Reconciling Simplicity and Realism in Parallel Disk Models

Peter Sanders*

Abstract

For the design and analysis of algorithms that process huge data sets, a machine model is needed that handles parallel disks. There seems to be a dilemma between simple and flexible use of such a model and accurate modelling of details of the hardware. This paper explains how many aspects of this problem can be resolved. The programming model implements one large logical disk allowing concurrent access to arbitrary sets of variable size blocks. This model can be implemented efficienctly on multiple independent disks even if zones with different speed, communication bottlenecks and failed disks are allowed. These results not only provide useful algorithmic tools but also imply a theoretical justification for studying external memory algorithms using simple abstract models.

The algorithmic approach is random redundant placement of data and optimal scheduling of accesses. The analysis generalizes a previous analysis for simple abstract external memory models in several ways (Higher efficiency, variable block sizes, more detailed disk model). As a side effect, an apparently new Chernoff bound for sums of weighted 0-1 random variables is derived.

1 Introduction

Ever larger data sets arise in important applications like data mining, electronic libraries, web servers, virtual reality, geographic information systems, or scientific computing. Often, no size limits are in sight. To process these massive data sets with high performance, many disk drives have to be operated in parallel. This situation is not likely to change with larger RAM chips since many applications are technology driven – they fill up whatever amount of memory we are able to pay and handle efficiently.

Operating parallel disks efficiently is more complicated than operating a single disk. We have to decide how to spread the data over the disks so that their capacity is not exceeded, and so that all important access patterns to the data can be served without bottlenecks. The situation is further complicated by the fact that not all individual disk accesses take the same time, that many applications require high performance even if a disk fails, and that the disk access itself is not always the only bottleneck in the system. Finally, users do not like constant factors hidden behind asymptotic notation but would prefer performance close to the peak performance of the system.

Handling this situation mathematically implies two competing requirements. On the one hand, our model should be as close to reality as possible. On the other hand, the complexity of the system model should be shielded from the application level view on the system.

A previous paper [16] describes a partial solution of this problem. Its system model is the independent parallel disk model of Vitter and Shriver [19], i.e, all block sizes and access times are assumed to be identical, no disk failures and no communication bottlenecks appear. A randomized technique is developed there which emulates the "multi-head" model of Aggarwal and Vitter [3]. The multi-head model is easier to use since the application program does not need to care about data placement. However, long data accesses still have to be broken down into requests of uniform size manually.

The purpose of this paper is to lift as many of these restrictions as possible. A more general system model is considered which allows disk failures, multi-zone disks that need different access times depending on which track of the disk the block is stored, and communication bottlenecks. The user view on the system is as simple as before except that requests may have variable length if desired. High probability performance guarantees are derived and the emulation is more efficient than before.

Section 1.1 explains the model and Section 1.2 summarizes the improvements made in the paper. Related work is discussed in Section 1.3. Section 1.4 reviews those tools from [16] which are needed to understand the rest of the paper.

1.1 The Basic Model: Our starting point for the system model is the *independent disk model* of Vitter and Shriver [19] with one processor, D identical disks and an internal memory of size M. The disks are

*Max Planck Insitut für Informatik Saarbrücken, Germany, sanders@mpi-sb.mpg.de, Partially supported by the IST Programme of the EU under contract number IST-1999-14186 (ALCOM-FT).

accessed in synchronized *I/O-steps* in which each disk can retrieve one block of size B. The primary concern in this model is to keep the number of I/O-steps small.

An easier to program yet physically unrealistic model is the *multi-head* model of Aggarwal and Vitter [3] where we have one large disk with N-fold capacity from which *any* N blocks can be retrieved in one I/O-step.

The main contribution of the previous paper [16] is to give a randomized algorithm for efficiently emulating the multi-head model on the independent disk model. In *random duplicate allocation* (RDA) two copies of each logical block are mapped to two random disks using a hash function assumed to behave like a truly random function. Writing is somewhat easier than reading since blocks contending for the same disk can be buffered in a shared pool of buffer blocks without delaying the progress of the application.

In what follows we therefore concentrate on the problem of reading logical blocks. Given N *requests* (logical blocks to be read), we have to decide which copy of each block should be read to minimize the number of I/O-steps needed. If several requests address the same block they can be combined into one request. If a request addresses a block which is cached or queued for writing, it can be serviced from internal memory. What remains are requests for different blocks which can be found on two independently placed copies.

There is an algorithm for this scheduling problem based on maximum flow computations. With high probability (whp) no more than $\lceil N/D \rceil + 1$ steps will be needed, i.e., the *inefficiency* $I \approx D/N$ can in principle be made arbitrarily small if the logical parallelism N is much larger than the physical parallelism D. This bound cannot be improved in general. In particular, if D is divisible by N and $N/D = o(\log D)$ then no placement strategy with at most a constant factor redundancy and no scheduling algorithm can achieve the trivial lower bound of $\lceil N/D \rceil$ even on the average.

1.2 Overview and Results: We first show in Section 2 that the inefficiency of emulating the previously described emulation can be dramatically reduced by carefully choosing N. For example, for $N = 3.84D$ we experimentally get an inefficiency as small as 4 %. Our theoretical bounds guarantee the same small inefficiency for a slightly larger $N = 5.76D$. Even this conservative estimate is four times smaller than the guarantees proven in [16] where four times larger parallelism would have been needed to achieve a similarly low inefficiency.

This high scheduling quality can even be maintained if a disk fails. In Section 3 it is shown that a system with a failed disk performs almost as good as an intact system with one disk less. We also discuss how the system can "recover" from disk failures.

High quality schedules can be obtained independently of the "sweet spots" for the ratios N/D used above if we allow requests to be split, i.e., the corresponding logical block may be assembled from pieces of both its copies. Section 4 addresses this issue.

In Section 5 the scheduling algorithm is generalized to a situation where each requested block i has its own access time t_i. Theoretical bounds are particularly important for such a model with N degrees of freedom because it cannot be adequately investigated with simulation. It turns out that optimal scheduling is NP-hard now unless we allow blocks to be split. Then, scheduling can be done as before and it is shown that the same bounds hold which we would get if all requests but possibly one have maximal size or zero size. As a side effect, a stronger form of the Chernoff bound for sums of weighted 0-1 random variables from [14] is derived.

