INTERNATIONAL COMPUTER SCIENCE INSTITUTE

1947 Center St. • Suite 600 • Berkeley, California 94704-1198 • (510) 643-9153 • FAX (510) 643-7684

Parallel Balanced Allocations

Volker Stemann

TR-96-020

June 1996

Abstract

We study the well known problem of throwing m balls into n bins. If each ball in the sequential game is allowed to select more than one bin, the maximum load of the bins can be exponentially reduced compared to the 'classical balls into bins' game.

We consider a static and a dynamic variant of a randomized parallel allocation where each ball can choose a constant number of bins. All results hold with high probability. In the static case all m balls arrive at the same time. We analyze for $m = n$ a very simple optimal class of protocols achieving maximum load $O\left(\sqrt[r]{\frac{\log n}{\log\log n}}\right)$ if r rounds of communication are allowed. This matches the lower bound of [ACMR95].

Furthermore, we generalize the protocols to the case of $m > n$ balls. An optimal load of $O(m/n)$ can be achieved using $\frac{\log\log n}{\log(m/n)}$ rounds of communication. Hence, for $m = n\frac{\log\log n}{\log\log\log n}$ balls this slackness allows to hide the amount of communication. In the 'classical balls into bins' game this optimal distribution can only be achieved for $m = n\log n$.

In the dynamic variant n of the m balls arrive at the same time and have to be allocated. Each of these initial n balls has a list of m/n successor-balls. As soon as a ball is allocated its successor will be processed. We present an optimal parallel process that allocates all $m = n\log n$ balls in $O(m/n)$ rounds. Hence, the expected allocation time is constant. The main contribution of this process is that the maximum allocation time is additionally bounded by $O(\log\log n)$.

1 Introduction

In the 'classical balls into bins' game m balls are thrown independently and uniformly at random (i.u.a.r.) into n bins. The distribution of the balls in the bins is well known ([KSC78]). For $m = n$ there exists a bin getting $\Theta\left(\frac{\log n}{\log \log n}\right)$ balls with high probability (w.h.p.)[1]. Azar et al. [ABKU94] consider a modified sequential game where each ball chooses i.u.a.r. $d \geq 2$ bins and is placed in the bin with the smaller load. They show a $\Theta(\log \log n / \log d)$ bound on the maximum load of the bins for this GREEDY strategy, w.h.p.. This game has many applications to computing problems, e.g. dynamic resource allocation, hashing, and competitive on-line load balancing (see [ABKU94]).

In this paper we consider the parallel version of the 'balls into bins' game. In the parallel case the balls do not arrive sequentially, instead several balls arrive simultaneously and have to be placed into the bins. We are interested in parallelizations without a global control, i.e., each ball is placed in a bin using only very local information. This leads to randomized strategies. In contrast to the sequential case we have a third parameter, the degree of parallelization, i.e., the number of balls that can be processed (thrown) at the same time.

Naturally, the question arises if the idea of choosing randomly more than one possible destination for a ball has a similar great impact on the maximum load in the parallel case as it had in the sequential case. To answer this question we consider a *static* and a *dynamic* variant of a parallelization.

In the *static* variant all m balls arrive at the same time. A *parallel protocol* has to place the balls in the bins minimizing the maximum load and using only a limited amount of communication. Obviously, there exists a tradeoff between the amount of communication available for the decision where to place the balls and the maximum load of each bin.

In the *dynamic* variant n of the m balls arrive at the same time. But as soon as a ball is allocated, a new ball, a successor, is processed. Each of the n balls at the beginning has m/n successors. A *parallel process* has to allocate all m balls using $O(m/n)$ rounds. That is, we focus on *optimal* processes with constant expected allocation time. Our goal is to minimize the maximum allocation time of a ball.

For $m = n$ a parallel protocol and a parallel process are the same. Therefore, they can be viewed as generalizations of this special case in different directions.

1.1 Parallel protocols

In the *classical balls into bins* game each of m balls is placed i.u.a.r. in one of n bins.

We consider an extension of this model described by Adler et al. [ACMR95]. At the beginning each of the m balls chooses i.u.a.r. d bins. In this paper we focus on the case $d = 2$. The balls decide their final destination using r rounds of communication. A round of communication consists of two phases. In a first phase each ball can send a message to the two bins it has chosen. In a second phase each bin sends back a message to all balls that it has received a message from. All these replies are done simultaneously. Finally, each ball decides in which of the two bins to be placed.

A more restricted version of this model suffices to achieve significant improvements compared to the 'classical balls into bins' game. We consider only protocols that are *non-*

[1] 'with high probability' means with probability at least $1 - n^{-\alpha}$ for a constant $\alpha \geq 1$.

adaptive and *symmetric*. Non-adaptive means that the possible destinations are chosen before any communication takes place. Symmetric means that all bins perform the same underlying algorithm and all possible destinations are chosen i.u.a.r.

Adler et al. ([ACMR95]) consider the case where $m = n$ balls are thrown into n bins. Their main result is a lower bound on the maximum load of a bin for the restricted class of protocols. They show an $\Omega\left(\sqrt[r]{\frac{\log n}{\log\log n}}\right)$ lower bound for the maximum load if r rounds of communication are allowed. For a constant number of rounds of communication they match the lower bound using a THRESHOLD protocol. If the number of rounds r grows with n, their best result is a maximum load of $\log\log n + O(1)$ using $r = \log\log n + O(1)$ rounds of communication. They achieve this result with a parallel version of the GREEDY algorithm described by Azar et al. ([ABKU94]).

