \leq 1) \cdot \text{Prob}(X^t_{j_1,k_1} = \ldots = X^t_{j_s,k_s} = 1 \mid Y \leq 1) \leq p^s .$$

(2) Dependent case ($2 \leq Y < 2s$):
Let $g_1, \ldots, g_s$ be fixed. Let $l_i$ be the number of $V_{TK}$-nodes of $\tilde{T}_i$ and $A(l_1, \ldots, l_s)$ be the number of embeddings of $T_1, \ldots, T_s$ given $l_1, \ldots, l_s$. Then it holds:

$$\text{Prob}(Y_1 = g_1, \ldots, Y_s = g_s) \leq \max_{l_1,\ldots,l_s \in \{c+1,\ldots,|V_{TK}|\}} \frac{A(l_1, \ldots, l_s)}{\prod_{i=1}^{s} \binom{n}{l_i}} . \quad (*)$$

Let $Z_i$ denote the number of $V_{TK}$-nodes in $\tilde{T}_i$ for all $i \in \{1, \ldots, s\}$, let $\gamma = \sum_{i=1}^{s} g_i$ and $l = |V_{TK}|$. The following items have to be considered in order to count all possible embeddings for $Y_1 = g_1, \ldots, Y_s = g_s$ and $Z_1 = l_1, \ldots, Z_s = l_s$:

(a) There are $\binom{n}{\sum_{i=1}^{s} l_i - \gamma}$ ways to choose $V_K$-nodes for the embeddings of $\tilde{T}_1, \ldots, \tilde{T}_s$.

(b) The $V_K$-nodes chosen in (a) can be distributed among the trees $\tilde{T}_i$ in $(\sum_{i=1}^{s} l_i - \gamma)! / \prod_{i=1}^{s}(l_i - g_i)!$ different ways.

(c) There are at most $(\sum_{i=1}^{s} l_i - \gamma)^\gamma$ possibilities to add $V_K$-nodes to trees $\tilde{T}_i$ that have already been used in trees $\tilde{T}_j$ with $j < i$.

Thus it holds:

$$\begin{aligned}
&\mathrm{Prob}(Y_1 = g_1, \ldots, Y_s = g_s \mid Z_1 = l_1, \ldots, Z_s = l_s) \leq \\
&\leq \frac{\binom{n}{\sum_{i=1}^{s} l_i - \gamma} \frac{(\sum_{i=1}^{s} l_i - \gamma)!}{\prod_{i=1}^{s}(l_i - g_i)!} (\sum_{i=1}^{s} l_i - \gamma)^\gamma}{\prod_{i=1}^{s} \binom{n}{l_i}} \\
&\leq \frac{n!}{(n - (\sum_{i=1}^{s} l_i - \gamma))!} \left(\prod_{i=1}^{s} \frac{(n-l_i)!}{n!}\right) \left(\prod_{i=1}^{s} \frac{l_i!}{(l_i - g_i)!}\right) \left(\sum_{i=1}^{s} l_i - \gamma\right)^\gamma \\
&\leq \left(\frac{1}{n-l}\right)^\gamma \left(\prod_{i=1}^{s} l^{g_i}\right) (sl - \gamma)^\gamma = \left(\frac{l(sl-\gamma)}{n-l}\right)^\gamma . \qquad (**)
\end{aligned}$$

Now we can estimate $(*)$ with the help of $(**)$:

$$\mathrm{Prob}(Y_1 = g_1, \ldots, Y_s = g_s) \leq \left(\frac{l(sl-\gamma)}{n-l}\right)^\gamma .$$

So it holds for $\gamma \geq 2$:

$$\begin{aligned}
\mathrm{Prob}(Y = \gamma) &\leq \sum_{\substack{g_1, \ldots, g_s \geq 0, \\ g_1 + \ldots + g_s = \gamma}} \mathrm{Prob}(Y_1 = g_1, \ldots, Y_s = g_s) \\
&\leq \sum_{\substack{g_1, \ldots, g_s \geq 0, \\ g_1 + \ldots + g_s = \gamma}} \left(\frac{l(sl-\gamma)}{n-l}\right)^\gamma \leq \binom{s+\gamma}{\gamma} \left(\frac{l(sl-\gamma)}{n-l}\right)^\gamma \\
&\leq \left(\frac{e(s+\gamma)l(sl-\gamma)}{\gamma(n-l)}\right)^\gamma \leq \left(\frac{1}{n}\right)^{\lfloor \frac{\gamma}{2} \rfloor}
\end{aligned}$$

if $n$ big enough, because $s \leq \alpha \log n$ and for $t \leq \frac{\log\log n}{\log(c(a-b))} + 2$ we have $l, \gamma = O((\log n)^{5/3})$. So for $Y = 2r$ or $Y = 2r+1$, $1 \leq r \leq s$, it suffices to remove $r$ base trees $\tilde{T}_i$ to be able to analyze the remaining trees independently. It follows:

$$\begin{aligned}
&\sum_{r=1}^{s-1} \mathrm{Prob}(Y = 2r \vee Y = 2r+1) \cdot \\
&\mathrm{Prob}(X^t_{j_1,k_1} = \ldots = X^t_{j_s,k_s} = 1 \mid Y = 2r \vee Y = 2r+1) \\
&\leq \sum_{r=1}^{s-1} \left(\frac{1}{n}\right)^r \cdot p^{s-r} \leq \sum_{r=1}^{s-1} \binom{s}{r} \left(\frac{1}{n}\right)^r \cdot p^{s-r}
\end{aligned}$$

(3) Dependent case ($Y \geq 2s$):

$$\begin{aligned}
&\sum_{\gamma=2s}^{(s-1)l} \mathrm{Prob}(Y = \gamma) \cdot \mathrm{Prob}(X^t_{j_1,k_1} = \ldots = X^t_{j_s,k_s} = 1 \mid Y = \gamma) \\
&\leq \sum_{\gamma=2s}^{(s-1)l} \mathrm{Prob}(Y = \gamma) \leq \sum_{\gamma=2s}^{(s-1)l} \left(\frac{e(s+\gamma)l(sl-\gamma)}{\gamma n}\right)^\gamma \leq \left(\frac{1}{n}\right)^s .
\end{aligned}$$

So altogether we get:

$$\begin{aligned}
&\mathrm{Prob}(X^t_{j_1,k_1} = \ldots = X^t_{j_s,k_s} = 1) \\
&= \sum_{\gamma=0}^{(s-1)l} \mathrm{Prob}(Y=\gamma) \cdot \mathrm{Prob}(X^t_{j_1,k_1} = \ldots = X^t_{j_s,k_s} = 1 \mid Y = \gamma) \\
&\leq p^s + \sum_{r=1}^{s-1} \binom{s}{r} \left(\frac{1}{n}\right)^r \cdot p^{s-r} + \left(\frac{1}{n}\right)^s \\
&= \sum_{r=0}^{s} \binom{s}{r} \left(\frac{1}{n}\right)^r \cdot p^{s-r} \leq \left(p + \frac{1}{n}\right)^s .
\end{aligned}$$

■

## 3.3 Avoiding Deadlocks

The Main Theorem implies that structures may appear in the access graph that prevent the $(n, \epsilon, a, b, c)$-process from terminating. Nevertheless, the expected number of rounds necessary to satisfy all $\epsilon n$ requests does not exceed $\frac{\log\log n}{\log(c(a-b))} + 3 + o(1)$ with the following strategy:

$s := 0$
**repeat**
  partition the $\epsilon n$ keys into sets $A^s_1, \ldots, A^s_{2^s}$ of $\frac{\epsilon n}{2^s}$ keys
  **for** $i = 1$ **to** $2^s$:
    run $\frac{\log\log n}{\log(c(a-b))} + 3$ rounds of the $(n, \frac{\epsilon}{2^s}, a, b, c)$-process on set $A^s_i$
  $s := s + 1$
**until** all keys are inactive

For $\epsilon$, $a$, $b$ and $c$ chosen as in the Main Theorem the average number of rounds required for this algorithm is at most :

$$\begin{aligned}
&\left(\frac{\log\log n}{\log(c(a-b))} + 3\right) \left(1 + \sum_{s=1}^{\log n} 2^s \cdot \mathrm{Prob}(\exists j \in \{1, \ldots, 2^{s-1}\} : \text{ deadlock in } A^{s-1}_j)\right) \\
&\leq \left(\frac{\log\log n}{\log(c(a-b))} + 3\right) \left(1 + \sum_{s=1}^{\log n} 2^s \cdot 2^{s-1} \left(\frac{1}{2^{s-1}}\right)^{c+1} (\log n)^2 \left(\frac{1}{n}\right)^{\frac{c^2(a-b)}{c+1}-1}\right) \\
&\leq \left(\frac{\log\log n}{\log(c(a-b))} + 3\right) \left(1 + 2(\log n)^3 \left(\frac{1}{n}\right)^{\frac{c^2(a-b)}{c+1}-1}\right) \\
&= \frac{\log\log n}{\log(c(a-b))} + 3 + o(1)
\end{aligned}$$

# 4 Implementations of the Basic Process

In this section we present efficient implementations of the basic process that yield fast static dictionaries and shared memory simulations.

