Fast Concurrent Access to Parallel Disks

Peter Sanders
Max-Planck-Institute for Computer Science
Im Stadtwald, 66123 Saarbrücken, Germany
sanders@mpi-sb.mpg.de

Sebastian Egner, Jan Korst
Philips Research Laboratories
Prof. Holstlaan 4, 5656 AA Eindhoven, The Netherlands
{egner,korst}@natlab.research.philips.com

Abstract

High performance applications involving large data sets require the efficient and flexible use of multiple disks. In an external memory machine with D parallel, independent disks, only one block can be accessed on each disk in one I/O step. This restriction leads to a load balancing problem that is perhaps the main inhibitor for the efficient adaptation of single-disk external memory algorithms to multiple disks. We solve this problem for arbitrary access patterns by randomly mapping blocks of a logical address space to the disks.

We show that a shared buffer of $\mathcal{O}(D)$ blocks suffices to support efficient writing. The analysis uses the properties of negative association to handle dependencies between the random variables involved. This approach might be of independent interest for probabilistic analysis in general.

If two randomly allocated copies of each block exist, N arbitrary blocks can be read within $\lceil N/D \rceil + 1$ I/O steps with high probability. The redundancy can be further reduced from 2 to $1 + 1/r$ for any integer r without a big impact on reading efficiency. From the point of view of external memory models, these results rehabilitate Aggarwal and Vitter's "single-disk multi-head" model [1] that allows access to D arbitrary blocks in each I/O step. This powerful model can be emulated on the physically more realistic independent disk model [2] with small constant overhead factors. Parallel disk external memory algorithms can therefore be developed in the multi-head model first. The emulation result can then be applied directly or further refinements can be added.

1 Introduction

Despite ever larger internal memories, even larger data sets arise in important applications like video on demand, data mining, electronic libraries, geographic information systems, computer graphics, or scientific computing. Often, no size limits are in sight. In this context, it is necessary to efficiently use multiple disks in parallel in order to achieve high bandwidth.

This situation can be modeled using the one processor version of Vitter and Shriver's *parallel disk model*: A processor with M words of *internal memory* is connected to D disks. In one *I/O step*, each disk can read or write one *block* of B words. For simplicity, we also assume that I/O steps are either pure read steps or pure write steps (Section 6.1 gives a more detailed discussion).

Efficient single-disk external memory algorithms are available for a wide spectrum of applications (e.g. [3]), yet parallel disk versions are not always easy to derive. We face two main tasks: firstly to expose enough parallelism so that at least D blocks can be processed concurrently and secondly to ensure that the blocks to be accessed are evenly distributed over the disks. In the worst case, load imbalance can completely spoil parallelism increasing the number of I/O steps by a factor of D. This paper solves the load balancing problem by placing blocks randomly and by using redundancy.

1.1 Summary of Results

In Section 2, we use queuing theory, Chernoff bounds and the concept of negative association [4] to show that writing can be made efficient if a pool of $W = \mathcal{O}(D/\epsilon)$ blocks of internal memory are reserved to support D *write queues*. This suffices to admit $(1-\epsilon)D$ new blocks to the write queues during nearly every write step. Subsequent read requests for blocks that have not yet been written, can be served from the write queues.

Since our model assumes separate read and write steps, we can analyze these two issues separately. Scheduling read accesses is more difficult since a parallel read has to wait until all requested blocks have been read. In Section 3, we investigate *random duplicate allocation* (RDA). Similar to *mirroring*, which is often used in practice to achieve fault tolerance, RDA uses two copies of each logical block but these copies are allocated to disks chosen randomly. Writing for RDA can be analyzed using the results from Section 3 keeping in mind that two physical blocks have to be written for each logical block. An optimal read scheduling algorithm based on maximum flow computations decides which of the two copies of each block is to be read. We show that N blocks can be retrieved using $L_{\max} = \lceil N/D \rceil + 1$ parallel read steps with high probability (whp). We call $L_{\max}$ the *maximum load*. Experiments reported in Section 3.5 and recent theoretical work [5, 6] indicate that $L_{\max} = \lceil N/D \rceil$ can be achieved if $\lceil N/D \rceil - N/D$ is sufficiently large. On the other hand, a lower bound derived in Section 3.4 shows that regardless of the allocation strategy $L_{\max} = \lceil N/D \rceil$ cannot be achieved if N is a multiple of D unless N/D is rather large. Furthermore, in Section 4 we show that schedules with $\lceil N/D \rceil + 1$ read steps can be found faster than the worst-case bounds of maximum flow algorithms would suggest.

In Section 5 we generalize RDA. Instead of writing two copies of each logical block, we split the logical block into r sub-blocks and produce an additional parity sub-block that is the bit-wise exclusive-or of these sub-blocks. These $r+1$ sub-blocks are then randomly placed as before. When reading a logical block, it suffices to retrieve any r out of the $r+1$ pieces. The scheduling algorithms allow simultaneous access to files that have been allocated with different degrees of redundancy. Much of the analysis also goes through as before. At the price of increasing the logical block size by a factor of r, we reduce the redundancy of RDA from 2 to $1 + 1/r$.

Our techniques for reading and writing can be joined to a quite far-reaching result. Aggarwal and Vitter's multi-head disk model [1] that allows access to D arbitrary blocks in each I/O step, can be efficiently emulated on the independent disk model [2]. In Section 6, we summarize how this can be exploited and adapted to yield improved parallel disk algorithms for many "classical" external memory algorithms for sorting, data structures and

computational geometry, as well as for newer applications like video-on-demand or interactive computer graphics. We also outline a further generalization of RDA which allows more fault tolerance.

1.2 Related Work

The predominant general technique to deal with parallel disks in practice is *striping* [7, 8]. In our terminology this means using logical blocks of size DB, which are split into D sub-blocks of size B—one for each disk. This yields a perfect load balance but is only effective if the application can make use of huge block sizes. For example, at currently realistic values of $D = 64$ and $B = 256$ KByte we would get logical blocks of 16 MByte. Since many external memory algorithms work best if thousands of I/O streams with separate buffer blocks are used, prohibitive internal memory requirements would result. Refer to [9] for a detailed discussion and simulation results.

Reducing access contention by random placement is a well-known technique. For example, Barve et al. [10] use it for a simple parallel disk sorting algorithm. However, in order to access N blocks in $(1+\epsilon)N/D$ steps, N must be at least $\Omega\left((D/\epsilon^2)\log D\right)$. If $N = \Theta(D)$, some disk will have to access $\Theta(\log D/\log\log D)$ blocks. Our result on writing from Section 2 shows how a buffer for $\mathcal{O}(D)$ blocks can solve this problem. Section 6.3 gives further details including references to subsequent sorting results building on Section 2 [11, 12].

Our results are also interesting from a more abstract point of view independent of the external memory model. Load balancing when two randomly chosen locations of load units are available has been studied using several models. Azar et al. [13] analyze an online strategy that commits each arriving request to the least loaded unit. Berenbrink et al. [14] analyzed this algorithm for the general case $N \neq D$ and showed that the maximum load is $L_{\max} = N/D + \log\ln D + \Theta(1)$.[1] Vöcking [15] showed that for $N = D$, a refined variant is better by a small constant factor and is optimal among all online strategies up to an additive constant. An obvious question is how much *better* we can do if we allow offline scheduling. The best previously known bound for offline algorithms was $L_{\max} = \mathcal{O}(N/D)$ which is *worse* than the online strategy for $N = \omega(D\log\log D)$. Refer to Section 4.4 for a discussion of some of these techniques which have the advantage of finding schedules in linear time. Our result yields an optimal offline strategy and shows that the gap between the online algorithm and an optimal offline strategy is $\Theta(\log\log D)$.

For PRAM simulation, fast parallel scheduling algorithms have been developed even earlier [16] achieving maximum load $\mathcal{O}(1)$. PRAM simulation using a 3-collision protocol achieves maximum load $\mathcal{O}(1)$ for $N = D$ using $\mathcal{O}(\log\log D)$ iterations [17],[18, Section 3]. This already works for $\mathcal{O}((\log D)^3)$-universal classes of hash functions. Similar results hold for allocation strategies with lower redundancy such as the ones we describe in Section 5.

Heuristic load balancing algorithms using redundant storage are used by a number of authors in multimedia applications [9, 19, 20, 21]. Even the idea of a parity sub-block built out of r data sub-blocks has been used by several researchers [22, 23]. The first optimal scheduling algorithm for RDA was presented in [9]. This and other papers give convincing experimental evidence that RDA is a good policy yet no closed form results were known which

[1] Throughout this paper $\log x = \log_2 x$.

prove that the same is true for systems of arbitrary size or which explain why RDA is so good. Our results close this gap. We prove the optimality of the scheduling algorithm, generalize it to parity encoding, analyze the quality achieved, and speed up the scheduling algorithm. A subsequent paper [5, 6] further refines our results: When is $L_{\max} = \lceil N/D \rceil$ possible? What about disk failures, more fault tolerance, variable length blocks, and communication bottlenecks? Another related issue are asynchronous disk scheduling algorithms that strive to minimize waiting times in a continous stream of requests [24].

There are several papers on deriving external memory algorithms from parallel algorithms, by emulating the parallel machine on an external memory machine [25, 26, 27, 28]. Most of these results also allow parallel disks [25, 27, 28] so that this is currently the main source of parallel disk external memory algorithms. Chiang et al. [25] look at PRAM algorithms with special structure. Dehne et al. [27, 28] look at BSP algorithms and obtain efficient external algorithms for the special case of CGM algorithms. CGM algorithms work in supersteps and communicate their entire memory content in every superstep. Our emulation result adds the multi-head model [1] as a new source of algorithms that can be efficiently emulated on parallel disks. A possible advantage of this model is that no processing parallelism and only rather limited data access parallelism is needed.

2 Queued Writing

This section shows that a fraction of $1-\varepsilon$ of the peak bandwidth for writing can be reached by making $W = \mathcal{O}(D/\varepsilon)$ blocks of internal memory available to buffer write requests. We first show this for any write-once access pattern that never writes a block twice. In Section 2.2 we explain how the multiple access case can be handled. We assume that if blocks are mapped to disks uniformly and independently at random. This could be implemented using a RAM resident directory keeping track of the physical positions of logical blocks. Note that the space requirements for the directory are not a big practical problem since we only need a few bytes for each block with hundreds of kilobytes. The modified allocation strategy described in Section 6.2 reduces the directory size by another factor of D. Nevertheless, in practice one may choose to map blocks using some more space efficient hash function h. Our randomness assumption is then only an approximation of the truth. However, viewing hash functions as fully random is a fairly standard assumption in the analysis of probabilistic algorithms.

Procedure `write`$((1-\varepsilon)D$ blocks):
  append blocks to $Q_1, \ldots, Q_D$;
  `write-to-disks`$(Q_1, \ldots, Q_D)$;
  while $|Q_1| + \cdots + |Q_D| > W$ **do**
    `write-to-disks`$(Q_1, \ldots, Q_D)$.

Figure 1: Queued Writing.

The buffer consists of queues $Q_1, \ldots, Q_D$, one for each disk. Initially, all queues are empty. Then the application invokes the procedure `write` shown in Figure 1 to write up to

$(1-\varepsilon)D$ blocks. After each invocation of `write`, the queues consume at most W blocks of internal memory.[2] The procedure `write-to-disks` stores the first block of each non-empty queue onto the disks in parallel. Note that read requests to blocks pending in the queues can be serviced directly from internal memory.[3]

Section 2.1 proves Theorem 1 which represents the main result on writing, namely that a global buffer size W which is linear in D suffices to ensure that on the average, a call of the write procedure incurs only about one I/O step, i.e., one invocation of `write-to-disks`. The theoretical treatment is complemented by experimental findings in Section 2.3 which suggest even better performance.

Theorem 1 *Consider $W = (\ln(2)+\delta)D/\varepsilon$ for some constant $\delta > 0$ and let $n^{(t)}$ be the number of calls to* `write-to-disks` *during the t-th invocation of* `write` *after the new blocks have been appended. Then $\mathbb{E}n^{(t)} \leq 1 + e^{-\Omega(D)}$.*

2.1 Analysis

By reducing the arrival rate to $1-\varepsilon$ we can bound the queues by the stationary distribution of a queuing system with batched arrivals. This means that the **while**-loop is entered infrequently (Lemma 3) for a suitably chosen W. As the first step, we derive the expected queue length and a Chernoff-type tail bound for one queue.

Lemma 2 *Let $Q_i^{(t)}$ be the length of Q_i at the t-th invocation of* `write`*. Then $\mathbb{E}\,Q_i^{(t)} \leq 1/(2\varepsilon)$ and*

$$\mathbb{P}\left[Q_i^{(t)} > q\right] < 2e^{-\varepsilon q} \quad \textit{for all } q > 0.$$

Proof: Clearly, the queues can only become shorter if the `while`-loop is entered. Hence, it is sufficient for an upper bound on the queue length to consider the case where W is so large that this never happens.

Let $X_i^{(t)}$ denote the number of blocks that are appended to Q_i at the t-th invocation of `write`. Then, $X_i^{(1)}, X_i^{(2)}, \ldots$ are independent $\mathcal{B}((1-\varepsilon)D, 1/D)$ binomially distributed random variables. We describe the queue Q_i together with its input $X_i^{(1)}, X_i^{(2)}, \ldots$ as a *queuing system with batched arrivals*. In particular, one block can leave per time unit and a $\mathcal{B}((1-\varepsilon)D, 1/D)$-distributed number of blocks arrive per time unit. We first derive the probability generating function (pgf) of $Q_i^{(t)}$ for the stationary state by adapting the derivation from [29, Section 12-2] to the case of batched arrivals. Let $G_t(z)$ be the pgf of $Q_i^{(t)}$. Then, $G_0(z) = 1$ and for all $t \in \{0, 1, \ldots\}$

$$G_{t+1}(z) = \left(z^{-1}G_t(z) + (1-z^{-1})G_t(0)\right) \cdot H(z)$$

[2] During the execution of `write` more than W blocks may reside in the queues. The additional memory is borrowed from the block buffers handed over by the calling application program.