We present a class of protocols, the COLLISION protocols, matching the lower bound of Adler et al. ([ACMR95]) for the whole range of r. In a k-COLLISION protocol a ball can only be placed in a bin if at most $k-1$ other balls have chosen this bin. Obviously this means that the maximum load of a bin is k. The simplicity of this protocol makes it very practical. Therefore, we analyze the constants in the number of rounds and in the maximum load. The COLLISION protocols can be reformulated such that they perform almost asynchronously. This answers the open question of Adler et al. [ACMR95] if there exists a simple protocol that achieves the same performance as their generalization of the GREEDY protocol.

Furthermore, we extend the k-COLLISION protocols to values $m > n$. It is well known that if $m = \Omega(n \log n)$ balls are thrown into n bins we get a distribution where the maximum load of the bins differs from the expected load only by a constant factor. That is, for large values of m the 'classical balls into bins' protocol performs very well. We show that for any m this distribution where the maximum load of all bins differs from the expected load only by a constant factor can be achieved with the k-collision protocol. We have to choose $k = \Theta(m/n)$ and allow a sufficient amount of communication, $\Omega\left(\log\log n / \log(m/n)\right)$ rounds. The protocols are optimal as the maximum load of each bin is obviously bounded from below by the expected load m/n. If we take $m = O(n\frac{\log\log n}{\log\log\log n})$ balls, a distribution as described above can be achieved using an optimal number of $r = O(\frac{\log\log n}{\log\log\log n})$ rounds.

The analysis of our protocols only needs $O(\log n)$-wise independent random variables, i.e., the destinations chosen by the balls only have to be $O(\log n)$-wise independent. Nevertheless, to keep the presentation simple we assume always that these variables are completely independent.

The problems appearing in shared memory simulations are related to this model of load balancing. In a shared memory simulation n requests have to be delivered to n memory modules. Contention at the memory modules results in a delay of the simulation of a parallel access. The main goal is to minimize the delivering time, i.e. the number of rounds to satisfy all requests ([KLM92, GMR94, MPR94, CMS95, MSS95]). Nevertheless, several ideas and techniques can be transferred to this problem. Especially, k-COLLISION protocols for $m \leq n$ and constant k are also studied in [DM93, GMR94, MPR94, MSS95]. They achieve the same bounds for this special setting of the parameters.

1.2 Parallel processes

A parallel process can be viewed as having n players and n bins. Each player has a list of τ balls to be allocated. At the beginning each of the $n\tau$ balls chooses i.u.a.r. a constant number d of bins where it can be allocated. It suffices if a ball is allocated at one of the d possible destinations.

The process performs in rounds. In each round each player can try to allocate the first ball in its list to the d possible destinations. If several players try to allocate a ball at the same bin, the balls are stored in a FIFO queue at the bin. The balls in the list of a player have to be allocated one after the other, i.e., a player is only able to try to allocate the next ball in the list if the previous one has successfully been processed. Hence, the situation can also be viewed as generating a new ball as soon as the previous one has successfully been allocated.

Our main goal is to minimize the *allocation time* of a ball, i.e. the difference between the first round the ball is the first one in the list of the player and the round this ball is finally allocated. We focus on *optimal* processes where all balls should be allocated in $O(\tau)$ rounds, i.e., the expected allocation time is constant. Within this class of processes we want to minimize the maximum allocation time.

In a naive approach the players just try to allocate one ball after the other, choosing for each ball $d = 1$ bin i.u.a.r.. Aiello et al. [LAB93] and independently Dietzfelbinger and Meyer auf der Heide [DM93] show that for $\tau = \log n$ this SIMPLE process is optimal. But the maximum allocation time is $\Omega(\log n / \log\log n)$.

On the other hand gluing the COLLISION protocols together decreases the maximum allocation time to $O(\log\log n)$ but the resulting process is not optimal.

We present an optimal process that needs $O(\tau)$ rounds for $\tau = \log n$ and has maximum allocation time $O(\log\log n)$. It chooses $d = 3$ possible destinations i.u.a.r. for each ball and decreases the maximum allocation time exponentially.

Similar processes which try to achieve constant expected allocation time have also been studied in other contexts, e.g. shared memory simulations (see e.g. [DM93, LAB93]).

1.3 Applications

Distributed load balancing

Consider a scenario described by Adler et al. [ACMR95] where m client workstations issue jobs (balls) that have to be allocated at decentralized compute-servers. The clients are ignorant about other clients submitting jobs. The main goal is to minimize the maximum load at each server. Using a random strategy avoids the high cost of a global coordination. The COLLISION protocols immediately apply to this scenario. In addition, this randomized protocol is highly fault-tolerant. If a server crashes a client just makes new random choices. The resulting maximum load is covered by our analysis for $m \geq n$ balls.

Also the parallel process fits in this application. Assume a client has to wait for a submitted job to be allocated until it can submit the next one. The parallel process gives a probabilistic guarantee that for a sufficiently large number of jobs the expected allocation time is constant. Also we get an $O(\log\log n)$ upper bound on the maximal allocation time.

Hence, a client knows an upper bound when he can start his k-th job in the worst case.

Parallel models with limited bandwidth

In the PRAM(n) model, a recently proposed limited bandwidth model ([MNV94, ABK95]), m processors communicate through a small, globally shared memory consisting of n memory cells. Viewing the requests of the processors as balls that have to be allocated to the bins, we can simulate a PRAM with large shared memory on this model. We use a constant time simulation of a PRAM(n) on a module parallel computer (MPC) with n memory modules described in [MNV94]. c copies of each shared memory cells of the large PRAM are i.u.a.r. distributed among the modules using c random functions. The memory accesses to the shared memory can be viewed as a 'balls into bins' game. Each request of a processor is a ball thrown to the c random locations in the modules of the memory cell that it wants to access. An extension of the majority technique due to Upfal and Wigderson [UW87] allows to restrict to the case that only one copy has to be allocated (see e.g. [CMS95]). Hence an m-processor PRAM can be simulated on an m-processor PRAM(n) with $O(m/n + \log\log n)$ delay, w.h.p..