## 4.1 Implementations on a $c$-collision DMM

Suppose we have an $n$-processor $c$-collision DMM, where $U$, the set of shared memory cells, is distributed among the modules with hash functions $h_1, \ldots, h_a : U \to \{1, \ldots, n\}$. So for every $i \in \{1, \ldots, a\}$ and $x \in U$, $M_{h_i(x)}$ contains the $i$-th copy of $x$. Each processor of the DMM knows at

most one active key and the hash functions $h_1, \ldots, h_a$. Such a processor is called *active*. Then the $(n, \epsilon, a, b, c)$-process can be implemented in such a way that every active processor tries to get access to all $a$ copies of its key sequentially until it got access to at least $b$. Because of the $c$-collision rule, after each trial, the active processors have to wait $c$ time steps to know whether their request was successful or not. Thus a round takes time $c \cdot a$. Now we can derive the following Theorem from part (b) of the Main Theorem:

**Theorem 4.1** *Let $h_1, \ldots, h_a$, $\epsilon$, $a$, $b$ and $c$ be chosen as in the Main Theorem. Then the above implementation of the $(n, \epsilon, a, b, c)$-process on a $c$-collision DMM needs at most time $c \cdot a \cdot (\frac{\log\log n}{\log(c(a-b))} + 3)$, w.h.p.*

The above theorem yields simulations of a static dictionary with access time $O(\log\log n)$, w.h.p., for example for $\epsilon = 1$, $a = 2$ and $b = 1$ on a 2-collision DMM, and for $\epsilon = \frac{1}{2}$, $a = 4$ and $b = 1$ on a 1-collision DMM.

With the help of the majority trick, that is $b > \frac{a}{2}$, the above theorem also yields simulations of shared memory, for instance for $\epsilon = \frac{1}{2}$ and $a = 3$ on a 2-collision DMM (a similar result is shown in [8], but with $\epsilon = 1/3$ and $c = 3$) or for $\epsilon = \frac{1}{40}$ and $a = 7$ on a 1-collision DMM. Thus Theorem 4.1 shows that an EREW PRAM can be simulated on an optical crossbar DMM with delay not exceeding $O(\log\log n)$, w.h.p.

Note that if the above implementation of the $(n, \epsilon, a, b, c)$-process successfully terminates then each module has to answer at most $c \cdot a$ requests.

## 4.2 Implementations on an arbitrary-DMM

We want to find a way to run a round of the $(n, 1, a, b, c)$-process much faster than in $a \cdot c$ steps. For this purpose we partition the DMM into $a$ groups $A_1, \ldots, A_a$ of $\frac{n}{a}$ processors and $a$ groups $B_1, \ldots, B_a$ of $\frac{n}{a}$ modules. Each function $h_j$, $j \in \{1, \ldots, a\}$, now maps $U$ to modules in $B_j$ only, i.e. has a range of size $\frac{n}{a}$ instead of $n$.

Let us first assume that we only have $\frac{n}{a}$ keys, the $i$-th of which is known by the $i$-th processor of all groups $A_1, \ldots, A_a$. Then the accesses to the $a$ copies of each key can be done in parallel, i.e. in time $c$. Checking whether key $x$ got at least $b$ answers needs time $O(b)$, because a threshold function on $a$ values with threshold $b$ (given a 0-1 vector of length $a$, test whether it contains at least $b$ 1's) has to be computed using $a$ processors and modules.

Let us now choose $c := \max\{\lceil \frac{a}{a-b} \rceil, 4\}$. If $\frac{a}{a-b} = O((\frac{\sqrt{\log n}}{a-b})^{1/3})$ then all conditions on $c$ in part (a) of the Main Theorem are fulfilled with $\epsilon = 1$. Thus part (a) of the Main Theorem implies that, after executing 4 rounds as described above, at most $n/2^{a^2}$ keys are still active, w.h.p. Each round needs time $O(c + b) = O(\frac{a}{a-b} + b)$.

Now assign $a \cdot 2^a$ processors to each active key. Since $a \cdot 2^a < 2^{a^2}$, this can be done in time $O(\log^* n)$, w.h.p., using the LAC algorithm, compare Section 2.2.

As $a \cdot 2^a$ processors and modules can compute threshold functions on $a$ values in constant time on an arbitrary-DMM, all remaining rounds of our process can be done in time $O(c)$, where $c$ can be chosen here to be $c = \lceil \log(\frac{2e(a-1)}{a-b}) \rceil$. It is again easy to check that this $c$ fulfills all preconditions demanded for part (b) of the Main Theorem. Thus this phase needs time

$$O\left(\log\left(\frac{a}{a-b}\right)\frac{\log\log n}{\log(a-b) + \log\log a}\right)$$

by part (b) of the Main Theorem. Altogether we have shown:

**Lemma 4.2** *If each of $\frac{n}{a}$ keys is known by $a$ processors then the algorithm described above needs time*

$$O\left(\left(b + \frac{a}{a-b}\right) + \log^* n + \log\left(\frac{a}{a-b}\right) \cdot \frac{\log\log n}{\log(a-b) + \log\log a}\right),$$

*w.h.p., to access $b$ out of $a$ copies of each of the $\frac{n}{a}$ keys.*

It remains to show how to reduce the number of active keys from $n$ to $m := \frac{n}{a}$. This is done by the following scheme.

Let $a \geq 4$, $b < a$, $c = \max\{\lceil \frac{a}{a-b} \rceil, 4\}$, and $a_0 := \min\{2b, a\}$.

**initial reduction scheme:**

simultaneously for each group $A_j$, $j \in \{1, \ldots, \frac{a}{a_s}\}$:
run 5 rounds of the $(m, 1, a_0, b, c)$-process, using hash functions
$h_j, \ldots, h_{(j+a_0-1) \bmod a}$

It is easy to organize these rounds in such a way that concurrent accesses to copies with respect to the same hash function never come from different groups $A_j$. Thus each round needs time $O(c \cdot a_0) = O(c \cdot b)$. By part (a) of the Main Theorem we can conclude the following lemma.

**Lemma 4.3** *The initial reduction scheme reduces the number of active keys from $m$ to at most $m/2^{a_0^3}$ for each group $A_j$ in time $O(\frac{a \cdot b}{a-b})$, w.h.p.*

**Proof**. The condition $c \geq \lceil \frac{a}{a-b} \rceil \geq 2$ and the choice of $a_0$ guarantee that $c(a_0 - b) \geq a_0$. ∎

Let us consider two different cases.

**Case 1:** $a_0 + 2^{a_0} > a$. Then it is easy to check that $2^{a_0} \geq \frac{3}{4}a$ for $a \geq 4$. Thus the initial reduction scheme reduces the number of active keys to at most

$$\frac{n}{2^{a_0^3}} \leq \frac{n}{(2^{a_0})^3} \leq \frac{n}{a^2},$$

w.h.p. So in this case the situation necessary for applying Lemma 4.2 can be established with the help of the LAC algorithm presented in Section 2.2.

**Case 2:** $a_0 + 2^{a_0} \leq a$. Then, in a final reduction scheme, we want to reduce the remaining $n/2^{a_0^3}$ active keys to at most $n/a^2$. Let $a_0$ be chosen as above and $a_s := 2^{a_s - 1}$ for all $s \geq 1$. For $s \geq 0$ let $m_s := a_s m$ and $A_j^s := \bigcup_{k=(j-1)a_s+1}^{j \cdot a_s} A_k$ be subsets of the set of processors, $j \in \{1, \ldots, \frac{a}{a_s}\}$, that is $|A_j^s| = m_s$. For the rest of this section the $\epsilon$ in the $(m, \epsilon, a, b, c)$-process will mean that our process is started with at most $\epsilon m$ active keys. Assume that in each $A_j$, $j \in \{1, \ldots, a\}$, at most $m/2^{a_0^3}$ keys are still active.

**final reduction scheme:**

$s := 1$
**while** $\sum_{i=0}^{s} a_i \leq a$ **do**
simultaneously for each $A_j^s$, $j \in \{1, \ldots, \frac{a}{a_s}\}$ :
(1) allocate $a_s$ processors for all but a fraction of at most $1/2^{3a_s+1}$ of the active keys in $A_j^s$
(2) run 5 rounds of the $(m, \frac{1}{a_s}, a_s, b, c)$-process on $A_j^s$ on hash functions not used before for keys in $A_j^s$
$s := s + 1$

Let us call one execution of the while-loop one *round*. We will now prove by induction on $s$ that in time $O(c \cdot b)$ each round $s$ reduces the number of active keys to at most $m_s/a_{s+1}^3$ in each group $A_j^s$, w.h.p.

The initial reduction scheme serves as a basis for this induction because it ensures that at the beginning of the final reduction scheme at most $m/2^{a_0^3}$ keys are still active in each group $A_j$ of processors, w.h.p.