[3] If one insists on finding the result of the entire computation in the external memory, then the queues have to be flushed at the very end of the program. However, this effort can be amortized over the entire computation, and using Lemma 2 it is easy to show that $\max(Q_1^{(t)}, \ldots, Q_D^{(t)}) = \mathcal{O}\left(\frac{\log D}{\varepsilon}\right)$ with high probability.

where $H(z) = (z/D + 1 - 1/D)^{(1-\varepsilon)D}$ is the binomial pgf of $X_i^{(t)}$. Since the average rate of arrival is $1 - \varepsilon$ and the rate of departure is 1, a stationary state exists. In the stationary state $G_{t+1} = G_t$. By applying L'Hôpital's rule and normalizing via $G(1) = 1$ we find that $G(0) = \varepsilon$ and the stationary pgf is

$$G(z) = \frac{(1-z)\varepsilon}{1 - zH(z)^{-1}}.$$

We now show that the stationary distribution is an upper bound on the distribution of $Q_i^{(t)}$ for all t in the sense

$$\mathbb{P}\left[Q_i^{(t)} > q\right] \leq \mathbb{P}\left[Q_i^{(\infty)} > q\right] \quad \text{for all } q > 0,$$

where $Q_i^{(\infty)}$ is a G-distributed random variable describing the steady state. To see the bound, consider two queues processing identical input but with different initial length. Then in any step, the difference in length either remains the same or gets reduced by one. This continues until (possibly) the lengths become equal for the first time and from then on the queues coincide for all time because they process the same input.

Thus, $\mathbb{E}Q_i^{(t)} \leq \mathbb{E}Q_i^{(\infty)} = G'(1)$ and by applying L'Hôpital's rule twice we obtain

$$G'(1) = \frac{1}{2\varepsilon} - \frac{1 - \varepsilon + D\varepsilon^2}{2D\varepsilon} \leq \frac{1}{2\varepsilon}.$$

For the tail bound, note that $\ln(1+x) < x$ for $x > 0$ implies $\ln H(e^\varepsilon) < (1-\varepsilon)(e^\varepsilon - 1)$. Thus

$$G(e^\varepsilon) < \frac{\varepsilon(1-e^\varepsilon)}{1 - \exp(\varepsilon - (1-\varepsilon)(e^\varepsilon - 1))} < 2.$$

The tail bound follows from the general tail inequality $\mathbb{P}\left[Q_i^{(\infty)} > q\right] < G(e^\varepsilon)e^{-\varepsilon q}$ for all $q > 0$ (from [30, Exercise 8.12a]). ■

Based on Lemma 2 we give an upper bound on the probability that the `while`-loop is entered for a given limit $W = qD$ of internal memory.

Lemma 3 *Let $Q^{(t)} = Q_1^{(t)} + \cdots + Q_D^{(t)}$ with $Q_i^{(t)}$ as in Lemma 2. Then $\mathbb{E}Q^{(t)} \leq D/(2\varepsilon)$ and*

$$\mathbb{P}\left[Q^{(t)} > qD\right] < e^{-(\varepsilon q - \ln 2)D} \quad \textit{for all } q > 0.$$

Proof: The technical problem here is that $Q_1^{(t)}, \ldots, Q_D^{(t)}$ are not independent. However, the variables are negatively associated (NA) in the sense of [4, Definition 3][4] as we will now show.

[4]For every two disjoint subsets of $\{Q_1^{(t)}, \ldots, Q_D^{(t)}\}$, A and B, and all functions $f : \mathbb{R}^{|A|} \to \mathbb{R}$ and $g : \mathbb{R}^{|B|} \to \mathbb{R}$ which are both nondecreasing or both nonincreasing,

$$\mathbb{E}[f(A)g(B)] \leq \mathbb{E}[f(A)]\mathbb{E}[g(B)].$$

Define the indicator variable $B_{i,k}^{(t)} = 1$ if the k-th request of the t-th invocation of `write` is placed in Q_i and $B_{i,k}^{(t)} = 0$ otherwise. Then [4, Proposition 12] states that all $B_{i,k}^{(t)}$ are NA. Furthermore, $Q_i^{(t)}$ is a non-decreasing function of all $B_{i,k}^{(t')}$ for all k and all $t' \leq t$, since adding a request to Q_i can only increase the queue length in the future. In this situation, [4, Proposition 8 (2.)] implies that $Q_1^{(t)}, \ldots, Q_D^{(t)}$ are NA.

Now we can use Chernoff's method to derive the tail bound. Consider Markov's inequality

$$\mathbb{P}\left[Q^{(t)} > W\right] = \mathbb{P}\left[e^{\varepsilon Q^{(t)}} > e^{\varepsilon W}\right] < e^{-\varepsilon W}\,\mathbb{E}e^{\varepsilon Q^{(t)}}.$$

Using the negative association

$$\mathbb{E}e^{\varepsilon Q^{(t)}} = \mathbb{E}e^{\varepsilon \sum_{i=1}^{n} Q_i^{(t)}} = \mathbb{E}\prod_{i=1}^{n} e^{\varepsilon Q_i^{(t)}} \leq \prod_{i=1}^{n} \mathbb{E}e^{\varepsilon Q_i^{(t)}} = \left(\mathbb{E}e^{\varepsilon Q_1^{(t)}}\right)^D.$$

Since $\mathbb{E}e^{\varepsilon Q_1^{(t)}} = G(e^{\varepsilon}) < 2$ (proof of Lemma 2) the tail bound follows. The bound on the expected value follows directly from Lemma 2 and the linearity of the expected value. ■

We are now ready to prove Theorem 1, the main result of this section.

Proof: `Write-to-disks` is called at least once during the t-th invocation of `write`. Lemma 3, with $W/D = q = (\ln(2) + \delta)/\varepsilon$, gives the probability that the body of the **while**-loop is entered

$$p = \mathbb{P}\left[Q^{(t)} > W\right] \leq e^{-(\varepsilon W/D - \ln(2))D} = e^{-\delta D}.$$

Even in the worst case after $W + D$ iterations all queues must be empty. Thus, the expected number of calls to `write-to-disks` is

$$\mathbb{E}n^{(t)} \leq 1 + p \cdot (W + D) = 1 + \mathcal{O}\left(\frac{D}{\varepsilon}\right) e^{-\delta D}$$

which is bounded by $1 + e^{-\Omega(D)}$. ■

2.2 Writing Blocks Multiple Times

We now generalize the analysis from Section 2.1 to access patterns where blocks can be written multiple times. The basic idea is simple. We *remap* blocks, i.e., whenever a block is written, we choose a fresh random location for it. Now the system behaves in the same way as a system where all blocks are only written once. Unfortunately, a direct implementation of this remapping idea would require a RAM resident directory mapping logical block IDs to their physical position.

We now describe how one can stick to a static allocation of *most* blocks using a fixed hash function h and achieve the same effect as full remapping at the price of a constant factor more I/Os and a small directory h' that can take $k = \Theta(D \log D)$ entries. Note that

the space consumption for the directory is small compared to the space needed for write buffers as long as the block size is large compared to $\log D$. Accesses to h' can be performed in constant expected time if h' is implemented as a hash table. We maintain the invariant that a block b not in the write queues can be found at physical position $h'(b)$ if $h'(b) \neq \perp$ and at position $h(b)$ else ($\perp$ stands for undefined).

Writing now works in *epochs*. At the beginning of an epoch, all write queues are empty and h' is empty, i.e., all blocks are mapped by the static hash function h. Write requests to blocks that are written for the first time in an epoch are independent of each other. Blocks rewritten within an epoch are remapped. Hence, within an epoch the analysis from Section 2.1 transfers. Remapping is implemented as follows: When a block b is written for the first time this can be detected by the test $h'(b) = \perp$. Block b need not yet be remapped but we remember the access by setting $h'(b) = h(b)$. If a block b is rewritten within the epoch, this can be detected because $h'(b) \neq \perp$. In this case b is remapped: A new block buffer f on a random disk is allocated and we set $h'(b) = f$. If b was already remapped ($h'(b) \neq h(b)$), its old position is deallocated.

An epoch ends when $|h'| > k$. Then the queues are emptied. By Lemma 3 this takes $\mathcal{O}(\log D)$ steps with high probability. Subsequently, all remapped blocks b are retrieved from $h'(b)$ and rewritten to their original position $h(b)$. Remapping alternates between reading and writing. It performs parallel input steps until at least $(1-\varepsilon)D$ blocks are available in $2D$ read buffers. Writing is done using queued writing. Using standard balanced allocation results for reading, our previous analysis for queued writing, and the fact that the write addresses are independent of the reading process, it can be seen that remapping takes at most $\mathcal{O}(\log D)$ I/O steps including time to empty all the write queues.

After an epoch ends, the next epoch starts. The same analysis as before applies since we are in the same system state — h' and the write queues are empty.

2.3 Experiments

The above closed form results leave open the behavior for small D and W and the constant factor relating memory requirements W and the admission rate $(1-\varepsilon)D$. To partially fill this gap, we have performed a small series of experiments with $D \in \{16, 64, 256\}$ disks and variable buffer space W. Since there is now an optimal write scheduling algorithm [12] that should be used to achieve good average case performance, we concentrate on the application in a soft real time system where we want high probability guarantees for every write step. The experiments summarized in Figure 2 performed 10^6 write steps writing $(1-\varepsilon)D$ blocks in each step for different values of ε, D and the ratio W/D. The plotted points show the smallest ε for which none of the operations in an experiment need more than on write step. One can see that for large D, the optimal choice approaches $\varepsilon = \frac{D}{2W}$. This is interesting since Lemma 2 implies that for this ε the *expected* total queue size approaches W. In other words, the total queue size is apparently sharply concentrated around its expectation. This suggests that the constant $\ln 2$ in Theorem 1 is an artifact of our analysis and could be replaced by $1/2$.

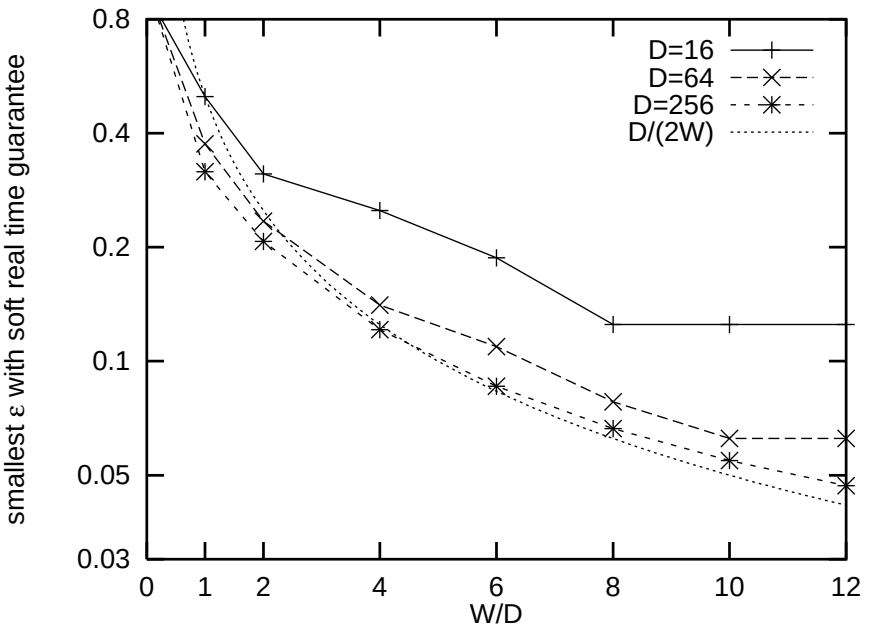

Figure 2: The smallest value for ε such that in 10^6 calls to `write`, passing $(1-\varepsilon)D$ blocks each, none needs more than one call of `write-to-disks`.

3 Random Duplicate Allocation

In this section, we investigate reading a *batch* of N logical blocks from D disks. There are copies of block i on disks u_i and v_i. The batch is described by the undirected *allocation multigraph* $G_a = (\{1..D\}, (\{u_1, v_1\}, \ldots, \{u_N, v_N\}))$. Observe that there can be multiple edges between two nodes. As in Section 2, logical blocks are mapped to the disks with a hash function assumed to be random. The logical block starting at external memory address kB is mapped to the disks $h(2k)$ and $h(2k+1)$ using the hash function h.[5] Therefore, G_a is a random multigraph with D nodes and N edges chosen independently and uniformly at random.

A *schedule* for the batch is a directed version G_s of G_a. (The directed edge (u_i, v_i) means that block i is read from disk u_i.) The *load* $L_u(G_s)$ of a node u is the outdegree of u in the schedule G_s. (We omit "(G_s)" when it is clear from the context which schedule is meant.) The maximum load $L_{\max}(G_s) := \max(L_1(G_s), \ldots, L_D(G_s))$ gives the number of read steps needed to execute the schedule. A schedule G_s for G_a is called *optimal* if there is no schedule G'_s with $L_{\max}(G'_s) < L_{\max}(G_s)$. The load of an optimal schedule is denoted by $L^*_{\max}$.

The main result of this section is the following theorem, which is proven in Section 3.2.

Theorem 4 *Consider a batch of N randomly and duplicately allocated blocks to be read from D disks. Then, abbreviating $b = \lceil N/D \rceil$, $L_{\max} \le b+1$ with probability at least $1 - \mathcal{O}(1/D)^{b+1}$.*

[5] We can additionally make sure that the two copies are always mapped to different disks. A refined analysis then yields a probability bound $\mathcal{O}(1/D)^{2b+1}$ in a strengthened version of Theorem 4. For the sake of simplicity, we do not go into this.

We want to point out that Lemma 6 below also provides more accurate bounds for small D and N that can be evaluated numerically. A corresponding lower bound which shows that $L_{\max} = \lceil N/D \rceil$ is unlikely for integer N/D is given in Section 3.4.

A difficulty in establishing Theorem 4 is that optimal schedules are complicated to analyze directly using probabilistic arguments because their structure is determined by a complicated scheduling algorithm. Therefore, we first give a characterization of optimal schedules in terms of the allocation graph G_a. Since this characterization is of completely combinatorial nature, and has nothing to do with the randomness of the allocation graph we have separated it out into Section 3.1.

In Section 3.3, we explain how an optimal schedule can be found in polynomial time using a small number of maximum flow computations. Section 4 will then show why optimal schedules can be found even faster than the worst-case bounds for maximum flow algorithms might suggest. Section 3.5 evaluates the scheduling quality experimentally.

3.1 Unavoidable Loads

Consider a subset Δ of disks and define the *unavoidable load* L_Δ as the number of blocks that have both copies allocated on a disk in Δ for a given batch of requests. Clearly, all these L_Δ blocks have to be read by some disk in Δ. The following theorem characterizes $L^*_{\max}$ in terms of the unavoidable load.

Theorem 5 *([31])* $L^*_{\max} = \max_{\emptyset \neq \Delta \subseteq \{1..D\}} \left\lceil \frac{L_\Delta}{|\Delta|} \right\rceil$.

The proof has been previously given by Schoenmaker [31, Theorem 1] who used the theorem for a different application. For self-containedness and as a warm-up for more complicated arguments yet to come, we nevertheless state a short proof here:

Proof: "$\geq$": For any Δ, a schedule fetches at least L_Δ blocks from the disks in Δ. Hence, there must be at least one disk $u \in \Delta$ with load $L_u \geq \lceil L_\Delta / |\Delta| \rceil$.