Universal dynamic mapping on MIMD machines

Consider the problem of mapping dynamically generated tasks onto processors of a MIMD-system. Decker et al. [DDLM95] look at this problem and want to construct an algorithm which can be integrated in a distributed runtime system like PVM or MPI. In [DDLM95] they consider only the sequential process of Azar et al. [ABKU94]. If more than one task arrives at the same time, the COLLISION protocols and the parallel allocation process can be used to bound the maximum load and the maximum allocation time.

1.4 Organization

In the next section we first introduce the class of COLLISION protocols. Then we analyze their performance for the case $m = n$, matching the existing lower bound. In the last part of Section 2 we extend the analysis to the case $m > n$. In Section 3 we finally present the optimal parallel process that achieves maximum allocation time $O(\log\log n)$.

2 COLLISION protocols

In this section we consider a class of protocols called COLLISION protocols. That is, a ball can only be allocated at a bin if at most $k-1$ other balls want to be allocated at this bin.

k-COLLISION

- *in parallel each ball b chooses i.u.a.r. two bins $i_1(b)$ and $i_2(b)$.*
- **while** *there is a ball that has not been allocated*
  - *in parallel each non-allocated ball b sends a request to $i_1(b)$ and $i_2(b)$*
  - *in parallel each bin getting at most k requests sends an acknowledgment to all requesting balls.*

– *each ball getting an acknowledgment is allocated to the respective bin (ties are broken arbitrarily)*

We call one iteration of the while-loop a *round*. Obviously in each round each ball only has to send two messages and each bin only has to handle $k+1$ messages. If a bin gets more than k messages it can stop receiving and stays silent. If it gets less than k requests it has to send at most k acknowledgments.

This protocol is synchronous, i.e., each ball has to wait for the end of each round to get to know if it will get an acknowledgment or not. But the protocol can also be converted into an almost asynchronous one.

ASYNCHRON-COLLISION

- *in parallel each ball b chooses i.u.a.r. two bins $i_1(b)$ and $i_2(b)$.*
- *in parallel each ball b sends a request to $i_1(b)$ and $i_2(b)$*
- **while** *there is a ball that has not been allocated*
  - *in parallel each bin having at most k requests sends an acknowledgment to all requesting balls.*
  - *each ball getting an acknowledgment is allocated in the respective bin (ties are broken arbitrarily) and sends a message to the other bin that it already has been allocated.*

The ASYNCHRON-COLLISION protocol only needs one synchronization just before the while-loop starts. This is because a bin has to know if it will get at least k requests. The while-loop can be performed asynchronously. Both protocols are from the point of view of our analysis the same. Therefore, we only analyze the first protocol. Obviously both protocols are non-adaptive and symmetric, i.e., the lower bound of Adler et al. ([ACMR95]) can be applied to both protocols.

2.1 The case $m = n$

Even if the general case $m \geq n$ subsumes the content of this subsection we want to present this special case in detail for two reasons. First, it clarifies the ideas of the proof and simplifies the presentation. Second we get a slightly better bound in this special case that matches the lower bound of Adler et al. ([ACMR95]) which only holds for this special case. We evaluate the tradeoff between r, the number of rounds of communication, and k, the maximum load of each bin, using n balls and n bins. The main result of this section is the following theorem.

Theorem 2.1 *Let $m = n$, α be a constant and $2 \leq r \leq \frac{\log\log n}{3\alpha}$. The k-COLLISION protocol is finished after r rounds for $k = \max\{\sqrt[r]{\frac{3\alpha r \log n}{\log\log n}}, 4(\alpha+7)\}$ with probability at least $1 - \frac{1}{n^{\alpha-1}}$.*

To analyze the k-COLLISION protocols we model the 'balls into bins' game as a game on a graph $G = (V, E)$. A similar modeling is also used in [KLM92, GMR94, CMS95]. As we need this model also in the subsequent sections we define it for the general case $m \geq n$. The bins are the nodes V of the graph G, $|V| = n$. The balls are represented by the edges E, $|E| = m$. There is an edge $(v_1, v_2) \in E$ labeled b if the ball b chooses i.u.a.r. the two bins $i_1(b) = v_1$ and $i_2(b) = v_2$. Since the bins are chosen i.u.a.r. from each ball, G is a random labeled graph with n vertices and m edges.

The k-COLLISION protocol (the while loop of the protocol) can now be viewed as a procedure of removing edges from the graph G. Again we perform in rounds. In each round in parallel each node that has degree at most k removes all incident edges from the graph. The procedure is finished when all edges are removed from the graph.

To analyze this procedure we need to know more about the structure of the random graph G. First we want to determine what kind of substructure has to be embedded into G such that an edge is still left after r rounds. This is the case if in the $(r-1)$st round both incident nodes had degree at least $k+1$. But this is only the case if these edges themselves were incident in round $r-2$ to nodes of degree $k+1$. Repeating this inductive argument it is easily seen that for an edge (or a node) being left after r rounds we get at least a complete k-ary tree of depth r that has to be embedded in G. Let us call this complete k-ary tree a *witness-tree* $W_{k,r}$. Unfortunately, $W_{k,r}$ does not have to be a subgraph of G. Two nodes of $W_{k,r}$ can be embedded to the same node of G. Then the subtree below the node on the lower level can be completely mapped onto the subtree below the other node. Hence, the size of the embedding of $W_{k,r}$ is much smaller than the original size. That is, if we identify nodes in the embedding, the tree is 'folded together'. On the other hand, we create a cycle if we embed two nodes of $W_{k,r}$ in the same node of G, i.e., we need for this embedding a *non-tree edge* of G.