So far we have considered the cost of actually reading a block from disk. But communicating a block from the disk to the requesting processor also costs time. Sometimes disk controllers or system buses can actually become a bottleneck. Section 6 explains how these effects can be incorporated into the scheduling algorithm. Section 7 touches some open problems.

1.3 Related Work: For an overview of the history of RDA and its alternatives refer to [16]. Aerts et al. [2] also study the problem of scheduling variable size requests. Using an LP-rounding technique by Lenstra et al. [11] they develop an approximate scheduling algorithm without splitting requests. Combining this technique with the approach from Section 5 to split large requests in subblocks might yield solutions better than either of the two approaches in isolation.

Disk failures in presence of redundancy have been considered experimentally in [9] leading to the predictions proven here. An extensive overview of results involving two random choices is given in [12]. Several studies consider more detailed models for external memory than the independent disk model (e.g., refer to the references in [18, Section 2.2]). A frequent observation is that detailed disk models do not become popular because it is unrealistic to require the wealth of external memory algorithms to be reanalyzed and redesigned for each new model. This is one reason why this paper addresses emulation of a simple model on a realistic model. On the other hand, it is often argued that analyzing external memory algorithms down to constant factors is not worth the effort as long as the assumed model does not reflect the actual system sufficiently accurately. The emulation techniques analyzed here may help to solve this problem since they promise emulation overheads which are small constant factors at least for large average block sizes.

An important issue not considered here is *asynchronous* scheduling of requests in RDA [15], i.e., request continously enter the system and we are interested in small delays of the individual requests. Simulations indicate that for this purpose quite simple algorithms already perform better than collecting requests into batches and scheduling them using synchronous algorithms. However, useful theoretical results are scarce for asynchronous scheduling even assuming a very simple machine model. Therefore, studying synchronous models can also be viewed as a first step towards understanding asynchronous scheduling.

1.4 Basic Tools:A batch of N different requests is conveniently represented by an *allocation graph* $G_a = (\{1..D\}, (\{u_1, v_1\}, \ldots, \{u_N, v_N\}))$ which behaves like a random multigraph with N edges assuming the hash function behaves like a truly random function. A *schedule* for the batch is a directed version G_s of G_a. (A 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. 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 value $L^*_{\max}$ can be characterized in terms of the *unavoidable load* L_Δ of subsets of disks Δ which is defined as the number of blocks that have both copies allocated on a disk in Δ. It can be shown [17] that

$$L^*_{\max} = \max_{\emptyset \neq \Delta \subseteq \{1..D\}} \left\lceil \frac{L_\Delta}{|\Delta|} \right\rceil \quad . \tag{1.1}$$

Using a simple combinatorial argument it follows that

$$\mathbb{P}\left[L^*_{\max} > k\right] \leq \sum_{d=1}^{D} \binom{D}{d} P_d \tag{1.2}$$

where $P_d = \mathbb{P}\left[L_\Delta \geq dk + 1\right]$ for a subset Δ of size d. It is the task of the probabilitic analysis to show that this sum is small. The binomial coefficient can be estimated using two forms of Stirling approximations. and P_d can be estimated using the Chernoff bound

$$\mathbb{P}\left[L_\Delta \geq x\right] \leq \left(\frac{Np^2}{x}\right)^x \left(\frac{1-p^2}{1-x/N}\right)^{N-x} \tag{1.3}$$

for any $x > Np^2$ where $p^2 = d^2/D^2$ if the blocks are allocated independently. This Chernoff bound is a rearrangement of a well known similar bound (e.g, [6]).

We will use the same basic approach but in Section 2 we need to calculate more accurately than in [16], in Section 3 we end up with a different value of p^2, and in Section 5 we need a more general form of the Chernoff bound.

An optimal schedule can be computed by a sequence of maximum flow computations. For a given tentative value L', the question $L^*_{\max} \leq L'$ can be answered constructively by setting up a flow network $\mathcal{N}$ derived from an arbitrary schedule graph G_s by adding source and target nodes s and t, edges (s, v) with capacity $c(s, v) = L_v(G_s) - L'$ for nodes with load exceeding L' and edges (u, t) with capacity $c(u, t) = L' - L_u(G_s)$ for nodes with load below L'. A schedule with $L_{\max} = L'$ exists if and only if a maximum flow through $\mathcal{N}$ saturates all edges leaving s. Such a schedule can be found by flipping all edges of G_s carrying flow. An optimal schedule can be found efficiently using binary search for arbitrary allocation graphs. For random schedule graphs the analysis shows that one can as well try the values $L' = \lceil N/D \rceil$ and $L' = \lceil N/D \rceil + 1$ first since these are the most likely values for $L^*_{\max}$.

2 Very High Disk Utilization

We first report a simple experiment which helps to understand why choosing the "right" value for N is crucial for good performance in Section 2.1. Section 2.2 explains this effect analytically.

2.1 Experiments: The bound $L^*_{\max} = \lceil N/D \rceil + 1$ is tight for small N/D if N is a multiple of D. It is intuitive that $L^*_{\max}$ does not change if we slightly increase N now, i.e., we have $L^*_{\max} = \lceil N/D \rceil$. If we further increase N there will be a point where $L^*_{\max}$ goes back to $\lceil N/D \rceil + 1$. The inefficiency $1 - N/(DL^*_{\max})$ is lowest just before this transition. Let us call this a *sweet spot* for N. Figure 1 shows that the sweet spots can easily be an order of magnitude less inefficient than other choices of N. The reason for this dramatic difference is that for increasing N/D the sweet spots move very close to integer values of N/D, so that only very few "holes" of less loaded disks remain.