Assume now that at most $m_{s-1}/a_s^3$ keys are left in each subset $A_i^{s-1}$ of processors at the beginning of round $s$. Then in (1), according to Lemma 2.4, for all but $m_s/2^{3a_s+1}$ active keys in each set $A_i^s$, $a_s$ processors can be allocated in constant time, w.h.p. We then are able to implement the $(m, \frac{1}{a_s}, a_s, b, c)$-process in (2) in such a way that each key that has $a_s$ processors can send messages to $a_s$ modules possessing its copies in constant time and can check in time $O(b)$ whether at least $b$ messages came through. So altogether step (2) needs $O(c+b)$ time, w.h.p., for $c$ chosen as for the initial reduction scheme to reduce the number of active keys in each $A_j^s$ to at most $m_s/2^{a_s^3}$. Since $a_s \geq a_1 = 2^{a_0} \geq 4$, we get

$$\frac{m_s}{2^{3a_s+1}} + \frac{m_s}{2^{a_s^3}} \leq \frac{m_s}{(2^{a_s})^3} = \frac{m_s}{a_{s+1}^3} \ .$$

Thus round $s$ of the reduction scheme reduces the active keys from at most $m_s/a_s^3$ to at most $m_s/a_{s+1}^3$ in each partition $A_i^s$ in time $O(c+b)$, w.h.p.

Consider now the last round of the final reduction scheme. It is easy to check that $a_{s+1} = 2^{a_s} \geq \frac{3}{4}a$ for all $a \geq 4$. So this round reduces the number of active keys from at most $m_s/a_s^3$ to at most

$$\frac{m_s}{2^{3a_s+1}} + \frac{m_s}{2^{a_s^3}} \leq \frac{m_s}{(2^{a_s})^3} \leq \frac{m_s}{a^2}$$

active keys in each group $A_i^s$, w.h.p. This results in the following lemma:

**Lemma 4.4** *The final reduction scheme reduces the number of active keys from at most $n/2^{a_0^3}$ to at most $n/a^2$ in time*

$$O\left(\left(b + \frac{a}{a-b}\right)(\log^* a - \log^* b + 1)\right),$$

*w.h.p.*

Note that the final reduction scheme works correctly even if $\frac{a}{a_s}$ is not an integer for some $s$. In this case the remaining keys of the incomplete group $A_i^s$ are put into one of the complete groups $A_j^s$. This at most doubles the upper bound for the number of remaining keys in $A_j^s$ which does not hurt our analysis.

Now an efficient way to access $b$ out of $a$ copies for each of $n$ keys on an arbitrary-DMM could look as follows:

- **Phase 1:** Reduce the number of active keys from $n$ to at most $\frac{n}{a^2}$ using the initial and final reduction scheme.
- **Phase 2:** Assign $a$ processors, one from each group $A_j$, to each of the $\frac{n}{a^2}$ active keys, using the LAC algorithm from Section 2.2.
- **Phase 3:** Finish the process (compare Lemma 4.2).

Unfortunately, we can not use the lemmas proven above to conclude the time bound

$$O\left(\tfrac{a \cdot b}{a-b} + \left(b + \tfrac{a}{a-b}\right)(log^* a - \log^* b + 1) + \log^* n + \log\left(\tfrac{a}{a-b}\right) \cdot \tfrac{\log\log n}{\log(a-b)+\log\log a}\right)$$

because the keys phase 3 is started with are not independent of the hash functions. The reason for this is that they remained active after phase 1, where the same hash functions are used as in phase 3.

In case that $b \leq \frac{a}{3}$ we can circumvent this problem easily: simply use disjoint halfs of the hash functions in phase 1 and 3.

In case $b > \frac{a}{3}$ we can proceed as follows. Let $p$ be chosen such that $\frac{(p-2)a}{p} \leq b \leq \frac{(p-1)a}{p+1}$. Thus $p \leq \lfloor \frac{2a}{a-b} \rfloor$. In order to simplify our presentation of the solution suppose that $p$ divides $a$. Then we can partition the $a$ groups $B_1, \ldots, B_a$ of $\frac{n}{a}$ modules each into groups $C_1, \ldots, C_{a/p}$ consisting of $p$ groups $B_j$, each. Select a pair $(C_i, C_j)$ out of these groups and perform the following scheme on them:

Carry out phase 1 on $C_i$ with $b' := \frac{b}{p-1}$ (note that $b' = \frac{b}{p-1} \leq \frac{a}{p+1}$) and phase 3 on $C_j$ with the same $b'$.

Since phases 1 and 3 are now run on different sets of hash functions we can combine the results in Lemma 4.2 and Lemma 4.4. Note that we have to choose $c = \lceil \frac{a/p}{a/p-b'} \rceil \leq p+1 \leq \lfloor \frac{2a}{a-b}+1 \rfloor$ for phase 1 and $c = \log(2e(p+1))$ for the last part of phase 3.

Of course, if we run this scheme only for one pair $(C_i, C_j)$ then each key got through only $b'$ instead of the required $b$ requests. But if we run this scheme for all pairs $(C_i, C_j)$ it can easily be seen that each key got through at least $b'$ requests in at least $p-1$ groups $C_i$, so altogether each key got through at least $b'(p-1) = b$ requests.

Let us call the algorithm described above the *advanced* $(n,a,b)$*-scheme*. Then the following Theorem holds.

**Theorem 4.5** *For $a$ and $b$ chosen such that $4 \leq a \leq \sqrt{\log n}$, $b < a$, and $\frac{a}{a-b} = O((\frac{\sqrt{\log n}}{a-b})^{1/3})$ the advanced $(n,a,b)$-scheme executed on an arbitrary-DMM needs time*

$$O\left(\left(\lfloor \tfrac{2a}{a-b} \rfloor\right)\left(\tfrac{a\cdot b}{a-b} + \left(b + \tfrac{a}{a-b}\right)(log^* a - \log^* b + 1) + \right.\right.$$

$$\left.\left. \log^* n + \log\left(\tfrac{a}{a-b}\right) \cdot \tfrac{\log\log n}{\log(a-b)+\log\log a}\right)\right),$$

*w.h.p. In particular, if $b \leq (1-\delta)a$ for constant $\delta > 0$, the time bound is*

$$O\left(\log^* n + b + \frac{\log\log n}{\log a}\right),$$

*w.h.p., for all $4 \leq a \leq \sqrt{\log n}$.*

In order to realize a static dictionary, $b = 1$ suffices. The number $a$ of hash functions used is the *redundancy* of the storage representation.

**Corollary 4.6** *For each $a$ such that $4 \leq a \leq \sqrt{\log n}$ the above strategy yields a static dictionary on an arbitrary-DMM with redundancy $a$ and parallel access time $O(\log^* n + \frac{\log\log n}{\log a})$, w.h.p. In particular, parallel access time $O(\log^* n)$ can be achieved with redundancy $(\log n)^{1/\log^* n}$.*

In order to realize shared memory simulations we have to choose $b > \frac{a}{2}$.

**Corollary 4.7** *For each $a$ such that $4 \leq a \leq \sqrt{\log n}$ the above strategy yields a shared memory simulation on an arbitrary-DMM with delay $O(a + \frac{\log\log n}{\log a})$, w.h.p. In particular, delay $O(\frac{\log\log n}{\log\log\log n})$ can be achieved with redundancy, e.g., $a = \sqrt{\log\log n}$.*

# 5 A lower bound for simple access protocols

In this section we ask whether there exist faster implementations of direct processes than ours. More precisely, we assume that the keys are distributed among the modules of an arbitrary DMM using $a$ independent, truly random hash functions. We demand that, given $n$ keys, only one copy of each of them has to be accessed. We allow the processors to try to access several copies of all keys they know in parallel and to communicate with other processors. The communication is restricted to the oblivious mode, i.e. the communication is independent of the hash functions and the input keys. Based on the information gathered about the topology of the access graph by such oblivious communication a processor may decide on which copies to request in the next round. Let us call this type of protocols the *simple access protocols*.

We want to describe a simple access protocol in a more formal way. As the communication is oblivious we can perform it in advance. We want to prove a lower bound for access protocols that need at most $\log\log n$ steps. After $t$ steps of the communication, $1 \leq t \leq \log\log n$, each processor knows a set $K_l$ of input keys, $|K_l| \leq 2^t$. As the communication is oblivious, $K_l$ is independent of the values of the keys, i.e. it is independent of the access graph. We assume w.l.o.g. that requests w.r.t. different hash functions never collide, i.e. we assume that each module $M_j$, $1 \leq j \leq n$, exists in $a$ copies, $M_{j,k}$, $1 \leq k \leq a$.

We transfer the capability of decisions from the processors to the modules, i.e. the processors always access all the active keys they have w.r.t. to all $a$ hash functions. A module $M_{j,k}$ knows a set $V_{j,k} = \bigcup_l K_l$ of keys, induced by the keys of the processors $P_l$ accessing it. $V_{j,k}$ can be partitioned in each step $t$ of the simple access protocol into two sets $A^t_{j,k}$ and $D^t_{j,k}$ of active and inactive keys, respectively. Let $G_{j,k}$ be the subgraph of the access graph induced by the set $V_{j,k}$ of keys and the neighbor modules of $V_{j,k}$.

In each step each module $M_{j,k}$ decides which of the active keys from $A^t_{j,k}$ it answers. This decision is only based on the topology of $G_{j,k}$ and the set $A^t_{j,k}$ of still active keys from $V_{j,k}$.