We want to describe the tradeoff between the size (number of nodes) of the embedding and the number of non-tree edges used. Consider an embedding of $W_{k,r}$ using w non-tree edges, i.e., the embedding is a tree with w additional edges. Only $2w$ of the k branches starting at the root of $W_{k,r}$ can use a non-tree edge for the embedding. That is, for $k \geq 2w$ at least $k - 2w$ branches have to be embedded one-to-one in G. Therefore a k-ary tree of depth r where only the root has degree $k - 2w$ has to be a subgraph of G. Note that the embedding with the smallest number of nodes is the tree having w self-loops at the root. For $w \geq k/2$ a deadlock can occur. This is a special case that has to be handled separately in the analysis.

The proof of Theorem 2.1 is split into two parts for different sizes of w. For small w we show that the 'large' k-ary tree with root-degree $k - 2w$ is very unlikely to be a subgraph of G. For larger values of w the embedding becomes smaller but more dense. We show that it is very unlikely for a dense graph to be a subgraph of G. Of course both arguments only hold for values of k and r that are large enough.

The following two technical lemmas are the basis of our proofs. The first lemma is already stated for the general case $m \geq n$. A simpler version for $m < n$ is already included in [KLM92].

Lemma 2.2 *Let G be a random graph with n nodes and $m = \lambda n$ edges, $1 \leq \lambda \leq \log n$. Each subgraph S consisting of $s \leq \beta \log n$ nodes has at most $s + w - 1$ edges with probability*

$1 - n^{-\alpha}$ *for* $w \geq 1 + \alpha + \beta(3 + \log\lambda)$ *and constant* $\alpha, \beta \geq 1$.

Proof: Consider the probability that a graph S with s nodes and $s + w - 1$ edges is a subgraph of the random graph G. We have at most $\binom{n}{s}$ possibilities to choose the s nodes, $\binom{s^2+s+w-2}{s+w-1}$ possibilities to choose the $s + w - 1$ edges, and at most m^{s+w-1} possibilities to label these edges. The probability that all $s+w-1$ labeled edges have the right connections is $\frac{1}{n^{2(s+w-1)}}$. Therefore, we can bound the probability that G has a subgraph S by

$$\begin{aligned}
&\binom{n}{s}\binom{s^2+s+w-2}{s+w-1}(\lambda n)^{s+w-1}\frac{1}{n^{2(s+w-1)}} \\
&\leq \left(\frac{ne}{s}\right)^s\left(\frac{(s^2+s+w-2)e}{s+w-1}\right)^{s+w-1}\lambda^{s+w-1}\frac{1}{n^{s+w-1}} \\
&\leq (e^2\lambda)^s\left(\frac{es\lambda}{n}\right)^{w-1} \\
&= 2^{(2\log e+\log\lambda)s}\left(\frac{es\lambda}{n}\right)^{w-1}
\end{aligned}$$

For $s \leq \beta\log n$ and $w \geq 1 + \alpha + \beta(3 + \log\lambda)$ this term is smaller than $n^{-\alpha}$. □

Lemma 2.3 *Let* $T_{k,r}$ *be a complete* k*-ary tree of depth* $2 \leq r \leq \frac{1}{3\alpha}\log\log n$ *where only the root has degree* $k/2$*. For* $k \geq \sqrt[r]{\frac{3\alpha r\log n}{\log\log n}}$ *the graph* $T_{k,r}$ *is not a subgraph of* G *with probability* $1 - n^{-\alpha+1}$ *for constant* α.

Proof: Let t be the size of $T_{k,r}$, i.e. the number of nodes. At least $\frac{1}{2}k^{r-1}$ nodes in $T_{k,r}$ have k children. We want to bound the probability that $T_{k,r}$ is a subgraph of G.

There are $\binom{n}{t}$ ways to choose the t nodes and $t!$ ways to order them. There are $\frac{n!}{(n-(t-1))!}$ ways to label the edges. Finally, as $T_{k,r}$ is a complete tree we have at least $(k!)^{\frac{1}{2}k^{r-1}}$ automorphism which can be eliminated. The probability that all edges have the correct connections is $\frac{1}{n^{2(t-1)}}$.

Hence, we can bound the probability that $T_{k,r}$ is a subgraph of G by

$$\begin{aligned}
&\binom{n}{t}t!\frac{n!}{(n-(t-1))!}\frac{1}{n^{2(t-1)}}\frac{1}{(k!)^{\frac{1}{2}k^{r-1}}} \\
&\leq \frac{n^t n^{t-1}}{n^{2t-2}}\frac{1}{(k!)^{\frac{1}{2}k^{r-1}}} = n\frac{1}{(k!)^{\frac{1}{2}k^{r-1}}} \leq n\left(\frac{e}{k}\right)^{\frac{k^r}{2}}
\end{aligned}$$

This expression is polynomially small for $k \geq \sqrt[r]{\frac{3\alpha r\log n}{\log\log n}}$ and $r \leq \frac{1}{3\alpha}\log\log n$:

$$\leq n\left(\frac{e}{\sqrt[r]{\frac{3\alpha r\log n}{\log\log n}}}\right)^{\frac{3\alpha r\log n}{2\log\log n}}$$

$$
\begin{aligned}
&\leq n\left(\frac{e^r \log\log n}{3\alpha r \log n}\right)^{\frac{3\alpha \log n}{2\log\log n}} \\
&\leq n\left(\frac{e^{\frac{1}{3\alpha}} \log\log n}{\log n}\right)^{\frac{3\alpha \log n}{2\log\log n}} \\
&\leq \left(\frac{1}{n}\right)^{(\frac{3}{2}\alpha - \frac{\log e}{2})-1} \leq \left(\frac{1}{n}\right)^{\alpha-1}
\end{aligned}
$$

□

Now we can prove Theorem 2.1.