Note that sweet spots are often easy to explore. External memory algorithms in the multi-head model are usually formulated so that they work with arbitrary parallelism as long as we do not exceed the parallelism inherent in the problem. So we can as well adjust N to a sweet spot. Let us consider another example. In a video-on-demand system we can adjust the system in such a way that under full load (maximal number of users requesting maximal bandwidth) the number of blocks accessed per round is just far enough away from the next sweet spot that with high probability

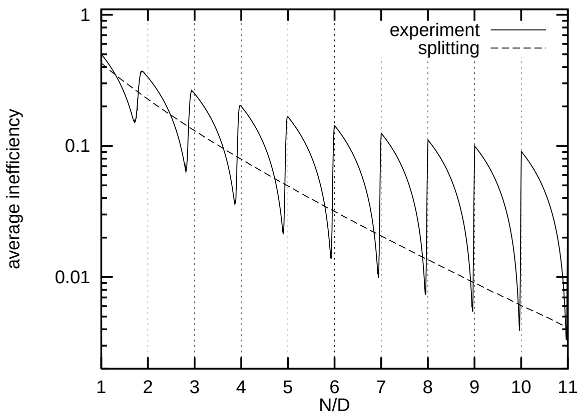

Figure 1: Average inefficiency $(1-N/(DL^*_{\max}))$ of RDA with N blocks to be retrieved. The sawtooth curve is for $D = 256$ and 1000 repetitions. Measurements for $D = 16$ and $D = 64$ indicate that D has little influence on the position of the extrema. The smooth curve is an upper bound for splittable requests (Section 4).

$L_{\max} = \lceil N/D \rceil$. Additional experiments not shown here indicate that such *real time sweet spots* also yield much lower inefficiency than integer N/D. If the load of the video server is lower we are not at a sweet spot but scheduling less blocks cannot overload the system either.

Sweet spots are also worthwhile to explore because the alternative way to achieve a comparable efficiency is to drastically increase N/D. This often is infeasible since many applications do not exhibit enough parallelism, because buffer memory may not be sufficient or because the resulting access latencies become too large for time critical tasks like interactive virtual reality. For example, for $D = 256$ the inefficiency for the sweet spot $N = 3.84D$ is less than 4 % or five times less than the 20 % inefficiency for the nearby $N = 4D$. To achieve the same inefficiency for N a multiple of D we would have to set $N = 24D$, i.e., more than six times larger implying six times larger required parallelism, six times larger buffer space and six times larger latency.

2.2 Analysis: Let us start with the special case $N < D/2$ because we get a simpler and tighter result than for the general case and because there is a particularly efficient scheduling algorithm for this case.

THEOREM 2.1. *For any constant $\epsilon > 0$, $N = (1/2-\epsilon)D$ requests can be scheduled with $L^*_{\max} = 1$ whp. This is possible in linear time.*

Proof. (Outline) From random graph theory [4] it is well known that whp a random graph with D nodes and $N = (1/2-\epsilon)D$ edges consists of connected components which are either trees or unicyclic (a tree plus one additional edge). To schedule the edges of a tree component, pick any root and orient all edges towards their parent. For unicyclic components, remove one edge $\{u, v\}$ from the cycle, orient it from u to v, and use u as a root for scheduling the resulting tree. ■

Another simple linear time scheduling algorithm iteratively removes minimum degree nodes from the allocation graph by changing all its incident edges into self-loops. Pittel Spencer and Wormald [13] have shown that for $N \le 1.67D$ this yields $L_{\max} = L^*_{\max} = 2$ and for $N \le 2.57D$ we get $L_{\max} = L^*_{\max} = 3$. For substantially larger N/D, this algorithm does not compute optimal schedules however since its $L_{\max}$ approaches $2N/D$ for $N/D \to \infty$.

For $N/D > 3$ we go back to computing optimal schedules and derive an upper bound on $L^*_{\max}$. The analysis is a generalization of [16] since it does not require N/D to be integer. The problem is reduced to bounding a one-dimensional smooth function which for fixed N/D does not depend on D.

LEMMA 2.1. *For any constant $b > 1$, let* $f_b : \left[e^{-3}, \frac{b}{\lceil b \rceil}\right) \to \mathbb{R}$,

$$f_b(p) = \frac{1}{p^p(1-p)^{1-p}} \left(p\frac{b}{\lceil b \rceil}\right)^{\lceil b \rceil p} \left(\frac{1-p^2}{1-p\frac{\lceil b \rceil}{b}}\right)^{b-p\lceil b \rceil}.$$

If $f_b(p) < 1$ for its specified range, then $L^*_{\max} \le \lceil b \rceil$ for $N \le bD$ with probability at least $1 - \mathcal{O}(1/D)^{\lceil b \rceil}$.

Bounding $f_b(p)$ can essentially be done "by inspection". Figure 2 shows the plots of the function $f_b(p)$ for $\lceil b \rceil \in \{4, 5, 6, 7, 8, 9\}$ where b was chosen to be the largest integral percentage of $\lceil b \rceil$ for which $f_b(p) < 1$. One can make such an argument more rigorous using interval arithmetic computations (e.g. [8]).

Combining Theorem 2.1 for $\lceil b \rceil = 1$, the heuristics from [13] for $\lceil b \rceil \in \{2, 3\}$ and Lemma 2.1 for larger b we get.

COROLLARY 2.1. *For $b = N/D \le 9$, $L^*_{\max} \le \lceil b \rceil$ whp if $b/\lceil b \rceil$ respects the following constraints:*

$\lceil b \rceil$	1	2	3	4	5	6	7	8
$\frac{b}{\lceil b \rceil} \le$	0.49	0.83	0.85	0.90	0.94	0.96	0.97	0.98

Note that the percentage given above is the efficiency of the resulting schedules.

It remains to prove Lemma 2.1.