"$\leq$": It remains to be shown that there is always a subset Δ with $\lceil L_\Delta / |\Delta| \rceil \geq L^*_{\max}$ witnessing that $L^*_{\max}$ cannot be improved. Consider an optimal schedule G_s, which has no directed paths of the form $(v, \ldots, w)$ with $L_v = L^*_{\max}$ and $L_w \leq L^*_{\max} - 2$. Such a schedule always exists, since in schedules with such paths, the number of maximally loaded nodes can be decreased by moving one unit of load from v to w by reversing the direction of all edges on the path.

Choose a node v with load $L^*_{\max}$ and let Δ denote the set containing v and all nodes to which a directed path from v exists. Using this construction, all edges leaving a node in Δ also have their target in Δ so that the unavoidable load L_Δ is simply $\sum_{u \in \Delta} L_u$. By definition of G_s and v, we get $L_\Delta \geq 1 + |\Delta| (L^*_{\max} - 1)$, i.e., $L_\Delta / |\Delta| \geq 1/|\Delta| + L^*_{\max} - 1$. Taking the ceiling on both sides yields $\left\lceil \frac{L_\Delta}{|\Delta|} \right\rceil \geq \left\lceil \frac{1}{|\Delta|} + L^*_{\max} - 1 \right\rceil = L^*_{\max}$ as desired. ∎

3.2 Proof of Theorem 4

The proof developed here could be considered a special case of the results obtained in Section 4.2 for fast scheduling. To obtain a better compromise between accessibility and conciseness we choose a different approach however. The main line of argument for the proof is developed here in detail so that Section 4.2 only needs to note the necessary modifications. We also achieve better constant factors inside the analysis of the simple case. On the other hand, the less interesting technical lemmata are proven for the general case.

It should first be noted that, without loss of generality, we can assume that N is a multiple of D, i.e., $b = \lceil N/D \rceil = N/D$, since it only makes the scheduling problem more difficult if we add $D\lceil N/D \rceil - N$ dummy blocks to the batch.

The starting point of our proof is the following simple probabilistic upper bound on the maximum load of optimal schedules, which is based on Theorem 5.

Lemma 6 $\mathbb{P}[L^*_{\max} > b+1] \leq \sum_{d=1}^{D} \binom{D}{d} P_d$
where $P_d := \mathbb{P}[L_\Delta \geq d(b+1)+1]$ *for a subset* Δ *of size* d.

Proof: By the principle of inclusion-exclusion and Theorem 5 it suffices to count the number of subsets of size d, $\binom{D}{d}$, multiply this with P_d and add over all possible set sizes d. ■

Lemma 6 is useful because L_Δ only depends on the allocation graph G_a and is binomially $\mathcal{B}(bD, d^2/D^2)$ distributed for $|\Delta| = d$. The bound already yields an efficient way to estimate $\mathbb{P}[L^*_{\max} > b+1]$ numerically since the cumulative distribution function of the binomial distribution can be efficiently evaluated by using a continued fraction development of the incomplete Beta-function [32, Section 6.4]. Furthermore, most summands will be very small so that it suffices to use simple upper bounds on $\binom{D}{d}P_d$ for them. Overall, we view it as likely that $\mathbb{P}[L^*_{\max} > b+1]$ can be well approximated in time $\mathcal{O}(D)$ yielding high probability bounds faster bound than using simulation.

Furthermore, good closed form tail bounds are known for the binomial distribution. We use the strongest possible Chernoff bound in order to bound P_d, the probability to overload a given set of disks of size d. Throughout this section let $p := d/D$ and $q = 1-p$.

Lemma 7 *For any* $x > \mathbb{E}L_\Delta$,

$$\mathbb{P}[L_\Delta \geq x] \leq \left(\frac{Np^2}{x}\right)^x \left(\frac{1-p^2}{1-x/N}\right)^{N-x} .$$

Proof: Define the independent identically distributed 0-1 random variables X_i that take the value one if both copies of block i are allocated to Δ. We have $L_\Delta = \sum_{i=1}^{D} X_i$ and $\mathbb{P}[X_i = 1] = p^2$. For this type of sum, Chernoff's technique can be applied without any

approximations beyond using Markov's inequality [33, Lemma 2.2].[6]

$$\mathbb{P}\left[L_\Delta \geq (p^2+t)N\right] \leq \left(\left(\frac{p^2}{p^2+t}\right)^{p^2+t}\left(\frac{1-p^2}{1-p^2-t}\right)^{1-p^2-t}\right)^N \quad .$$

Solving $(p^2+t)N = x$ for t yields $t = x/N - p^2$. Substituting this value into the above equations yields the desired bound after straightforward simplifications. ∎

The technically most challenging part is to further bound the resulting expressions to obtain easy to interpret asymptotic estimates. We do this by splitting the summation over d into three partial sums for $d \leq D/8$ (Lemma 8 with $\alpha = 1/8$), $D/8 < d < Db/(b+1)$ (Lemma 9) and $\sum_{d \geq Db/(b+1)} \binom{D}{d} P_d$ which is simply zero.

Lemma 8 *For any constant $\alpha < e^{-2}$,*

$$\sum_{d \leq \alpha D} \binom{D}{d} P_d = \mathcal{O}(1/D)^{b+1}$$

Proof: Lemma 15 proves a bound for small Δ which we can apply in its simplest form (setting $\epsilon = 0$) to see that

$$\binom{D}{d} P_d \leq \left(\frac{d}{D}\right)^{db+1} e^{d(b+1)+1} \quad .$$

Viewing this bound as a function $f(d)$ of d, it can be verified that $f''(d) \geq 0$ (differentiate, remove obviously growing factors and differentiate again). Therefore, f assumes its maximum over any positive interval at one of the borders of that interval. We get $\sum_{d \leq \alpha D} \binom{D}{d} P_d \leq f(1) + \alpha D \max\{f(2), f(\alpha D)\}$.

$$\begin{aligned} f(1) &= D^{-b-1}e^{b+2} = e(e/D)^{b+1} = \mathcal{O}(1/D)^{b+1} \\ \alpha D f(2) &= \alpha D (2/D)^{2b+1} e^{2b+3} = \mathcal{O}(1/D)^{2b} \\ \alpha D f(\alpha D) &= \alpha D \alpha^{\alpha D b+1} e^{\alpha D(b+1)+1} = \mathcal{O}(D)\, e^{\alpha D(b(1+\ln\alpha)+1)} = e^{-\Omega(D)} \text{ if } \alpha < e^{-2}. \end{aligned}$$

All these values are in $\mathcal{O}(1/D)^{b+1}$. ∎

When $|\Delta|$ is at least a constant fraction of D, P_d actually decreases exponentially with D.

Lemma 9 $$\sum_{\frac{D}{8} < d < \frac{Db}{b+1}} \binom{D}{d} P_d = \mathcal{O}\left(\sqrt{D} \cdot 0.9^D\right) \quad .$$

[6] Several more well-known simpler forms do not suffice for our purposes. This bound is strongest possible in the sense that it only uses the Markov inequality once and no further estimates.

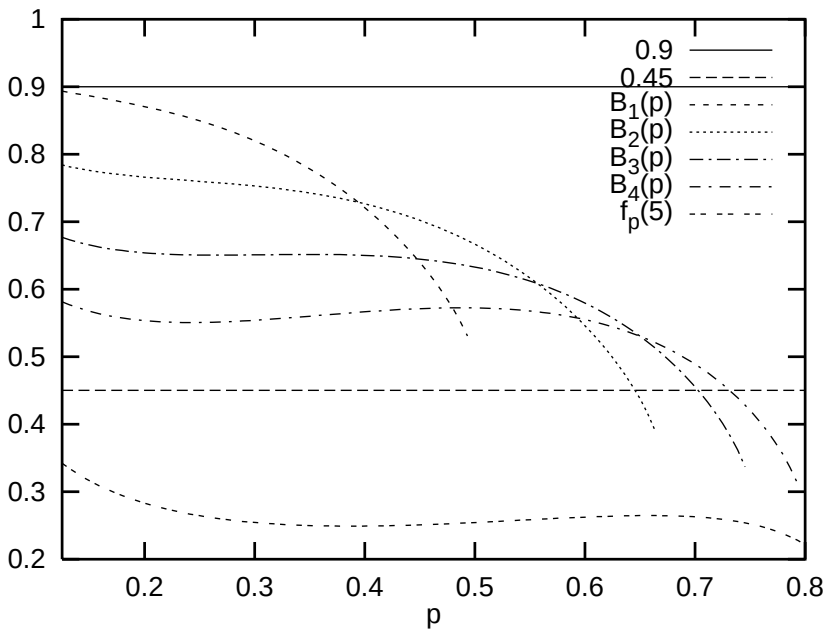

Figure 3: Behavior of $B_b(p)$ for small b.

Proof: Remembering that $p = d/D$ and $N = bD$ we get

$$d(b+1)+1 \leq d(b+1) = pD(b+1)$$

and using Lemma 7 we get

$$P_d \leq \left(\frac{bDp^2}{pD(b+1)}\right)^{pD(b+1)} \left(\frac{1-p^2}{1-\frac{pD(b+1)}{bD}}\right)^{bD-pD(b+1)} = \left(\left(\frac{bp}{b+1}\right)^{p(b+1)} \left(\frac{1-p^2}{1-p-p/b}\right)^{b-p(b+1)}\right)^D .$$

Note that D only appears as an exponent now. $\binom{D}{d} = \binom{D}{pD}$ can be brought into a similar form. Using the Stirling approximation (e.g. [34]) it can be seen that

$$\binom{D}{pD} = \mathcal{O}\left(\sqrt{\frac{D}{pD(D-pD)}} \left(\frac{D}{pD}\right)^{pD} \left(\frac{D}{D-pD}\right)^{D-pD}\right) = \mathcal{O}\left(\sqrt{\frac{1}{Dpq}}(p^{-p}q^{-q})^D\right) = \mathcal{O}\left(\sqrt{\frac{1}{D}}(p^{-p}q^{-q})^D\right)$$

for $1/8 < p < b/(b+1)$.

Since we are summing $\mathcal{O}(D)$ terms it remains to show that

$$B_b(p) := \frac{\left(\frac{bp}{b+1}\right)^{p(b+1)} \left(\frac{1-p^2}{1-p-p/b}\right)^{b-p(b+1)}}{p^p q^q} \leq 0.9$$

for all $1/8 < p < b/(b+1)$. For fixed b, this is easy. $B_b(p)$ is a smooth function and the open right border of the interval is no problem since $\lim_{p \to b/(b+1)} B_b(p) = (b/(b+1))^{2b^2/(b+1)} < 0.9$. For $b \in \{1,2,3,4\}$ we have verified the claim using interval arithmetics. We have written a

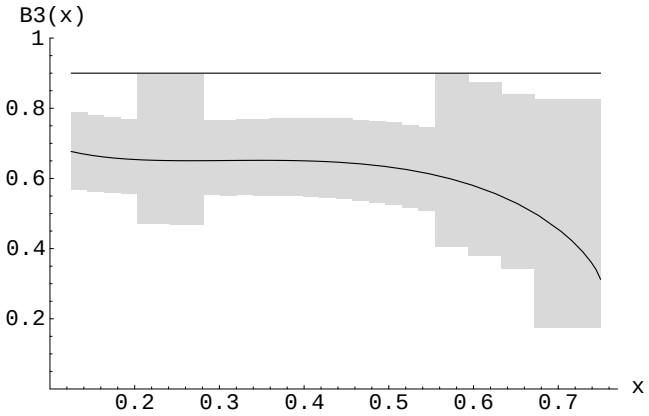

Figure 4: Behavior of $B_3(p)$ together with bounds obtained by our interval arithmetics verification.

small Mathematica program that adaptively subdivides the interval $[\frac{1}{8}, \frac{b}{b+1}]$. Note that this approach yields a rigorous proof since interval arithmetics produces conservative upper and lower bounds. The right border was handled by implementing interval arithmetics for the function x^x with $0^0 = 1$ and expressing B_b in terms of this function and the builtin functions. Figure 4 shows B_3 together with the bounds computed by our program. To save space, we only give plain plots for B_1, B_2, and B_4 in Figure 3.

For $b \geq 5$ we exploit that $p^{-p}q^{-q} \leq 2$ so that it also suffices to show that

$$f_p(b) := \left(\tfrac{pb}{b+1}\right)^{p(b+1)} \left(\tfrac{1-p^2}{1-p-p/b}\right)^{b-p(b+1)} \leq 0.45 \ .$$

In Figure 3 it can be seen that this relation holds for $b = 5$ and Lemma 17 (setting $\epsilon = 0$) implies that for a larger b the maximum of $f_p(b)$ can only decrease (again, we used interval arithmetics to verify the result rigorously). ■

3.3 Finding Optimal Schedules

We can efficiently find an optimal schedule by transforming the problem into a sequence of maximum flow computations: Suppose we have a schedule $G_s = (V, E)$ for a given batch G_a, and we try to find an improved schedule G'_s with $L_{\max}(G'_s) = L' < L_{\max}(G_s)$. Consider the flow network $\mathcal{N} = ((V \cup \{s,t\}, E^+), c, s, t)$ where $E^+ = E \cup \{(s,v) : L_v(G_s) > L'\} \cup \{(u,t) : L_u(G_s) < L'\}$. Edges (u,v) stemming from E have unit flow capacity $c(u,v) = 1$; $c(s,v) = L_v(G_s) - L'$ for $(s,v) \in E^+$; $c(u,t) = L' - L_u(G_s)$ for $(u,t) \in E^+$. s and t are artificial source and sink nodes, respectively. The edges leaving the source indicate how much load should flow away from an overloaded node. Edges into the sink indicate how much additional load can be accepted by underloaded nodes.

