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 provably efficiently using the following approach: A buffer of $\mathcal{O}(D)$ blocks suffices to support efficient writing of arbitrary blocks if blocks are distributed uniformly at random to the disks (e.g., by hashing). 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. 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 [42] 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 of 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 modelled 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. [41]), 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, in the case of reading, by using redundancy.

1.1 Summary of Results

In Section 2, we use queuing theory, Chernoff bounds and the concept of negative association [16] 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 to blocks that have not yet been written, can be served from the write queues. Furthermore, experiments indicate that if we simply admit D blocks every time step only every $2W/D$ steps we need to invest an additional wait step on the average.

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 *random* disks. Which of the two copies is to be read is optimally scheduled using maximum flow computations. We show that N blocks can be retrieved using $\lceil N/D \rceil + 1$ parallel read steps with high probability (whp). Certain values for N which are not multiples of D yield even higher efficiency. Furthermore, in Section 4 we explain why the optimal schedules 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 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—a missing sub-block is always the exclusive-or of the retrieved sub-blocks. We allow mixed workloads 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, namely that 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 [42]. 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* [34, 31]. 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 [24] for a detailed discussion and simulation results.

Reducing access contention by random placement is a well-known technique. For example, Barve et al. [6] 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. Apparently, it has not been proven before that in the case of writing, a small buffer solves this problem.

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. [4] show that an optimal symmetric online strategy commits each arriving request to the least loaded unit. Berenbrink et al. [7] recently analyzed this algorithm for the general case $N \neq D$ and showed that the maximum load is $L_{\max} = N/D + \log\ln D + \Theta(1)$. Vöcking [43] showed that for $N = D$, an asymmetric 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 bounds for offline algorithms were $L_{\max} = \mathcal{O}(N/D)$ 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 to find 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 [22]. PRAM simulation using a 3-collision protocol achieves maximum load $\mathcal{O}(1)$ for $N = D$ using $\mathcal{O}(\log\log D)$ iterations [27, 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 [38, 39, 24, 28]. Even the idea of a parity sub-block built out of r data sub-blocks has been used by several researchers [8, 9]. The first optimal scheduling algorithm for RDA was presented in [24]. This and other papers give convincing experimental evidence that RDA is a good policy yet no closed form results were known which 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.

2 Queued Writing

This section shows that a fraction of $1-\epsilon$ of the peak bandwidth for writing can be reached by making $W = \mathcal{O}(D/\epsilon)$ blocks of internal memory available to buffer write requests. This holds for any access pattern (Theorem 1), assuming that logical blocks are mapped to disks with a random hash function[1]. 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 $(1-\epsilon)D$ blocks.

Procedure `write`($(1-\epsilon)D$ 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.

After each invocation of `write`, the queues consume at most W internal memory.[2] The procedure `write-to-disks` stores all first blocks of the non-empty queues 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 the following statement 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. The theoretical treatement is complemented by experimental findings in Section 2.2 which suggest even better performance.

Theorem 1 *Consider $W = (\ln(2)+\delta)D/\epsilon$ for some $\delta > 0$ 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 $\mathbb{E}n^{(t)} \le 1 + e^{-\Omega(D)}$.*

[1]The hash function h maps block number i, starting at external memory address iB, to disk $h(i)$. The assumption that the hash function behaves like a true random function is quite similar to the usual assumption of randomized algorithms that the pseudo-random number generators used in practice produces true random numbers and the same assumption seems to be quite common in other works relying on hash function like PRAM emulation. However, in our case we can do even better. We could simply use a RAM resident directory with random entries for each block. This is possible since we need only a few bytes of RAM for a disk block with hundreds of kilobytes. The additional hardware cost for this RAM is negligible in many practical situations.

[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 $\max(Q_1^{(t)}, \ldots, Q_D^{(t)}) = \mathcal{O}\left(\frac{\log D}{\epsilon}\right)$ with high probability.

2.1 Analysis

The idea behind the analysis: By reducing the arrival rate to $1-\epsilon$ 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 $\mathbb{E}Q_i^{(t)} \leq 1/(2\epsilon)$ and*

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

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 $\mathcal{B}((1-\epsilon)D, 1/D)$ binomially distributed random variables. We describe the queue Q_i together with its input $X_i^{(1)}, X_i^{(2)}, \ldots$ as a *queueing system with batched arrivals.* In particular, one block can leave per time unit and a $\mathcal{B}((1-\epsilon)D, 1/D)$-distributed number of blocks arrives per time unit. We first derive the probability generating function (pgf) of Q_i for the stationary state by adapting the derivation from [30, 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 $t \in \{0, 1, \ldots\}$

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

where $H(z) = (z/D + 1 - 1/D)^{(1-\epsilon)D}$ is the binomial pgf of $X_i^{(t)}$. Since the average rate of arrival is $1-\epsilon$ and the rate of departure is 1, a stationary state exists. In the stationary state $G_{t+1} = G_t$ and by normalizing $G(1) = 1$ we find the stationary pgf

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

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

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

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

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

The tail bound follows from the general tail inequality $\mathbb{P}\left[Q_i^{(\infty)} > q\right] < G(e^\epsilon)e^{-\epsilon q}$ for all $q > 0$ (from [20, 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 $\mathbb{E}Q^{(t)} \le D/(2\epsilon)$ and*

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

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 [16, Definition 3][4] as we will now show.

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 [16, 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 $t' \le t$, since adding a request to Q_i can only increase the queue length in the future. In this situation, [16, 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

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

Using the negative association

$$\mathbb{E}e^{\epsilon Q^{(t)}} = \mathbb{E}e^{\epsilon \sum_i Q_i^{(t)}} \le \prod_i \mathbb{E}e^{\epsilon Q_i^{(t)}} = \left(\mathbb{E}e^{\epsilon Q_1^{(t)}}\right)^D.$$

Since $\mathbb{E}e^{\epsilon Q_1^{(t)}} = G(e^\epsilon) < 2$ (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.

[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,

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

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

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

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

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

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

2.2 Experiments

The above closed form results leave open the behavior for small D and W. We now fill this gap using simulation. Furthermore, there are a number of practical refinements which we want to investigate.