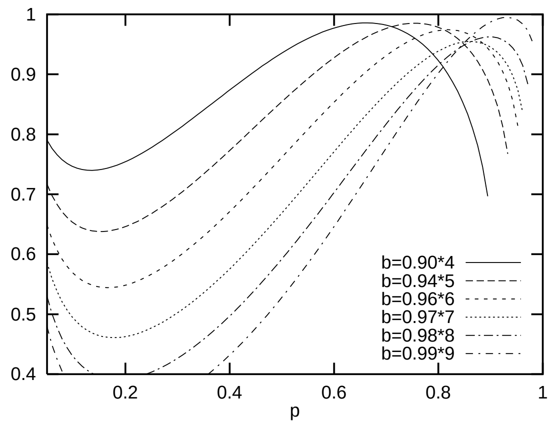

Figure 2: Behavior of $f_b(p)$ for $p \geq e^{-3}$ and small $b = N/D$.

Proof. The schema used in [16] for integral b and for fast scheduling can be adapted to our case of arbitrary b. We therefore only outline the analysis and mention the appropriate form of the most important expressions.

First note that we can concentrate on the case $N = bD$ since adding requests can only increase $L^*_{\max}$. As already explained in Section 1.4, it suffices to bound the sum $\sum_{d=1}^{D}\binom{D}{d}\mathbb{P}\left[L_\Delta \geq d\lceil b\rceil + 1\right]$ from Relation 1.2. This sum is now split into three parts. For very large $d \geq bD/\lceil b\rceil$, $d\lceil b\rceil \geq N$ so that $P_d = 0$.

For small d one can use the estimate

(2.4)
$$\binom{D}{d}\mathbb{P}\left[L_\Delta \geq d\lceil b\rceil + 1\right] \leq \left(\frac{d}{D}\right)^{d(\lceil b\rceil - 1)+1} e^{d(b+1)+1}$$

which follows from elementary yet long-winded calculations with the Chernoff bound (1.3) and the Stirling approximation $\binom{D}{d} \leq (De/d)^d$ for the binomial coefficient. A similar calculation can be found in the proof of Lemma 3.2. Further study of Relation 2.4 shows that for $d < e^{-3}$ and sufficiently large D, the term for $d = 1$ dominates all other summands.[1] We get

$$P_1 \leq \left(\frac{1}{D}\right)^{\lceil b\rceil} e^{b+2} = \mathcal{O}(1/D)^{\lceil b\rceil}$$

For the remaining case $De^{-3} \leq d < \frac{b}{\lceil b\rceil}D$ we set $p := d/D$. Using the Stirling approximation $\binom{D}{pD} = p^{-p}(1-p)^{1-p}\mathcal{O}\left(D^{-1/2}\right)$ and the Chernoff bound (1.3) for $N = bD$ and $x = \lceil b\rceil pD$, P_d can be brought into the form

$$\begin{aligned}
&\mathbb{P}[L_\Delta \geq d\lceil b\rceil] \\
&\leq \left(\frac{bDp^2}{\lceil b\rceil pD}\right)^{\lceil b\rceil pD}\left(\frac{1-p^2}{1-\frac{\lceil b\rceil pD}{bD}}\right)^{bD-\lceil b\rceil pD}\mathcal{O}\left(D^{-1/2}\right) \\
&= \left(\left(p\frac{b}{\lceil b\rceil}\right)^{\lceil b\rceil p}\left(\frac{1-p^2}{1-p\frac{\lceil b\rceil}{b}}\right)^{b-\lceil b\rceil p}\right)^{D}\mathcal{O}\left(D^{-1/2}\right) \\
&= f_b(p)^D\mathcal{O}\left(D^{-1/2}\right) \;.
\end{aligned}$$

If $f_b(p) < 1$, this expression decreases exponentially with D. Even summing $\mathcal{O}(D)$ of these terms yield only an exponentially small probability. ∎

3 Disk Failures

We study what happens when one disk u fails. For simplicity, we so far assumed *independent* RDA where the copies of a block are allocated independently so that it can happen that both end up on the same disk. If this is the failed disk, independent RDA cannot service this request. Therefore we now modify the placement scheme so that the copies are placed on randomly chosen *different* disks. The question arises how the scheduling quality is affected. We show that the same scheduling quality can be achieved as in an intact system with $D-1$ disks for independent RDA whp. This is not obvious since there is no scheduling flexibility for those edges which have one copy allocated on the failed disk.

LEMMA 3.1. *For any constant $b > 1$, let f_b be defined as in Lemma 2.1. If $f_b(p) < 1$ for its specified range, then $L^*_{\max} \leq \lceil b\rceil$ for $N \leq b(D-1)$ with probability at least* $1 - \mathcal{O}(1/D)^{\lceil b\rceil}$.

Hence, Corollary 2.1 also applies for the case of a failed disk if we replace D by $D-1$.

Proof. Witout loss of generality assume disk number D failed. We can model the scheduling problem by an allocation graph with node set $\{1, \ldots, D-1\}$ where requests $\{D, v\}$ affecting the failed disk are represented as a self loop $\{v, v\}$ on the mirror disk v. Now the concept of unavoidable load L_Δ of a set of disks can be applied again. For any request, the probability that it has to be served by some disk in Δ with $|\Delta| = d$ is

$$p^2 = 2\cdot\frac{1}{D}\cdot\frac{d}{D-1} + \frac{d}{D}\cdot\frac{d-1}{D-1} = \frac{d(d+1)}{D(D-1)}$$

Hence, L_Δ is binomially $\mathcal{B}(b(D-1), d(d+1)/(D(D-1)))$ distributed. We can apply the Chernoff bound (1.3) with $p^2 = d(d+1)/(D(D-1)$ and $N = b(D-1)$.

[1] This even holds for $d < \exp((b+1)/(\lceil b\rceil - 1)) < e^{-2}$. The border $d < e^{-3}$ was chosen for compatibility with Section 3.