If an integral maximum flow through $\mathcal{N}$ saturates the edges leaving s, we can construct a new schedule G'_s with $L_{\max}(G'_s) = L'$ by flipping all edges in G_s that carry flow. Furthermore, if the edges leaving s are not saturated, $L_{\max}$ cannot be reduced to L':

Lemma 10 *If a maximum flow in $\mathcal{N}$ does not saturate all edges leaving s, then $L^*_{\max} > L'$.*

Proof: It suffices to identify a subset Δ with unavoidable load $L_\Delta > L'|\Delta|$. Consider a minimal s-t-cut (S,T). Define $\Delta := S - \{s\}$. Since not all edges leaving s are saturated, Δ is nonempty. Let $c_s := \sum_{(s,v)\in E'} c(s,v)$ denote the capacity of the edges leaving s and let $c_{ST} := \sum_{\{(u,v):u\in S, v\in T\}} c(u,v)$ denote the capacity of the cut. The unavoidable load of Δ is $L_\Delta = L'|\Delta| + c_s - c_{ST}$ (by definition of the flow network). By the max-flow min-cut Theorem, c_{ST} is identical to the maximum flow. By construction we get $c_s > c_{ST}$. Therefore, $L_\Delta > L'|\Delta|$ and by Theorem 5, $L^*_{\max} > L'$. ∎

An optimal schedule can now be found using binary search in at most $\log N$ steps and much less if a good heuristic initialization scheme is used [9]. Moreover, Theorem 4 shows that the optimal solution is almost always $\lceil N/D \rceil$ or $\lceil N/D \rceil + 1$ so that we only need to try these two values for L' most of the time.

3.4 A Lower Bound

We have seen that maximum load $L_{\max} = \lceil N/D \rceil + 1$ is almost always possible. A natural question is whether a perfect balance of $L_{\max} = \lceil N/D \rceil$ can also be achieved perhaps using a different allocation strategy. The following Theorem answers this question negatively for small integer N/D even for average case problems and even if we allow more redundancy.

Theorem 11 *Assume that w copies of each of U logical blocks have been placed on D disks. Define a positive integer $b \le \frac{\ln D}{3w}$ Then for sufficiently large U, an access to a subset of bD logical blocks chosen uniformly at random needs $L^*_{\max} \ge b+1$ read steps with probability $1 - \mathcal{O}(1/D)$ regardless of how the blocks have been placed.*

Proof: Let w_i/D denote the fraction of the logical blocks which are present on disk i with at least one copy and note that $\sum_{i=1}^{D} w_i \le wD$. Now consider a set of requested logical blocks chosen uniformly at random without replacement. We show that with high probability at least one disk i_0 holds no copy of any of the requested blocks so that the set $\Delta = \{1,\ldots,D\} \setminus \{i_0\}$ is overloaded.

Let X_i denote the number of blocks which could be served by disk i. We have

$$\mathbb{P}[X_i = 0] = \prod_{j<bD} \left(1 - \frac{w_i U/D}{U-j}\right) \ge \prod_{j<bD} \left(1 - \frac{w_i U}{D(U-bD)}\right) = \left(1 - \frac{w_i U}{D(U-bD)}\right)^{bD} \approx e^{-w_i b}$$

as $U \to \infty$ and for sufficiently large D. Let X denote the number of disks without usable blocks. We have

$$\mathbb{E}X := \sum_{i=1}^{D} \mathbb{P}[X_i = 0] \ge \sum_{i=1}^{D} e^{-w_i b} \ge De^{-wb}$$

The last step can be verified by minimizing the function $g(w_1,\ldots,w_D) = \sum_{i=1}^{D} e^{-w_i b}$ under the constraint $\sum_{i=1}^{D} w_i \le wD$ using calculus.

Now we use the method of bounded differences [35, Theorem 4.18] to show that X is sufficiently sharply concentrated around its mean that it is improbable that all X_i are nonzero. We view X as a function f of the bD random variables denoting the requested blocks. Fixing one of these variables changes $\mathbb{E}X$ by at most w. We get

$$\begin{aligned}\mathbb{P}[X<1] &= \mathbb{P}\left[X < \mathbb{E}X - (De^{-wb}-1)\right] \le \exp\left(-\frac{(De^{-wb}-1)^2}{2bDw^2}\right) \\ &\le \exp\left(-\frac{e^{-wb}(De^{-wb}-2)}{2bw^2}\right) \le \exp\left(-\frac{D^{\frac{1}{3}}-2D^{-\frac{1}{3}}}{\frac{2}{3}w\ln D}\right) = \mathcal{O}(1/D) \quad .\end{aligned}$$

The last "$\le$" uses $e^{-wb} \le e^{-w\ln(D)/(3w)} = D^{-1/3}$ and $De^{-wb} - 2 \ge 0$ for $D \ge 5$ and $b \le \ln(D)/(3w)$. The last "=" makes use of the fact that $1 \le b \le \ln(D)/(3w)$ and hence $w \le \ln(D)/3$. ■

3.5 Experiments

Similar to the case of queued writing, it is of practical interest to complement the asymptotic analysis for RDA with concrete numbers for small D. We can do that using a combination of simulation and numerical evaluation of the tail bound from Lemma 6. Simulation quickly yields approximations for the average performance and estimates for not-so-small failure probabilities. On the other hand, the tail bound makes it possible to estimate large deviations which would be very expensive to approximate using simulation. Figure 5 shows the overhead (one minus efficiency) of RDA for $D = 16$ and $D = 64$ based on expected performance and high probability performance.

It can be seen that average performance does not grow monotonically with N/D but achieves local optima shortly before N becomes divisible by D. So, if an application has some freedom regarding the number of blocks to be submitted for a parallel read request, it can be wise to submit fewer blocks than maximally possible. The curves for the average performance exhibit little dependence on D. To get high probability guarantees for good performance other choices for N/D can be useful. In particular, bad average performance means that there will usually be just a few disks with load $L_{\max}$. But this also means that it is quite improbable that there are any disks with even more load. Using Figure 6 this behavior can be studied in more detail. For $D = 64$ the failure rates are already so low that in most cases a hardware failure is much more probable than a request set which is difficult to schedule. For $D = 16$, we can achieve similarly low failure rates if N/D is large enough or if we are willing to accept a load of $\lceil N/D \rceil + 2$.

We have also made experiments regarding the question when perfect balance $L_{\max} = N/D$ for integer N/D is achievable. It looks like for $N \approx D\lceil 2.3 \log D \rceil$ perfect balance can be achieved in 90 % of all cases.

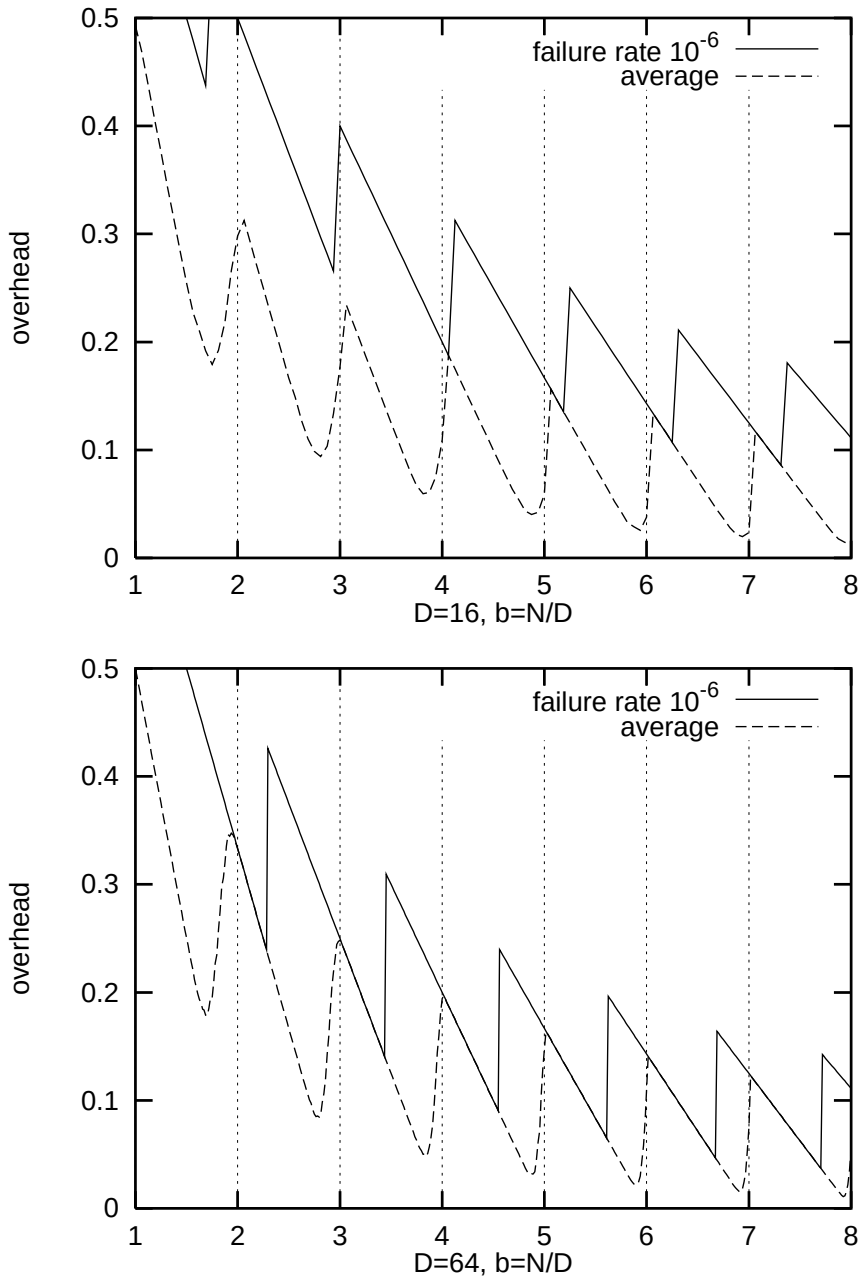

Figure 5: Overhead $1 - N/L^*_{\max}$ of RDA with N blocks to be retrieved.

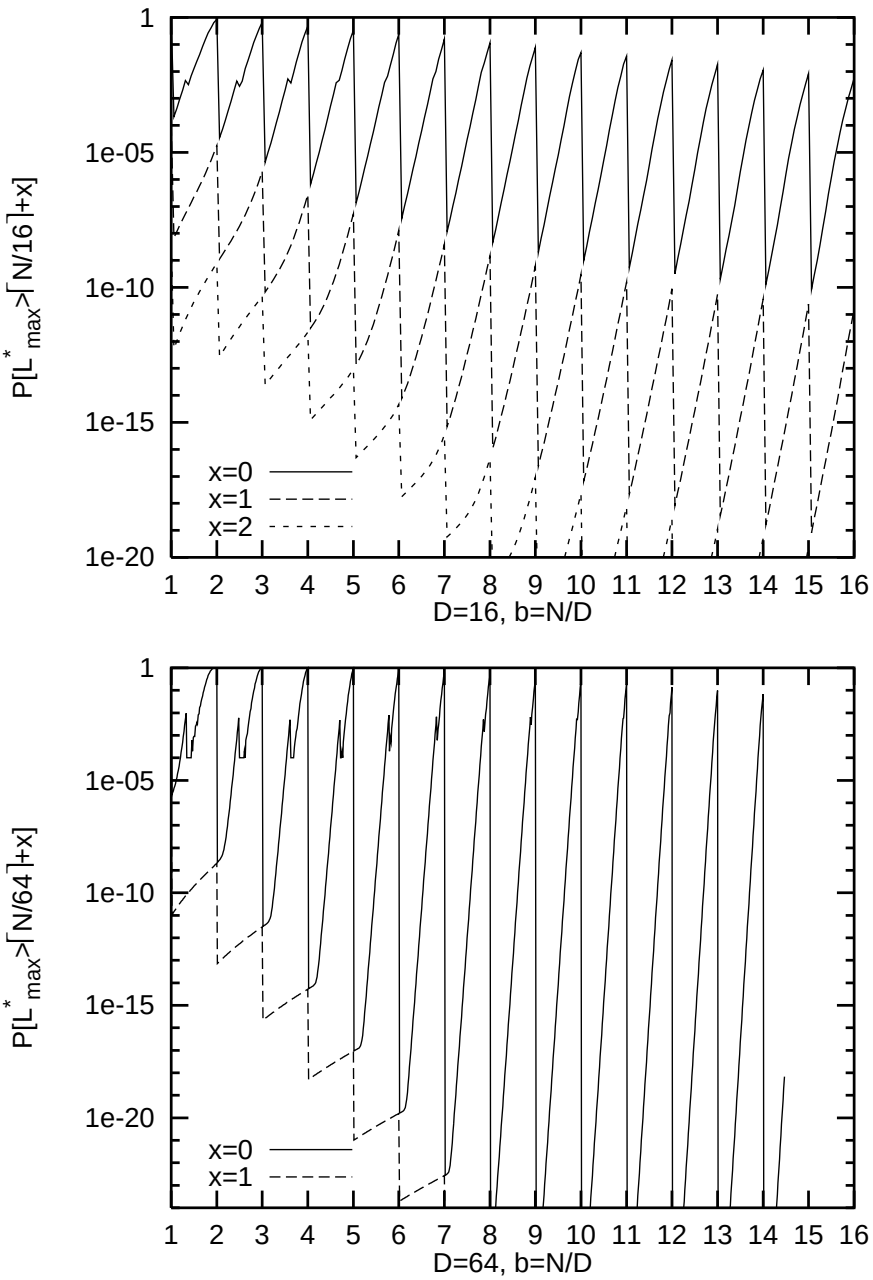

Figure 6: Failure probabilities of RDA with N blocks to be retrieved. Probabilities exceeding 0.01 are estimated using simulation. Smaller probabilities use the tail bound from Lemma 6.

4 Fast Scheduling

For very large D, the worst-case bounds for maximum flow computations ($\Omega\left(D^{3/2}\right)$, [36]) might become too expensive, since eventually, the scheduling time exceeds the access time.[7] Therefore, we will now explain, why slightly modified maximum flow algorithms can actually find a schedule with $L_{\max} = \lceil N/D \rceil + 1$ efficiently with high probability. In Section 4.4 even faster linear time approximations are discussed.

Theorem 12 *Consider a batch of $N = \Theta(D)$ blocks. Let $b = \lceil N/D \rceil$ and define a constant $0 < \epsilon \leq 1/5$. A schedule with $L_{\max} = b + 1$ can then be found in time $\mathcal{O}(D \log D)$ with probability $1 - \mathcal{O}(1/D)^{b+1-\epsilon}$.*

The proof is executed similarly to Section 3 and starts with graph theoretic arguments in Section 4.1, continues with a probabilistic analysis in Section 4.2 and only then considers algorithmic questions in Section 4.3.

The general idea is based on the observation that maximum flow algorithms essentially compute optimal schedules by removing all paths from overloaded to underloaded nodes. We call such paths *augmenting* paths following the tradition in flow computations. The key observation is that it is actually sufficient to perform flow augmentations that remove all augmenting paths of at most logarithmic length. Why is this sufficient? Consider a schedule without augmenting paths of length $\leq c \log D$. Assume $L_{\max} > b + 1$ and let v denote a disk with load $L_v \geq b + 2$. Section 4.1 establishes that in that case, a set of disks Δ with $L_\Delta > |\Delta|\,(b+1-\epsilon)$ must also exist. We then prove that such a subset is unlikely to exist for a random allocation graph G_a. This requires a slightly strengthened version of the probabilistic analysis done in Section 3.2. Finally, in Section 4.3 we explain how maximum flow algorithms can be adapted to find augmenting paths of logarithmic length very efficiently. In particular, even a simple preflow-push algorithm solves the task in $\mathcal{O}(D \log D)$ steps.

