Fast Concurrent Access to Parallel Disks

Peter Sanders*, Sebastian Egner and Jan Korst†

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 adapting single-disk external memory algorithms to multiple disks. This paper shows that this problem can be solved efficiently using a combination of randomized placement, redundancy and an optimal scheduling algorithm. 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. In addition, the redundancy can be reduced from 2 to $1 + 1/r$ for any integer r. These results can be used to emulate the simple and powerful "single-disk multi-head" model of external computing [1] on the physically more realistic independent disk model [33] with small constant overhead. This is faster than a lower bound for deterministic emulation [3].

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

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

†Philips Research Laboratories, Prof. Holstlaan 4, 5656 AA Eindhoven, The Netherlands, {sebastian.egner,jan.korst}@philips.com.

Efficient single-disk external memory algorithms are available for a wide spectrum of applications (e.g. [32]), 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 [10] to show that writing can be made efficient if a pool of $\mathcal{O}(D/\epsilon)$ 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.

Since our model assumes separate read and write steps, these two issues can be analyzed separately. A parallel read is more difficult to schedule since it has to wait until all requested blocks have been read. In Section 3, we investigate *random duplicate allocation* (RDA) that uses two randomly allocated copies of each logical block. 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). In Section 4 we explain why the optimal schedules can be found much faster than the worst-case bounds of maximum flow algorithms would suggest.

RDA is generalized in Section 5. 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. Mixed workloads with different degrees of redundancy are possible. 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 $1 + 1/r$.

The techniques for reading and writing can be joined to a 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 emulated on the independent disk model [33] using only small linear overhead factors. This is faster than a lower bound on deterministic emulation by Armen [3] which shows that $\Omega\left(T\frac{\log(N/D)}{\log\log(N/D)}\right)$ I/O steps are needed for emulating T I/O steps of a multi-head algorithm for a problem of size N. 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.

1.2 Related Work. The predominant general technique to deal with parallel disks in practice is *striping* [27, 24]. 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. Refer to [17] for a detailed discussion why this is costly in video servers. In many external memory algorithms striping converts a factor $\log_{M/B} N$ in the I/O bound to a factor $\log_{M/(DB)} N$ which can be significantly larger.

Reducing access contention by random placement is a well-known technique. For example, Barve et al. [5] 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 – usually for the case $N = D$ or $N = \Theta(D)$. Azar et al. [4] show that an optimal online strategy commits each arriving request to the least loaded unit. This strategy achieves a maximum load of $\mathcal{O}(\log\log D)$ whp. They also state that load $\mathcal{O}(1)$ can be achieved using offline scheduling. In Section 4.2 we review how such offline algorithms can be used to get approximately optimal schedules in linear time. For PRAM simulation, fast parallel scheduling algorithms have been developed even earlier [15]. PRAM simulation using a 3-collision protocol achieves maximum load 3 for $N = D$ using $\mathcal{O}(\log\log D)$ iterations [20, Section 3]. This already works for $\mathcal{O}\big((\log D)^3\big)$-universal classes of hash functions. Similar results hold for allocation strategies with lower redundancy such as the ones we describe in Section 5. We reduce the gap between $\lceil N/D\rceil$ and the maximal load from a a constant factor to the additive constant one