Average Case Performance

For non-real time algorithms we are interested in achieving a large average throughput and do not care if we sometimes have to wait to get rid of blocks to be written. In this case, we can set $\epsilon = 0$ and admit all blocks to the queue whenever possible. Figure 2 shows the overhead for a limited buffer size W for different values of D and W/D. The overhead is one minus the *efficiency* N/t, i.e., the ratio between the number of blocks DN actually written and the theoretical peak performance of Dt blocks in t steps. In every iteration, D new blocks are passed to the routine `write` above. In order to measure the steady state behavior of the system, we first run it without counting until more than one call to `write-to-disks` is needed for the first time. Then `write` is called $N = 10^6$ times with D more blocks and the number t of calls to `write-to-disks` is counted.

These results can be interpreted beyond the qualitative observation that high efficiency is reached even for quite small buffers like $W = 2D$. First, it should be noted that for small D, performance is even better than for large D. Except for small W, the number of disks hardly matters. This justifies our asymptotic approach for analyzing the average performance. Furthermore, a very simple rule for the relation between buffer size and efficiency suggests itself. Namely, the efficiency seems to approach $1 - \frac{D}{2W}$ (Note the logarithmic scale in Figure 2). This formula was not found by curve fitting but using an analogy to our theoretical analysis. Namely, if we allow $(1 - \frac{D}{2W})D$ blocks per call of `write` using unlimited buffers, Lemma 2 tells us that we get an expected total queue length of W. So, our eager scheme that allows D blocks every time and a *worst case* memory use of W has similar but slighly better performance than a scheme which admits less blocks, has unlimited worst case buffer space, and the same *average* memory use. To test this hypothesis, we performed further simulations. Since we were already quite convinced that D has little influence except for small D, we fixed $D = 256$ and only increased the range of measured buffer sizes. Between $W/D =$

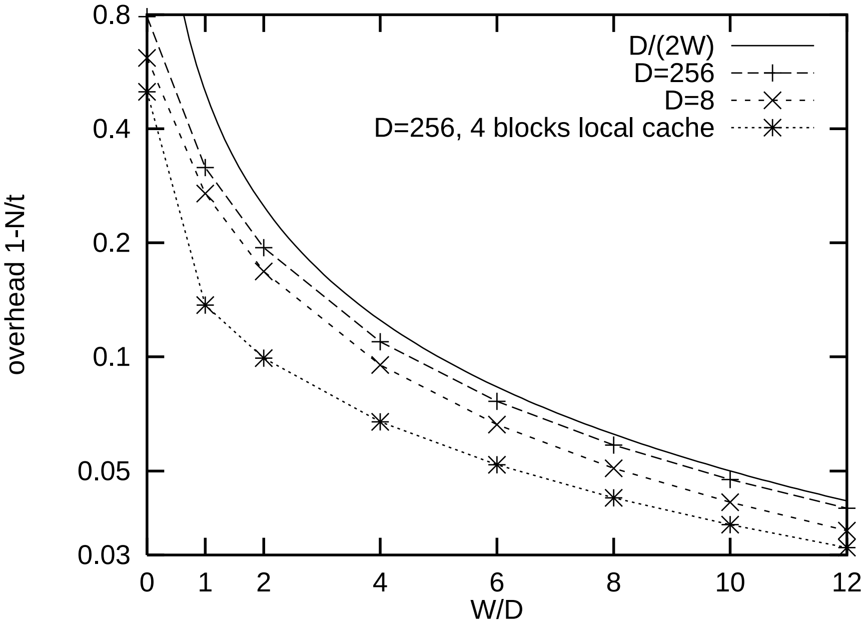

Figure 2: Overhead (i.e., 1−efficiency) of queued writing to D disks using a global queue of size W and no admission control ($\epsilon = 0$).

12 and $W/D = 32$ the difference between the measured efficiency and the approximation $1 - \frac{D}{2W}$ decreases from 0.001 to 0.0003. The simulations and the heuristic connection to our theoretical results lead us to formulate the following conjecture:

Conjecture 4 *Queued writing with $\epsilon = 0$ and buffer size W achieves average efficiency at least* $1 - \frac{D}{2W}$.

Even higher efficiency can be reached if we additionally exploit that the disks usually have local caches. The topmost curve in Figure 2 shows the effect of an additional cache for up to 4 blocks. It should be noted however that local cache alone is less effective than global cache. With global cache $W = 4D$ and $D = 256$ the efficiency is already 89 % whereas the same amount of memory completely invested in local caches yields only an efficiency of 50 %.

High Probability Performance Guarantees

For real time operation, we are less interested in the average throughput than in the probability to miss deadlines. For example, assume we want to save video data from a camera. Then on buffer overflow we may loose some picture frames so that we are interested in the probability that this happens. In this case, our approach to limit the number of blocks admitted per time step to $(1 - \epsilon)D$ is also useful in practice.

How should ϵ be choosen? Below $\epsilon = \frac{D}{2W}$ Lemma 2 predicts that the expected total queue size approaches W and therefore buffer overflows will be frequent. On the other hand, there are good reasons to believe that Theorem 1 can be strengthened to prove that the total