Proof: [of Theorem 2.1] For an edge to be left after r rounds of the k-COLLISION protocol a witness-tree $W_{k,r}$ has to be embedded into G. First, as $r \leq \frac{\log\log n}{3\alpha}$ and $k \geq \sqrt[r]{\frac{3\alpha r \log n}{\log\log n}}$ the size of $W_{k,r}$ is at most $2k^r \leq 2\log n$.

Hence, Lemma 2.2 states for $m = n$ that the embedding of $W_{k,r}$ uses at most $w = 7 + \alpha$ non-tree edges with probability $1 - n^{-\alpha}$. If we choose $k \geq 4w$ at least a k-ary tree $T_{k,r}$ of depth r where only the root has degree $k/2$ has to be a subgraph of G. For $k \geq \sqrt[r]{\frac{3\alpha r \log n}{\log\log n}}$ Lemma 2.3 states that w.h.p. $T_{k,r}$ is not a subgraph of G, i.e., there is no embedding of $W_{k,r}$ using less than w non-tree edges. Hence, for k and r in the stated bounds the theorem follows.

□

As $\sqrt[r]{r}$ can be bounded by two for $r \geq 2$, Theorem 2.1 matches the lower bound of Adler et al. [ACMR95]. Interesting special cases of Theorem 2.1 are for constant k, i.e., we want to keep the maximum load in each bin as small as possible, and also for a constant number of rounds r.

Corollary 2.4 *The k-COLLISION protocol achieves w.h.p.*

- *maximum load $k = 32$ using $0.17\log\log n$ rounds and*
- *maximum load $k = \sqrt{\frac{12\log n}{\log\log n}}$ using two rounds.*

2.2 The case $m > n$

In this section we consider the case where $m > n$ balls are thrown i.u.a.r. into n bins. Again, we use the k-COLLISION protocol to distribute the balls.

Obviously, we have to choose k at least as large as the expected optimal load m/n. To avoid dead-locks of the protocol we choose $k = c \cdot \frac{m}{n}$ for a constant c to be specified below. For the rest of this section let $m = \lambda n$. We only analyze the case $1 \leq \lambda \leq O(\log n)$. For larger values of λ already the 'classical balls into bins' game performs optimal.

Theorem 2.5 *Let $m = \lambda n$, $k^r = \log n$, and α be a constant. The k-COLLISION protocol places m balls into n bins using r rounds for $k \geq \max\{2^{2\alpha+2}\lambda, 28 + 4\alpha + 8\log\lambda\}$, with probability at least $1 - \frac{1}{n^{\alpha-1}}$.*

We use the same proof techniques as in Subsection 2.1 to prove Theorem 2.5. We reformulate a slightly weaker version of Lemma 2.3 for the general case of $m > n$.

Lemma 2.6 *Let $G = (V, E)$ be a random graph with $|V| = n$ vertices and $|E| = \lambda n$ edges, $\lambda \geq 1$. Let $T_{k,r}$ be a complete k-ary tree of depth $r \geq 1$ where only the root has degree $k/2$. For $k^r \geq \log n$ and $k \geq 2^{2\alpha+2}\lambda$ the probability that $T_{k,r}$ is a subgraph of G is at most $n^{-\alpha+1}$.*

Proof: Let t be the size of $T_{k,r}$, i.e. the number of nodes. At least $\frac{1}{2}k^{r-1}$ nodes in $T_{k,r}$ have k children. We want to bound the probability that $T_{k,r}$ is a subgraph of G.

We have $\binom{n}{t}$ possibilities to choose the t nodes and $t!$ possibilities to order them. There are $\frac{m!}{(m-(t-1))!}$ possibilities to label the edges. Finally, as $T_{k,r}$ is a complete tree, we have at least $(k!)^{k^{r-1}}$ automorphism which can be eliminated.

Hence, we can bound the probability that $T_{k,r}$ is a subgraph of G by

$$\begin{aligned}
&\binom{n}{t} t! \frac{m!}{(m-(t-1))!} \frac{1}{n^{2(t-1)}} \frac{1}{(k!)^{\frac{k^r-1}{2}}} \\
\leq\ & \frac{n^t m^{t-1}}{n^{2t-2}} \frac{1}{(k!)^{\frac{k^r-1}{2}}} \\
=\ & n\lambda^{t-1} \frac{1}{(k!)^{\frac{k^r-1}{2}}} \\
\leq\ & n\lambda^{\frac{k^{r+1}-1}{2(k-1)}} \left(\frac{e}{k}\right)^{\frac{1}{2}k^r} \\
\leq\ & n\lambda^{\frac{1}{2}\frac{k^r+1}{k-1}} \left(\frac{e}{k}\right)^{\frac{1}{2}k^r} \\
\leq\ & n \left(\frac{e\lambda^{\frac{k}{k-1}}}{k}\right)^{\frac{1}{2}k^r}
\end{aligned}$$

As $k^r \geq \log n$ and $k \geq 2^{2\alpha+2}\lambda$ the probability is at most $n^{-\alpha+1}$. □

Now the proof of Theorem 2.5 can be done in the same way as the proof of Theorem 2.1 using Lemma 2.2 and Lemma 2.6.