Finally the partition $A^t_{j,k} \cup D^t_{j,k}$ is adjusted, i.e. if $x \in A^t_{j,k}$ has been answered in step $t$ by $M_{j,k}$ or some other module, it moves to $D^{t+1}_{j,k}$. Obviously a module can simulate the decisions of a processor accessing it because it has at least as much information as the processor.

In the following we first show that the decisions made by the modules are in essence random. Then we analyze the process where each module $M_{j,k}$ answers a random request from $A^t_{j,k}$ in each step $t$. We allow at most $\log n$ hash functions, i.e. $a \leq \log n$.

**Lemma 5.1** *There is a subset $A$ of the input set, $|A| \geq \frac{n}{3\log^2 n}$, such that no processor knows more than one key from $A$ during the $t$ steps of the simple protocol, $1 \leq t \leq \log\log n$. $A$ is independent of the hash functions and the access graph $G$ restricted to $A$ contains no cycle of length less than 4, with probability at least $\frac{1}{4}$.*

**Proof**. As we mentioned above we can perform the communication steps in advance, because they are oblivious, i.e. independent of the hash functions and the input keys.

Define the communication graph $\tilde{G} = (K, E)$, where $K$ is the set of all keys, $x_1, \ldots, x_n$. There is an edge $\{x_i, x_j\} \in E$ if some processor with key $x_i \in K$ communicates with some processor with key $x_j \in K$ during a round $t$, for $1 \leq t \leq \log\log n$. For $t$ in the stated bounds we can bound the number of edges in the graph by $n \log n$. We restrict the set of keys to a maximum independent set $A$. Using Turan's Theorem (see e.g. [2]) we get

$$A \geq \frac{|K|^2}{|K| + 2|E|} \geq \frac{n^2}{n + 2n\log n} \geq \frac{n}{3\log n} .$$

As we consider different sets of modules to store the copies of keys w.r.t. different hash functions the access graph $G$ has no cycles of length 2. Thus, we only have to show that there exist no cycles of length 4. Note that the set $A$ is independent of the hash functions. Hence the probability of the occurrence of a 4-cycle can be bounded by

$$\binom{\frac{n}{3\log n}}{2} \left(\frac{a}{n}\right)^2 \leq \left(\frac{ne}{6\log n}\right)^2 \left(\frac{a}{n}\right)^2 \leq \frac{1}{4} .$$

The last inequality holds for $a \leq \log n$.

∎

From now on we assume that there are no communication steps, and only requests from the set $A$ have to be answered. In this case the subgraph $G'_{j,k}$ of $G_{j,k}$ induced by $V'_{j,k} = V_{j,k} \cap A$ reduces to the 2-neighborhood of $M_{j,k}$ in the access graph. Lemma 5.1 ensures that no cycles exist in this neighborhood. Therefore it is completely symmetric around $M_{j,k}$, and each decision of $M_{j,k}$ based on the topology of this 2-neighborhood is random. Hence, in the following we can view the simulation as acting on $a$ independent games. In each game all processors try to access their active key w.r.t. to all $a$ hash functions and the active modules randomly and independently choose one request for processing. This procedure is called an *access step*.

We first prove a technical lemma that bounds the number of keys left in a single game after one access step:

**Lemma 5.2** *Assume we have $\frac{n}{q}$ keys, $3 \leq q \leq n^{\frac{1}{6}}$, randomly distributed among $n$ modules. If each module removes one of its keys, with high probability at least $\frac{n}{q^3}$ keys will remain.*

**Proof**. We have $m = n/q$ requests and $n$ modules. Define the following random variables for $i = 1, \ldots, n$:

$$Y_i := \begin{cases} 1 & \text{if module } M_i \text{ gets a request} \\ 0 & \text{else} \end{cases}$$

$Y := \sum_{i=1}^{n} Y_i$ is the number of modules which receive a request. Obviously the $Y_i$ are self-weakening (compare Definition 2.5). First we have to find an upper bound $\mu^*$ for $E(Y)$ to apply Theorem 2.6. Define the random variable $X := n - Y$ for the number of modules which do not get a request. One can easily see:

$$\begin{aligned} E(X) &= n(1-\frac{1}{n})^m \geq n(1-\frac{1}{n})e^{-\frac{m}{n}} \\ \Rightarrow E(Y) &= n - E(X) \leq n\left(1-\left(1-\frac{1}{n}\right)e^{-\frac{m}{n}}\right) =: \mu^* . \end{aligned}$$

We can bound $\mu^*$ by $m/2$:

$$\begin{aligned} \mu^* &= n\left(1-\left(1-\frac{1}{n}\right)e^{-\frac{m}{n}}\right) \\ &\geq n\left(1-e^{-\frac{m}{n}}\right) \\ &\geq n\left(\frac{1}{2}\frac{m}{n}\right) = \frac{m}{2} . \end{aligned}$$

Therefore Theorem 2.6 yields for $0 \leq \varepsilon \leq 1$:

$$P(Y \geq (1+\varepsilon)\mu^*) \leq \left(\frac{1}{e}\right)^{\varepsilon^2\mu^*/3} \leq \left(\frac{1}{e}\right)^{\varepsilon^2 m/6} .$$

We choose $\varepsilon := \frac{1}{2}\left(\frac{m}{n}\right)^2$ to make this probability polynomial small:

$$P(Y \geq (1+\frac{1}{2}\left(\frac{m}{n}\right)^2)\mu^*) \leq \left(\frac{1}{e}\right)^{\frac{m^5}{24n^4}} = \left(\frac{1}{e}\right)^{\frac{n}{24q^5}} .$$

This probability is polynomial small for $3 \leq q \leq n^{\frac{1}{6}}$.

It remains to compute the number of requests that are left, i.e. $m - (1+\varepsilon)\mu^*$. We want to show that this expression is greater than $n/q^3$ using Taylor's expansion of the exponential function.

$$\begin{aligned} m-(1+\varepsilon)\mu^* &= m-(1+\varepsilon)n\left(1-\left(1-\frac{1}{n}\right)e^{-\frac{m}{n}}\right) \\ &= n\left(\frac{m}{n}-(1+\varepsilon)+(1+\varepsilon)\left(1-\frac{1}{n}\right)e^{-\frac{m}{n}}\right) \\ &= n\left(\frac{m}{n}-1-\varepsilon+e^{-\frac{m}{n}}+\varepsilon e^{-\frac{m}{n}}\right)-\left(e^{-\frac{m}{n}}+\varepsilon e^{-\frac{m}{n}}\right) \\ &\geq n\left(-1+\frac{m}{n}-\varepsilon+\varepsilon e^{-\frac{m}{n}}+\sum_{j=0}^{\infty}\frac{\left(-\frac{m}{n}\right)^j}{j!}\right)-2 \\ &\overset{\varepsilon:=\frac{1}{2}\left(\frac{m}{n}\right)^2}{=} n\left(\frac{1}{2}\left(\frac{m}{n}\right)^2 e^{-\frac{m}{n}}-\frac{1}{6}\left(\frac{m}{n}\right)^3+\sum_{j=4}^{\infty}\frac{\left(-\frac{m}{n}\right)^j}{j!}\right)-2 \\ &\geq n\left(\frac{m}{n}\right)^3 = \frac{m^3}{n^2} = \frac{n}{q^3} . \end{aligned}$$

The last inequality holds for $m = n/q$ and $q \geq 3$.

∎

With the help of this lemma we are able to bound the number of requests that are left after one access step:

**Lemma 5.3** *Let the number of active keys be at least $\frac{n}{q}$. After performing one access step, i.e. all modules independently answer one key randomly, at least $\frac{n}{q^{5a}}$ active keys are left, w.h.p., as long as $q \leq n^{\frac{1}{4a+2}}$.*

**Proof**. W.l.o.g. we can make the analysis using new random hash functions at each step. The old hash function can be viewed in step $t$ as random function distributing the active keys randomly among the active modules. A new random hash function has the range of all $n$ modules in each step. Hence the probability for an active key to be answered increases.