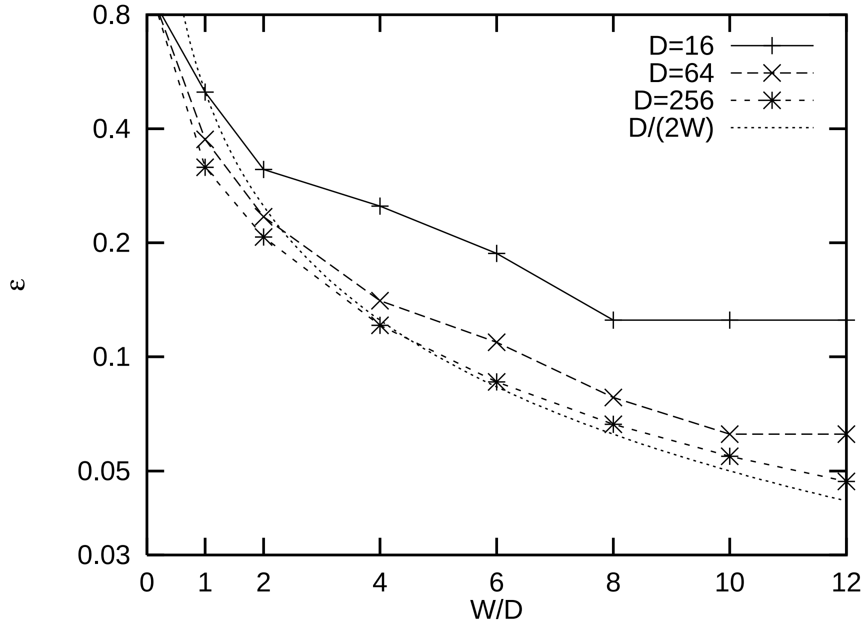

Figure 3: The largest value for ϵ such that in 10^6 calls to `write`, passing $(1-\epsilon)D$ blocks each, none needs more than one call of `write-to-disks`.

queue size is sharply concentrated around $\frac{D}{2\epsilon}$ so that an ϵ close to $\frac{D}{2W}$ should do. Whether this hypothesis is warranted and what "slightly larger" actually means is the subject of the following experiment. Figure 3 shows the minimum ϵ needed so that 10^6 subsequent calls to `write` need only call of `write-to-disks` each. As before, counting is started when the buffer is almost full. Only for small D do we need to choose ϵ significantly smaller than $D/(2W)$ to achieve a single-step `write` with high probability.

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* $G_a = (\{1..D\}, (\{u_1, v_1\}, \ldots, \{u_N, v_N\}))$ —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

[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 5. For the sake of simplicity, we do not go into this.

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 $L^*_{\max}$.

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

Theorem 5 *Consider a batch of N randomly and duplicately allocated blocks to be read from D disks. Then, abbreviating $b = \lceil N/D \rceil$, $L_{\max} \leq b+1$ with probability at least $1 - \mathcal{O}(1/D)^{b+1}$.*

Note that Lemma 7 below also provides more accurate bounds for small D and N that can be evaluated numerically. A corresponding lower bound which shows that $L_{\max} = N/D$ is impossible is given in Section 3.4.

A difficulty in establishing Theorem 5 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 to 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). The following Theorem characterizes $L^*_{\max}$ in terms of the unavoidable load.

Theorem 6 *([36])* $L^*_{\max} = \max_{\emptyset \neq \Delta \subseteq \{1..D\}} \left\lceil \frac{L_\Delta}{|\Delta|} \right\rceil$.

The proof has been previously given by Schoenmaker [36, 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 show 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 5

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 6.

Lemma 7 $\mathbb{P}[L^*_{\max} > b + 1] \leq \sum_{d=1}^{D} \binom{D}{d} P_d$
where $P_d := \mathbb{P}[L_\Delta \geq d(b+1) + 1]$ *for a subset* Δ *of size* d.[6]

Proof: By the principle of inclusion-exclusion and Theorem 6 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 7 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 $|\Delta| = d$.

We use an optimally accurate Chernoff bound for the tail of the binomial distribution in order to bound P_d, the probability to overload a given set of disks of size d. Throughout this section let $p := d/D$.

Lemma 8 *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 $L_\Delta = \sum_i X_i$ 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 [26, Lemma 2.2].[7]

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

[6] Note that this 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 [33, Section 6.4]. Furthermore, most summands will be very small so that is 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 a more accurate and faster bound than our simulation results from [24].

[7] Several more well-known simpler forms do not suffice for our purposes.

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 straigtforward 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 $d \le D/8$ (Lemma 9 with $\alpha = 1/8$), $D/8 < d < Db/(b+1)$ (Lemma 10) and $\sum_{d \ge Db/(b+1)} \binom{D}{d} P_d$ which is simply zero.

Lemma 9 *For any constant $\alpha < e^{-2}$,*

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

Proof: Lemma 16 proves a bound for small Δ which we can apply in its simplest form (setting $\epsilon = 0$) to see that

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

Viewing this bound as a function $f(d)$ of d, it can be verified that $f''(d) \ge 0$ (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 $\sum_{d \le \alpha D} \binom{D}{d} P_d \le f(1) + \alpha D \max\{f(2), f(\alpha D)\}$.

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

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 10 $$\sum_{\frac{D}{8} < d < \frac{Db}{b+1}} \binom{D}{d} P_d = \mathcal{O}\left(\sqrt{D} \cdot 0.9^D\right) \quad .$$

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

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

and using Lemma 8 we get

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

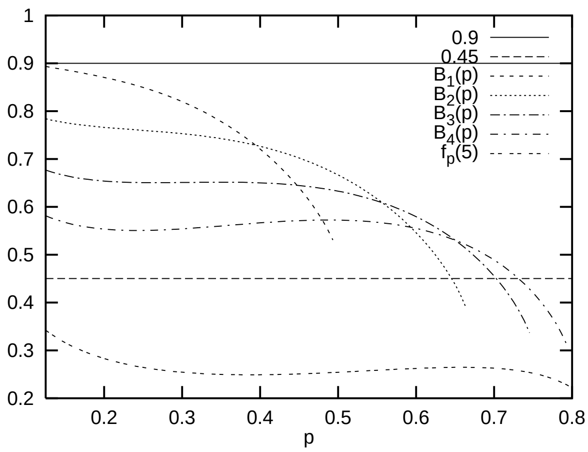

Figure 4: Behavior of $B_b(p)$ for small b.

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. [44]) 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 $\lim_{p\to b/(b+1)} B_b(p) = (b/(b+1))^{2b^2/(b+1)} < 0.9$. Essentially, for fixed b, the proof can be done "by inspection".