Proof: [of Theorem 2.5] For an edge to be left after r rounds of the k-Collision protocol a witness-tree $W_{k,r}$ has to be embedded into G. We choose λ such that $(2^{2\alpha+2}\lambda)^r \leq \log n$. As $k^r \leq \log n$ the size of $W_{k,r}$ is at most $2k^r \leq 2\log n$.

Hence, Lemma 2.2 states that the embedding of $W_{k,r}$ uses at most $w = 7 + \alpha + 2\log\lambda$ non-tree edges with probability $1 - n^{-\alpha}$. If we choose $k \geq 4w$ at least a k-ary tree $T_{k,r}$ of depth r where only the root has degree $k/2$ has to be a subgraph of G. For $k^r = \log n$ and $k \geq 2^{2\alpha+2}\lambda$ Lemma 2.6 states that w.h.p. $T_{k,r}$ is not a subgraph of G, i.e., there is no embedding of $W_{k,r}$ using less than w non-tree edges. Hence, for $k^r = \log n$ and $k \geq \max\{2^{2\alpha+2}\lambda, 28 + 4\alpha + 8\log\lambda\}$ the theorem follows.

□

Theorem 2.5 says that it is always possible to place the balls such that the maximum load k of each bin is at most constant times larger than the expected load λ. We need

$\Omega(\log n / \log \lambda)$ rounds of communication for this purpose. However, an interesting special case is when the number of rounds r used is at most a constant factor away from the maximum load of each bin. This distribution can be achieved in the classical 'balls into bins' game only for $\lambda = \Omega(\log n)$.

Corollary 2.7 *For $\lambda = O(\log \log n / \log \log \log n)$ the maximum load k of each bin and the number of rounds r of the k-*Collision *protocol are $O(\lambda)$.*

3 Optimal parallel processes

In this section we extend the protocols described in the last section and by Adler et al. [ACMR95] to the case where each player wants to allocate more than one ball. Each of the n players has a list of τ balls to be placed in the n bins (servers). The balls in the list of a player have to be allocated one after the other, i.e., a player is only able to try to allocate the next ball in the list if the previous one has successfully been processed. Hence, the situation can also be viewed as generating a new ball as soon as the previous one has successfully been allocated.

We are interested in *optimal* processes, i.e., if each player has a list of τ balls, all balls should be allocated in time $O(\tau)$. Therefore the expected allocation time is constant. Our main goal is to minimize the *allocation time* of a ball, i.e. the difference between the first round the ball is first in the list of the player and the round this ball is finally allocated.

A naive approach is the τ-Simple process: Each player has a list of τ balls. The process performs in rounds and at the beginning all players are not busy. In each round each player that is not busy takes the first ball in its list, sends it to an i.u.a.r. chosen bin, and becomes busy. Each bin has a FIFO queue. Balls arriving in the same round are placed in an arbitrary order in the queue. In each round each bin accepts one ball from the queue. All players of successfully allocated balls become non-busy. The protocol stops after $O(\tau)$ rounds or when all balls are allocated.

Aiello et al. [LAB93] and independently Dietzfelbinger and Meyer auf der Heide [DM93] show the following result.

Lemma 3.1 *For $\tau = \log n$ the τ-*Simple *process allocates all balls within $O(\log n)$ rounds, w.h.p..*

Hence, the τ-Simple process is optimal but the maximum allocation can only be bounded by $\Omega(\log n / \log \log n)$. On the other hand, we can just glue together τ of the protocols described in Section 2 to a process. This gives an $O(\log \log n)$ maximum allocation time but the process needs $O(\tau \log \log n)$ rounds to finish, i.e., it is not optimal. We combine both techniques to get an optimal process with $O(\log \log n)$ maximum allocation time. First, we describe a very simple process that is part of the final process.

(τ, t)-Allocation process

Input: *Each player P_j posses a list of τ balls, $b_{j,1}, \ldots, b_{j,\tau}$. Each ball b has two i.u.a.r. chosen destinations $i_1(b)$ and $i_2(b)$.*

Task: *Allocate each ball to at least one of the two chosen bins.*

Process: *All players are non-busy.*

Each player performs the following round $c \cdot t$ times for a constant c or until all its balls are allocated.

- *If the player is non-busy it takes the next ball in its list, sends a copy to the two destinations, and becomes busy.*
- *The balls are stored in two FIFO queues at the bins. One queue for balls that are first copies and one for second copies. Balls arriving in the same round are stored in arbitrary order.*
- *Each non-empty bin accepts one ball from each FIFO queue.*
- *A player whose ball has been allocated becomes non-busy and the other copy of the ball will be eliminated.*

As this process consists in essence of two parallel SIMPLE processes, the $(\log n, \log n)$-ALLOCATION process allocates all balls in $O(\log n)$ time because of Lemma 3.1. A process similar to $(\log n, \log n)$-ALLOCATION but with a stronger collision rule for the bins is analyzed in [DM93]. They split the players into a constant number of groups and handle the groups sequentially to achieve the stronger result. Moreover, one can only get an $O(\log n)$ upper bound for the maximum allocation time from their proof.

Unfortunately, we are not able to analyze the maximum allocation time of a ball in the (τ, t)-ALLOCATION process for all values of τ and t. We have to split the process into phases such that during some periods of time several processors stay idle and wait for the others.