Heuristic load balancing algorithms using redundant storage are used by a number of authors in multimedia applications [30, 31, 17, 21]. Even the idea of a parity sub-block built out of r data sub-blocks has been used by several researchers [6, 7]. The first optimal scheduling algorithm for RDA was presented in [17]. 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 2.1), 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 following procedure to write up to $(1-\epsilon)D$ blocks. (In practice one can simply use $\epsilon = 0$ refer to Section 6.2 for a discussion.)

```
write((1−ε)D blocks):
  append blocks to Q_1, ..., Q_D;
  write-to-disks(Q_1, ..., Q_D);
  while |Q_1| + ··· + |Q_D| > W do
      write-to-disks(Q_1, ..., Q_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. 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.

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]

The remainder of this section contains the proof of 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.

THEOREM 2.1. *Consider $W = (\ln 2 + \delta)D/\epsilon$ for some constant $\delta > 0$ and let $n^{(t)}$ be the number of calls to* `write-to-disks` *during the t-th invocation of* `write`. *Then $\mathbb{E}n^{(t)} \le 1 + e^{-\Omega(D)}$.*

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 2.2) 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 2.1. *Let $Q_i^{(t)}$ be the length of Q_i at the t-th invocation of* `write` *after the new blocks have been appended Then $\mathbb{E}Q_i^{(t)} \le 1/(2\epsilon)$ 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 *queuing 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 [23, 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 pgf of the binomially distributed variable $X_i^{(t)}$. 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

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

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, $\mathbb{E}Q_i^{(t)} \le \mathbb{E}Q_i^{(\infty)}) = G'(1)$ and

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

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 [13, Exercise 8.12a]). ∎

Based on Lemma 2.1 we give an upper bound on the probability that the `while`-loop is entered for a given limit $W = qD$ of internal memory.

LEMMA 2.2. *Let $Q^{(t)} = Q_1^{(t)} + \cdots + Q_D^{(t)}$ with $Q_i^{(t)}$ as in Lemma 2.1. Then $\mathbb{E}Q^{(t)} \le D/(2\epsilon)$ and*

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

[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 2.1 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.

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 [10, 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 [10, 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, [10, 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 2.1) the tail bound follows. The bound on the expected value follows directly from Lemma 2.1 and the linearity of the expected value. ∎

We are now ready to prove Theorem 2.1, the main result of this section.

Proof. `Write-to-disks` is called at least once during the t-th invocation of `write`. Lemma 2.2, with $W/D = q = (\ln(2)+\delta)/\epsilon$, gives the probability that the body of the **while**-loop is entered

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

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

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

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

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

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

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 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. Finally, $L^*_{\max}$ is the load of an *optimal schedule*. This is a schedule G_s for G_a with minimal $L_{\max}(G_s)$.

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

THEOREM 3.1. *Consider a batch of N randomly and duplicately allocated blocks to be read from D disks. Then, abbreviating $b = \lceil N/D \rceil$,*

$$\mathbb{P}\left[L^*_{\max} > b + 1\right] = \mathcal{O}(1/D)^{b+1}.$$

Note that Lemma 3.1 below also provides more accurate bounds for small D and N that can be evaluated numerically.

A difficulty in establishing Theorem 3.1 is that optimal schedules are hard to analyze directly using probabilistic arguments because their structure is determined by a complicated scheduling algorithm. Therefore, we first derive a characterization of optimal schedules in terms of the allocation graph G_a which is simply a random graph. Since this result is of some independent interest and of completely combinatorial nature, 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.

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

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 3.2. $L^*_{\max} = \max_{\emptyset \neq \Delta \subseteq \{1..D\}} \left\lceil \frac{L_\Delta}{|\Delta|} \right\rceil$.

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. ∎

An important consequence of Theorem 3.2 is that *perfect* load balance (i.e. $L^*_{\max} = N/D$ whp) is not possible unless $N = \Omega(D \log D)$ since the allocation graph G_a contains isolated nodes.

3.2 Proof Outline of Theorem 3.1. 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 the proof is the following simple probabilistic upper bound on the maximum load of optimal schedules.

LEMMA 3.1. $\mathbb{P}[L^*_{\max} > b + 1] \leq \sum_{d=1}^{D} \binom{D}{d} P_d$

where $P_d := \mathbb{P}[L_\Delta \geq d(b+1) + 1]$ *for a subset* Δ *of size* d.[6]

Proof. By Theorem 3.2 and the principle of inclusion-exclusion, $\mathbb{P}[L^*_{\max} > b + 1] = \mathbb{P}[\exists \Delta : L_\Delta > |\Delta|(b+1)] \leq \sum_{d=1}^{D} \binom{D}{d} P_d$ since $\binom{D}{d}$ is the number of subsets of size d. ∎

Lemma 3.1 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 the following 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.

LEMMA 3.2. *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. The full paper gives a detailed proof which basically consists in transforming [19, Lemma 2.2]. ∎