4.1 Unavoidable Loads

Our key argument is a counterpart to Theorem 5:

Lemma 13 *Consider a schedule graph $G_s = (\{1..D\}, E)$, any disk v with load L_v and a parameter $\gamma \in (0, 1)$. If there is no directed path $(v, \ldots, u)$ from v to a disk u with $L_u \leq L_v - 2$ and a path length $|(v, \ldots, u)| \leq \log_{1+\gamma} D + 1$, then there must be a subset Δ of disks with unavoidable load $L_\Delta > |\Delta|\,(1-\gamma)(L_v - 1)$.*

Proof: Consider the neighborhoods of v reached by i steps of breadth first search: $\Delta_0 := \{v\}$ and $\Delta_{i+1} := \Delta_i \cup \{u : \exists w \in \Delta_i \mid \exists (w, u) \in E\}$. Let $j := \min\{i : |\Delta_{i+1}| < (1+\gamma)\,|\Delta_i|\}$ denote the first neighborhood that grows by a factor less then $1+\gamma$. We have $D \geq |\Delta| \geq (1+\gamma)^j$ and hence $j \leq \log_{1+\gamma} D$. Let $\Delta' := \Delta_{j+1} - \Delta_j$ and let $\bar{\Delta}$ denote the set of disks in Δ' that have at least L_v incoming edges from Δ_j. We argue that $\Delta := \Delta_j \cup \bar{\Delta}$ has $L_\Delta > |\Delta|\,(1-\gamma)(L_v - 1)$.

[7] We have a prototype server with eight disks. At least for this machine the scheduling time is still negligible.

By assumption, the disks in Δ_j have total load exceeding $|\Delta_j|(L_v - 1)$. Load can only be moved out of Δ over at most $\left|\Delta' - \bar{\Delta}\right|(L_v - 1)$ edges leaving Δ, i.e., Δ has unavoidable load

$$\begin{aligned} L_\Delta &> |\Delta_j|(L_v - 1) - \left|\Delta' - \bar{\Delta}\right|(L_v - 1) \\ &= (|\Delta_j| + \left|\bar{\Delta}\right| - |\Delta'|)(L_v - 1) \\ &= (|\Delta| - |\Delta'|)(L_v - 1) \\ &\geq (|\Delta| - \gamma\,|\Delta_j|)(L_v - 1) \\ &\geq |\Delta|\,(1 - \gamma)(L_v - 1) \end{aligned}$$

■

We proceed as follows: Set $\gamma = \frac{\epsilon}{b+1}$. Set up a maximum flow problem for the algorithm from Section 3.3 with target maximum load $L' = b + 1$. Now run a modified maximum flow algorithm, which stops when no augmenting paths of length $\log_{1+\gamma} D+1 \approx 1+(b+1)\log(D)/\epsilon$ exist.

When the flow is computed, a schedule G_s is derived from it as described in Section 3.3. If the flow saturates the source node, we have a maximum flow and $L' = b + 1$ as desired. Otherwise, there must be a node with load at least $b + 2$ and Lemma 13 tells us that there must also be set of disks Δ with unavoidable load $L_\Delta > |\Delta|\,(b + 1 - \epsilon)$.

4.2 Proof of Theorem 12

Let us introduce the abbreviations $b_\epsilon := b+1-\epsilon$ and $P_d^\epsilon := \mathbb{P}[L_\Delta \geq db_\epsilon + 1]$ for a subset Δ of size d. Analogous to Lemma 6 and its proof, we have to prove that $\sum_{d=1}^{D} \binom{D}{d} P_d^\epsilon = \mathcal{O}(1/D)^{b_\epsilon}$.

As in Section 3.2, the sum $\sum_{d=1}^{D} \binom{D}{d} P_d^\epsilon$ is split into three parts. Now, small Δ are between 0 and $\lfloor D/16 \rfloor$. P_d^ϵ disappears for very large Δ with at least $\frac{b}{b_\epsilon}$ disks.

4.2.1 Small Δ

Lemma 14 $\sum_{d \leq D/16} \binom{D}{d} P_d = \mathcal{O}(1/D)^{b_\epsilon}$

The proof is very similar to the proof of Lemma 8 and can be found in Appendix A.1. It is based on the following bound which we prove here in detail since it is also needed for the proof of Lemma 8.

Lemma 15 *For any* $0 \leq \epsilon < 1$, *and* $b_\epsilon = (b + 1 - \epsilon)$

$$\binom{D}{d} \mathbb{P}[L_\Delta \geq db_\epsilon + 1] \leq \left(\frac{d}{D}\right)^{d(b-\epsilon)+1} e^{d(b+1)+1} \quad .$$

Proof: First, we estimate

$$\binom{D}{d} \leq \left(\frac{De}{d}\right)^d = \left(\frac{D}{d}\right)^d e^d$$

using the Stirling approximation. Now, setting $x = db_\epsilon + 1$, $p = d/D$, $N = bD$ in Lemma 7, we get $\mathbb{P}[L_\Delta \geq db_\epsilon + 1] \leq f \cdot g$ where

$$f = \left(\frac{bd^2/D}{db_\epsilon + 1}\right)^{db_\epsilon+1} \text{ and } g = \left(\frac{1 - d^2/D^2}{1 - \frac{db_\epsilon+1}{bD}}\right)^{bD-db_\epsilon-1} .$$

We have

$$f \leq \left(\frac{bd}{Db_\epsilon}\right)^{db_\epsilon+1} = \left(\frac{d}{D}\right)^{db_\epsilon+1} \left(\frac{b}{b_\epsilon}\right)^{db_\epsilon+1} \leq \left(\frac{d}{D}\right)^{db_\epsilon+1} \left(\frac{b}{b_\epsilon}\right)^{db_\epsilon} \leq \left(\frac{d}{D}\right)^{db_\epsilon+1} e^{-d(1-\epsilon)}$$

where the last estimate stems from the relation

$$\left(\frac{b}{b_\epsilon}\right)^{b_\epsilon} = \left(1 - \frac{1-\epsilon}{b_\epsilon}\right)^{b_\epsilon} \leq e^{-(1-\epsilon)} .$$

Since $1 - d^2/D^2 = (1 + d/D)(1 - d/D)$, we can write the second factor, g, as $g = g_1 \cdot g_2$ where

$$g_1 = \left(1 + \frac{d}{D}\right)^{bD-db_\epsilon-1} \leq \left(1 + \frac{d}{D}\right)^{bD} \leq e^{bd} \text{ and}$$
$$g_2 = \left(\frac{1 - \frac{d}{D}}{1 - \frac{db_\epsilon+1}{bD}}\right)^{bD-db_\epsilon-1} = \left(1 + \frac{db_\epsilon - db + 1}{bD - db_\epsilon - 1}\right)^{bD-db_\epsilon-1} \leq e^{db_\epsilon - db + 1} .$$

Multiplying the bounds for $\binom{D}{d}$, f, g_1, and g_2 yields

$$\binom{D}{d}\mathbb{P}[L_\Delta \geq d(b+1-\epsilon)+1] \leq \left(\frac{D}{d}\right)^d e^d \left(\frac{d}{D}\right)^{db_\epsilon+1} e^{-d(1-\epsilon)} e^{bd} e^{db_\epsilon - db + 1}$$
$$= \left(\frac{d}{D}\right)^{d(b+1-\epsilon)-d} e^{d-d(1-\epsilon)+bd+d(b+1-\epsilon)-db+1} = \left(\frac{d}{D}\right)^{d(b-\epsilon)+1} e^{d(b+1)+1} .$$

■

4.2.2 Larger Δ

Lemma 16 *For $\epsilon \leq 1/5$,* $\displaystyle\sum_{D/16<d<Db/b_\epsilon} \binom{D}{d} P_d^\epsilon = e^{-\Omega(D)}$.

Proof: Using an analogous argument as in the proof of Lemma 9 we can see that it suffices to evaluate

$$B_b(p) := \left(\frac{bp}{b_\epsilon}\right)^{pb_\epsilon} \left(\frac{1-p^2}{1-p-\frac{p(1-\epsilon)}{b}}\right)^{b-pb_\epsilon} p^{-p} q^{-q} < 1 .$$

on the interval $[\frac{1}{16}, \frac{b}{b_\epsilon})$. Since $\frac{\partial}{\partial \epsilon} B_b(p) \geq 0$ it suffices to consider the case $\epsilon = 1/5$. Figure 9 shows the plots for $b \leq 4$. For $b = 5$, we even have

$$f_p(b) := \left(\frac{bp}{b_\epsilon}\right)^{pb_\epsilon} \left(\frac{1-p^2}{1-p-\frac{p(1-\epsilon)}{b}}\right)^{b-pb_\epsilon} < 0.5,$$

and Lemma 17 shows that the maximum of $f_p(b)$ can only decrease for larger b. ■

Lemma 17 *Given constants $0 < \alpha \leq 1/2$ and $0 \leq \epsilon < 1$ and the abbreviation $b_\epsilon = (b+1-\epsilon)$, consider the function*

$$f_p(b) := \left(\frac{bp}{b_\epsilon}\right)^{pb_\epsilon} \left(\frac{1-p^2}{1-p-\frac{p(1-\epsilon)}{b}}\right)^{b-pb_\epsilon} .$$

Then, $\sup_{\alpha \leq p < b/b_\epsilon} f_p(b)$ is monotonically decreasing for integer $b \geq 5$.

Proof: Consider any $b > 5$ and any p where $f_p(b)$ is maximized. Such a value must exist in the interior of $[\alpha p, b/b_\epsilon)$ since $\lim_{p \to b/b_\epsilon} \frac{\partial}{\partial p} f_p(b) = -\infty$.
Case $p \leq (b-2)/b$**:** In Lemma 23 it is shown that $f_p(b)$ is non-increasing for $p \leq (b-1)/(b+1)$. In particular, it can only decrease on the interval $[b-1, b]$.
Case $p > (b-2)/b$**:** We make the substitution $p := \frac{b-\delta}{b_\epsilon}$, i.e., $\delta = b - pb_\epsilon$ and the condition $p > (b-2)/b$ becomes $\delta < 1 + \epsilon + \frac{2(1+\epsilon)}{b} \leq 4$. In Lemma 24 it is shown that

$$g_\delta(b) := f_p(b)\left[p \leftarrow \frac{b-\delta}{b_\epsilon}\right]$$

is non-increasing for its range of definition $b \geq \delta$.[8] In particular, for $b \geq 5$ and $\delta \leq 5$, $g_\delta(b)$ is defined and non-increasing on the interval $[b-1, b]$. We get

$$\begin{aligned} f_p(b) &= g_{b-p(b+1-\epsilon)}(b) \leq g_{b-p(b+1-\epsilon)}(b-1) \\ &= f_{\frac{(b-1)-(b-p(b+1-\epsilon))}{(b-1)+1-\epsilon}}(b-1) = f_{p-(1-p)/(b-\epsilon)}(b-1) \end{aligned}$$

since $p-(1-p)/(b-\epsilon) \geq 1/2$ for $b \geq 5$, $\epsilon \leq 1$, and $p > (b-2)/b \geq 3/5$. ■
The technical lemmata 23 and 24 are proven in Appendix A.2.

4.3 Maximum Flow with Short Augmenting Paths

What remains to be done to establish Theorem 12 is to explain how all augmenting paths of at most logarithmic length can be removed in time $\mathcal{O}(N \log D)$ time where $N = \mathcal{O}(D)$ is the number of edges of the allocation graph.

[8] The notation $e[a \leftarrow b]$ stands for the expression e where a is consistently subsituted by b.

To explain why flow computations can be easier if only augmenting paths of logarithmic length need to be considered we start with a simple example. Dinitz' algorithm [37] removes all augmenting paths of length i in the i-th iteration. Each iteration computes a *blocking* flow. Even a simple backtracking implementation of the blocking flow routine can do that in time $\mathcal{O}(iN)$ so that the time for the $\mathcal{O}(\log D)$ first iterations is $\mathcal{O}\big(N \log^2 D\big)$. Note that the same simplistic implementation needs $\mathcal{O}(D^3)$ steps for unconstrained maximum flows.

We can prove an even better bound for preflow-push algorithms by using the fact that we are essentially dealing with a unit capacity flow problem. This can be made precise by transforming the flow problem as formulated in Section 3.3 into a problem with only unit capacity edges: Replacing an edge (s, v) or (u, t) with integer capacity c by c parallel unit capacity edges. For target load $L' = \mathcal{O}(N/D)$, the number of additional edges will be in $\mathcal{O}(N)$.

Since detailed treatments of the preflow push algorithm are standard textbook material [38, 39], we only sketch the changes needed for our analysis: A preflow push algorithm maintains a *preflow*, which respects the capacity constraints of the flow network but relaxes the flow conservation constraints. Nodes with excess flow are called *active*. The difference between the original flow network and the preflow is the *residual network* that defines which edges are still able to carry flow. The algorithm also maintains a height $H(v)$ which is a lower bound for the distance of a node v to the sink node t, i.e. the minimum number of residual edges needed to connect v to t. Units of flow can be *pushed* downward from active nodes. Active nodes that lack downward residual edges can be *lifted*.

In the standard preflow push algorithm, $H(s)$ is initialized to D to make sure that flow can only return to the source if no path to the sink is left. If we are only interested in augmenting paths of length at most $H_{\max}$, we can initialize $H(s)$ to $H_{\max}$. The standard analysis of preflow-push is straightforward to adapt so that it takes the additional parameter $H_{\max}$ into account. It turns out that the number of lift operations is bounded by $2DH_{\max}$ and the number of saturating push operations is bounded by $NH_{\max}$. Furthermore, the algorithm can be implemented to spend only constant time per push operation and a total of $\mathcal{O}(NH_{\max})$ operations in other operations. The most difficult part in the analysis of general preflow-push algorithms, namely bounding the number of nonsaturating push operations, is simple here. Since there are only unit capacity edges, no nonsaturating pushes occur. Alltogether, preflow push can be implemented to run in time $\mathcal{O}(NH_{\max})$ for unit capacity flow networks. Since $N = \mathcal{O}(D)$ and $H_{\max} = \mathcal{O}(\log D)$ in our case, we get the desired $\mathcal{O}(D \log D)$ bound. ■

4.4 Linear Time Approximation.

Azar et al. [40] give a construction that achieves maximum load 10 for $N = D$. This is mainly of theoretical interest but they attribute a method that achieves maximum load 2 for $N \leq 1.6D$ to Frieze. A similar result is described in more detail by Czumaj and Stemann in the full paper [41, Section 7] using a result by Pittel, Spencer, and Wormald on "k-cores" [42]. For $N \leq 1.67D$ it is unlikely that there is any 3-core, i.e., a subset of nodes of G_a which induces a subgraph with minimum degree 3. Therefore, an algorithm which repeatedly removes nodes v with minimal degree by committing all its incident requests to

v will yield a schedule with maximum load 2 whp.