Define the following random variables:

$$\begin{aligned} X &= \#\text{ answered keys in this round} \\ X &= \sum_{i=1}^{n/q} X_i,\ X_i := \begin{cases} 1 & \text{key i will be answered} \\ 0 & \text{else} \end{cases} \\ Y &= m - X = \#\text{ not answered keys} \\ Y &= \sum_{i=1}^{n/q} Y_i,\ Y_i := \begin{cases} 1 & \text{key i will be left} \\ 0 & \text{else} \end{cases} \end{aligned}$$

Apply Lemma 5.2 to all hash functions to obtain that, if we consider the hash functions independently, with respect to every hash function at least $\frac{n}{q^3}$ keys remain. To handle the dependencies between the hash functions we have to compute the probability that a key is answered by at least one hash function or not answered by any hash functions.

$$\begin{aligned} P(Y_i = 1) &= P(\text{ key i will be left in all hash functions}) \\ &\geq \left(\frac{\frac{n}{q^3}}{\frac{n}{q}}\right)^a = \left(\frac{1}{q^2}\right)^a. \end{aligned}$$

$$\begin{aligned} \Rightarrow E(Y) &\geq \frac{n}{q}\left(\frac{1}{q^2}\right)^a = \frac{n}{q^{2a+1}} \\ \Rightarrow E(X) &= \frac{n}{q} - E(Y) \leq \frac{n}{q}\left(1 - \frac{1}{q^{2a}}\right) = \mu^*. \end{aligned}$$

The $X_i$ are self-weakening. Applying Theorem 2.6 yields:

$$\begin{aligned} P\left(X \geq (1+\varepsilon)\mu^*\right) &= P\left(X \geq (1+\varepsilon)\frac{n}{q}\left(1 - \frac{1}{q^{2a}}\right)\right) \\ &\leq e^{-\frac{\varepsilon^2}{3}\frac{n}{q}\left(1-\frac{1}{q^{2a}}\right)} \text{ for } 0 \leq \varepsilon \leq 1. \end{aligned}$$

Choose $\varepsilon := \frac{1}{q^{2a}}$:

$$\begin{aligned} P\left(X \geq \frac{n}{q}\left(1 - \frac{1}{q^{4a}}\right)\right) &\leq \left(\frac{1}{e}\right)^{\frac{n}{3q^{2a+1}}\left(1-\frac{1}{q^{2a}}\right)} \\ &\leq \left(\frac{1}{e}\right)^{\frac{n}{6q^{2a+1}}} \text{ for } q > 1. \end{aligned}$$

The probability is at least polynomial small for $q \leq n^{\frac{1}{4a+2}}$. We want to derive from this expression a bound for $Y$:

$$P\left(Y \leq \frac{n}{q^{4a+1}}\right) \leq P\left(\frac{n}{q} - X \leq \frac{n}{q^{4a+1}}\right) \leq P\left(X \geq \frac{n}{q}\left(1 - \frac{1}{q^{4a+1}}\right)\right).$$

Hence the probability for $Y$ being smaller than $\frac{n}{q^{4a+1}}$ is polynomial small.

∎

If we consider only a set $A$ of keys satisfying the assumptions in Lemma 5.1, Lemma 5.3 immediately implies the following theorem:

**Theorem 5.4** *Any simple scheme based on a random, independent hash functions to distribute the shared memory needs expected time* $\Omega\left(\frac{\log\log n}{\log a}\right)$, *for* $a \leq (\log n)^\delta$, $\delta < 1$.

**Proof**. We start with the set $A$ of keys described in Lemma 5.1, i.e. with $\frac{n}{q}$ keys, $q = 3\log n$. Lemma 5.3 states that after $t$ rounds we are left with at least $\frac{n}{q^{5a^t}}$ requests as long as $q^{5a^t} \leq n^{\frac{1}{4a+2}}$. This condition holds for

$$\begin{aligned} t &\leq \frac{\log\log n - \log(4a+2) - \log\log q}{\log 5a} \\ &= \Omega\left(\frac{\log\log n}{\log a}\right) \text{ for } a \leq (\log n)^\delta,\ \delta < 1. \end{aligned}$$

Because of Lemma 5.1 this bound holds with constant probability and therefore the expected time follows.

∎

For shared memory simulations, also $a/2$ is a lower bound, because at least $\frac{a}{2}$ copies of each of the $n$ keys have to be updated, but only $n$ updates can be done in one step.

**Corollary 5.5** *Any shared memory simulation within the class of simple access protocols based on a random, independent hash functions has expected delay* $\Omega(a + \frac{\log\log n}{\log a})$. *Thus any choice of a can only yield simulations with expected delay* $\Omega(\frac{\log\log n}{\log\log\log n})$, *i.e. the result from Theorem 4.5 is optimal.*

# 6 Conclusions

Note that we can get rid of the factor $\binom{\lfloor \frac{2a}{a-b} \rfloor}{2}$ in the running time stated in Theorem 4.5. In the Appendix it is proved that the following lemma holds even if phase 3 of the advanced $(n, a, b)$-scheme is perfomed on hash functions that have already been used in phase 1. Thus we do not have to use a partitioning of the hash functions as necessary for our scheme in Section 4.2.

**Lemma 6.1** *If phase 3 of the advanced* $(n, a, b)$*-scheme is started with the same hash functions as phase 1 then at most* $n/2^{a^t-2}$ *keys are still active after* $t$ *rounds of the process we execute in phase 3, w.h.p. In particular, phase 3 finishes within* $\frac{\log\log n}{\log(a-b)+\log\log a} + 3$ *rounds, w.h.p.*

Note that if in this case the advanced $(n, a, b)$-scheme successfully terminates then each module has to answer at most $c_1(\log^* a - \log^* b + 2) + c_2$ requests, where $c_1 = \max\{\lceil \frac{a}{a-b} \rceil, 4\}$ and $c_2 = \log(\frac{2e(a-1)}{a-b})$. For $b = 1$ this reduces to $4\log^* a + 11$, for $b = \frac{a}{2} + 1$ this reduces to 16.

# References


[1] R.J. Anderson and G.L. Miller, Optical communication for pointer based algorithms, Technical Report CRI 88-14, Computer Science Department, University of Southern Carolina, 1988.

[2] C. Berge, *Graphs and Hypergraphs* (North-Holland, Amsterdam, 1973).

[3] H. Bast and T. Hagerup, Fast and reliable parallel hashing, in: *Proc. SPAA'95*, 50-61.

[4] A. Czumaj, F. Meyer auf der Heide and V. Stemann, Shared memory simulations with triple-logarithmic delay, in: *Proc. ESA'95*, 46-59.

[5] J.L. Carter and M.N. Wegman, Universal classes of hash functions, *J. Comput. Syst. Sci.* **18** (1979) 143-154.

[6] M. Dietzfelbinger, A. Karlin, K. Mehlhorn, F. Meyer auf der Heide, H. Rohnert and R. Tarjan, Dynamic perfect hashing: upper and lower bounds, *SIAM J. of Comput.* **4** (1994) 738-761.

[7] M. Dietzfelbinger and F. Meyer auf der Heide, How to distribute a hash table in a complete network, in: *Proc. STOC'90*, 117-127.

[8] M. Dietzfelbinger and F. Meyer auf der Heide, Simple efficient shared memory simulations, in: *Proc. SPAA'93*, 110-119.

[9] M. Geréb-Graus and T. Tsantilas, Efficient optical communication in parallel computers, in: *Proc. SPAA'92*, 41-48.

[10] L. A. Goldberg, M. Jerrum and T. Leighton, A doubly logarithmic communication algorithm for the completely connected optical communication parallel computer, in: *Proc. SPAA'93*, 300-309.

[11] L.A. Goldberg, Y. Matias and S. Rao, An optical simulation of shared memory, in: *Proc. SPAA'94*, 257-267.

[12] T. Hagerup and C. Rüb, A guided tour of Chernoff bounds, *Inform. Proc. Letters* **33** (1989/90) 305-308.

[13] R. Karp, M. Luby and F. Meyer auf der Heide, Efficient PRAM simulation on a distributed memory machine, Technical Report TR-RI-93-134, University of Paderborn, Sept. 1993. To appear in Algorithmica. A preliminary version appeared in *Proc. STOC'92*, 318-326.

[14] F. Meyer auf der Heide, Hashing strategies for simulating shared memory on distributed memory machines, in: F. Meyer auf der Heide, B. Monien and A.L. Rosenberg, ed., *Proc. of the 1st Heinz Nixdorf Symposium "Parallel Architectures and Their Efficient Use"*, LNCS, Vol. 678 (Springer, Berlin, 1992) 20-29.

[15] P.D. MacKenzie, C.G. Plaxton and R. Rajamaran, On contention resolution protocols and associated phenomena, in: *Proc. STOC'94*, 153-162.

[16] Y. Matias and U. Vishkin, Converting high probability into nearly-constant time – with applications to parallel hashing, in: *Proc. STOC'91*, 307-316.

[17] A. Panconesi and A. Srinivasan, Fast algorithms for distributed edge coloring, in: *Proc. PODC'92*, 251-262.

[18] A. Siegel, On universal classes of fast high performance hash functions, their time-space tradeoff, and their applications, in: *Proc. FOCS'89*, 20-25.

[19] J. Schmidt, A. Siegel and A. Srinivasan, Chernoff-Hoeffding bounds for applications with limited independence, in: *Proc. SODA'93*, 331-340.

[20] E. Upfal and A. Wigderson, How to share memory in a distributed system, *J. Assoc. Comput. Mach.* **34** (1987) 116-127.

[21] L. Valiant, Parallelism in comparison problems, *SIAM J. Comp.* **4** (1975) 348-355.

[22] L. Valiant, General purpose parallel architectures, in: J. van Leeuwen, ed., *Handbook of Computer Science* (Elsevier, Amsterdam, 1990), Chapter 18.