τ-ALLOCATION-PHASE process

Input: *Each player C_j posses a list of τ balls $b_{j,1}, \ldots, b_{j,\tau}$. Each ball b has three i.u.a.r. chosen destinations $i_1(b)$, $i_2(b)$, and $i_3(b)$.*

Task: *Allocate each ball to at least one of the three chosen bins.*

Process: *Repeat $\frac{\tau}{\log\log n}$ times the following phase:*

- *Each player only uses a cluster consisting of the next $\log\log n$ non-allocated balls in its list. It performs the $\log\log n$-SIMPLE process on this cluster using the first random destination $i_1(b)$ for each ball b.*
- *Each player that is still busy with a ball b performs the $(1, \log\log n)$-ALLOCATION-process with b using the two random destinations $i_2(b)$, and $i_3(b)$.*

Perform the $(\log n, \log n)$-ALLOCATION process with the remaining balls.

It seems that this process is not easier to analyze because the (τ, t)-ALLOCATION Process is a subroutine. However, we only use special settings of the parameter pair (τ, t). The splitting of the process into phases allows to analyze each phase separately. To bound the running time we need the following lemma:

Lemma 3.2 *For $\tau \leq \gamma \log n$ the τ-SIMPLE process allocates in $O(\tau)$ rounds all but $\frac{n}{2^\tau}$ balls, w.h.p., for a sufficiently small constant γ.*

For the proof we need a lemma that is implicitly stated in [LAB93] and also [DM93].

Lemma 3.3 *A player has not allocated all its τ balls after $c_1\tau$ rounds of the τ-*SIMPLE *process with probability $1/2^{2\tau}$ for $c_1 > 8$ and $\tau \leq \log n$.*

Proof: [of Lemma 3.2] Let X_i, $1 \leq i \leq n$, be the random $0-1$ variable indicating if player i has allocated all its τ balls after c_1 rounds of the τ-SIMPLE process. Lemma 3.3 bounds the expected value of $X = \sum_{i=1}^{n} X_i$.

$$E(X) \leq \frac{n}{2^{2\tau}}.$$

We want to bound the deviation from the expected value using a martingale tale estimate [McD89]. Let ω_j, $j = 1, \ldots, \tau n$, be the random choice made by ball b_j. The ω_j are independent and uniformly distributed over $\{1, \ldots, n\}$. X is a function f in these random choices: $X = f(\omega_1, \ldots, \omega_{\tau n})$. For the martingale tale estimate we have to bound the change C_j of X if we change the value of one ω_j. The ball b_j that changes its value from ω_j^{old} to ω_j^{new} has direct influence to the next ball that should have been removed from the bin $B_{\omega_j^{old}}$, from the bin $B_{\omega_j^{new}}$, and the next ball in the list of the processor b_j belongs to. Hence, as we run the protocol at most $c_1\tau$ rounds, the change of ω_j has influence to at most $3^{c_1\tau}$ balls. Therefore, at most $C_j = 3^{c_1\tau}$ of the processors change their value X_i. The formula for the martingale tail estimate,

$$Pr(|X - E(X)| \geq t) \leq e^{-\frac{2t^2}{\sum C_j}},$$

gives the following bound:

$$\begin{aligned} Pr(X \geq \frac{n}{2^\tau}) &\leq Pr(X - \frac{n}{2^{2\tau}} \geq \frac{n}{2^{2\tau}}) \\ &\leq e^{-\frac{2n^2}{2^{4\tau}\tau n 3^{2c_1\tau}}} \\ &= e^{-\frac{2n}{2^{4\tau+\log(\tau)+2\log 3c_1\tau}}} \\ &\leq e^{-\sqrt{n}} \end{aligned}$$

The last inequality holds for $\tau \leq \gamma \log n$ and γ being a sufficiently small constant. □

We are now ready to formulate the main theorem of this section:

Theorem 3.4 *The $\log n$-*ALLOCATION-PHASE *process allocates all balls in $O(\log n)$ rounds with maximum allocation time $O(\log\log n)$, w.h.p..*

Proof: The first step of each phase of the τ-ALLOCATION-PHASE, the $\log\log n$-SIMPLE process, takes time $O(\log\log n)$. Using Lemma 3.2 we get that it allocates at least $n\log\log n - \frac{n}{2^{\log\log n}}$ balls, w.h.p.. Therefore, at most $\frac{n}{\log n}$ processors are still busy with a ball after this step.

As $\tau = 1$, the $(1, \log\log n)$-ALLOCATION process in step 2 of each phase is a protocol in the sense described in Section 2. The running time of the $(1, \log\log n)$-ALLOCATION process is dominated by the running time of a k-COLLISION protocol for constant k because

the FIFO rule of the $(1, \log\log n)$-ALLOCATION process is dominated by the collision rule of the k-COLLISION protocol. This means that one step of the k-COLLISION protocol can be simulated within $O(k)$ steps of the $(1, \log\log n)$-ALLOCATION process. Hence, Corollary 2.4 says that the allocation of these balls can be done in time $O(\log\log n)$, w.h.p., using the other two random destinations of these balls. The main effect of this second step is that all remaining non-allocated balls have never been touched before, i.e., they are completely random and we are in the same situation as in the beginning of the phase. Only the lists of many players are a little bit shorter.

Each player now takes the next $\log\log n$ balls in its list and starts with a new phase. (Note that balls which have not been allocated yet, remain in the original list.) After $\log n / \log\log n$ phases at least

$$(n \log\log n - \frac{n}{\log n}) \frac{\log n}{\log\log n} = n \log n - \frac{n}{\log\log n}$$

balls have been allocated.