Figure 4 shows the plots of the function $B_b(p)$ for $b \in \{1, 2, 3, 4\}$. One can make such an argument more rigorous using interval arithmetic computations (e.g. [21]). For $b \geq 5$ we exploit that $p^{-p}q^{-q} \leq 2$ so that it also suffices to show that

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

In Figure 4 it can be seen that this relation holds for $b = 5$ and Lemma 18 (setting $\epsilon = 0$) implies that for a larger b the maximum of $f_p(b)$ can only decrease. ■

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 $L_{\max}(G'_s) = L' < L_{\max}(G_s)$. Then, 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 11 *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 6, $L^*_{\max} > L'$. ■

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 [24]. Moreover, Theorem 5 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 perfect balance $L_{\max} = \lceil N/D \rceil$ can also be achieved perhaps using a different strategy. The following Theorem answers this question negatively for small integer N/D even for average case problems and if we allow more redundancy.

Theorem 12 *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 choosen 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: (Outline) 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 $\sum_i w_i \le wD$. Now consider a set of requested logical blocks choosen 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

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

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

$$\mathbb{E}X := \sum_i \mathbb{P}[X_i = 0] \geq \sum_i e^{-w_i b} \geq De^{-wb}$$

Where the latter estimate can be obtained by minimizing the function $g(w_1, \ldots, w_D) = \sum_i e^{-w_i b}$ under the constraint $\sum_i w_i \leq wD$ using calculus.

Now we use the method of bounded differences [17, Theorem 4.18] to show that X is sufficiently sharply concentrated around its mean to make it 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

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

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

3.5 Experiments

Similar to the case of queued writing, it is of practical interest to augment 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 7. 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 less blocks than maximally possible. The curves for the average performance exhibit little dependence on D.[8] To get high probability guarantees for good

[8]For small D the average performance is slightly better mainly because we use the practically more logical strategy to place blocks on different disks. High probability guarentees are easier to obtain for large D. This is not astonishing since the failure probability depends on D.

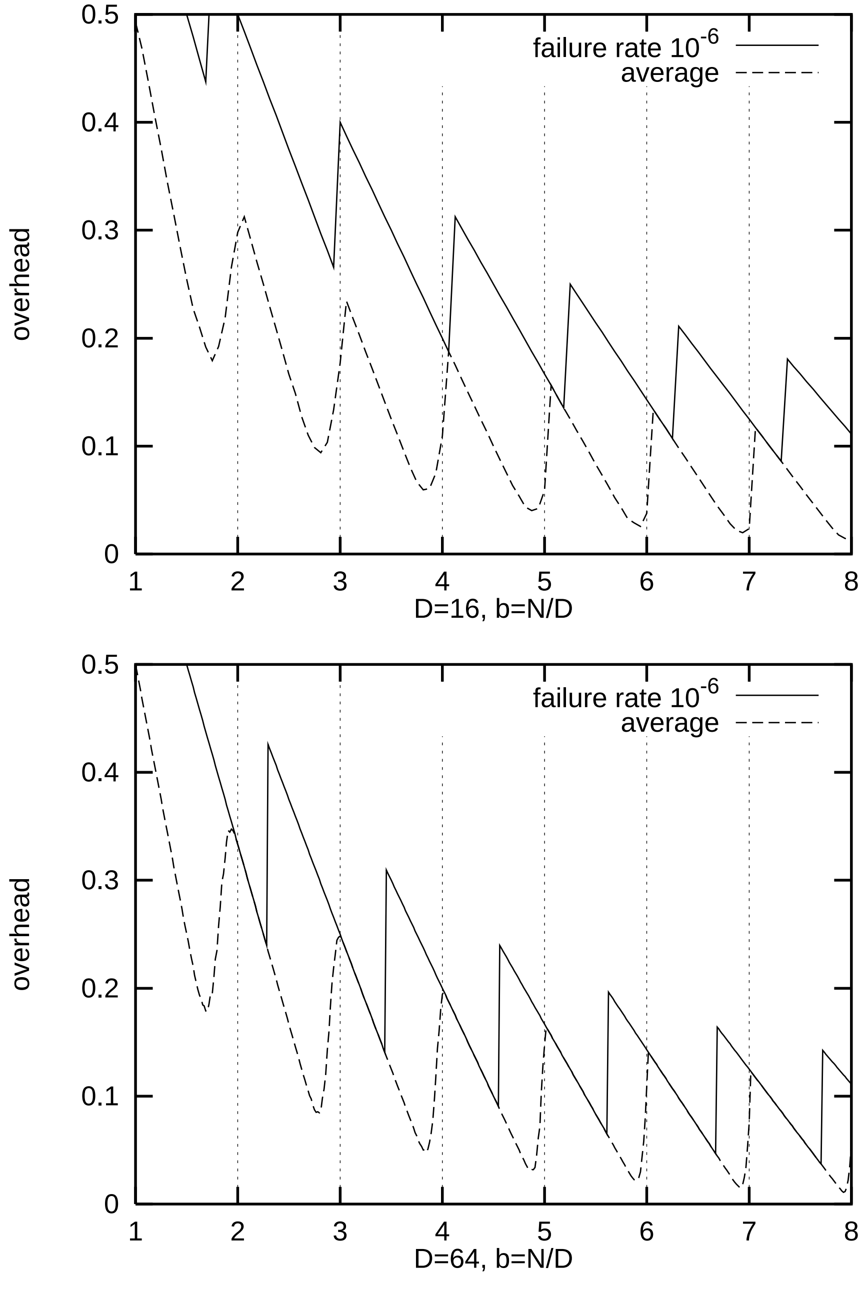

Figure 5: Overhead $1 - N/L^*_{\max}$ of RDA with N blocks to be retrieved.