From now on, the same machinery as in the proof of Lemma 2.1 can be used but some of the calculations are more complicated so that we execute them in more detail. We have

$$\mathbb{P}\left[L^*_{\max} > \lceil b \rceil\right] \leq \sum_{d=1}^{D-1} \binom{D-1}{d} P_d$$

where $P_d = \mathbb{P}\left[L_\Delta \geq d\lceil b \rceil + 1\right]$ for a subset Δ of size d. For $d \geq b(D-1)/\lceil b \rceil$, $P_d = 0$. For small d, Lemma 3.2 yields the bound

$$\binom{D-1}{d}\mathbb{P}\left[L_\Delta \geq d\lceil b \rceil + 1\right] \leq \left(\frac{d+1}{D}\right)^{d(\lceil b \rceil - 1)+1} e^{d(b+2)+1} \ .$$

As before, it can be shown that the term for $d = 1$, dominates all other terms for $d < e^{-3}$ and sufficiently large D.[2] We get

$$P_1 \leq \left(\frac{2}{D}\right)^{\lceil b \rceil} e^{b+3} = \mathcal{O}(1/D)^{\lceil b \rceil}$$

For the remaining case of intermediate d, Lemma 3.3 and the Stirling approximation $\binom{D-1}{\alpha(D-1)} = \alpha^{-\alpha}(1-\alpha)^{(-(1-\alpha)}\mathcal{O}\left(D^{-1/2}\right)$ yields

$$\binom{D-1}{d}\mathbb{P}\left[L_\Delta \geq \alpha(D-1)\lceil b \rceil\right] = \mathcal{O}\left(f_b(\alpha)^{D-1}D^{-1/2}\right)$$

and even $\mathcal{O}(D)$ of these terms yield only an exponentially small probability if $f_b < 1$. ■

It remains to prove the two technical lemmata 3.2 and 3.3. Proofs are given in the Appendix.

LEMMA 3.2.

$$\binom{D-1}{d}\mathbb{P}\left[L_\Delta \geq d\lceil b \rceil + 1\right] \leq \left(\tfrac{d+1}{D}\right)^{d(\lceil b \rceil - 1)+1} e^{d(b+2)+1}.$$

LEMMA 3.3. $\mathbb{P}\left[L_\Delta \geq \alpha(D-1)\lceil b \rceil\right] =$

$$\mathcal{O}\left(\left(\left(p\frac{b}{\lceil b \rceil}\right)^{\lceil b \rceil p}\left(\frac{1-p^2}{1-p\frac{\lceil b \rceil}{b}}\right)^{b-p\lceil b \rceil}\right)^{D-1}\right)$$

3.1 Dynamic Spares: Once one disk i fails, the system is no more safe against further disk failures. However, a previously unused spare disk i' can be brought in to take the role of disk i. To make this possible, the data stored on disk i has to be gathered from the intact disks which store the duplicates. Even if we do this copying at the full speed of disk i', we can expect a performance comparable to that of an intact system with $D-2$ disks. The analysis can proceed as before because the copied blocks are spread over all intact disks and since their location is independent of the location of the blocks accessed for the regular service of the system. Note the difference to classical mirroring systems where the mirror of a failed disk becomes a hot spot because it has to service the ordinary traffic and the copying traffic.

[2] Even $d < \exp((b+2)/(\lceil b \rceil - 1))$ is sufficient.

3.2 Virtual Spares: If we are not willing to keep spare disks idle just for the case of a disk failure we can evenly spread the the data of the failed disk over the intact disks. As for dynamic spares, recovery results in an evenly distributed workload for all disks.

Remapping the data allocated on the failed disk can be done as follows: When the number of intact disks drops from D to $D-1$ by failure of disk i, we invent a *forwarding hash function* h_i mapping logical block numbers to the range $0..D-3$. If a block k has previously been mapped to disks i and j, it will now be mapped to disk j and to a disk determined by $h_i(k)$, namely the $_i(k)$-th disk in some ordering of the set of disks different from j which were intact when h_i was installed.

After the subsequent failure of several disks, it may happen that a block is forwarded from one failed disk to another failed disk. But this process must eventually terminate since a block is never forwarded back to a previously failed disk. Furthermore, as long as only a constant fraction of the disks fails, the expected number of forwarding steps is constant. Even if all disks but one disk eventually fail, the expected number of forwarding steps is only $\mathcal{O}(\log D)$.

Another adaptive way to map blocks using hash functions is descibed in [5].

4 Splittable Requests

So far we have assumed that only one copy of a requested block should be accessed. If the blocks are very long, it makes sense however to assemble the requested data from pieces of both copies. For example, if we have only one block, we need only about half the time if we retrieve half of it from each copy. We imagine each block to be subdivided into U subblocks for some parameter U. Accessing k subblocks is assumed to take time k/U.

This situation can be modeled by an allocation graph as before by representing each request as U parallel edges. Note that the allocation graph is no random graph any more. Parallel edges can be stored implicitly. In a schedule graph we are only interested in the number of parallel edges oriented in each of the two possible directions. The maximum flow computations used for finding optimal schedules can also work with this implicit representation of parallel edges at the small penalty of turning unit capacity flow problems into general flow problems with integer capacities.

The main advantage of allowing splits is that we do not have to rely on the sweet spots. We can target any $L_{\max} = b'$ which is a multiple of $1/U$ and the following Lemma tells us that deciding whether we can achieve that can be done as before if we substitute $\lceil b \rceil$ by b'. Figure 1 plots an upper bound on the achievable inefficiency as a function of b.

LEMMA 4.1. *For any constant $b > 1$ and any $b' > b$ such that $b'U \in \mathbb{N}$, let $f_{bb'} := f_b[\lceil b \rceil \leftarrow b']$. If $f_{bb'}(p) < 1$ for its specified range, then if requests are splittable, $L^*_{\max} \leq b'$ for $N \leq bD$ with probability at least $1 - \mathcal{O}(1/D)^{b'-1}$. The same is true in case of a disk failure for $N \leq b(D-1)$.*

The proof is almost completely analogous to the proofs of lemmata 2.1 and 3.1. We get a higher failure probability however because now from $L_\Delta > |\Delta|\, b'$ we cannot conclude that $L_\Delta \geq |\Delta|\, b' + 1$. For the case $|\Delta| = 1$ this makes a big difference.