The final step of the τ-ALLOCATION-PHASE process, the $(\log n, \log n)$-ALLOCATION process, allocates all remaining $O(\frac{n}{\log\log n})$ balls in $O(\log n)$ rounds, w.h.p.. This follows from Lemma 3.1 since the $\log n$-SIMPLE process is a part of this final step. It remains to show that these balls have maximum delivering time $O(\log\log n)$, w.h.p.. We want to view the $\frac{n}{\log\log n}$ remaining balls of the final stage as a graph like in Subsection 2.1 with respect to the first and the second copy of each ball. As edges (balls) are added and eliminated from the graph we view the $(\log n, \log n)$-ALLOCATION process as a family of graphs over the time, $G_r, r = 1, \ldots, \log n$. Let $G = \bigcup_r G_r$ be the graph consisting of all $m = \frac{n}{\log\log n}$ edges.

Using a modification of Lemma 2.2 for $m < n$ that already appeared in [KLM93] we get that G has connected components of size at most $O(\log n)$ and the connected components are trees with only a constant number of additional edges.

Consider an edge (ball) that is the first time at the first position in the list of the player at time $\tilde{r}$ (thrown at time $\tilde{r}$). Since we have FIFO queues at the bins, a ball can only be blocked by balls that arrived before or at the same time, i.e., we only have to consider the connected components of $\bigcup_{r \leq \tilde{r}} G_r \subset G$. In a constant number of steps the constant number of additional edges in each connected component can be removed by the process. Hence, we only have to deal with trees as connected components. As in each step each bin allocates a ball, the size of each connected component decreases in each step by at least a half Therefore, a ball will be allocated in $O(\log\log n)$ rounds (see also the proof of Theorem 6.5 in [KLM93]). □

4 Conclusions

We have studied parallel allocation strategies, i.e. parallel 'balls into bins' games. Especially, we have investigated the effects if each ball chooses not only one but a constant number of possible destinations where it can be allocated. If the number of balls equals the number of bins we have shown a tight upper bound for the trade-off between the maximum load of each bin and the amount of communication used. Extensions of this protocol in two different directions for different settings of the parameters have shown that the multiple

choices of the balls either effect the maximum load of a bin or the maximum allocation time of the balls. As mentioned in [ACMR95] a future direction would be to associate a weight to each ball and minimize the maximum weight over all bins. Also we believe that the natural process $(\log n, \log n)$-ALLOCATION, or a slight extension, should achieve the same maximum allocation time as the more technical $(\log n)$-ALLOCATION-PHASE process.

Acknowledgments

The author wants to thank Micah Adler, Johannes Blömer, John Byers, Mike Mitzenmacher, and Mike Luby for helpful discussions.

References

[ABK95] M. Adler, J.W. Byers, and R.M. Karp. Parallel sorting with limited bandwith. In *Proceedings of the 7th Annual ACM Symposium on Parallel Algorithms and Architectures*, 1995.

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

[ACMR95] M. Adler, S. Chakrabarti, M. Mitzenmacher, and L. Rasmussen. Parallel randomized load balancing. In *Proceedings of the 27th Annual ACM Symposium on Theory of Computing*, pages 238–247, 1995.

[CMS95] A. Czumaj, F. Meyer auf der Heide, and V. Stemann. Improved optimal shared memory simulations, and the power of reconfiguration. In *Proceedings of the 3rd Israel Symposium on Theory of Computing and Systems*, pages 11–19, 1995.

[DDLM95] T. Decker, R. Diekmann, R. Lueling, and B. Monien. Towards developing universal dynamic mapping algorithms. In *Proceedings of the 7th Symposium on Parallel and Distributed Processing*, 1995.

[DM93] M. Dietzfelbinger and F. Meyer auf der Heide. Simple, efficient shared memory simulations. In *Proceedings of the 5th Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 110–119, 1993.

[GMR94] L. A. Goldberg, Y. Matias, and S. Rao. An optical simulation of shared memory. In *Proceedings of the 6th Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 257–267, 1994.

[KLM92] R. M. Karp, M. Luby, and F. Meyer auf der Heide. Efficient PRAM simulation on a distributed memory machine. In *Proceedings of the 24th Annual ACM Symposium on Theory of Computing*, pages 318–326, 1992.

[KLM93] R. M. Karp, M. Luby, and F. Meyer auf der Heide. Efficient PRAM simulation on a distributed memory machine. Technical report, University of Paderborn, September 1993. To appear in *Algorithmica*. A preliminary version is [KLM92].

[KSC78] V.F. Kolchin, B.A. Sevsatyanov, and V.P. Chistyakov. *Random Allocation*. V.H. Winston and Sons, 1978.

[LAB93] P. Liu, W. Aiello, and S. Bhatt. An atomic model for message-passing. In *Proceedings of the 5th Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 153–163, 1993.

[McD89] C. McDiarmid. On the method of bounded differences. In J. Siemons, editor, *Surveys in Combinatorics*, London Mathematical Society Lecture Note Series 141, pages 148–188. Cambridge University Press, 1989.

[MNV94] Y. Mansour, N. Nisan, and U. Vishkin. Trade-offs between communication throughput and parallel time. In *Proceedings of the 26th Annual ACM Symposium on Theory of Computing*, pages 372–381, 1994.

[MPR94] P. D. MacKenzie, C. G. Plaxton, and R. Rajaraman. On contention resolution protocols and associated probabilistic phenomena. In *Proceedings of the 26th Annual ACM Symposium on Theory of Computing*, pages 153–162, 1994.

[MSS95] F. Meyer auf der Heide, C. Scheideler, and V. Stemann. Exploiting storage redundancy to speed up randomized shared memory simulations. In *Proceedings of the 12th Annual Symposium on Theoretical Aspects of Computer Science*, pages 267–278, 1995.

[UW87] E. Upfal and A. Wigderson. How to share memory in a distributed system. *Journal of the ACM*, 34:116–127, 1987.