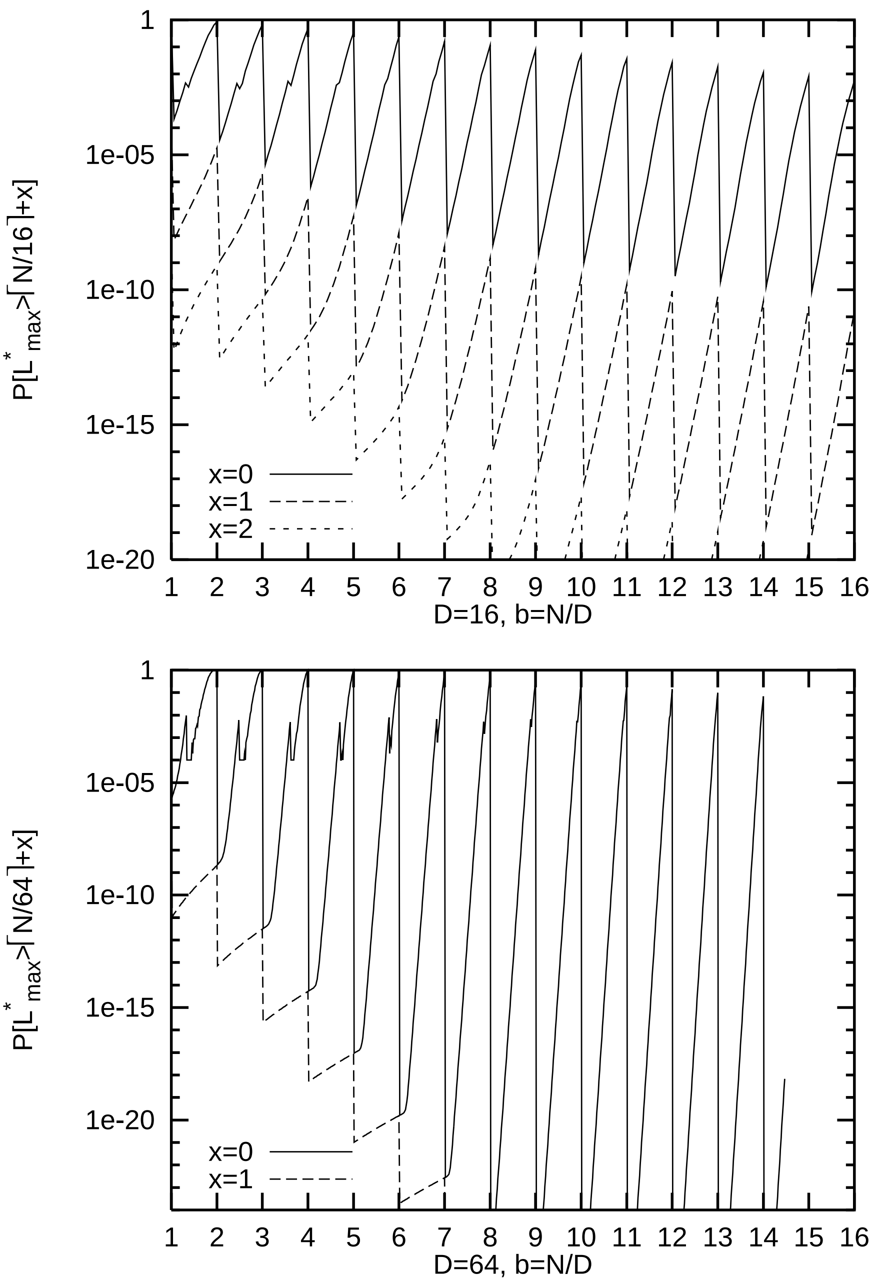

Figure 6: Failure probabilities of RDA with N blocks to be retrieved. Probabilities exceeding 0.01 are estimated using simulation. Smaller probabilities use the tail bound from Lemma 7.

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.

4 Fast Scheduling

For very large D, the worst-case bounds for maximum flow computations ($\Omega\left(D^{3/2}\right)$, [18]) might become too expensive, since eventually, the scheduling time exceeds the access time.[9] 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 13 *Given a batch of $N = \Theta(D)$ blocks.*[10] *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 $1 - \mathcal{O}(1/D)^{b+1-\epsilon}$.*[11]

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 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 then there must also exist a set of disks Δ with $L_\Delta > |\Delta| (b + 1 - \epsilon)$. 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.

[9] In our small prototype server with eight disks scheduling time is still negligible however.

[10] The assumption $N = \Theta(D)$ is for technical convenience only. Note that it encompasses the most interesting case.

[11] Using more careful rounding in lemmata 14 and 16, even sharper probabilistic bounds can be obtained because it turns out that we do not need to take small overloaded sets into account.

4.1 Unavoidable Loads

Our key argument is a counterpart to Theorem 6:

Lemma 14 *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 $(v, \dots, u)$ from v to a disk u with $L_u \le L_v - 2$ and a path length $|(v, \dots, u)| \le \log_{1+\gamma} D + 1$, 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 $D \ge |\Delta| \ge (1+\gamma)^j$ and hence $j \le \log_{1+\gamma} D$. 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 $L_\Delta > |\Delta|(1-\gamma)(L_v - 1)$. 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

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

■

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 14 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 13

Let us introduce the abbreviations $b_\epsilon := b + 1 - \epsilon$ and $P_d^\epsilon := \mathbb{P}[L_\Delta \ge d b_\epsilon + 1]$ for a subset Δ of size d. Analogous to Lemma 7 and its proof, we have to prove that $\sum_{d=1}^{D} \binom{D}{d} P_d^\epsilon = \mathcal{O}(1/D)^{b_\epsilon}$. In principle, we could replace Section 3.2 by the simple remark that it is the special case $\epsilon = 0$ of the present analysis. However, this would reduce the accessibility of the basic result for $\epsilon = 0$, which is perhaps more important than the refinement presented here. We therefore choose the following compromise between understandability and low redundancy: The less interesting technical lemmata are proven for the general case. The main line of argument for the proof is done in detail for the case $\epsilon = 0$ in Section 3.2. This has the additional advantage to yield more favourable constant factors inside the analysis. Here, we only outline the necessary modifications.

As before, the sum $\sum \binom{D}{d} P_d^\epsilon$ 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 15 $\sum_{d \le D/16} \binom{D}{d} P_d = \mathcal{O}(1/D)^{b_\epsilon}$

The proof is very similar to the proof of Lemma 9 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 9.