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

LEMMA 3.3.

$$\sum_{d \leq D/8} \binom{D}{d} P_d = \mathcal{O}(1/D)^{b+1}$$

Proof. In the full paper we prove that

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

Viewing this bound as a function $f(d)$ of d, it is easy to check that $f''(d) \geq 0$ (differentiate, remove obviously growing factors and differentiate again). Therefore, f assumes its maximum over any positive interval at one of the borders of that interval. We get $\sum_{d \leq D/8} \binom{D}{d} P_d \leq f(1) + D \max\{f(2), f(\alpha D)\}/8$.

$$\begin{aligned} f(1) &= D^{-b-1} e^{b+2} = e(e/D)^{b+1} = \mathcal{O}(1/D)^{b+1} \\ Df(2)/8 &= D(2/D)^{2b+1} e^{2b+3}/8 = \mathcal{O}(1/D)^{2b} \\ Df(D/8)/8 &= D(1/8)^{Db/8+1} e^{D(b+1)/8+1}/8 \\ &= \mathcal{O}(D)\, e^{D(b(1+\ln(1/8))+1)/8} \\ &= e^{-\Omega(D)} . \end{aligned}$$

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

[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 [26, 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 an faster bound than our simulation results from [17].

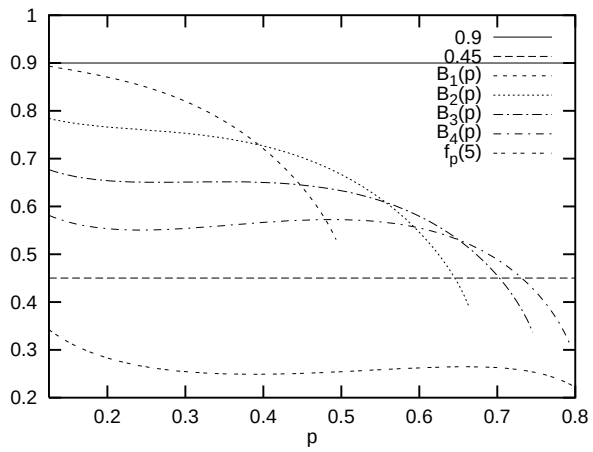

Figure 1: Behavior of $B_b(p)$ for small b.

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

LEMMA 3.4. $$\sum_{\frac{D}{8}<d<\frac{Db}{b+1}} \binom{D}{d} P_d = \mathcal{O}\left(\sqrt{D}\cdot 0.9^D\right) \quad .$$

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 3.2 we get

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

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

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

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

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

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

for all $1/8 < p < b/(b+1)$. For fixed b, this is easy since $B_b(p)$ is a smooth function and because 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 1 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. [14]).

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

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

In Figure 1 it can be seen that this relation holds for $b = 5$. In the full paper we show that for larger b the maximum of $f_p(b)$ can only decrease. Here we only outline the basic approach. For $p \le (b-2)/b$, $f_p(b)$ can be shown to be non-increasing using calculus. For $p > (b-2)/b$, we make the substitution $p := \frac{b-\delta}{b+1}$ and then show that

$$g_\delta(b) := f_p(b)\left[p \leftarrow \frac{b-\delta}{b+1}\right]$$

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

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

since $p-(1-p)/b \ge 1/2$ for $b \ge 5$, $\epsilon \le 1$, and $p > (b-2)/b \ge 3/5$. ∎

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 3.5. *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 3.2, $L^*_{\max} > L'$. ∎

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 [17]. Moreover, Theorem 3.1 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.

4 Fast Scheduling

For very large D, the worst-case bounds for maximum flow computations might become too expensive, since eventually, the scheduling time exceeds the access time.[7] Therefore, we explain in Section 4.1, why slightly modified maximum flow algorithms can actually find an optimal schedule in time $\mathcal{O}(D \log D)$ with high probability. In Section 4.2 we outline how previous results can be adapted to yield approximate schedules in linear time.

4.1 Fast Flows

THEOREM 4.1. *Given a batch of $N = \Theta(D)$ blocks.*[8] *Let $b = \lceil N/D \rceil$ and define a constant $0 < \epsilon \leq 1/5$. An schedule with maximum load $b+1$ can then be found in time $\mathcal{O}(D \log D)$ with probability $1 - \mathcal{O}(1/D)^{b+1-\epsilon}$.*

The full paper develops a proof. Here we give a summary. First note, that maximum flow algorithms essentially compute optimal schedules by removing all paths from overloaded to underloaded nodes. Our approach is to remove only paths with logarithmic length. We show that this either yields a schedule with maximal load at most $b+1$ or implies the existence of a set of disks Δ with rather high load. Although this load is slightly smaller than the load predicted by Theorem 3.2 for the case that paths of arbitrary length have been considered, we demonstrate how the probabilistic analysis from Section 3.2 can be generalized to show that a set Δ with such a high load is improbable.

Finally, we explain why the required flow computations are easier if only paths of logarithmic lengths are needed. In particular, it is almost trivial to see why Dinic' algorithm can solve the problem in time $\mathcal{O}\big(D \log^2 D\big)$ and a revision of the analysis of preflow push algorithms shows that those even work in time $\mathcal{O}(D \log D)$.

4.2 Linear Time Approximation. In their forthcoming full paper on balanced allocation, Azar et al. [4] 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 [8, Section 7] using a result by Pittel, Spencer, and Wormald on "k-cores" [25]. 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.

This acutally yields a reasonable practical algorithm. First note that it can be implemented in linear time using a bucketed priority queue with one bucket for each possible node degree. 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

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

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

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 3.2 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 3.5 is also easy to generalize.

The most difficult part is again the probabilistic analysis. We would like to generalize Theorem 3.1 for arbitrary r. Indeed, we have no analysis yet which holds for all values of r and N/D. Yet, in the full paper, we outline an analysis which can be applied for any fixed r (we do that for $r \leq 10$) and yield the desired bound for sufficiently large N/D but still for all D. This already suffices to analyze a concrete application in a scalable way, and to establish a general emulation result between the multi-head model and independent disks.