By splitting the input into $\lceil N/1.67 \rceil$ subbatches one gets a schedule with maximum load $2\lceil N/1.67 \rceil$ in linear time. A further improvement is possible by using subbatches of size up to $2.57D$. Using similar arguments as before it can be shown that those can be scheduled with maximum load 3, yielding a slightly better load balance. One should not apply the algorithm to larger subbatches however since it then deteriorates, approaching a maximum load of $2N/D$ for $N \gg D \log D$.

5 Reducing Redundancy

We model this more general storage scheme already outlined in the introduction in analogy to RDA: The allocation of $r+1$ sub-blocks of a logical block is coded into a hyperedge $e \in E$ of a hypergraph $H_a = (\{1..D\}, E)$ connecting the $r+1$ nodes (disks), to which sub-blocks have been allocated. Both e and E are multisets. A schedule is a directed version of this hypergraph H_s, where each hyperedge points to the disk which need *not* access the sub-block. RDA is the special case where all hyperedges connect exactly two nodes. Note that not all edges need to connect the same number of nodes. On a general purpose server, different files might use different trade-offs between storage overhead and logical block size. A logical block without redundancy can be modeled by an undirected hyperedge incident to only one node.

The unavoidable load of a subset of disks Δ is the difference between the number of times an element of Δ appears in an edge and the number of incident edges. Formally, $L_\Delta := \sum_{e \in E} |\Delta \cap \{e\}| - |\{e \in E : \Delta \cap E \neq \emptyset\}|$. With these definitions, Theorem 5 can be adapted to hypergraphs and the proof can be copied almost verbatim. Maximum flow algorithms for ordinary graphs can be applied by coding the hypergraph into a bipartite graph in the obvious way. Lemma 10 is also easy to generalize.

The most difficult part is again the probabilistic analysis. We would like to generalize Theorem 4 for arbitrary r. Indeed, we have no analysis yet which holds for all values of r and N/D. Yet, in Section 5.1, we outline an analysis which can be applied for any fixed r (we do that for $r \leq 10$) and yields the desired bound for sufficiently large N/D but still for all D. This already suffices to analyze a concrete application in a scalable way, and to establish a general emulation result between the multi-head model and independent disks. This result is summarized by the following lemma:

Lemma 18 *For given $b = N/D$ and r, let*

$$B_{br}(p) : [\frac{1}{14r}, \frac{rb}{rb+1}) \to \mathbb{R}, B_{br}(p) = \frac{(q^R + \frac{(pT+q)^R - q^R}{T})^b}{T^{p(br+1)} p^p q^q}$$

where $R := r+1$, $q = 1-p$, and $T = \frac{q}{p} \cdot \frac{1+Rp}{qr-p/b}$. If $B_{br} < 1$ in its range of definition, then $L_{\max} \leq \lceil br \rceil + 1$ with probability at least $1 - \mathcal{O}(1/D)^{br+1}$.

Using a simple trick, we can study the behavior of $B_{br}(p)$ for fixed r and arbitrarily large b. We simply substitute $y \leftarrow 1/b$ and plot the resulting twodimensional function $g_r(y,p)$.

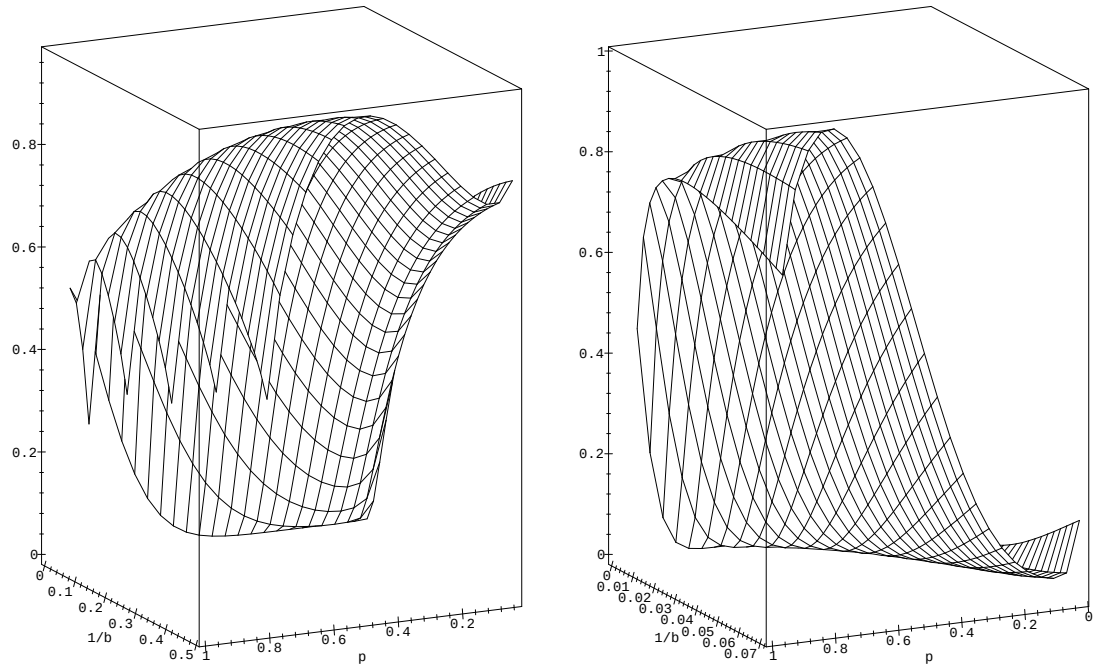

Figure 7: Behavior of $B_{b2}(p)$ for $b \geq 2$ and $B_{b4}(p)$ for $b \geq 14$.

Using this approach, Figure 7 shows the behavior of $B_{b2}(p)$ and $B_{b4}(p)$ for values of b which are large enough to ensure a value less than one. The following table gives the smallest b which ensures that $B_{br} < 1$ for $r \in \{2, \ldots, 10\}$.

r	2	3	4	5	6	7	8	9	10
b	2	6	14	24	38	56	77	101	130

Section 5.2 provides simulation result which indicate that even smaller N/D work well.

5.1 Proof of Lemma 18

Let Δ, $d = |\Delta|$, $p = d/D$ be defined as in Section 3 and introduce the abbreviations $q := 1-p$, $R := r + 1$ and $P_d := \mathbb{P}[L_\Delta \geq d(rb + 1) + 1]$ for a subset Δ of size d. The structure of the analysis is analogous to the proof of Theorem 4. Lemma 6 still applies. As before, if X_i denotes the unavoidable load incurred by logical block i, we have $L_\Delta = \sum_{i=1}^{N} X_i$. However, for $r \geq 2$, the X_i are not 0-1 random variables and L_Δ is *not* binomially distributed. Instead X_i has the shifted binomial distribution $\max\{0, \mathcal{B}(R, d/D) - 1\}$. Fortunately, the X_i are independent and we can use Chernoff's technique to develop a tail bound for L_Δ:

Lemma 19 *For any $x \geq \mathbb{E}[L_\Delta]$ and any $T \geq 1$, $\mathbb{P}[L_\Delta > x] \leq \dfrac{(q^R + \frac{(pT+q)^R - q^R}{T})^N}{T^x}$.*

Proof: We have $\mathbb{P}[L_\Delta > x] = \mathbb{P}\left[T^{L_\Delta} > T^x\right]$ and hence, using Markov's inequality, $\mathbb{P}[L_\Delta > x] \leq \mathbb{E}[T^{L_\Delta}]/T^x$. By definition of L_Δ, $\mathbb{E}[T^{L_\Delta}] = \mathbb{E}[T^{\sum_i X_i}] = \mathbb{E}[\prod_i T^{X_i}] = \mathbb{E}[T^{X_1}]^N$. Using the binomial theorem, it is easy to evaluate $\mathbb{E}[T^{X_i}] = q^R + ((pT+q)^R - q^R)/T$. ∎

For greater flexibility, we have left the parameter T unspecified. (There seems to be no closed form optimal choice for T and general r.) Still, by picking an appropriate T, we can use Lemma 19 in a similar way as we used Lemma 7 in the proof for $r = 1$.

We split the sum from Lemma 6 into the intervals $\{0..\frac{D}{14r}\}$, $\{\frac{D}{14r}..\frac{Drb}{rb+1}\}$ and $\{\frac{Drb}{rb+1}..D\}$ where the last interval contributes only zero summands.

Appendix A.3 proves the following generalization of Lemma 8 by setting $T = 1 + \frac{1}{rp}$ in the Chernoff bound from Lemma 19.

Lemma 20 *For $r \geq 2$,* $\displaystyle\sum_{d \leq D/(14r)} \binom{D}{d} P_d = \mathcal{O}(1/D)^{br+1}$.

Concerning larger Δ we argue similarly to Lemma 9 that for $r \geq 2$,

$$\sum_{D/(14r) < d < Db/(rb+1)} \binom{D}{d} P_d = e^{-\Omega(D)} \tag{1}$$

for sufficiently large b depending on r.

We start the computation by setting

$$T = \frac{q}{p} \cdot \frac{N+x}{rN-x} = \frac{q}{p} \cdot \frac{1+Rp}{qr - p/b}$$

where $N = bD$ and $x = pD(rb+1) < pD(rb+1) + 1$. Lemma 19 then yields

$$P_d < \mathbb{P}[L_\Delta > x] < \left(\frac{(q^R + \frac{(pT+q)^R - q^R}{T})^b}{T^{p(br+1)}} \right)^D .$$

Since T does not depend on D, Relation (1) can be established by showing that

$$B_{br}(p) := \frac{(q^R + \frac{(pT+q)^R - q^R}{T})^b}{T^{p(br+1)} p^p q^q}$$

is bounded by some constant $\hat{B} < 1$ for $1/(14r) \leq p < \frac{rb}{rb+1}$. The factor $1/(p^p q^q)$ stems from the Stirling approximation of the binomial coefficient that was already used in the proof of Lemma 9. ∎

5.2 Experiments

Figure 8 compares the efficiency of the r out of $r+1$ coding scheme for $r = 1$ (RDA), $r = 4$ and $r = 8$, always using $D = 64$. The abscissa uses the scale rN/D so that all the points with the same abscissa involve the the same number of subblocks per disk. For $r = 4$ and N divisible by D the performance is quite close to the performance of RDA. However, choosing a "clever" value for N shows less dramatic performance improvement than for RDA. For $r = 8$, we always need somewhat larger batches of inputs for good performance.

The measured performance of the r out of $r+1$ scheme is significantly better than can be proven using the upper bounds. For example, we have performed simulations with $D = 64$

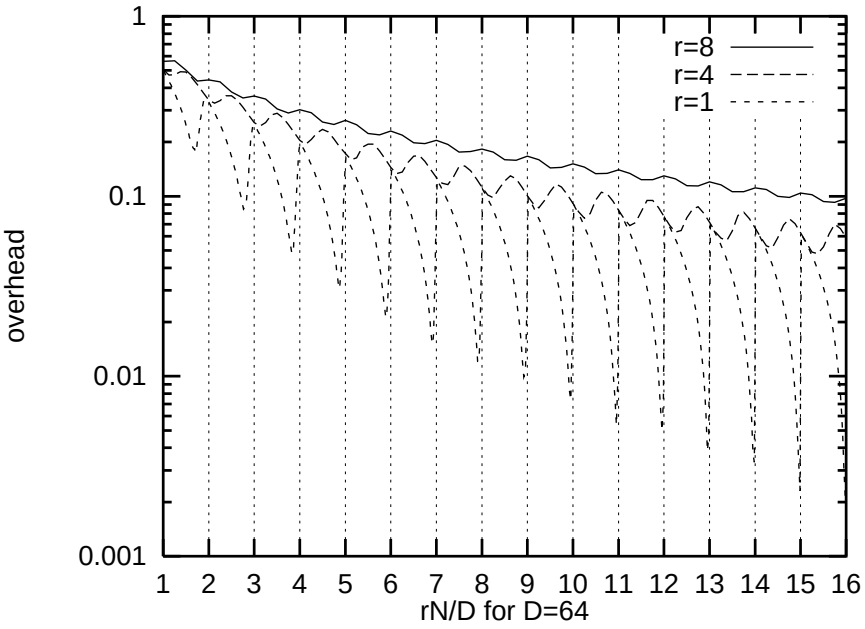

Figure 8: Average overhead (logarithmic scale) $1 - rN/L^*_{\max}$ of the r out for $r+1$ scheme with N blocks to be retrieved.

and different values for r and $b = N/D$ to find out when the average $L^*_{\max}$ goes down to $rb + 1$ in order to compare this to the analytical performance guarantees. For $r = 4$, $b = 4$ suffices whereas the theoretical bound requires $b = 14$. For $r = 8$, $b = 16$ suffices and the theoretical bound requires $b = 77$.

6 Applications and Refinements

Whereas sections 2 and 3 treat queued writing and reading with RDA as two independent techniques, we combine them into a general result on emulating multi-headed disks in Section 6.1. Further refinements that combine advantages of randomization and striping are outlined in Section 6.2. Then we give some examples of how our results can be used to improve the known bounds for external memory problems. Applications for multimedia are singled out in Section 6.4, since they served as a "breeding ground" for the algorithms described here. In Section 6.5 we explain how the coding scheme can be further generalized to allow reconstruction of a logical block from r out of $w \geq r$ subblocks using *Maximum Distance Separable codes* [43, 44]. This allows more flexible tradeoffs between low redundancy and high fault tolerance.

6.1 Emulating Multi-Headed Disks

Let us compare the independent disk model and the concurrent access multi-headed disk model under the simplifying assumption that I/O steps are either read steps or write steps.

Definition 21 *Let* MHDM-I-O$_{D,B,M}(i,o)$ *denote the set of problems*[9] *solvable on a D-head disk with block size B and internal memory of size M using i parallel read steps and o parallel write steps. Let* IPDM-I-O$_{D,B,M}(i,o)$ *denote the corresponding set of problems solvable with D independent single headed disks with expected complexity i and o assuming the availability of a random hash function.*

Using queued writing (Theorem 1) and RDA (Theorem 4), we can immediately conclude:

Corollary 22 *For any $0 < \epsilon < 1$ and $b \in \mathbb{N}$,*

$$\text{MHDM-I-O}_{bD,B,M}(i,o) \quad \subseteq \quad \text{IPDM-I-O}_{D,B,M+\mathcal{O}(D/\epsilon+bD)}(i',o')$$

where $i' = i \cdot (b+1) + \mathcal{O}(i/D)$ and $o' = \mathcal{O}(ob)$.