# A Appendix

**Proof of Lemma 6.1:**
Let $s$ be the biggest integer such that $\sum_{i=0}^{s} a_i \leq a$, and let $l \geq 2$. In order to analyze phase 3 of the advanced $(n, a, b)$-scheme under the assumption that the same hash functions as in phase 1 are used, we want to consider the following events:

- $E_t$ : Module $M_{x,y}$ is still active after $t$ rounds
- $E'_{t,k}$ : the $k$ selected keys provide an embedding for $T_t$
- $E''_k$ : the $k$ selected keys survived phase 1

Let

$$p'_{a,t} := (a+1) \cdot \frac{(\log m)^2}{2} \left(\frac{1}{m}\right)^{\frac{c^2(a-b)}{c+1}} + \left(\frac{1}{2}\right)^{\sum_{j=1}^{t-2}(c(a-b))^j}$$

We want to show that $\text{Prob}(E_t) \leq p'_{a,t}$ for all $t$, $2 \leq t \leq \frac{\log\log n}{\log(c(a-b))} + 3$. The proof for the upper bound for the number of keys that are still active after $t$ rounds will then be analogous to the proof of the Main Theorem. It holds:

$$\text{Prob}(E_t) \leq \sum_{k=c+1}^{|V_{TK}|} \binom{n}{k} \cdot \text{Prob}(E'_{t,k}) \cdot \text{Prob}(E''_k \mid E'_{t,k})$$

According to the proof of Lemma 3.3 we get (note that $n$ has to be substituted by $m = \frac{n}{a}$)

$$\text{Prob}(E'_{t,k}) \leq \sum_{i=u(k)}^{k} |B_{k,i}| \cdot \phi_{k,i} \cdot p_{k,i}.$$

It remains to calculate $\text{Prob}(E''_k \mid E'_{t,k})$. The active keys consist of those keys surviving for $s = 0$ the initial reduction scheme in one of $a$ groups $A_j^0 := A_j$ or for $s > 0$ round $s$ of the final reduction scheme in one of $\frac{a}{a_s}$ groups $A_j^s$. To keep sentences in the following short let us simply call the situation for $s \geq 0$ *iteration* $s$. Let

$$g := \begin{cases} a & : \quad s = 0 \\ \frac{a}{a_s} & : \quad s > 0 \end{cases}$$

be the number of groups that are melted together to one group in phase 3 and let $\tilde{T}$ be a base tree for the event $E'_{t,k}$, that is, $\tilde{T}$ has $k$ $V_{TK}$-nodes. Furthermore, let $X \in \{0, \ldots, k-1\}$ be a random variable for the number of keys embedded in a $V_{TK}$-node $v$ of $\tilde{T}$ that have a key embedded in the grandfather of $v$ which comes from a different group $A_j^s$ of keys than itself. The probability for such an event is $1 - \frac{1}{g}$ for each $V_{TK}$-node in the base tree independently from the others. There are $\binom{k-1}{r}$ possibilities to choose $V_{TK}$-nodes for the event $X = r$ (the leftmost $V_{TK}$-node at the top of $T_t$ serves as a basis for the partition of $\tilde{T}$). Thus it holds :

$$\text{Prob}(E''_k \mid E'_{t,k}) \leq \sum_{r=0}^{k-1} \binom{k-1}{r} \left(1 - \frac{1}{g}\right)^r \left(\frac{1}{g}\right)^{(k-1)-r} \cdot \text{Prob}(E''_k \mid E'_{t,k} \ \wedge \ X = r)$$

According to the definition of $X$, for the case $X = r$ $\tilde{T}$ can be partitioned into $r+1$ subtrees with $V_{TK}$-nodes as roots and with keys embedded only from one group $A_j^s$ in each subtree. Because up to $c$ subtrees with keys embedded only from one group $A_j^s$ can be connected with each other by a $V_{TM}$-node in $\tilde{T}$ we can partition $\tilde{T}$ into at least $\lceil \frac{r}{c} \rceil + 1$ subtrees with $V_{TM}$-nodes as roots and with keys embedded only from one group $A_j^s$.

We call $\tilde{T}$ a *minimal deadlock tree* if either $X = 0$ and the embedding of $\tilde{T}$ induces a deadlock structure in the access graph, or $X > 0$ and the embedding of $\tilde{T}$ does *not* induce a deadlock structure within any group $A_j^s$ of keys. Surely, if there exists a deadlock structure in the access graph then there

exists a valid embedding for a minimal deadlock tree. This means that we can restrict ourselves to counting minimal deadlock trees instead of all deadlock trees.

In the following we will only consider deadlock-free trees or minimal deadlock trees $\tilde{T}$. First, let $\tilde{T}$ be a tree with $X > 0$. Then, according to our assumptions, $\tilde{T}$ has at least $\lceil \frac{X}{c} \rceil + 1$ deadlock-free subtrees with $V_{TM}$-nodes as roots and with keys embedded only from one group $A_j^s$. Let $T'$ be one of these subtrees. Then one of the following two cases holds:

(1) There is a key $x$ embedded in $T'$ for which at most $a_s - b$ modules are known by $T'$ that store a copy of $x$. (Otherwise $x$ has a complete neighbourhood of $a_s - b + 1$ modules in $T'$.) For this case we refer to Lemma A.1.

(2) There is a module $M$ embedded in $T'$ for which at most $c$ keys are known by $T'$ that have a copy stored in it. (Otherwise $M$ has a complete neighbourhood of $c + 1$ keys in $T'$.) For this case we refer to Lemma A.2

It is not difficult to see that, if no such key $x$ or module $M$ exists in $T'$ then the embedding of $T'$ implies a deadlock structure for group $A_j^s$, that is, our assumptions for $\tilde{T}$ are not fulfilled.

**Lemma A.1** *Let $k$ be the number of $V_{TK}$-nodes in a base tree $\tilde{T}$ for the situation $E'_{t,k}$. Let $T'$ and $x$ be defined as in case (1) above, and let $E_x$ be the event that $x$ is still active after $t'$ rounds of the $(m, \frac{1}{a_s}, a_s, b, c)$-process at iteration $s$ of the advanced $(n, a, b)$-scheme. Then it holds :*

$$Prob(E_x) \leq \frac{1}{2^{(c(a_s-b))^{t'-2}}}$$

**Proof**. Let all keys embedded in $T'$ belong to the group $A_j^s$. According to our assumptions at most $a_s - b$ modules are known by $T'$ that store copies of $x$. As a worst case for Prob$(E_x)$ we will suppose that for each module $M'$ out of these at most $a_s - b$ modules the embedding of $T'$ already provides an embedding of $T_{t'}$ into the access graph of $A_j^s$ with root embedding $M'$. But we still have to find one more embedding of $T_{t'}$ into the access graph of $A_j^s$ with one of the remaining $b$ modules that store copies of $x$ embedded in its root to fulfill the event $E_x$. Let $M'$ be one of these remaining modules and $E_{M'}$ be the event that $M'$ is still blocked at round $t'$ of iteration $s$. As we will see, if we know the probability for $E_{M'}$ we can easily estimate the probability for the event $E_x$. In order to get an upper bound for the probability that $E_{M'}$ is true we need to estimate the probability that $T_{t'}$ can be embedded in the access graph of $A_j^s$ with root embedding $M'$ under the assumption that the embedding of $\tilde{T}$ is valid.

Let $k'$ be the number of keys from $A_j^s$ embedded in $T_{t'}$ that are not embedded in $\tilde{T}$. We have to distinguish three cases.

$k' = 0$ **:** For this case the upper bound of the probability that $T_{t'}$ can be embedded completely in the access graph of $A_j^s$ is

$$\text{Prob}(E_{M'} \mid k' = 0) \leq \frac{k}{m} + \left(\frac{k}{m}\right)^c$$

$\tilde{T}$ can have at most $k$ $V_{TM}$-nodes embedded in the same hash function as $M'$, so the probability that $x$ is stored in a module already used in $\tilde{T}$ is at most $\frac{k}{m}$. If this is not the case then there must be at least $c$ keys besides $x$ that are already used in $\tilde{T}$ and are stored in $M'$. The probability for this event is at most $(\frac{k}{m})^c$.

$k' = 1..c$ **:** Let $\tilde{T}'$ be a base tree in $T_{t'}$ with $k'$ $V_{TK}$-nodes, $i$ of them inner nodes and $\mathcal{E}'$ be its set of expansion nodes. None of the keys embedded in $V_{TK}$-nodes in $\tilde{T}'$ is allowed to be a key already embedded in $\tilde{T}$, but in the expansion nodes all keys from $A_j^s$ are allowed to be embedded. Let us call keys or modules that are not already embedded into $\tilde{T}$ *new* and otherwise *old*. Let $p$ be the number of new keys that are embedded in the highest row of $V_{TK}$-nodes in $T_{t'}$. If we consider all possibilities for base trees $\tilde{T}'$ for any triple $(k', i, p)$ then we obviously cover all the possibilities for valid base trees in $T_{t'}$. Let $r(k', i, p)$ be the number of expansion nodes a base

tree with parameters $(k', i, p)$ possesses and $j$ be the number of old nodes embedded in $\mathcal{E}'$. Then the probability for the expansion nodes is bounded above by

$$\begin{aligned} & \sum_{j=0}^{r(k',i,p)} \binom{r(k',i,p)}{j} \left(\frac{1}{m}\right)^{\lceil \frac{r(k',i,p)-j}{c+1} \rceil} \left(\frac{k}{m} + \sqrt[c]{\frac{k}{m}}\right)^j \\ \leq & \left(\sqrt[c+1]{\frac{1}{m}} + \left(\frac{k}{m} + \sqrt[c]{\frac{k}{m}}\right)\right)^{r(k',i,p)}, \end{aligned}$$

because it holds:

- $\binom{r(k',i,p)}{j}$ is the number of ways to distribute the old keys among the expansion nodes.
- $\left(\frac{1}{m}\right)^{\lceil \frac{r(k',i,p)-j}{c+1} \rceil}$ is the overall probability for the new keys, see Proof of Lemma 3.3.
- The probability that a key already used in $\tilde{T}$ is embedded in an expansion node is at most $\frac{k}{m}$. Furthermore, the probability that the module embedded in a $V_{TM}$-node above an expansion node is already embedded in $\tilde{T}$ is at most $\frac{k}{m}$. Because at most $c$ expansion nodes can have one common $V_{TM}$-node as father we get an additional factor of $\sqrt[c]{k/m}$ for each expansion node with an old key.