Refinements: Splittable requests could be analyzed even more elegantly by allowing continuously subdividable blocks, i.e., by making the limit transition $U \to \infty$. There is a problem however. In reality, accessing a block in two pieces needs more work than accessing it in one piece because there is a startup overhead (seek time etc.). If the "optimal" schedule assigns many very small pieces to some disk u, these unaccounted startup overheads can accumulate to a load much larger than the load due to accessing the actual data. For example, for $N = 0.4D$ we know from random graph theory that there is a node u with degree $\Theta(\log D/\log\log D)$ which is part of a tree component whp. An optimal schedule for this tree would result in a schedule with the same number of startup overheads accumulating at node u.

This problem can be limited by choosing the parameter U as a small constant so that the access time for a subblock dominates the startup overhead. There is a tradeoff between a large U with high scheduling flexibility and a small U with a small error in the model. Here is a heuristics which partially combines the advantages of both models: First find a schedule G_1 for a small U. Commit those blocks which are not split by G_1 and account for their access time and for the startup overheads of the split blocks. Then set U to a large value and find an optimal schedule for the remaining problem.

5 Variable Size Requests

Now assume that a disk needs time t_i to read the data corresponding to block i. We call t_i the *size* of request i. The maximum request size is used as the unit, i.e., $t_i \leq 1$. Let $S := \sum_i t_i$ denote the sum of the request times. If the requests are additionally unsplittable, optimal scheduling becomes a difficult problem:

THEOREM 5.1. *Optimal scheduling of unsplittable variable size requests is NP-hard even for different disk random placement and $D = 2$.*

Proof. For $D = 2$ and different RDA allcoation, the problem becomes equivalent to the PARTITION problem [7]. ∎

Indeed, the main motivation for studying splittable requests in the first place was to escape the difficulty of scheduling variable size unsplittable requests. When splitting is allowed, scheduling is quite easy. It can be done in the same way as described in the previous section using $\mathcal{O}(\log(L^*_{\max}U))$ maximum flow computations if requests are split into subblocks of size $1/U$.

It remains the question of how large $L^*_{\max}$ is for RDA. The answer turns out to be remarkably simple: We can use the same bounds as derived before if we replace N by S. So, for the worst case combination of request sizes we get the same bounds as for an instance where request sizes are "maximally" uneven, i.e., there are $\lfloor S \rfloor$ unit size requests and $N - \lceil S \rceil$ requests of size zero.

LEMMA 5.1. *Lemma 4.1 also applies to variable request sizes if N is substituted by S.*

Proof. The probabilistic analysis of RDA for unit size requests is based on the Chernoff bound [6]

$$\mathbb{P}\left[X \geq (p+t)N\right] \leq \left(\left(\frac{p}{p+t}\right)^{p+t}\left(\frac{q}{q-t}\right)^{q-t}\right)^N$$

for the sum of N unweighted independent 0-1 random variables. Therefore, the proof for the variable size case is a simple corollary of the generalization proved in Lemma 5.2 which is a stronger form of the weighted Chernoff bound by Raghavan [14]. ∎

LEMMA 5.2. *Let X_i, $i \leq i \leq N$ denote independent 0-1 random variables with $\mathbb{P}[X_i = 1] = p$. Let $X = \sum_i t_i X_i$ where t_i is a real valued weight between zero and one. Let $S = \sum_i t_i$, $q = 1 - p$ and $t \geq 0$. Then,*

$$\mathbb{P}\left[X \geq (p+t)S\right] \leq \left(\left(\frac{p}{p+t}\right)^{p+t}\left(\frac{q}{q-t}\right)^{q-t}\right)^S.$$

Proof. The proof begins like every proof of Chernoff bounds by exponentiation of λX for a positive parameter λ we are free to choose and an application of the Markov inequality.

$$\mathbb{P}\left[X \geq (p+t)S\right] = \mathbb{P}\left[e^X \geq e^{(p+t)S}\right] \leq \frac{\mathbb{E}e^{\lambda X}}{e^{\lambda(p+t)S}}$$

Next, $\mathbb{E}e^{\lambda X}$ is evaluated using the properties of the exponential function and the independence of the variables.

$$\begin{aligned}
\mathbb{E}e^{\lambda X} &= \mathbb{E}e^{\lambda \sum_{i=1}^N t_i X_i} = \mathbb{E}\prod_{i=1}^N e^{\lambda t_i X_i} \\
&= \prod_{i=1}^N \mathbb{E}e^{\lambda t_i X_i} = \prod_{i=1}^N (pe^{\lambda t_i} + q) \\
&\leq \prod_{i=1}^N (pe^{\lambda} + q)^{t_i} = (pe^{\lambda} + q)^{\sum_{i=1}^N t_i} = (pe^{\lambda} + q)^S
\end{aligned}$$

The estimate is based on the fact that $f(p) := (pe^{\lambda t} + q) \leq g(p) := (pe^{\lambda} + q)^t$ for $t \in [0,1]$ since both agree for $p \in \{0,1\}$, $f(p)$ is linear, and $g(p)$ is convex.

The rest of the argument is as in [6]. By optimizing λ using calculus we get

$$\begin{aligned}
\mathbb{P}\left[X \geq (p+t)S\right] &\leq \frac{(pe^{\lambda} + q)^S}{e^{\lambda(p+t)S}} \\
&\leq \left(\left(\frac{p}{p+t}\right)^{p+t}\left(\frac{q}{q-t}\right)^{q-t}\right)^S .
\end{aligned}$$

∎

6 Communication Bottlenecks

In a system with many disks, it is unrealistic to assume that all disks are directly connected to the processor. We now study a more realistic configuration which allows scaling to many disks with current technology. We assume that the processors are tightly coupled over a shared memory or a high speed interconnect. The SUN E10000 or the SGI Origin2000 would be typical examples. Several I/O busses (e.g. 64bit PCI) are connected to the high speed interconnect. Each I/O bus supports several disk controllers (e.g. SCSI or FibreChannel) and each disk controller supports several disks. More generally, we have a tree-shaped hierarchy of components. In the basic case [1] we have $h = 1$ levels of hierarchy and in our example $h = 3$.