6 Applications and Refinements

Sections 2 and 3 treat queued writing and reading with RDA as two independent techniques. We now 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. In Section 6.3 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 the full paper 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* [18, 12]. This allows more flexible tradeoffs between low redundancy and high fault tolerance.

6.1 Emulating Multi-Headed Disks. We 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 6.1. *Let* $\text{MHDM-I-O}_{D,B,M}(i,o)$ *denote the set of problems*[9] *solvable on a D-head disk with block size B and internal memory of size M using i parallel read steps and o parallel write steps. Let* $\text{IPDM-I-O}_{D,B,M}(i,o)$ *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*[10] *hash function.*

Using queued writing (Theorem 2.1) and RDA (Theorem 3.1), we can immediately conclude:

COROLLARY 6.1. *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.

6.2 Refinements. 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* [5, 16, 30], 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.

[9] In a complexity theoretic sense.

[10] Using the collision protocols from [20] the same asymptotic bounds can be proven for $\mathcal{O}((\log D)^3)$-universal classes of hash functions at the cost of larger constant factors.

We have no formal proof yet but conjecture that our analysis extends to this *random striped placement*. Some applications are described below.

A more radical measure is to 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 the 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 [32]. 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 [16]. Ingenious deterministic algorithms have been developed that adapt multi-way merging to independent disks [22]. Since the known deterministic algorithms increase the number of I/Os by a considerable factor, Barve et al. [5] have developed a more practical algorithm based on randomized striping, which also achieves $\mathcal{O}\left(\frac{N}{DB}\log_{M/B}\frac{N}{M}\right)$ I/Os if $M = \Omega\left(D\log D\right)$. Our general emulation result does not have this restriction and achieves $2(1+\frac{1}{r}+\epsilon)\frac{N}{DB}\log_{\Omega(M/B)}\frac{N}{M}$ for $\epsilon > 0$. 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 [32, Section 2.1] and select $\mathcal{O}\left(\frac{M}{B}\right)$ partitioning elements $\{s_0, s_1, \ldots, s_{k-1}, s_k\}$ based on a random 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 any constant $\epsilon > 0$. Since the buckets are again striped, we can apply the algorithm recursively to each bucket. Overall we get $\frac{2N}{DB(1-\epsilon)}\log_{\Omega(M/B)}\frac{N}{M}$ I/Os plus a small overhead for retrieving samples. The full paper gives more detail.

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 [32] (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 [2].

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 so much faster [28] that a fifty percent overhead for duplicate writing is not an issue.

Parallel algorithms are a productive source of external memory algorithms. There are even formal frameworks for this approach which emulate parallel algorithms for the BSP model [29, 9] or PRAMs [11] using a single disk. Using Corollary 6.1 this result extends to parallel disks.

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. Similar properties hold for interactive graphics applications [21]. 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.

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. Petra Berenbrink, Artur Czumaj, Martin Dietzfelbinger and Friedhelm Meyer auf der Heide advised us regarding the probabilistic foundations of the problem and related work.

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] 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.

[3] Armen, C. Bounds on the separation of two parallel disk models. In *IOPADS* (Philadelphia, May 1996), ACM Press, pp. 122–127.

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

[6] 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).

[7] 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).

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

[9] 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.

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

[11] et al., Y.-J. C. External memory graph algorithms. In *6th Annual ACM-SIAM Symposium on Discrete Algorithms* (1995), pp. 139–149.

[12] 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.

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

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

[15] 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.

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

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

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

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

[20] 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.

[21] 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.

[22] 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.

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

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

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

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

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

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

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

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

[31] 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.

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

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

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