Aggarwal and Vitter's original multi-head model [1] allows read and write operation to be mixed in one I/O step. By buffering write operations this more general model could be emulated on the above MHDM-model with an additional slowdown factor of at most two. However, nobody prevents us from mixing reads and writes in the emulation. The write queues can even be used to saturate underloaded disks during reading. We have only avoided considering mixed reading and writing to keep the analysis simple.

The parity encoding from Section 5 can be used to reduce the overhead for write operations from two to $1 + 1/r$ at the price of increasing the logical (emulated) block size by a factor of r.

6.2 Refined Allocation Strategies

It may be argued that striping, i.e., allocating logical block i to disk $i \bmod D$ is more efficient than random placement for applications accessing only few, long data streams, since striping achieves perfect load balance in this case. We can get the best of both worlds by generalizing *randomized striping* [10, 45, 20], where long sequences of blocks are striped using a random disk for the first block.

We propose to allocate short strips of D consecutive blocks in a round robin fashion. A hash function h is only applied to the start of the strip: Block i is allocated to disk $(h(i \operatorname{div} D) + i \bmod D) + 1$. This placement policy has the property that two arbitrary physical blocks i' and j' are either placed on random independent disks or on different disks, and similar properties hold for any subset of blocks. In the case of redundant allocation, each copy is striped independently.

6.3 External Memory Algorithms

We first consider the classical problem of sorting N keys, since many problems can be solved externally using sorting as a subroutine [3]. Perhaps the best algorithm for both a single disk and a parallel multi-head disk is multi-way merge sort. This algorithm can be implemented using about $2\frac{N}{DB}\log_{M/B}\frac{N}{M}$ I/Os [45]. Ingenious deterministic algorithms have

[9] In a complexity theoretic sense.

been developed that adapt multi-way merging to independent disks [46]. Since the known deterministic algorithms increase the number of I/Os by a considerable factor, Barve et al. [10] have developed a more practical algorithm based on randomized striping, which also achieves $\mathcal{O}\left(\frac{N}{DB}\log_{M/B}\frac{N}{M}\right)$ I/Os if $M = \Omega\left(D\log D\right)$. Our general emulation result does not have this restriction and achieves $2(1+\frac{1}{r}+\epsilon)\frac{N}{DB}\log_{\Omega(M/B)}\frac{N}{M}$ for $\epsilon > DB/M$. (Note that during a merging operation, blocks are only written once so that bound for writing from Theorem 1 can be applied directly.)

Since the publication of the conference version [47] of this paper, the bounds for sorting have further improved. Hutchinson and Vitter [11] have proven a conjecture in the conference paper that the distribution sorting algorithm of Barve et al. can be improved using our analysis of writing. The key to this success was a generalization of randomized striping that combines the advantages of striping and fully random allocation. Subsequently, Hutchinson, Sanders, and Vitter [12] have improved the constant factors involved and explain how merge sort can be implemented optimally without redundancy. The key to this result is the observation that algorithms for write scheduling yield read schedules if they are applied to the reverse of the sequence of blocks to be read. The resulting algorithms for distribution sorting and merge sort meet the lower bound for sorting up to lower order terms.

Efficient external memory algorithms for more complicated problems than sorting, have so far mainly been developed for the single disk case. However, many of them are easily adapted to the multi-head model so that our emulation result yields randomized algorithms for parallel independent disks, which need a factor $\Theta(D)$ fewer I/O steps than using one disk. Many external memory algorithms have easily predictable read accesses and only writing is not predictable. For example, this is true for the batched geometric problems mentioned in [3] (orthogonal range queries, line segment intersection, 3D convex hulls, triangulation of point sets, point location, etc.) and also for data structures like buffer trees buffer trees [48]. In this case, redundancy can be avoided using the methods outlined above for distribution sorting.

Despite some overhead for redundancy, algorithms based on reading from multiple sources can still be the best choice. For example, although buffer trees yield an asymptotically optimal algorithm for priority queues, specialized algorithms based on multi-way merging can be a large constant factor faster [49]. A fifty percent overhead for duplicate writing is not an issue in this case.

6.4 Interactive Multimedia

In video-on-demand applications, almost all I/O steps concern reading. Hence, the disadvantage of RDA of having to write two copies of each block is of little significance to these applications. In addition, if many users have to be serviced simultaneously by a video-on-demand server, then disk bandwidth, rather than disk storage space tends to be the limiting resource. In that case, the duplicate storage of RDA need not imply that more disks are required for storage. Otherwise, the redundancy can be reduced as shown in Section 5. Also bear in mind, that similar kinds of redundancy (mirroring, parity blocks) are even needed in current systems to ensure fault tolerance.

Similar properties hold for interactive graphics applications [19]. In these applications

it is very important to be able to handle arbitrary access patterns while at the same time to realize small response times. In this respect, RDA clearly outperforms striping and also random allocation without redundancy.

6.5 More General Encodings

The parallel disk system (the redundant storage strategy together with the protocol to read and write) can be seen as a communication system in the sense of Shannon. The channel is represented by the read-protocol which deliberately introduces *erasures* in order to be able to balance the load on the disks. Another possible source of erasures is disk failure.

Consider the following mechanism: Each block is split into k equally sized parts to which another $n-k$ redundant parts are added as linear combinations of the first k parts. The linear combinations are described by an $[n,k,d]$ error correcting block code with minimum distance $d = n-k+1$. Such a code is called maximum distance separable (MDS).[10] MDS codes are optimal in the sense that the original block can be reconstructed from *any* set of at least k parts. The use of MDS-codes for fault tolerance has been investigated for example in [44].

All storage strategies mentioned in this article are special cases of binary MDS encoding: Striping uses the $[D,D,1]$ trivial code where D is the number of disks, RDA uses the $[2,1,2]$ repetition code and "r-out-of-$(r+1)$" uses the $[r+1,r,2]$ parity check code. In fact, it is known that the only existing binary MDS codes are the $[n,n,1]$ trivial, $[n,n-1,2]$ parity and $[n,1,n]$ repetition codes (from [50, Corollary 1]). Over larger alphabets, however, other MDS codes exist (e.g. Reed-Solomon codes). By the choice of an appropriate MDS code one can protect against disk failure (as in [44]), even against failure of multiple disks, and guarantee efficient load balancing at the same time.

Acknowledgements

The authors would like to thank David Maslen and Mike Keane for contributions to the analysis of RDA and Ludo Tolhuizen for advice on error correcting codes. Jeff Vitter and David Irwin helped to develop the rewriting algorithm in Section 2.2.

References

[1] A. Aggarwal and J. S. Vitter. The input/output complexity of sorting and related problems. *Communications of the ACM*, 31(9):1116–1127, 1988.

[2] J. S. Vitter and E. A. M. Shriver. Algorithms for parallel memory I: Two level memories. *Algorithmica*, 12(2–3):110–147, 1994.

[3] J. S. Vitter. External memory algorithms and data structures: dealing with massive data. *ACM Computing Surveys*, 33(2):209–271, June 2001.

[10] For a treatment of coding theory refer to the book of MacWilliams and Sloane, [43], in particular to Chapter 1 ("Linear codes") and Chapter 11 ("MDS codes"). The symbol $[n,k,d]$ denotes the parameters of a linear block code encoding k information symbols into n code symbols with a minimum distance of d.

[4] D. P. Dubhashi and D. Ranjan. Balls and bins: A study in negative dependence. *RSA: Random Structures & Algorithms*, 13:99–124, 1998.

[5] P. Sanders. Reconciling simplicity and realism in parallel disk models. In *12th ACM-SIAM Symposium on Discrete Algorithms*, pages 67–76, Washington DC, 2001.

[6] P. Sanders. Reconciling simplicity and realism in parallel disk models. *Parallel Computing*, 28(5):705–723, 2002. Special Issue on Parallel Data Intensive Algorithms and Applications.

[7] K. Salem and H. Garcia-Molina. Disk striping. *Proceedings of Data Engineering'86*, pages 336–342, 1986.

[8] D. Patterson, G. Gibson, and R. Katz. A case for redundant arrays of inexpensive disks (RAID). *Proceedings of ACM SIGMOD'88*, 1988.

[9] J. Korst. Random duplicate assignment: An alternative to striping in video servers. In *ACM Multimedia*, pages 219–226, Seattle, 1997.

[10] R. D. Barve, E. F. Grove, and J. S. Vitter. Simple randomized mergesort on parallel disks. *Parallel Computing*, 23(4):601–631, 1997.

[11] J. S. Vitter and D. A. Hutchinson. Distribution sort with randomized cycling. In *12th ACM-SIAM Symposium on Discrete Algorithms*, pages 77–86, 2001.

[12] D. A. Hutchinson, P. Sanders, and J. S. Vitter. Duality between prefetching and queued writing with parallel disks. In *9th European Symposium on Algorithms (ESA)*, number 2161 in LNCS, pages 62–73. Springer, 2001.

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

[14] P. Berenbrink, A. Czumaj, A. Steger, and B. Vöcking. Balanced allocations: The heavily loaded case. In *32th Annual ACM Symposium on Theory of Computing*, pages 745–754, 2000.

[15] B. Vöcking. How asymmetry helps load balancing. In *40th Symposium on Foundations of Computer Science*, pages 131–140. IEEE, 1999.

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

[17] M. Dietzfelbinger and F. Meyer auf der Heide. Simple, efficient shared memory simulations. In *5th ACM Symposium on Parallel Algorithms and Architectures*, pages 110–119, Velen, Germany, June 30–July 2, 1993. SIGACT and SIGARCH.

[18] F. Meyer auf der Heide, C. Scheideler, and V. Stemann. Exploiting storage redundancy to speed up randomized shared memory simulations. *Theoret. Comput. Sci.*, 162(2):245–281, August 1996.

[19] R. Muntz, J.R. Santos, and S. Berson. A parallel disk storage system for real-time multimedia applications. *International Journal of Intelligent Systems*, 13:1137–1174, 1998.

[20] W. Tetzlaff and R. Flynn. Block allocation in video servers for availability and throughput. *Proceedings Multimedia Computing and Networking*, 1996.

[21] R. Tewari, R. Mukherjee, D.M. Dias, and H.M. Vin. Design and performance tradeoffs in clustered video servers. *Proceedings of the International Conference on Multimedia Computing and Systems*, pages 144–150, 1996.

[22] S. Berson, R.R. Muntz, and W.R. Wong. Randomized data allocation for real-time disk I/O. *Proceedings of the 41st IEEE Computer Society Conference, COMPCON'96, Santa Clara, CA, February 25-28, pp. 286-290*, 1996.

[23] Y. Birk. Random RAIDs with selective exploitation of redundancy for high performance video servers. *NOSSDAV'97, St. Louis, MO, May, 1997, pp. 13-23*, 1997.

[24] P. Sanders. Asynchronous scheduling of redundant disk arrays. In *12th ACM Symposium on Parallel Algorithms and Architectures*, pages 89–98, 2000.

[25] Y.-J. Chiang, M. T. Goodrich, E. F. Grove, R. Tamasia, D. E. Vengroff, and J. S. Vitter. External memory graph algorithms. In *6th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 139–149, 1995.

[26] J. Sibeyn and M. Kaufmann. BSP-like external-memory computation. In *3rd Italian Conference on Algorithms and Complexity*, pages 229–240, 1997.

[27] F. Dehne, W. Dittrich, and D. Hutchinson. Efficient external memory algorithms by simulating coarse-grained parallel algorithms. In *ACM Symposium on Parallel Architectures and Algorithms*, pages 106–115, Newport, RI, 1997.

[28] F. Dehne, W. Dittrich, D. Hutchinson, and A. Maheshwari. Reducing i/o complexity by simulating coarse grained parallel algorithms. In *13th International Parallel Processing Symposium*, pages 14–20. IEEE, 1999.

[29] A. Papoulis. *Probability, random variables, and stochastic processes.* McGraw-Hill, 2nd ed., 1984.

[30] R. L. Graham, D. E. Knuth, and O. Patashnik. *Concrete Mathematics.* Addison-Wesley, 1989.

[31] L. A. M. Schoenmakers. A new algorithm for the recognition of series parallel graphs. Technical Report CS-R9504, CWI - Centrum voor Wiskunde en Informatica, January 31, 1995.

[32] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes in C (2nd Ed.)*. Cambridge University Press, 1992.

[33] C. McDiarmid. Concentration. In M. Habib, C. McDiarmid, and J. Ramirez-Alfonsin, editors, *Probabilistic Methods for Algorithmic Discrete Mathematics*, pages 195–247. Springer, 1998.

[34] T. Worsch. Lower and upper bounds for (sums of) binomial coefficients. Technical Report IB 31/94, Universität Karlsruhe, 1994.

[35] D. P. Dubhashi and A. Panconesi. Concentration of measure for computer scientists. Draft Manuscript, `http://www.brics.dk/~ale/papers.html`, February 1998.

[36] S. Even and E. Tarjan. Network flow and testing graph connectivity. *SIAM J. Comput.*, 4:507–518, 1975.

[37] E. A. Dinic. Algorithm for solution of a problem of maximum flow. *Soviet Math. Dokl.*, 11:1277–1280, 1970. now spelled 'Dinitz'.

[38] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. McGraw-Hill, 1990.

[39] R. K. Ahuja, R. L. Magnanti, and J. B. Orlin. *Network Flows*. Prentice Hall, 1993.

[40] Y. Azar, A. Z. Broder, A. R. Karlin, and Eli Upfal. Balanced allocations. *SIAM Journal on Computing*, 29(1):180–200, February 2000.

[41] A. Czumaj and V. Stemann. Randomized allocation processes. In *38th Symposium on Foundations of Computer Science (FOCS)*, pages 194–203. IEEE, 1997.

[42] B. Pittel, J. Spencer, and N. Wormald. Sudden emergence of a giant k-core in random graph. *J. Combinatorial Theory, Series B*, 67:111–151, 1996.

[43] F.J. MacWilliams and N.J.A. Sloane. *Theory of error-correcting codes*. North-Holland, 1988.

[44] G. A. Gibson, L. Hellerstein, R. M. Karp, R. H. Katz, and D. A. Patterson. Coding techniques for handling failures in large disk arrays, csd-88-477. Technical report, U. C. Berkley, 1988.