We can improve this probability bound for $k' \in \{1, \ldots, c\}$. Suppose that $p$ new keys are embedded into $V_{TK}$-nodes of the highest row of $T_{t'}$. For each of the $a_s - b$ $V_{TM}$-nodes $v$ below these $p$ $V_{TK}$-nodes it holds (recall that only $k - p \leq c - p$ new keys are left):
Either the module embedded in $v$ is old (this is true with probability $\frac{k}{m}$) or if the module embedded in $v$ is new at least $c - (c - p) = p$ old keys have to be embedded below $v$ to get a valid embedding for $\tilde{T}'$ (this is true with probability $(\frac{k}{m})^p$).

Let $u(k', p)$ be a lower bound for $i$ for fixed values of $k'$ and $p$. Obviously, $u(k', p) \geq p \geq 1$ and $r(k', i, p) \geq p \cdot c(a_s - b) - (c - p)$ (recall the definition of $r(k, i)$ in Proposition 3.5), that is, $r(k', i, p) \geq \frac{r(k',i,p)}{3} + p(a_s - b)$. So we get the following probability bound for the expansion nodes :

$$\begin{aligned} & \left(\sqrt[c+1]{\frac{1}{m}} + \left(\frac{k}{m} + \sqrt[c]{\frac{k}{m}}\right)\right)^{r(k',i,p)-p(a_s-b)} \cdot \left[\left(\frac{k}{m}\right)^p + \frac{k}{m}\right]^{p(a_s-b)} \\ \leq & \left(\sqrt[c+1]{\frac{1}{m}} + \left(\frac{k}{m} + \sqrt[c]{\frac{k}{m}}\right)\right)^{\frac{r(k',i,p)}{3}} \cdot \left[\left(\frac{k}{m}\right)^p + \frac{k}{m}\right]^{p(a_s-b)} \end{aligned}$$

Because the hash functions used at round $s$ have never been used before with keys in $A_j^s$ we get analogous to the proof of Lemma 3.3:

$$\begin{aligned} & \text{Prob}(E_{M'} \mid k' = 1..c) \\ \leq & \sum_{k'=1}^{c} \binom{m_s}{k'} \left(\frac{1}{a_s}\right)^{k'} \sum_{p=1}^{k'} \binom{k'}{p} (k'-p)! \left(\frac{k}{m}\right)^{c-p} \sum_{i=u(k',p)}^{k'} \left[\binom{a_s-1}{a_s-b} m^{a_s-b}\right]^i \cdot \\ & \left(\frac{1}{c!}\right)^{i(a_s-b)} \binom{i \cdot c(a_s-b)}{r(k',i,p)} \left(\sqrt[c+1]{\frac{1}{m}} + \left(\frac{k}{m} + \sqrt[c]{\frac{k}{m}}\right)\right)^{\frac{r(k',i,p)}{3}} \cdot \\ & \left[\left(\frac{k}{m}\right)^p + \frac{k}{m}\right]^{p(a_s-b)} \left(\frac{1}{m}\right)^{k'+i(a_s-b)} \\ \leq & \sum_{k'=1}^{c} \sum_{p=1}^{k'} \frac{1}{p!} \left(\frac{k}{m}\right)^{c-p} \sum_{i=1}^{k'} \left(\frac{1}{2}\right)^i \left(\frac{k}{m}\right)^{2p-1} < \frac{1}{m} \end{aligned}$$

$k' > c$ **:** Let $p_{a_s,t'}$ be defined as in the proof of the Main Theorem. Then we get:

$$\begin{aligned}
&\mathrm{Prob}(E_{M'} \mid k' > c) \\
&\leq \sum_{k'=c+1}^{|V_{TK}|} \binom{m_s}{k'} \left(\frac{1}{a_s}\right)^{k'} \sum_{p=1}^{c} \binom{k'}{p} (k'-p)! \left(\frac{k}{m}\right)^{c-p} \sum_{i=u(k',p)}^{k'} \left[\binom{a_s-1}{a_s-b} m^{a_s-b}\right]^i \cdot \\
&\quad \left(\frac{1}{c!}\right)^{i(a_s-b)} \binom{i \cdot c(a_s-b)}{r(k',i,p)} \left(k'^{\,\frac{1}{c+1}} \sqrt{\frac{1}{m}} + \left(\frac{k}{m} + \sqrt[c]{\frac{k}{m}}\right)\right)^{r(k',i,p)} \left(\frac{1}{m}\right)^{k'+i(a_s-b)} \\
&\leq p_{a_s,t'} + \sum_{p=1}^{c-1} \frac{1}{p!} \left(\frac{k}{m}\right)^{c-p} \sum_{k'=c+1}^{|V_{TK}|} \sum_{i=u(k',p)}^{k'} \left(\frac{1}{2}\right)^i \left(\sqrt[c+1]{\frac{1}{m}} + \left(\frac{k}{m} + \sqrt[c]{\frac{k}{m}}\right)\right)^{r(k',i,p)/2} \\
&\leq p_{a_s,t'} + \sum_{p=1}^{c-1} \frac{1}{p!} \left(\frac{k}{m}\right)^{c-p} \left(|V_{TK}| \left(\frac{1}{m}\right)^{\frac{1}{2(c+1)}} + c \left(\frac{1}{2}\right)^{c(a_s-b)}\right) \\
&\leq p_{a_s,t'} + \sum_{p=1}^{c-1} \frac{1}{p!} \left(\frac{k}{m}\right)^{c-p} \leq p_{a_s,t'} + \frac{k}{m}
\end{aligned}$$

The key $x$ has edges to $a_s$ modules $M'_1, \ldots, M'_{a_s}$ in the access graph of the group $A^s_j$. So altogether it holds :

$$\begin{aligned}
\mathrm{Prob}(E_x) &\leq \sum_{i=1}^{a_s} \mathrm{Prob}(E_{M'_i}) \\
&= \sum_{i=1}^{a_s} \Big(\mathrm{Prob}(E_{M'_i} \mid k' = 0) + \mathrm{Prob}(E_{M'_i} \mid k' = 1..c) + \mathrm{Prob}(E_{M'_i} \mid k' > c)\Big) \\
&\leq a_s \cdot \left(p_{a_s,t'} + \frac{2k+1}{m}\right) \leq \frac{1}{2^{(c(a_s-b))^{t'-2}}}
\end{aligned}$$

∎

**Lemma A.2** *Let $k$ be the number of $V_{TK}$-nodes in a base tree $\tilde{T}$ for the situation $E'_{t,k}$. Let $T'$ and $M$ be defined as in case (2) above, and let $E_M$ be the event that $M$ is still blocked at round $t'$ of the $(m, \frac{1}{a_s}, a_s, b, c)$-process at iteration $s$ of the advanced $(n, a, b)$-scheme. Then it holds :*

$$Prob(E_M) \leq \frac{1}{2^{(c(a_s-b))^{t'-3}}}$$

**Proof**. Let all keys embedded in $T'$ belong to the group $A^s_j$. According to our assumptions at most $c$ keys are known that have a copy stored in $M$. Thus, as an upper bound for $\mathrm{Prob}(E_M)$ we can use the probability that there exists a key $y$ with a copy in $M$ that was still active after $t'-1$ rounds of iteration $s$. Let us define the following events:

- $E^1_y$: the key $y$ has a copy in $M$ and was still active at the beginning of iteration $s$.
- $E^2_y$: the key $y$ was still active after $t'-1$ rounds of iteration $s$.

For the calculation of the probability of $E^1_y$ we have to consider two cases:

- Case 1: $s = 0$. Then $\mathrm{Prob}(E^1_y) \leq \frac{1}{m}$.
- Case 2: $s > 0$. Then $\mathrm{Prob}(E^1_y) \leq \frac{1}{m} \cdot \frac{1}{a_s}$.