Fault tolerance now demands that copies of a block are in different subtrees of the root (e.g. on disks connected to controllers on different PCI-busses). Our probabilistic analysis can consider each level separately. For a fixed level, the only difference to our previous calculations is that the nodes servicing a copy are chosen from two disjoint sets. We do not execute the new calculations here since they are involved and unlikely to yield new insights. Vöcking has analysed a related situation [20] and concludes that placing copies into disjoint sets is not much different to choosing arbitrary locations for large D.

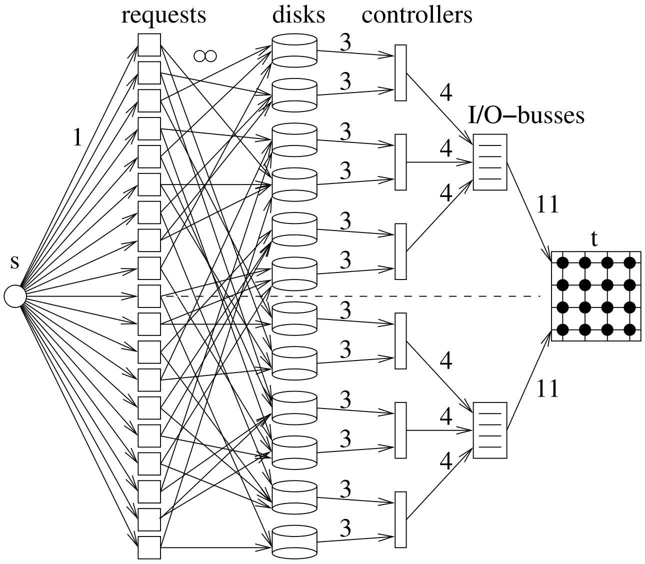

Figure 3: Example for a flow network that can be used to schedule 19 requests to 12 disks connected to the processors via a tree of 6 controllers and 2 I/O-busses. In this system it is likely that the controllers are the most difficult to schedule resource.

What has to be changed for the scheduling algorithm? We go away from the formulation using edge orientation and generalize a formulation due to Aerts, Korst and Egner [1] which uses a flow network with separate nodes for requests and disk. We have $h + 2$ layers. Layer 0 only contains the source node s and layer $h + 1$ only contains the target node t. Layer 1 contains one node for each request and the layers $2 \leq i \leq h + 1$ contain one node for each hardware resource (e.g. disk, SCSI bus, PCI bus). Edges connect subsequent layers. There is an edge from s to request i with capacity t_i and an infinitye capacity edge from each request to the disks able to retrieve this copy. Nodes u and v on levels j and $j + 1$, $j > 1$ are connected if resource u is connected to component v. The capacity of (u, v) is the number of subblocks component u is able to transfer in the available time T. As before, T can be found by binary search or by trying values suggested by the probabilistic analysis. Figure 3 gives an example.

Refinements: The layered scheduling approach presented above can handle even more complex systems. Technology is evolving so fast that disks and controllers bought one or two years after the initial machine are likely to be different (larger, faster) from the original parts. This can be partially modeled by adjusting the edge capacities accordingly.

The network can be an arbitrary directed acyclic graph. Even cycles can be modeled in certain cases. For example, consider the important case that processors are connected over a tree hierarchy of crossbars. (A fat tree [10] where the bandwidth of the upper levels of the tree may not completely scale with the number of subordinate processors.) Data first travels up the hierarchy (not necessarily all levels) and then down. Therefore it suffices to split the hierarchy into an upward copy and a downward copy to get an acyclic graph. The penalty we have to pay is that the capacity of the links has to be split a priori between upward and downward direction although this may not always be optimal. The network we get is a collection of small trees leading from the disks to the processors they are attached to. Then the upward part of the interprocessor interconnection network and then the downward part with the processors as the last level. (Note that there will be infinity capacity links between the two incarnations of each processor and each node of the fat tree network.) The capacity of the edges leaving the processors say how many requests each processor can handle. Unfortunately, we have no control over which block ends up on what processor. For some applications this is not a restriction but an additional feature since the resulting schedule will be one with high locality. In this setting it also makes sense to add edges from requests to additional cached copies of the block.

A probabilistic analysis of such a system is considerably more complicated than in the previous sections. Still, there are some rules. For example, the probability to place a block at some disk should be chosen in such a way that the ratio between the expected load of each resource and its capacity is everywhere about the same. One interesting consequence is that new high capacity disks are not easy to fill with data since the speed of disks typically evolves more slowly than their capacity. The excess storage capacity can be used to store infrequently accessed data. We get the seemingly paradoxical situation that the most advanced disks will (partially) be used for the least important data.

If we want to predetermine the target of some or all request we get a more complicated scheduling problem that can still be solved using linear programming or multicommodity flow.

7 Discussion

Although we have modeled important aspects of practical parallel disk systems, there remain a few problems. In particular, we always assume that the time t_i needed to service a request is known in advance. In reality, due to seek overhead and rotational delays, the actual access time depends on what other requests are serviced on this disk. Finding optimal schedules which take this interdependence into account is NP-hard even on a single disk. No scheduling algorithms exploiting the additional flexibility of RDA to minimize overheads have been considered yet.

Although this restriction relativizes the efficiencies close to one predicted by our model there are still important situations where very high efficiencies can be achieved. If the blocks occupy one or more entire tracks there is no rotational delay and the seek overhead can be partially amortized over the requests in a batch by ordering the requests by track number on each disk.

Acknowledgements

I would like to thank Sebastian Egner and Jan Korst for valuable discussions.

References

[1] Aerts, J., Korst, J., and Egner, S. Random duplicate storage for load balancing in multimedia servers. *Information Processing Letters* (2000). accepted.

[2] Aerts, J., Korst, J., and Verhaegh, W. Load balancing in multimedia servers. In *Workshop on Project Management and Scheduling PMS* (2000), pp. 25–28.

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

[4] Bollobás, B. *Random graphs.* Academic Press, 1985.

[5] Brinkmann, A., Salzwedel, K., and Scheideler, C. Efficient, distributed data placement strategies for storage area networks. In *12th ACM Symposium on Parallel Algorithms and Architectures* (2000), pp. 119–128.