Lemma 16 *For any* $0 \le \epsilon < 1$, *and* $b_\epsilon = (b + 1 - \epsilon)$

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

Proof: First, we estimate

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

using the Stirling approximation. Now, setting $x = db_\epsilon + 1$, $p = d/D$, $N = bD$ in Lemma 8, we get $\mathbb{P}\left[L_\Delta \ge db_\epsilon + 1\right] \le f \cdot g$ where

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

We have

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

where the last estimate stems from the relation

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

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

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

$$g_2 = \left(\frac{1 - \frac{d}{D}}{1 - \frac{db_\epsilon+1}{bD}}\right)^{bD-db_\epsilon-1} = \left(1 + \frac{db_\epsilon - db + 1}{bD - db_\epsilon - 1}\right)^{bD-db_\epsilon-1} \le e^{db_\epsilon - db + 1} \quad .$$

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

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

$$= \left(\frac{d}{D}\right)^{d(b+1-\epsilon)-d} e^{d-d(1-\epsilon)+bd+d(b+1-\epsilon)-db+1} = \left(\frac{d}{D}\right)^{d(b-\epsilon)+1} e^{d(b+1)+1} .$$

■

4.2.2 Larger Δ

Lemma 17 *For $\epsilon \leq 1/5$,* $\sum_{D/16<d<Db/b_\epsilon} \binom{D}{d} P_d^\epsilon = e^{-\Omega(D)}$.

Proof: Using an analogous argument as in the proof of Lemma 10 we can see that it suffices to evaluate

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

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

Lemma 18 *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 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 24 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 25 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 $b \geq \delta$. 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 24 and 25 are proven in Appendix A.2.

4.3 Maximum Flow with Short Augmenting Paths

What remains to be done to establish Theorem 13 is to explain how all augmenting paths of 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.

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. Dinic' algorithm [15] 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 additionally exploiting that we are essentially dealing with a unit capacity flow problem. This 'essentially' 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 [12, 2], 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. [5] 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 [13, Section 7] using a result by Pittel, Spencer, and Wormald on "k-cores" [32]. 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 edge without an outgoing connection.

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 6 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 11 is also easy to generalize.

The most difficult part is again the probabilistic analysis. We would like to generalize Theorem 5 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

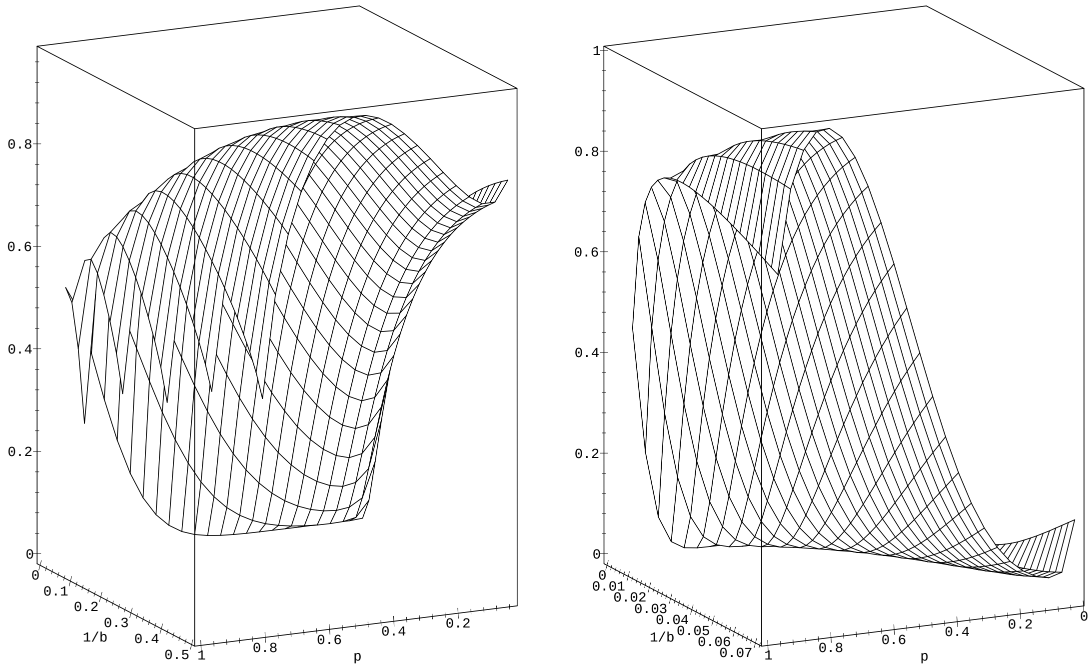

Figure 7: Behavior of $B_{b2}(p)$ for $b \geq 2$ and $B_{b4}(p)$ for $b \geq 14$.

establish a general emulation result between the multi-head model and independent disks. This result is summarized by the following lemma:

Lemma 19 *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 $g_r(y, p)$. 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 19

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 5. Lemma 7 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 20 *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 20 in a similar way as we used Lemma 8 in the proof for $r = 1$.

We split the sum from Lemma 7 into the intervals $\left\{0..\frac{D}{14r}\right\}$, $\left\{\frac{D}{14r}..\frac{Drb}{rb+1}\right\}$ and $\left\{\frac{Drb}{rb+1}..D\right\}$ where the last interval contributes only zero summands.

Appendix A.3 proves the following generalization of Lemma 9 by setting $T = 1 + \frac{1}{rp}$ in the Chernoff bound from Lemma 20.

Lemma 21 *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 10 that for $r \geq 2$,

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

However, this only holds for sufficienly large b depending on r. Furthermore, we only know how to show this analytically if r and b are fixed. Still, the result holds for all D, and by evaluating a two-dimensional function we will come very close to a proof for arbitrarily large b and fixed 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 20 then yields

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

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 (refer to the proof of Lemma 10 for details). ■

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 $D = 64$ 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 $b = 77$.