Let $m_0 := m$ (recall that $m_s = a_s m$ for $s > 0$) and $y_1, \ldots, y_{m_s}$ be all keys in $A_j^s$. Then we get:

$$\begin{aligned} \text{Prob}(E_M) &\leq \sum_{i=1}^{m_s} \text{Prob}(E_{y_i}^1) \cdot \text{Prob}(E_{y_i}^2 \mid E_{y_i}^1) \\ &\leq \sum_{i=1}^{m_s} \frac{1}{m_s} \cdot \text{Prob}(E_{y_i}^2 \mid E_{y_i}^1) \end{aligned}$$

In order to get an upper bound for the probability that $E_y^2$ is true under the assumption that $E_y^1$ is true we need to estimate the probability that a key $y$ in $A_j^s$ that has one copy in module $M$ has copies in at least $a_s - b$ other modules for which there is an embedding of $T_{t'-1}$ into the access graph of $A_j^s$. Thus according to Lemma A.1 it holds for $t \geq 2$ (recall that $a_s - b > 1$):

$$\begin{aligned} \text{Prob}(E_y^2 \mid E_y^1) &\leq \binom{a_s - 1}{a_s - b} \cdot \left( p_{a_s, t'-1} + \frac{2k+1}{m} \right)^{a_s - b} \\ &\leq \frac{1}{2^{(c(a_s-b))^{t'-3}}} \end{aligned}$$

Now we can estimate the probability that $E_M$ is true:

$$\begin{aligned} \text{Prob}(E_M) &\leq \sum_{i=1}^{m_s} \frac{1}{m_s} \cdot \frac{1}{2^{(c(a_s-b))^{t'-3}}} \\ &= \frac{1}{2^{(c(a_s-b))^{t'-3}}} \end{aligned}$$

∎

If we combine the results of Lemma A.1 and Lemma A.2 we see that we get an additional probability of at least $1/2^{(c(a_s-b))^{t'-3}}$ for every subtree $T'$ in $\tilde{T}$ as defined above. In the following let $t' = 5$. Now we can start to prove an upper bound for $\text{Prob}(E_k'' \mid X = r)$. We will distinguish here between the case $s = 0$ and $s > 0$.

**Case 1:** $s > 0$.

**Lemma A.3** *Let $a$, $b$ and $c$ be chosen as in Lemma 3.4. Then it holds for $s > 0$ and $r > 0$:*

$$Prob(E_k'' \mid X = r) \leq \left( \frac{1}{2a_s} \right)^k \left( \frac{1}{g} \right)^{r+1}$$

**Proof**. Let $T'$ be defined as above, let $q$ be the number of different keys embedded in $T'$. Suppose that none of the modules embedded in $T'$ belongs to a hash function used in round $s-1$. Suppose round $s-1$ took $l+2$ rounds of our process. Then for $l \geq 2$ according to Lemma 3.4 the probability that all keys embedded in $T'$ are still active after round $s-1$ is at most

$$\left( p_{a_{s-1}, l+2} + \frac{1}{m} \right)^q \leq \left( \frac{1}{2a_s} \right)^q$$

Let $v$ be a $V_{TM}$-node in $T'$. If we change the embedding of $v$ in a way that we embed a module in $v$ from a hash function already used in round $s-1$ for $d$ keys embedded in neighbors of $v$, $2 \leq d \leq c+1$ then we can partition $T'$ into $r$ trees, that is, we get an addition probability of

$$\left( \frac{1}{2^{(c(a_s-b))^{t'-3}}} \right)^{d-1} / \left( \frac{1}{2a_s} \right)^d$$

which is at most 1 for $t' = 5$. Thus as a worst case we assume that all hash functions embedded in $T'$ have only been used in iteration $s$. Therefore it holds for all $l' \geq 3$ and $r > 0$:

$$\begin{aligned}\text{Prob}(E''_k \mid X = r) &\leq \left(\frac{1}{2a_s}\right)^k \cdot (\text{Prob}(E))^{\frac{r}{c}+1} \\ &\leq \left(\frac{1}{2a_s}\right)^k \left(\frac{1}{2^{(c(a_s-b))^2}}\right)^{\frac{r}{c}+1} \leq \left(\frac{1}{2a_s}\right)^k \left(\frac{1}{g}\right)^{r+1}\end{aligned}$$

∎

So altogether we get:

$$\text{Prob}(E''_k \mid E'_{t,k} \wedge X > 0) \leq \frac{1}{g} \cdot \left(\frac{1}{g}\left(1 - \frac{1}{g}\right) + \frac{1}{g}\right)^{k-1} \cdot \left(\frac{1}{2a_s}\right)^k \leq \frac{1}{a_s}\left(\frac{1}{a}\right)^k$$

Let $p_{a,t}$ be defined as

$$p_{a,t} := \frac{(\log m)^2}{2}\left(\frac{1}{m}\right)^{\frac{c^2(a-b)}{c+1}} + \left(\frac{1}{2}\right)^{\sum_{j=1}^{t-2}(c(a-b))^j}$$

Then it holds (recall that $n = a \cdot m$):

$$\begin{aligned}\text{Prob}(E_t \mid X > 0) &\leq \sum_{k=c+1}^{|V_{TK}|} \frac{n^k}{k!} \cdot \text{Prob}(E'_{t,k}) \cdot \text{Prob}(E''_k \mid E'_{t,k} \wedge X > 0) \\ &\leq \sum_{k=c+1}^{|V_{TK}|} \frac{m^k}{k!} \cdot \text{Prob}(E'_{t,k}) \cdot \frac{1}{a_s} \\ &\leq \frac{1}{a_s} \cdot p_{a,t}\end{aligned}$$

We still have to consider the case where $\tilde{T}$ is only embedded into keys from one group $A_j^s$, that is, $X = 0$. If $\tilde{T}$ has $k$ $V_{TK}$-nodes then for this case a similar argumentation as in Lemma A.3 yields that $\text{Prob}(E''_k \mid X = 0) \leq (\frac{1}{a_s})^k$. So if we count over all $g$ groups $A_j^s$ we get:

$$\begin{aligned}\text{Prob}(E_t \mid X = 0) &\leq g \cdot \sum_{k=c+1}^{|V_{TK}|} \frac{m_s^k}{k!} \cdot \text{Prob}(E'_{t,k}) \cdot \text{Prob}(E''_k \mid E'_{t,k} \wedge X = 0) \\ &\leq g \cdot \sum_{k=c+1}^{|V_{TK}|} \frac{m^k}{k!} \cdot \text{Prob}(E'_{t,k}) \cdot \left(\frac{1}{2}\right)^k \leq \ldots \\ &\leq g \cdot \frac{(\log m)^2}{2}\left(\frac{1}{m}\right)^{\frac{c^2(a-b)}{c+1}} + \frac{1}{2} \cdot \left(\frac{1}{2}\right)^{\sum_{j=1}^{t-2}(c(a-b))^j}\end{aligned}$$

So altogether we get $\text{Prob}(E_t) \leq p'_{a,t}$ for the case $s > 0$.

**Case 2:** $s = 0$.
For $c(a-b) \geq a$ and $a - b > 1$ it holds for $t' = 5$:

$$\text{Prob}(E''_k \mid X = r) \leq \left(\frac{1}{2^{(c(a_s-b))^2}}\right)^{\frac{r}{c}+1} \leq \left(\frac{1}{g}\right)^{r+1}$$

Thus we get:

$$\text{Prob}(E''_k \mid E'_{t,k} \wedge X > 0) \leq \frac{1}{g} \cdot \left(\frac{1}{g}\left(1 - \frac{1}{g}\right) + \frac{1}{g}\right)^{k-1} \leq \frac{1}{a} \cdot \left(\frac{2}{a}\right)^{k-1}$$

Let us now suppose that we have $\frac{n}{2}$ active keys instead of $n$ active keys at the beginning of the advanced $(n,a,b)$-scheme. We can get this situation by running phase 1 to 3 seperately for the first and the second half of the keys. Then we get for each half :

$$\begin{aligned}\text{Prob}(E_t \mid X>0) &\leq \sum_{k=c+1}^{|V_{TK}|} \frac{(n/2)^k}{k!} \cdot \text{Prob}(E'_{t,k}) \cdot \text{Prob}(E''_k \mid E'_{t,k} \wedge X>0) \\ &\leq \sum_{k=c+1}^{|V_{TK}|} \frac{m^k}{2 \cdot k!} \cdot \text{Prob}(E'_{t,k}) \leq \frac{1}{2} p_{a,t}\end{aligned}$$

We still have to consider the case $X=0$. For this case it is easy to see that we get:

$$\begin{aligned}\text{Prob}(E_t \mid X=0) &\leq g \cdot \sum_{k=c+1}^{|V_{TK}|} \frac{(m/2)^k}{k!} \cdot \text{Prob}(E'_{t,k}) \cdot \text{Prob}(E''_k \mid E'_{t,k} \wedge X=0) \\ &\leq g \cdot \sum_{k=c+1}^{|V_{TK}|} \frac{m^k}{2 \cdot k!} \cdot \text{Prob}(E'_{t,k}) \leq \ldots \\ &\leq g \cdot \frac{(\log m)^2}{2} \left(\frac{1}{m}\right)^{\frac{c^2(a-b)}{c+1}} + \frac{1}{2} \cdot \left(\frac{1}{2}\right)^{\sum_{j=1}^{t-2}(c(a-b))^j}\end{aligned}$$

So altogether it also holds $\text{Prob}(E_t) \leq p'_{a,t}$ for the case $s=0$. Note that if at most $(1-\frac{1}{a})n$ keys are active at the beginning of the advanced $(n,a,b)$-scheme then we do not have to partition the set of keys any more to get $\text{Prob}(E_t) \leq p'_{a,t}$ for the case $s=0$. ∎