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

[7] Garey, M. R., and Johnson, D. S. *Computers and Intractability.* Freeman and Company, New York, 1979.

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

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

[10] Leiserson, C. E. Fat trees: Universal networks for hardware efficient supercomputing. In *International Conference on Parallel Processing* (1985), pp. 393–402.

[11] Lenstra, J. K., Shmoys, D. B., and Tardos, E. Approximation algorithms for scheduling unrelated parallel machines. *Mathematical Programming 46* (1990), 259–271.

[12] Mitzenmacher, M., Richa, A., and Sitaraman, R. The power of two random choices: A survey of the techniques and results. In *Handbook of Randomized Computing*, P. Pardalos, S. Rajasekaran, and J. Rolim, Eds. Kluwer, 2000.

[13] 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.

[14] Raghavan, P. Probabilistic construction of deterministic algorithms: Approximating packing integer programs. *Journal of Computer and System Sciences 37* (1988), 130–143.

[15] Sanders, P. Asynchronous scheduling of redundant disk array. In *12th ACM Symposium on Parallel Algorithms and Architectures* (2000), pp. 98–98.

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

[17] 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.

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

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

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

A Proof of Lemma 3.2

Proof. First, we estimate $\binom{D-1}{d} \leq ((D-1)e/d)^d = ((D-1)/d)^d e^d$ using the Stirling approximation. Now, setting $x = d\lceil b\rceil + 1$, $p^2 = \frac{d(d+1)}{D(D-1)}$, and $N = b(D-1)$ in Relation (1.3), we get $\mathbb{P}\left[L_\Delta \geq d\lceil b\rceil + 1\right] \leq f \cdot g$ where

$$f = \left(\frac{bd(d+1)/D}{d\lceil b\rceil + 1}\right)^{d\lceil b\rceil+1} \quad \text{and}$$

$$g = \left(\frac{1 - \frac{d(d+1)}{D(D-1)}}{1 - \frac{d\lceil b\rceil+1}{b(D-1)}}\right)^{b(D-1)-d\lceil b\rceil-1} .$$

We have

$$f \leq \left(\frac{b(d+1)}{D\lceil b\rceil}\right)^{d\lceil b\rceil+1} = \left(\frac{d+1}{D}\right)^{d\lceil b\rceil+1}\left(\frac{b}{\lceil b\rceil}\right)^{d\lceil b\rceil+1}$$

$$\leq \left(\frac{d+1}{D}\right)^{d\lceil b\rceil+1}\left(\frac{b}{\lceil b\rceil}\right)^{d\lceil b\rceil} \leq \left(\frac{d+1}{D}\right)^{d\lceil b\rceil+1} e^{d(b-\lceil b\rceil)}$$

where the last estimate stems from the relation

$$\left(\frac{b}{\lceil b\rceil}\right)^{\lceil b\rceil} = \left(1 + \frac{b-\lceil b\rceil}{\lceil b\rceil}\right)^{\lceil b\rceil} \leq e^{b-\lceil b\rceil} .$$

Now we turn to g.

$$g = \left(1 + \frac{d\lceil b\rceil + 1 - bd(d+1)/D}{b(D-1) - d\lceil b\rceil - 1}\right)^{b(D-1)-d\lceil b\rceil-1}$$

$$\leq \left(1 + \frac{d\lceil b\rceil + 1}{b(D-1) - d\lceil b\rceil - 1}\right)^{b(D-1)-d\lceil b\rceil-1} \leq e^{d\lceil b\rceil+1}$$

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

$$\binom{D-1}{d}\mathbb{P}\left[L_\Delta \geq d\lceil b\rceil + 1\right]$$

$$\leq \left(\frac{(D-1)(d+1)}{dD}\right)^d \left(\frac{d+1}{D}\right)^{d(\lceil b\rceil-1)+1} e^{d+d(b-\lceil b\rceil)+d\lceil b\rceil+1}$$

$$\leq \left(1 + \frac{1}{d}\right)^d \left(\frac{d+1}{D}\right)^{d(\lceil b\rceil-1)+1} e^{d(b+1)+1}$$

$$\leq \left(\frac{d+1}{D}\right)^{d(\lceil b\rceil-1)+1} e^{d(b+2)+1}$$

■

B Proof of Lemma 3.3

Proof. Setting $d = \alpha(D-1)$, $x = d\lceil b\rceil$, $p^2 = d(d+1)/(D(D-1))$, and $N = b(D-1)$ in Relation (1.3), we get $\mathbb{P}\left[L_\Delta \geq \alpha(D-1)\lceil b\rceil\right] \leq f \cdot g$ where

$$f = \left(\frac{b}{\lceil b\rceil} \cdot \frac{\alpha(D-1)+1}{D}\right)^{\alpha\lceil b\rceil(D-1)} \quad \text{and}$$

$$g = \left((1 - \frac{\alpha(\alpha(D-1)+1)}{D})/(1 - \frac{\alpha\lceil b\rceil}{b})\right)^{(D-1)(b-\alpha\lceil b\rceil)} .$$

We have

$$f = \left(\alpha\frac{b}{\lceil b\rceil}\right)^{\alpha\lceil b\rceil(D-1)} \left(1 + \frac{1/\alpha}{D} - \frac{1}{D}\right)^{(D-1)\alpha\lceil b\rceil}$$

$$\leq \left(\alpha\frac{b}{\lceil b\rceil}\right)^{\alpha(D-1)\lceil b\rceil} e^{\lceil b\rceil} \quad \text{and}$$

$$g = \left(\frac{1 - \alpha^2 - \frac{\alpha(1-\alpha)}{D}}{1 - \frac{\alpha\lceil b\rceil}{b}}\right)^{(b-\alpha\lceil b\rceil)(D-1)}$$

$$\leq \left(\frac{1-\alpha^2}{1 - \frac{\alpha\lceil b\rceil}{b}}\right)^{(b-\alpha\lceil b\rceil)(D-1)} .$$

Since $\lceil b\rceil$ is a constant the Lemma follows by multiplying the bounds for f and g. ■