[45] D. E. Knuth. *The Art of Computer Programming — Sorting and Searching*, volume 3. Addison Wesley, 2nd edition, 1998.

[46] M. H. Nodine and J. S. Vitter. Greed sort: An optimal sorting algorithm for multiple disks. *Journal of the ACM*, 42(4):919–933, 1995.

[47] P. Sanders, S. Egner, and J. Korst. Fast concurrent access to parallel disks. In *11th ACM-SIAM Symposium on Discrete Algorithms*, pages 849–858, 2000.

[48] L. Arge. The buffer tree: A new technique for optimal I/O-algorithms. In *4th Workshop on Algorithms and Data Structures*, number 955 in LNCS, pages 334–345. Springer, 1995.

[49] P. Sanders. Fast priority queues for cached memory. In *ALENEX '99, Workshop on Algorithm Engineering and Experimentation*, number 1619 in LNCS, pages 312–327. Springer, 1999.

[50] L. Tolhuizen. On maximum distance separable codes over alphabets of arbitrary size. In *IEEE International Symposium on Information Theory*, page 431, 1994.

A Proof Details

A.1 Proof of Lemma 14

Proof: Lemma 15 is now applied in its full generality. Setting

$$f(d) := \left(\frac{d}{D}\right)^{d(b-\epsilon)+1} e^{d(b+1)+1},$$

we can see that $f''(d)$ is positive as before if $d \geq 3$ and $\epsilon \leq 1/2$, so that it suffices to consider values at the boundary of the interval $[3, D/16]$. We get $\sum_{d \leq \alpha D} \binom{D}{d} P_d^\epsilon \leq f(1) + f(2) + \alpha D \max\{f(3), f(\alpha D)\}$.

$$\begin{aligned}
f(1) &= (1/D)^{b_\epsilon} e^{b+2} = e^{1+\epsilon}(e/D)^{b_\epsilon} = \mathcal{O}(1/D)^{b_\epsilon}. \text{ Similarly,} \\
f(2) &= (2/D)^{2(b-\epsilon)+1} e^{2b+3} = \mathcal{O}(1/D)^{2(b-\epsilon)+1} \\
\alpha D f(3) &= \alpha D (3/D)^{3(b-\epsilon)} e^{3b+4} = \mathcal{O}(1/D)^{3(b-\epsilon)} \\
\alpha D f(\alpha D) &= \alpha D \alpha^{\alpha D(b-\epsilon)+1} e^{\alpha D(b+1)+1} = \mathcal{O}(D)\, e^{\alpha D((b-\epsilon)\ln\alpha + b + 1)} = e^{-\Omega(D)} \text{ if } \alpha < e^{-2/(1-\epsilon)}.
\end{aligned}$$

All these values are in $\mathcal{O}(1/D)^{b_\epsilon}$ for $\epsilon < 1/5$ and $\alpha < 1/16$. ∎

A.2 Auxiliary Lemmata for the Proof of Lemma 16

Lemma 23 *For $p < \frac{b-1}{b+1}$ and any $0 \leq \epsilon < 1$,*

$$f_b(p) = \left(\frac{bp}{b_\epsilon}\right)^{pb_\epsilon} \left(\frac{1-p^2}{1-p-\frac{p(1-\epsilon)}{b}}\right)^{b-pb_\epsilon}$$

is non-increasing.

Proof: Consider the derivative of $f_p(b)$,

$$f_p'(b) = f_p(b)\left(p\ln\left(\tfrac{bp}{b_\epsilon}\right) + (1-p)\ln\left(\tfrac{1-p^2}{1-p-\frac{p(1-\epsilon)}{b}}\right)\right).$$

Since $f_p(b)$ is positive, we have to verify that

$$l_b(p) := p \ln\left(\frac{bp}{b_\epsilon}\right) + (1-p)\ln\left(\frac{1-p^2}{1-p-\frac{p(1-\epsilon)}{b}}\right) \leq 0$$

for $p \leq \frac{b-1}{b+1}$. However, since $\frac{\partial}{\partial\epsilon} l_b(p) \leq 0$ for $p < b/b_\epsilon$, it suffices to consider the case $\epsilon = 0$ within the rest of this proof.

Lets first consider extreme values of p: We have $l_b(0) = 0$ and

$$l_b\left(\tfrac{b-1}{b+1}\right) = \tfrac{4}{b+1}\ln\left(\tfrac{2b}{b+1}\right) + \tfrac{b-1}{b+1}\ln\left(\tfrac{b(b-1)}{(b+1)^2}\right) \quad .$$

By inspection, it can be seen that this is indeed negative for $b \leq 34$. For larger b, we use $2b/(b+1) \leq 2$ and estimate

$$\ln\left(\frac{b(b-1)}{(b+1)^2}\right) = \ln\left(1 - \frac{3b+1}{(b+1)^2}\right) \leq -\frac{3b+1}{(b+1)^2}$$

using series development. We get

$$l_b\left(\frac{b-1}{b+1}\right) \leq \frac{4\ln(2)}{b+1} - \frac{(b-1)(3b+1)}{(b+1)^3} \quad .$$

This can be shown to be negative for $b \geq 34$ by solving a simple quadratic equation.

To complete the proof, we show that $l_b(p)$ cannot assume larger values for $0 < p < \frac{b-1}{b+1}$ because $l_b(p)$ is concave, i.e., $l_b''(p) > 0$. $l_b''(p)$ is a rational function and has the positive denominator $(p+1)^2(1-p)(b-bp-p)^2 p$ so that its sign only depends on the numerator, the polynomial $P_b(p) := (p^4 - 4p^3 + 6p^2 - 4p + 1)b^2 + (2p^3 - 6p^2 + 6p - 2)pb + p^4 - p^3 + 3p^2 + p$. Since the b-independent summand $p^4 - p^3 + 3p^2 + p$ is nonnegative for $p \in [0, 1]$, it suffices to show that

$$\begin{aligned} Q_b(p) &:= (P_b(p) - p^4 - p^3 + 3p^2 + p)/b \\ &= (p^4 - 4p^3 + 6p^2 - 4p + 1)b + (2p^3 - 6p^2 + 6p^1 - 2)p \\ &= (1-p)^3(b - p(b+2)) \end{aligned}$$

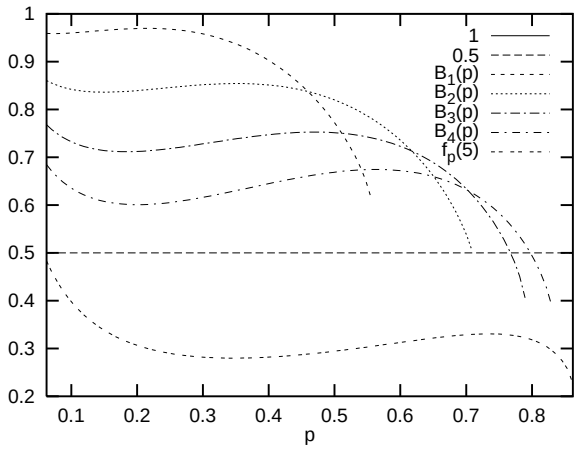

Figure 9: Behavior of $B_b(p)$ for $\epsilon = \frac{1}{5}$ and small b.

is nonnegative. This is the case for $p \leq b/(b+2)$, i.e., even beyond $(b-1)/(b+1)$. Rolling up our chain of arguments, we conclude that $P_b(p) \geq 0$ and $l_b''(p) \geq 0$ for $p \in [0, \frac{b-1}{b+1}]$, i.e., $l_b(p)$ is concave. Therefore, it was sufficient to prove that $l_b(0) \leq 0$ and $l_b(\frac{b-1}{b+1}) \leq 0$ to establish that $f_p(b)$ is non-increasing. ■

Lemma 24 *$g_\delta(b) := \left(\frac{b(b-\delta)}{b_\epsilon^2}\right)^{b-\delta} \left(\frac{b}{\delta}\left(1-\frac{(b-\delta)^2}{b_\epsilon^2}\right)\right)^{\delta}$ is non-increasing for $b \geq \delta$.*

Proof: Consider

$$g_\delta'(b) = \frac{g_\delta(b) u_b(\delta)}{b_\epsilon(b + b_\epsilon - \delta)}$$

where $u_b(\delta) := b(2\delta + 4(1-\epsilon)) + 2 - 4\epsilon + ((1-\epsilon)^2 + 3b(1-\epsilon) - db_\epsilon + 2b^2) \ln \frac{b(b-\delta)}{b_\epsilon^2}$ is the only term that can become negative for $b \geq \delta$. We have

$$u_b(0) = 2(b + b_\epsilon)(1 - \epsilon + b_\epsilon \ln \frac{b}{b_\epsilon}) \ .$$

Using series development, we get $\ln(b/b_\epsilon) \leq -\frac{1-\varepsilon}{b_\epsilon}$ and hence $u_b(0) \leq 0$. Furthermore, using series development again yields

$$\begin{aligned} u_b'(0) &= 2b_\epsilon \ln(1 + \frac{1-\epsilon}{b}) - 3(1-\epsilon) - \frac{(1-\epsilon)^2}{b} \\ &\leq 2b_\epsilon \frac{1-\epsilon}{b} - 3(1-\epsilon) - \frac{(1-\epsilon)^2}{b} \\ &= -\frac{(1-\epsilon)b_\epsilon}{b} \leq 0 \end{aligned}$$

Finally

$$u_b''(\delta) = -\frac{(1+\delta-\epsilon)b_\epsilon}{(b-\delta)^2} \leq 0,$$

i.e., $u_b(\delta)$ is convex. Together with $u_b'(0) \leq 0$ and $u_b(0) \leq 0$ this implies that $u_b(\delta) \leq 0$ for all $0 \leq \delta < b$ and the same holds for $g_\delta'(b)$. ■

A.3 Proof of Lemma 20

First, we further simplify the Chernoff bound from Lemma 19 for $N = bD$, $p = d/D$ and $x = d(br+1)+1$.

Lemma 25 *For $N = bD$, $|\Delta| = d$ and $p = \frac{d}{D}$,*

$$\mathbb{P}[L_\Delta \geq x] \leq e^{bd(r+1)(e-1)} \left(\frac{dr}{D}\right)^x \ .$$

Proof: Choosing $T = 1 + \frac{1}{rp}$ in Lemma 19 yields

$$\begin{aligned}
\mathbb{P}[L_\Delta > x] &\leq \frac{\left(q^R + \frac{(p(1+\frac{1}{rp})+q)^R - q^R}{1+\frac{1}{rp}}\right)^N}{(1+\frac{1}{rp})^x} = \frac{\left((R/r)^R + \frac{q^R}{rp}\right)^N}{(1+\frac{1}{rp})^{N+x}} \text{ since } 1+\tfrac{1}{rp} \geq \tfrac{1}{rp} \\
&\leq \left((R/r)^R + \frac{q^R}{rp}\right)^N (rp)^{N+x} = (Rp(R/r)^r + q^R)^N (rp)^x \\
&\leq e^{bdR((R/r)^r-1))}(rp)^x \leq e^{bd(r+1)(e-1)} \left(\frac{dr}{D}\right)^x
\end{aligned}$$

The latter two estimates are based on Lemma 26 and the fact that $(R/r)^r = (1+1/r)^r \leq e$. ■

We now set $x = d(rb+1)+1$ and use the Stirling approximation $\binom{D}{d} \leq (De/d)^d$ to get an overall bound

$$\binom{D}{d} P_d \leq (De/d)^d e^{bd(r+1)(e-1)} \left(\frac{dr}{D}\right)^{d(rb+1)+1} = (er)^d e^{bd(r+1)(e-1)} \left(\frac{dr}{D}\right)^{dbr+1}$$

Completing the proof of Lemma 20 is only slightly more complicated than it was in Lemma 8. Let $f(d) = (er)^d e^{bd(r+1)(e-1)} \left(\frac{dr}{D}\right)^{dbr+1}$. It is easy to check that $f'''(d) \geq 0$ and $f'(1) \leq 0$ for $D > re^{e+e/r+\ln(r)/r}$. Therefore, for sufficiently large D, f assumes its maximum over an interval $[d_{\min} \geq 1, d_{\max}]$ at one of the borders of that interval if $d_{\min} \geq 1$. For any constant $0 < \alpha < 1$, we get
$\sum_{d \leq \alpha D} \binom{D}{d} P_d \leq f(1) + \alpha D \max\{f(2), f(\alpha D)\}$.

$$\begin{aligned}
f(1) &= ere^{b(r+1)(e-1)} \left(\frac{r}{D}\right)^{br+1} = \mathcal{O}(1/D)^{br+1} \\
\alpha D f(2) &= \alpha D (er)^2 e^{2b(r+1)(e-1)} \left(\frac{2r}{D}\right)^{2br+1} = \mathcal{O}(1/D)^{2br} \\
\alpha D f(\alpha D) &= (er)^{\alpha D} e^{b\alpha D(r+1)(e-1)} (\alpha r)^{\alpha D br+1} \\
&= \mathcal{O}(D)\, e^{\alpha D(1+\ln(r)+b(r+1)(e-1)+\ln(\alpha r)br)} = e^{-\Omega(D)}
\end{aligned}$$

if $\alpha < \frac{1}{r} e^{-\frac{1+\ln r}{br} - (e-1)(1+\frac{1}{r})}$ or, if we prefer to choose α independently of b and proportional to $1/r$, $\alpha \leq 1/(14r) < \frac{1}{r} e^{-(e-1)\frac{3}{2}}$ for $r \geq 2$. ■

It remains to prove the following technical lemma:

Lemma 26 $(Rp(R/r)^r + q^R)^{bD} \leq e^{bdR((R/r)^r-1)}$.

Proof: (Outline)
Let $f(D) = (Rp(R/r)^r + q^R)^{bD} \leq e^{bdR((R/r)^r-1)}$. First observe, that $\lim_{D\to\infty} f(D) = e^{bdR((R/r)^r-1)}$. Therefore, it suffices to show that f grows monotonically. We have $f'(D) = f(D) b g(p)$ where $g(p) = \ln(pR(R/r)^r + q^R) + \frac{Rpq^R - pR(R/r)^r}{pR(R/r)^r + q^R}$, and it suffices to show that

$g(p) \geq 0$. Note that g only depends on r and $p = d/D$. In particular, for fixed r, it suffices to discuss a onedimensional function. Showing the $g(p) \geq 0$ for arbitrary r is tedious but possible. One way is to show that $g'(p) \geq 0$ in order to argue $g(p) \geq g(0) = 0$. The derivative $g'(p)$ is a rational function and its numerator can be further simplified by using $1 - rp \leq q^r \leq 1$ in the appropriate way. The denominator of the resulting function is a quadratic polynomial in p and can be minimized analytically. ■