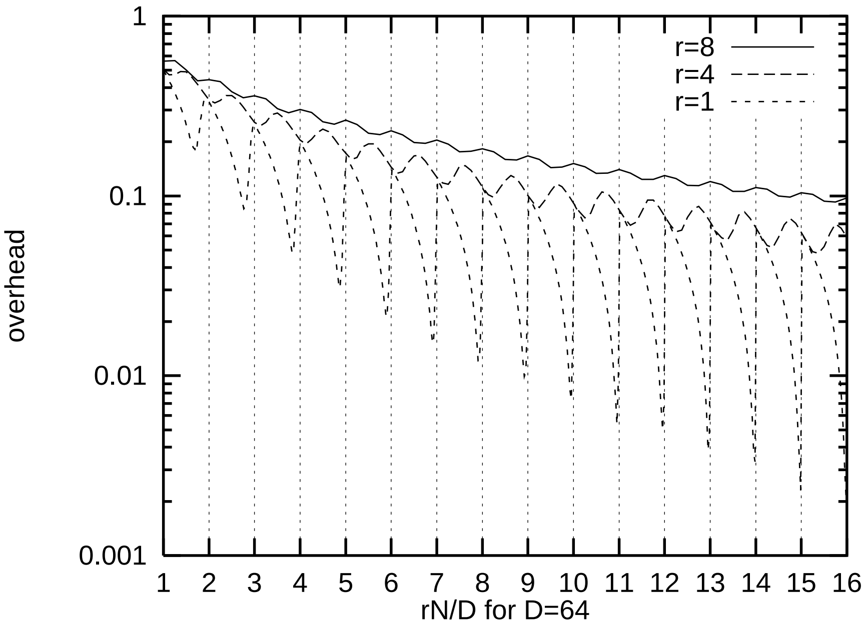

Figure 8: Average overhead (logarithmic scale) $1 - rN/L^*_{\max}$ of the r out for $r+1$ scheme with N blocks to be retrieved.

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* [25, 19]. 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 22 *Let* $\text{MHDM-I-O}_{D,B,M}(i,o)$ *denote the set of problems*[12] *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* $\text{IPDM-I-O}_{D,B,M}(i,o)$ *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 5), we can immediately conclude:

Corollary 23 *For any* $0 < \epsilon < 1$ *and* $b \in \mathbb{N}$,

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

where $i' = i \cdot (b+1) + \mathcal{O}(i/D)$ *and* $o' = o \cdot 2(b/(1-\epsilon) + e^{-\Omega(D)})$.

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.

[12] In a complexity theoretic sense.

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* [6, 23, 38], 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.

Writing can be made more efficient if we replace the hash function by a directory that maps logical blocks to disks. We can then dynamically remap blocks. In particular, we can write exactly D blocks in a single parallel write step by generating a random permutation of the disk indices, and mapping the blocks to be written to these disks. Note that, in practice, the additional hardware cost for a directory is relatively small, because a block on a disk is much more expensive than a directory entry in RAM.

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 [41]. 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 [23]. Ingenious deterministic algorithms have been developed that adapt multi-way merging to independent disks [29]. Since the known deterministic algorithms increase the number of I/Os by a considerable factor, Barve et al. [6] have developed a more practical algorithm based on randomized striping, which also achieves $\mathcal{O}\big(\frac{N}{DB}\log_{M/B}\frac{N}{M}\big)$ I/Os if $M = \Omega\,(D\log D)$. 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 $\epsilon > DB/M$. Further practical improvements are possible using prefetching, randomized striping and mixing of input and output steps.

Using randomized striping and the fact that queued writing does not require redundant allocation, we can even avoid redundant storage. We use distribution sort [41, Section 2.1] and select $k = \min(\alpha\frac{M}{B}, \beta\frac{N}{M})$ partitioning elements $\{s_0, s_1, \ldots, s_{k-1}, s_k\}$ for appropriate constants α and β. The partitioning elements are found by sorting a random sample of size $K = \Theta(k\log(N/M))$ and then choosing s_i as the element with rank iK/k in the sample. The input sequence is read using striping and all elements are classified into k buckets such that bucket j contains all elements x with $s_{j-1} \le x < s_j$. The buckets are files organized by randomized striping without redundancy. This can be done using $\frac{N}{BD}$ read steps and $\frac{N}{BD(1-\epsilon)}$ write steps using queued writing for some small positive constant ϵ determined by the memory available for the write buffer. Since the buckets are again striped, we can apply the algorithm recursively to each bucket. Using Chernoff bounds it can be shown that all buckets will have about the same size (e.g. [10]) and hence $\log_{\Omega(M/B)}\frac{N}{M}$ levels of recursion are sufficient. Overall, we get $\frac{2N}{DB(1-\epsilon)} \log_{\Omega(M/B)} \frac{N}{M}$ I/Os plus the overhead for retrieving samples. Our choice of k makes sure that for $\log \frac{M}{B} \ll \frac{M}{B}$ the latter overhead is small compared to the I/Os needed for classification. This is the case whenever $M \gg B$.

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. The batched geometric problems mentioned in [41] (orthogonal range queries, line segment intersection, 3D convex hulls, triangulation of point sets, point location, etc.) can even be handled without redundancy using randomized striping and queued writing. The same is true for many data structure problems, e.g., buffer trees [3].

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 [35]. A fifty percent overhead for duplicate writing is not an issue in this case.

Parallel algorithms are a productive source of external memory algorithms. Several researchers give frameworks for emulating PRAM algorithms [11] or BSP algorithms [37, 14] on the external memory. Using Corollary 23 these result extends to parallel disks. Some graph problems like list ranking can be solved efficiently using emulation of parallel algorithms.

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 [28]. 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).[13] 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 [19].

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 [40, Corollary 1]). Over larger alphabets, however, other MDS codes exists such as Reed-Solomon codes. By the choice of an appropriate MDS code one can protect against disk failure (as in [19]), 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.

References

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

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

[3] Arge, L. The buffer tree: A new technique for optimal I/O-algorithms. In *4th WADS* (1995), no. 955 in LNCS, Springer, pp. 334–345.

[4] Azar, Y., Broder, A. Z., Karlin, A. R., and Upfal, E. Balanced allocations. In *26th ACM Symposium on the Theory of Computing* (1994), pp. 593–602. Full version to appear in SIAM J. Computing.

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

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

[7] Berenbrink, P., Czumaj, A., Steger, A., and Vöcking, B. Balanced allocations: The heavily loaded case. In *32th Annual ACM Symposium on Theory of Computing* (2000). accepted for publication.

[13] For a treatment of coding theory refer to the book of MacWilliams and Sloane, [25], 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.

[8] Berson, S., Muntz, R., and Wong, W. 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).

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

[10] Blelloch, G. E., Leiserson, C. E., Maggs, B. M., Plaxton, C. G., Smith, S. J., and Zagha, M. A comparison of sorting algorithms for the connection machine CM-2. In *ACM Symposium on Parallel Architectures and Algorithms* (1991), pp. 3–16.

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

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

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

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

[15] Dinic, E. A. Algorithm for solution of a problem of maximum flow. *Soviet Math. Dokl. 11* (1970), 1277–1280.

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

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

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

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

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

[21] Hansen, E. Global optimization using interval analysis – the multidimensional case. *Numerische Mathematik 34* (1980), 247–270.

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

[23] Knuth, D. E. *The Art of Computer Programming — Sorting and Searching*, 2nd ed., vol. 3. Addison Wesley, 1998.

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

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

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

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

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

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

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

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

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

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

[34] Salem, K., and Garcia-Molina, H. Disk striping. *Proceedings of Data Engineering'86* (1986).

[35] Sanders, P. Fast priority queues for cached memory. In *ALENEX '99, Workshop on Algorithm Engineering and Experimentation* (1999), no. 1619 in LNCS, Springer, pp. 312–327.

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

[37] SIBEYN, J., AND KAUFMANN, M. BSP-like external-memory computation. In *3rd Italian Conference on Algorithms and Complexity* (1997), pp. 229–240.

[38] TETZLAFF, W., AND FLYNN, R. Block allocation in video servers for availability and throughput. *Proceedings Multimedia Computing and Networking* (1996).

[39] TEWARI, R., MUKHERJEE, R., DIAS, D., AND VIN, H. Design and performance tradeoffs in clustered video servers. *Proceedings of the International Conference on Multimedia Computing and Systems* (1996), 144–150.

[40] TOLHUIZEN, L. On maximum distance separable codes over alphabets of arbitrary size. In *IEEE International Symposium on Information Theory* (1994), p. 431.

[41] VITTER, J. S. External memory algorithms. In *6th European Symposium on Algorithms* (1998), no. 1461 in LNCS, Springer, pp. 1–25.

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

[43] VÖCKING, B. How asymmetry helps load balancing. In *40th FOCS* (1999), pp. 131–140.

[44] WORSCH, T. Lower and upper bounds for (sums of) binomial coefficients. Tech. Rep. IB 31/94, Universität Karlsruhe, 1994.

A Proof Details

A.1 Proof of Lemma 15

Proof: Lemma 16 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 17

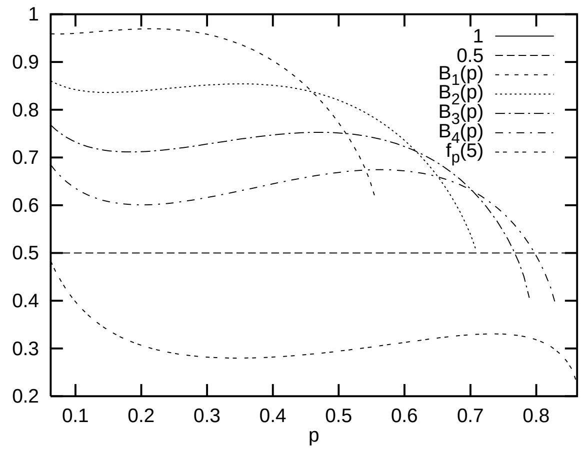

Figure 9: Behavior of $B_b(p)$ for $\epsilon = \frac{1}{5}$ and small b.

Lemma 24 *For* $p < \frac{b-1}{b+1}$ *and any* $0 \le \epsilon < 1$,

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

is non-increasing.

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

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

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

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

for $p \le \frac{b-1}{b+1}$. However, since $\frac{\partial}{\partial\epsilon} l_b(p) \le 0$ 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

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

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

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

using series development. We get

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

This can be shown to be negative for $b \ge 34$ 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 $(p+1)^2(1-p)(b-bp-p)^2p$ 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

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

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

Lemma 25 $g_\delta(b) := \left(\frac{b(b-\delta)}{b_\epsilon^2}\right)^{b-\delta}\left(\frac{b}{\delta}\left(1-\frac{(b-\delta)^2}{b_\epsilon^2}\right)\right)^\delta$ *is non-increasing for* $b \ge \delta$.

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 $b \ge \delta$. We have

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

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

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

Finally

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

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

A.3 Proof of Lemma 21

First, we further simplify the Chernoff bound from Lemma 20 for $N = bD$, $p = d/D$ and $x = d(br+1)+1$.

Lemma 26 *For $N = bD$, $|\Delta| = d$ and $p = \frac{d}{D}$,*

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

Proof: Choosing $T = 1 + \frac{1}{rp}$ in Lemma 20 yields

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

The latter two estimates are based on Lemma 27 and the fact that $(R/r)^r = (1+1/r)^r \le e$. ∎

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

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

Completing the proof of Lemma 21 is only slightly more complicated than it was in Lemma 9. Let $f(d) = (er)^d e^{bd(r+1)(e-1)} \left(\frac{dr}{D}\right)^{dbr+1}$. It is easy to check that $f'''(d) \ge 0$ and $f'(1) \le 0$ for $D > re^{e+e/r+\ln(r)/r}$. Therefore, for sufficiently large D, f assumes its maximum over an interval $[d_{\min} \ge 1, d_{\max}]$ at one of the borders of that interval if $d_{\min} \ge 1$. For any constant $0 < \alpha < 1$, we get
$\sum_{d \le \alpha D} \binom{D}{d} P_d \le f(1) + \alpha D \max\{f(2), f(\alpha D)\}$.

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

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 27 $(Rp(R/r)^r + q^R)^{bD} \leq e^{bdR((R/r)^r-1)}$.

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 $f'(D) = f(D)bg(p)$ 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. ■